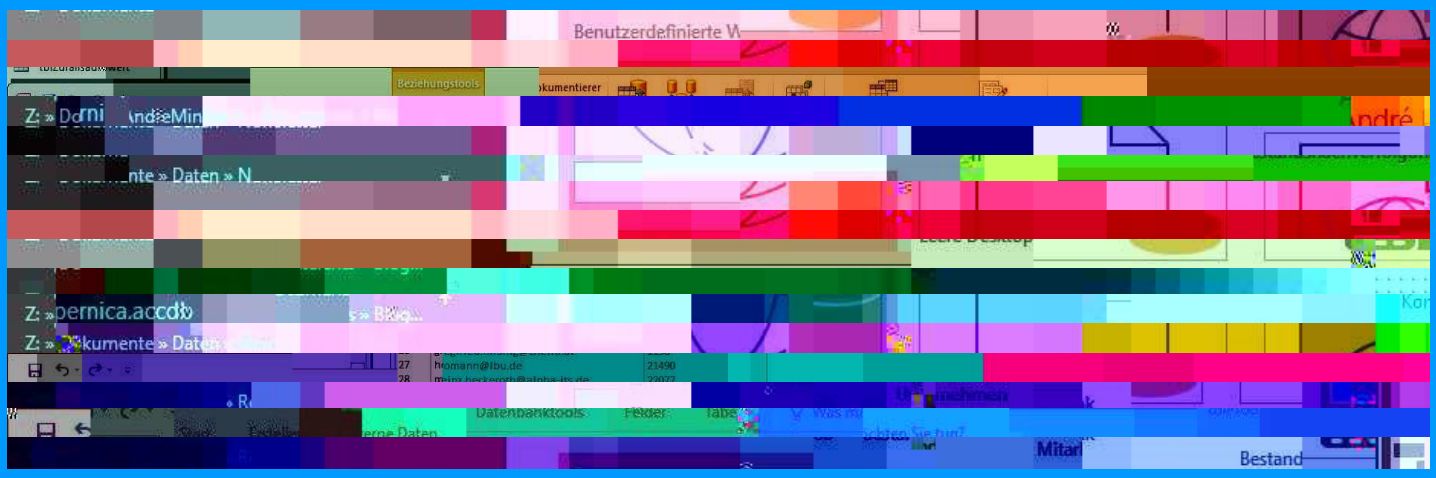
KOMPENDIUM

ACCESS BASICS

DAS ACCESS-MAGAZIN FÜR ALLE,
DIE VON 0 AUF 100 WOLLEN



KOMPENDIUM

ALLE AUSGABEN VON 2010-2019

FAST 2.500 SEITEN ACCESS-KNOW-HOW

ÜBER 300 BEISPIELDATENBANKEN

www.access-basic.de

André Minhorst Verlag

André Minhorst – Access [basics] Kompendium 2019

ISBN 978-3-944216-09-6
© 2020 André Minhorst Verlag, Borkhofer Straße 17, 47137 Duisburg/Deutschland
1. Auflage 2020
Lektorat Carsten Gromberg/André Minhorst
Korrektur André Minhorst
Cover/Titelbild André Minhorst
Typographie, Layout und Satz André Minhorst
Herstellung André Minhorst

Das vorliegende Werk ist in all seinen Teilen urheberrechtlich geschützt. Alle Rechte vorbehalten, insbesondere das Recht der Übersetzung, des Vortrags, der Reproduktion, der Vervielfältigung auf fotomechanischem oder anderen Wegen und der Speicherung in elektronischen Medien. Ungeachtet der Sorgfalt, die auf die Erstellung von Text, Abbildungen und Programmen verwendet wurde, können weder Verlag noch Autor, Herausgeber oder Übersetzer für mögliche Fehler und deren Folgen eine juristische Verantwortung oder irgendeine Haftung übernehmen.
Die in diesem Werk wiedergegebenen Gebrauchsnamen, Handelsnamen, Warenbezeichnungen et cetera können auch ohne besondere Kennzeichnung Marken sein und als solche den gesetzlichen Bestimmungen unterliegen.
Die in den Beispielen verwendeten Namen von Firmen, Produkten, Personen oder E-Mail-Adressen sind frei erfunden, soweit nichts anderes angegeben ist. Jede Ähnlichkeit mit tatsächlichen Firmen, Produkten, Personen oder E-Mail-Adressen ist rein zufällig.

Inhalt

Access - eine Einführung 27
Kunden verwalten 52
Entwurf einer einfachen Kundentabelle 57
Formulare für einfache Tabellen 60
Datenbankfenster und Navigationsbereich im Detail 67
Kunden und Projekte verwalten 72
Einfache Auswahlabfragen kennenlernen 81
Darstellung von 1:n-Beziehungen 88
Formulare für Daten aus 1:n-Beziehungen 91
Anreden und Co. mit Wertlisten oder Lookup-Tabellen verwalten 98
Einfache Tabelle in der Detailansicht 102
Anfügeabfragen 107
Daten aus CSV-Dateien importieren 114
Mitarbeiter per m:n-Beziehung zu Projektteams zusammenstellen 122
m:n-Beziehung per Unterformular 128
Fremdschlüsselfelder mit Nachschlage-Funktion: Pro und Kontra 133
Vom Textfeld zur Lookuptabelle 137
Formularereignisse nutzen 142
Daten aus verknüpften Tabellen darstellen 147
Einführung in VBA 151
Formulare öffnen und schließen 157
Die Schaltfläche 164
Das Listenfeld 167
Löschabfragen 175
Eigene Routinen schreiben 178
If...Then- und Select Case-Bedingungen 181
Filtern von Textfeldern 187
Einsatz von Parametern 192
Filtern nach Zahlen 194
Variablen in VBA 199
m:n-Beziehungen in Listenfeldern darstellen 203
Felder und ihre Eigenschaften 211
Daten zusammenführen mit UNION-Abfragen 220
Das Textfeld 224
Das Kombinationsfeld 228
Datensatzauswahl per Kombinationsfeld 233
Eigenschaften von Tabellen 240

Der Ausdrucks-Generator 243
Berechnete Felder in Abfragen und Eigenschaften 248
Benutzerdefinierte Funktionen in Abfragen 252
Domänenfunktionen 254
Abfrageeigenschaften 261
Adressblöcke in Berichten 264
Bedingungen: Wenn, Schalter und Wahl 269
Abhängige Kombinationsfelder 271
Standardwerte statisch und dynamisch einstellen 275
Projektzeiterfassung - Aufgaben und Tätigkeiten verwalten 277
Indizierung von Feldern 286
Duplikate per Abfrage finden 291
Einfache Tabelle in der tabellarischen Ansicht darstellen 293
Darstellung von 1:n-Beziehungen per Gruppierung 298
Datum und Zeit, Teil I 302
Validieren von Benutzereingaben 308
Access-Funktionen: Datum und Zeit 316
Abfragen: Datum und Zeit 322
Schleifen in VBA 325
Werte zu Kombinationsfeldern hinzufügen 330
Aktionsabfragen per VBA ausführen 336
Daten per Kombinationsfeld auswählen und löschen 341
Meldungsfenster anzeigen und auswerten 343
Daten in der Endlosansicht von Formularen anzeigen 346
CSV-Export für Adressdaten 351
Im- und Export-Spezifikationen definieren und verwenden 357
Datenbanken aufteilen 360
Datenbanken im Mehrbenutzerbetrieb 364
Tabellenverknüpfungen pflegen 367
Aktion beim Start: Das Autoexec-Makro 373
Dialog zur Auswahl von Dateien anzeigen 375
Fehler unter VBA 378
Fehlerbehandlung unter VBA 382
Lookup-Daten per Formular verwalten 388
Recordsets in der Datenblattansicht 395
Formular- und Berichtsvorlagen 399
Datensätze in Unterformularen anlegen 403
Fehlerbehandlung in der Praxis 406
Fehlerbehandlung in Formularen 411
Kombinationsfeld als Eingabehilfe für Textfelder 417

Optionsgruppen und Optionsfelder 420
Berichtsbereiche 426
Sortieren in Berichten 429
Tipps und Tricks: Standardverzeichnis einstellen 432
Tipps und Tricks: Objekte ausblenden 432
Tipps und Tricks: Ausgeblendete Objekte anzeigen 433
Tipps und Tricks: Aktuelles Datum und Zeit blitzschnell 433
Tipps und Tricks: Wert aus vorherigem Datensatz übernehmen 434
Tipps und Tricks: Datum und Zeit im Formulartitel 434
Tipps und Tricks: Memofeld als Sortierungs- und Gruppierungsfeld 435
Tipps und Tricks: Datenbankkennwort vergeben 435
Tipps und Tricks: Textbausteine verwenden 436
Datenblattansicht im Griff 438
Aktuellen Datensatz kopieren 442
Formulare und Steuerelemente referenzieren 444
Berichte öffnen 448
DAO: Datenbank im Griff mit dem Database-Objekt 453
For Each-Schleifen 456
Bedingte Formatierung 460
Mehrspaltige Berichte 465
Zeichenketten-Funktionen 470
Zeilenumbruch und Tabulator in Access 475
DAO: Recordsets öffnen, lesen und bearbeiten 476
Mehrfachauswahl in Listenfeldern auslesen 482
DAO: Recordsets filtern, durchsuchen und sortieren 486
SQL: SELECT-Abfragen, Teil I 490
Kunden per Bezeichnung verwalten 496
SQL: SELECT-Abfragen, Teil II 503
Einfache Suchfunktion für Formulare 509
Arrays 520
Kommunikationsdaten im Griff 526
Bestellungen in Tabellen verwalten 531
Bestellungen in Formularen verwalten 536
Rechnen in Formularen 543
Rechnen in Berichten 548
Gruppierungen in Berichten 553
Bericht als PDF speichern 561
Tipps und Tricks zu Berichten: Duplikate ausblenden 564
Tipps und Tricks zu Berichten: Vergrößerbar 565
Tipps und Tricks zu Berichten: Verkleinerbar 565

Tipps und Tricks zu Berichten: Berichte zusammenfassen 567
Daten für die Ausgabe formatieren 569
Die Format-Funktion 573
Farbige und vom Wert abhängige Formatierung 578
Listenfeldauswahl speichern und wiederherstellen 580
Ereignisse in Formularen, Teil 1: Reihenfolge 585
Null-Werte und wie man damit umgeht 590
Gruppierungen in Abfragen 593
Kreuztabellenabfragen 598
Schnellsuche im Listenfeld 604
Ereignisse in Formularen: Öffnen und Schließen 608
Kreuztabellenabfragen nach Datum 613
Dateibefehle 616
Textdateien schreiben und lesen 620
Rechnungsbericht 626
Die Dir-Funktion 632
Binärzahlen in VBA-Eigenschaften 634
Kombinations- und Listenfelder mit Wertliste füllen 638
Kombinations- und Listenfelder per Callback füllen 641
Das ListView-Steuerelement - Grundlagen 647
Access und Excel: Import und Export 653
Die Umschaltfläche 660
ListView-Steuerelement mit Daten füllen 666
Zugriff auf Daten im ListView-Steuerelement 670
AutoKeys: Globale Tastenkombinationen 676
Access und Excel: Verknüpfungen 679
Beispieldaten-Assistent 682
Early Binding und Late Binding 687
ListView-Steuerelement: Daten bearbeiten 691
Datensätze individuell auswählen 696
Synchrone Unterformulare 701
Internetmarke mit Access nutzen 706
Das Registersteuerelement 717
ListView-Steuerelement: Kontrollkästchen nutzen 722
Aktionsabfragen per VBA 725
Dynamische Datenherkunft mit Formularbezug 728
Internetmarke mit Access nutzen, Teil II 732
Das Registersteuerelement in der Praxis 737
SQL-Grundlagen: Aktionsabfragen 744
DAO: QueryDefs und Parameterabfragen unter VBA 748

ListView-Steuerelement: Drag and Drop, Teil I 752
Programmieren mit Arrays 761
Reihenfolge individuell einstellen 766
Feiertage verwalten, Teil 1 774
Debugging im VBA-Editor, Teil 1 783
ListView-Steuerelement: Drag and Drop, Teil II 788
Feiertage verwalten, Teil 2 796
Tipps und Tricks: Ereignisprozeduren schneller anlegen 801
Tipps und Tricks: Monatsnamen und Wochentage 802
Tipps und Tricks: Texte in leeren Feldern vorblenden 802
Steuerelementlayouts mit Access 2007 und jünger 804
Balkendiagramme mit Bordmitteln 811
Haupt- und Unterformular synchron 817
Debugging im VBA-Editor, Teil 2 820
Filterkriterien für Formulare, Teil I 826
Textfelder: Value, OldValue und Text 834
Änderungen protokollieren, Teil I 838
Berichte per E-Mail verschicken, Teil I 843
Filterkriterien für Formulare, Teil II 849
E-Mails mit Outlook versenden 857
Berichte per E-Mail versenden, Teil II 863
E-Mails sicher versenden 872
Ereignisse abgreifen 877
Filterkriterien für Formulare, Teil III: Kombinationsfelder 881
Textfelder im Griff, Teil II: Markierungen 887
Rekursion mit VBA 894
Platzhalter in Texten ersetzen 901
Berichte per E-Mail verschicken, Teil III 906
Das TreeView-Steuerelement: Grundlagen, Teil I 909
Das TreeView-Steuerelement: Grundlagen, Teil II 916
Bilder in Access, Teil I 924
Das Anlage-Feld 930
Anlage-Felder und VBA 937
Das TreeView-Steuerelement: Verknüpfte Daten, Teil I 943
TreeView und Unterformulare 948
Bilder in Access, Teil II: Bilder in OLE-Feldern 953
Kontextmenüs 959
Das TreeView-Steuerelement: Verknüpfte Daten, Teil II 967
Das Bild-Steuerelement 973
Bilder in Access, Teil III: Bilder aus OLE-Feldern anzeigen 979

Das ImageList-Steuerelement 982
Kontextmenüs im TreeView, Teil I: Anlegen und Löschen 989
Das Kontrollkästchen 1000
Anwendungsparts 1004
Autowerte 1010
Datensätze kopieren mit Bordmitteln 1015
Neue Autowerte ermitteln und RecordsAffected 1021
GUIDs 1024
Datensätze per VBA kopieren 1028
Verknüpfte Daten kopieren 1033
Programmieren mit Makros, Teil I 1041
Programmieren mit Makros, Teil II 1062
Fehlerbehandlung in Makros 1070
Datenmakros I: Grundlagen 1075
Vertrauenswürdige Datenbanken 1084
Vertrauenswürdigkeit per Makro prüfen 1090
Datenmakros II: Datenoperationen 1092
TempVars unter VBA 1104
Programmieren mit Collections 1109
Mehrere Formularinstanzen anzeigen 1113
Ereignisse in Formularen: Zeitgeber 1121
Detailformulare aufrufen 1124
Datenbankereignis beim Schließen 1129
Datenmakros III: Ereignisse 1133
Datensätze filtern per Kombinationsfeld 1145
Von Formular zu Formular mit OpenArgs 1151
Arrays mit Split und Join 1158
Finetuning für Steuerelemente 1160
Access-Berichte drucken per VBA 1164
Anwendungseinstellungen abspeichern 1171
Registry-Zugriff per VBA 1177
Filtern und Suchen 1184
Zufallszahlen generieren und verwenden 1192
Tabellen bearbeiten mit dem TableDef-Objekt 1199
Etiketten drucken per Assistent 1206
Ex- und Import zu Microsoft Word 1214
Rund um den VBA-Editor 1223
Adressen konsolidieren 1229
VBA-Funktionen in Abfragen 1236
Access-Systemtabellen 1243

Diagrammformulare 1251
Datenbanken und Objekte sichern 1262
Tabellen und Abfragen über DAO verwalten 1268
Die VBA-With-Anweisung 1278
Zippen mit Bordmitteln 1284
Navigationsbereiche gestalten 1292
Access als MP3-Player 1296
Textdateien exportieren und einlesen 1304
Kodierte Texte mit VBA umwandeln 1310
Navigieren über Formulartricks 1324
Nützliche Helferlein per VBScript 1334
Daten ins Web exportieren 1345
Reimen leicht gemacht 1356
Datenbankvorlagen und Designs 1365
Datentypen konvertieren unter VBA 1374
Mathematische VBA-Funktionen 1380
Arbeiten mit externen Bibliotheken 1385
Das Web im Formular 1389
Besseres Code-Layout 1397
ADODB als Alternative zu DAO 1405
Frontends aktualisieren 1416
Arrays als Funktionsergebnis 1421
Rekursive Tabellen 1426
XML-Import und -Export 1432
Recordset-Typen 1446
DAO-Objekte und -Auflistungen 1452
Datenbanken über Verknüpfung starten 1458
DoCmd- und RunCommand-Anweisungen 1467
Navigation in und zwischen Formularen 1477
DAO-Objekte und -Auflistungen, Teil II 1484
Kommunikationsdaten als Bericht 1491
Mehrspaltiger gruppierter Bericht 1497
Bericht als Unterformular 1503
Zusätzliche Steuerelemente mit MSForms 1507
Bubble Sort 1515
Klassenmodule und benutzerdefinierte Typen 1517
Horoskopgenerator 1527
Alles plausibel? 1541
SELECT-Abfragen in UNION 1550
Tastensteuerung abfangen 1557

Google- und Bing-Maps im Formular 1562
Elemente für die Bildanzeige 1573
Unterabfragen als Feldinhalt 1582
Textboxen im Detail 1594
Designs für Steuerelemente 1599
Währungen runden 1609
Daten sichern über XML 1616
Fixierte Formulare 1620
Seitennavigation 1626
Datenbanken von Altlasten säubern 1631
Flying Controls 1637
Zugriff auf Server über ODBC 1643
Ribbon-Programmierung, Teil I 1651
Zugriff auf Server über ODBC, Teil II 1657
Lokaler Webshop 1665
Steuerelemente im Eigenbau: Range Control 1678
Rezepte verwalten 1685
Nährwertdatenbank 1692
Lokaler Webshop, Teil II 1701
Datei- und Verzeichnisoperationen mit der Shell 1712
Erfahrungen mit LibreOffice Base 1722
Outlook-E-Mails importieren 1730
Ribbons, Teil II: Bilder im Ribbon 1741
Ereignisse anderer Objekte nutzen 1746
Formularpositionen lesen und setzen 1751
Mehrere Instanzen eines Formulars 1757
Anwendungs-Design mit positionierten Formularen 1768
Berichte zur Laufzeit filtern 1778
Tabelleninhalte verschlüsseln 1786
Ribbons, Teil III: Interaktion im Ribbon 1794
Platzhalter in Textdateien füllen 1802
Der Webbrowser als Bildanzeige 1812
Indizierung in der Praxis 1824
Sortieralgorithmen 1833
Lokaler Webshop, Teil III 1843
Mit Binärzahlen arbeiten 1855
Transaktionen in Access 1859
Selektion im Datenblatt 1865
Individuelle Auswahl mit eigener Tabelle 1875
Access-Add-Ins programmieren 1880

Access-Objekt-Assistenten programmieren, Teil I 1888
Tabellen erstellen mit SQL und DDL 1895
Alle Ereignisse im Hauptformular 1906
Text- und Bezeichnungsfelder 1911
Handyverwaltung I: Datenmodell 1916
Handyverwaltung II: Personen verwalten 1925
Meldungen bei Datensatzänderungen 1935
Handyverwaltung III: Provider verwalten 1940
Steuerelemente verankern 1949
Handyverwaltung IV: Verträge und SIM-Karten 1953
Detailformular öffnen und abfragen 1966
Tipps und Tricks für Memofelder 1972
Suche in Lookupfeldern 1975
Zeilenvorschub in Berichten steuern 1982
Mehrere Datensätze pro Zeile im Bericht 1988
Haupt- und Unterformular synchron 1997
Handyverwaltung IV: Mobilfunkgeräte verwalten 2005
Das Webbrowser-Steuerelement 2018
Berichtsansicht 2024
Berichtsansicht im Unterformular 2030
Dokumente per VBA öffnen 2037
Aufgabenplaner 2038
Handyverwaltung: Personen, Verträge, Handys 2047
Ribbon für die Handyverwaltung 2058
Daten im Ribbon anzeigen 2069
Autowert festlegen 2079
Access und Excel: Makros aufzeichnen 2082
Lookup-Daten löschen 2086
Mehrere Datensätze im Register-Steuerelement 2098
Details aus dem Unterformular öffnen 2108
Tabellarische Berichtslayouts 2113
Datenaustausch mit Excel per VBA 2119
Excel fernsteuern, Teil 1: Initialisierung 2126
Exceldaten per SQL und DAO 2131
Datensätze markieren mit bedingter Formatierung 2138
Markierte Datensätze verarbeiten 2149
Excel fernsteuern, Teil 2: Workbooks und -sheets 2155
Das Navigationssteuerelement 2166
Der neue Tabellenverknüpfungs-Manager 2178
Eingabeformate 2183

Kombinationsfeld um Suche erweitern 2188
Formulare per VBA referenzieren 2194
Steuerelemente per VBA referenzieren 2199
Bedingte Formatierung per VBA 2207
Suchen und ersetzen per Access-Dialog 2218
1:1-Beziehungen 2222
Einfacher Listenfeldfilter 2231
1:1-Beziehungen in Formularen 2234
Formularansicht filtern 2246
Datenblatt per VBA: Spalten ein- und ausblenden 2250
Berechnete Felder in Tabellen 2258
Laufende Summe in Tabellen 2263
Berechnete Felder in Abfragen 2271
Linien und Rechtecke 2279
Formularansicht filtern, Teil II 2281
VBA-Funktionen der Information-Klasse 2286
Das FileSystemObject: Verzeichnisse 2298
Das FileSystemObject: Dateien 2307
Das FileSystemObject: Textstream 2313
Auflistungen mit dem Dictionary-Objekt 2318
Formulare mehrfach öffnen mit Dictionary 2325
Access-Daten nach Excel verknüpfen 2329
Standards für Steuerelemente einfach festlegen 2335
XML: Eine Einführung 2340
XML: Austausch mit Access-Bordmitteln 2346
XML exportieren per VBA 2355
Kategorie-XML-Export mit Transformation 2362
XML-Dokumente transformieren mit XSL 2367
XML-Dokumente mit VBA lesen 2380
Performance: Verbindung aktiv halten 2388
Erweitern der VBA-IDE: Projekte, Module und Co. 2391
Berichte per VBA steuern 2400
Ligaverwaltung, Teil 1: Datenmodell 2409
Zoomfenster: Einsatz und Neuerungen 2420
Feldinhalte aufteilen, Teil I 2422
VBA-Funktionen testen 2427
MSForms: Formulare und Steuerelemente 2431
Ligaverwaltung, Teil 2: Formulare 2439
XML mit VBA lesen: Nodes selektieren 2452
Erweitern der VBA-IDE: Mit Code arbeiten 2460

Erweitern der VBA-IDE: CommandBars 2467
Ligaverwaltung, Teil 3: Spielpläne generieren 2475
Verweise beim Start prüfen 2487
Mit Prozentzahlen arbeiten 2495

Inhaltsverzeichnis nach Kategorien

Grundlagen

Access - eine Einführung 25
Kunden verwalten 52
Datum und Zeit, Teil I 302
Datenbanken aufteilen 360
Datenbanken im Mehrbenutzerbetrieb 364
Tabellenverknüpfungen pflegen 365
Dialog zur Auswahl von Dateien anzeigen 375
Recordsets in der Datenblattansicht 395
Daten für die Ausgabe formatieren 567
Null-Werte und wie man damit umgeht 590
Datenbankvorlagen und Designs 1365
Etiketten drucken per Assistent 1206
Anwendungsparts 1004
Datensätze kopieren mit Bordmitteln 1013
Vertrauenswürdige Datenbanken 1082
Datenbanken und Objekte sichern 1262
Datenbanken über Verknüpfung starten 1456
Elemente für die Bildanzeige 1571
Währungen runden 1607
Daten sichern über XML 1614
Datei- und Verzeichnisoperationen mit der Shell 1712
Meldungen bei Datensatzänderungen 1933
Der neue Tabellenverknüpfungs-Manager 2176
Suchen und ersetzen per Access-Dialog 2218
Eingabeformate 2183
Performance: Verbindung aktiv halten 2388

Tabellen entwerfen

Entwurf einer einfachen Kundentabelle 57
Kunden und Projekte verwalten 72
Anreden und Co. mit Wertlisten oder Lookup-Tabellen verwalten 98
Mitarbeiter per m:n-Beziehung zu Projektteams zusammenstellen 122
Fremdschlüsselfelder mit Nachschlage-Funktion: Pro und Kontra 131
Felder und ihre Eigenschaften 211

Eigenschaften von Tabellen 240
Indizierung von Feldern 286
Bestellungen in Tabellen verwalten 531
Autowerte 1010
Neue Autowerte ermitteln und RecordsAffected 1021
GUIDs 1024
Adressen konsolidieren 1229
Indizierung in der Praxis 1824
Tabellen erstellen mit SQL und DDL 1895
1:1-Beziehungen 2222
Berechnete Felder in Tabellen 2258

Abfragen für die Datenauswahl

Einfache Auswahlabfragen kennenlernen 81
Darstellung von 1:n-Beziehungen 88
Daten aus verknüpften Tabellen darstellen 145
Filtern von Textfeldern 187
Einsatz von Parametern 192
Filtern nach Zahlen 194
Daten zusammenführen mit UNION-Abfragen 218
Berechnete Felder in Abfragen und Eigenschaften 248
Benutzerdefinierte Funktionen in Abfragen 252
Abfrageeigenschaften 261
Duplikate per Abfrage finden 291
Abfragen: Datum und Zeit 322
Gruppierungen in Abfragen 593
Kreuztabellenabfragen 598
Kreuztabellenabfragen nach Datum 613
Balkendiagramme mit Bordmitteln 811
Filterkriterien für Formulare, Teil I 824
VBA-Funktionen in Abfragen 1236
Rekursive Tabellen 1426
SELECT-Abfragen in Union 1550
Unterabfragen als Feldinhalt 1582

Aktionsabfragen

Anfügeabfragen 105
Löschabfragen 173
Aktionsabfragen per VBA ausführen 334
Aktionsabfragen per VBA 725

SQL-Grundlagen

SQL: SELECT-Abfragen, Teil I 490
SQL: SELECT-Abfragen, Teil II 503
SQL-Grundlagen: Aktionsabfragen 744

Formulare für die Dateneingabe

Formulare für einfache Tabellen 58
Formulare für Daten aus 1:n-Beziehungen 89
m:n-Beziehung per Unterformular 128
Formulare öffnen und schließen 155
m:n-Beziehungen in Listenfeldern darstellen 203
Datensatzauswahl per Kombinationsfeld 233
Abhängige Kombinationsfelder 269
Standardwerte statisch und dynamisch einstellen 275
Validieren von Benutzereingaben 308
Werte zu Kombinationsfeldern hinzufügen 328
Daten per Kombinationsfeld auswählen und löschen 339
Daten in der Endlosansicht von Formularen anzeigen 346
Lookup-Daten per Formular verwalten 388
Datensätze in Unterformularen anlegen 401
Kombinationsfeld als Eingabehilfe für Textfelder 415
Bedingte Formatierung 460
Mehrfachauswahl in Listenfeldern auslesen 482
Bestellungen in Formularen verwalten 536
Rechnen in Formularen 543
Listenfeldauswahl speichern und wiederherstellen 580
Schnellsuche im Listenfeld 604
Synchrone Unterformulare 701
Datensätze individuell auswählen 694
Dynamische Datenherkunft mit Formularbezug 728
Haupt- und Unterformular synchron 1997
Filterkriterien für Formulare, Teil II 847

Filterkriterien für Formulare, Teil III: Kombinationsfelder 879
Detailformulare aufrufen 1124
Navigation in und zwischen Formularen 1477
Zusätzliche Steuerelemente mit MSForms 1505
Alles plausibel? 1541
Berichte zur Laufzeit filtern 1778
Tabelleninhalte verschlüsseln 1784
Text- und Bezeichnungsfelder 1911
Steuerelemente verankern 1949
Berichtsansicht im Unterformular 2030
Mehrere Datensätze im Register-Steuerelement 2096
Details aus dem Unterformular öffnen 2106
Datensätze markieren mit bedingter Formatierung 2136
Kombinationsfeld um Suche erweitern 2188
Einfacher Listenfeldfilter 2231
1:1-Beziehungen in Formularen 2234
Formularansicht filtern 2246
Formularansicht filtern, Teil II 2281
MSForms: Formulare und Steuerelemente 2431

Steuerelemente

Die Schaltfläche 162
Das Listenfeld 167
Das Textfeld 224
Das Kombinationsfeld 228
Optionsgruppen und Optionsfelder 420
Das Registersteuerelement 717
Das Bild-Steuerelement 973
Die Umschaltfläche 658
Das Kontrollkästchen 998
Finetuning für Steuerelemente 1158
Kombinations- und Listenfelder mit Wertliste füllen 636
Kombinations- und Listenfelder per Callback füllen 639
ListView-Steuerelement mit Daten füllen 664
ListView-Steuerelement: Daten bearbeiten 691
ListView-Steuerelement: Kontrollkästchen nutzen 720
Das Registersteuerelement in der Praxis 737
ListView-Steuerelement: Drag and Drop, Teil I 752
ListView-Steuerelement: Drag and Drop, Teil II 788

Steuerelementlayouts mit Access 2007 und jünger 802
Textfelder: Value, OldValue und Text 834
Textfelder im Griff, Teil II: Markierungen 887
Das TreeView-Steuerelement: Grundlagen, Teil I 909
Das TreeView-Steuerelement: Grundlagen, Teil II 916
Das Anlage-Feld 930
Anlage-Felder und VBA 937
Das TreeView-Steuerelement: Verknüpfte Daten, Teil I 941
TreeView und Unterformulare 948
Das TreeView-Steuerelement: Verknüpfte Daten, Teil II 965
Das ImageList-Steuerelement 982
Kontextmenüs im TreeView, Teil I: Anlegen und Löschen 987
Das Web im Formular 1389
Designs für Steuerelemente 1597
Textboxen im Detail 1594
Seitennavigation 1626
Steuerelemente im Eigenbau: Range Control 1678
Das Webbrowser-Steuerelement 2018
Das Navigationssteuerelement 2166
Bedingte Formatierung per VBA 2207
Linien und Rechtecke 2279

Reporting mit Berichten

Einfache Tabelle in der Detailansicht 102
Adressblöcke in Berichten 262
Einfache Tabelle in der tabellarischen Ansicht darstellen 293
Darstellung von 1:n-Beziehungen per Gruppierung 298
Berichtsbereiche 426
Sortieren in Berichten 429
Berichte öffnen 446
Mehrspaltige Berichte 465
Rechnen in Berichten 548
Gruppierungen in Berichten 553
Bericht als PDF speichern 561
Rechnungsbericht 626
Access-Berichte drucken per VBA 1164
Kommunikationsdaten als Bericht 1491
Mehrspaltiger gruppierter Bericht 1497
Bericht als Unterformular 1503

Zeilenvorschub in Berichten steuern 1982
Mehrere Datensätze pro Zeile im Bericht 1986
Berichtsansicht 2024
Tipps und Tricks zu Berichten: Duplikate ausblenden 564
Tipps und Tricks zu Berichten: Vergrößerbar 563
Tipps und Tricks zu Berichten: Verkleinerbar 565
Tipps und Tricks zu Berichten: Berichte zusammenfassen 567
Tabellarische Berichtslayouts 2113
Markierte Datensätze verarbeiten 2147
Berichte per VBA steuern 2400

Makros

Aktion beim Start: Das Autoexec-Makro 373
AutoKeys: Globale Tastenkombinationen 676
Programmieren mit Makros, Teil I 1041
Programmieren mit Makros, Teil II 1062
Datenmakros I: Grundlagen 1075
Vertrauenswürdigkeit per Makro prüfen 1088
Datenmakros II: Datenoperationen 1092
Datenmakros III: Ereignisse 1133

VBA-Grundlagen

Einführung in VBA 149
Eigene Routinen schreiben 178
If...Then- und Select Case-Bedingungen 181
Variablen in VBA 199
Schleifen in VBA 325
Fehler unter VBA 378
Fehlerbehandlung unter VBA 382
Fehlerbehandlung in der Praxis 406
For Each-Schleifen 456
Zeichenketten-Funktionen 470
Zeilenumbruch und Tabulator in Access 475
Textdateien schreiben und lesen 620
Zufallszahlen generieren und verwenden 1192
Diagrammformulare 1251
Dateibefehle 616
Die Dir-Funktion 632

Binärzahlen in VBA-Eigenschaften 632
Early Binding und Late Binding 687
Programmieren mit Arrays 761
Debugging im VBA-Editor, Teil 1 783
Debugging im VBA-Editor, Teil 2 820
Ereignisse abgreifen 877
Rekursion mit VBA 894
TempVars unter VBA 1104
Programmieren mit Collections 1109
Arrays mit Split und Join 1158
Anwendungseinstellungen abspeichern 1171
Registry-Zugriff per VBA 1177
Rund um den VBA-Editor 1223
Die VBA-With-Anweisung 1278
Nützliche Helferlein per VBScript 1332
Arbeiten mit externen Bibliotheken 1385
Besseres Code-Layout 1397
Arrays als Funktionsergebnis 1421
Bubble Sort 1515
Klassenmodule und benutzerdefinierte Typen 1517
Ereignisse anderer Objekte nutzen 1746
Sortieralgorithmen 1833
Mit Binärzahlen arbeiten 1853
Dokumente per VBA öffnen 2035
VBA-Funktionen der Information-Klasse 2286
Das FileSystemObject: Verzeichnisse 2298
Das FileSystemObject: Dateien 2307
Das FileSystemObject: Textstream 2313
Auflistungen mit dem Dictionary-Objekt 2318
Erweitern der VBA-IDE: Projekte, Module und Co. 2391
VBA-Funktionen testen 2427
Erweitern der VBA-IDE: CommandBars 2467
Erweitern der VBA-IDE: Mit Code arbeiten 2460

Access-Funktionen

Domänenfunktionen 252
Bedingungen: Wenn, Schalter und Wahl 269
Access-Funktionen: Datum und Zeit 316
Meldungsfenster anzeigen und auswerten 343
Die Format-Funktion 573
Datentypen konvertieren unter VBA 1374
Mathematische VBA-Funktionen 1380
DoCmd- und RunCommand-Anweisungen 1467

Formulare mit VBA programmieren

Formularereignisse nutzen 142
Fehlerbehandlung in Formularen 411
Aktuellen Datensatz kopieren 442
Formulare und Steuerelemente referenzieren 444
Einfache Suchfunktion für Formulare 507
Farbige und vom Wert abhängige Formatierung 576
Ereignisse in Formularen: Öffnen und Schließen 606
Mehrere Formularinstanzen anzeigen 1113
Ereignisse in Formularen: Zeitgeber 1121
Datenbankereignis beim Schließen 1127
Datensätze filtern per Kombinationsfeld 1143
Von Formular zu Formular mit OpenArgs 1151
Navigieren über Formulartricks 1322
Tastensteuerung abfangen 1557
Flying Controls 1637
Formularpositionen lesen und setzen 1751
Mehrere Instanzen eines Formulars 1757
Anwendungs-Design mit positionierten Formularen 1768
Selektion im Datenblatt 1865
Individuelle Auswahl mit eigener Tabelle 1875
Alle Ereignisse im Hauptformular 1906
Detailformular öffnen und abfragen 1964
Suche in Lookupfeldern 1975
Steuerelemente per VBA referenzieren 2199
Formulare per VBA referenzieren 2194
Datenblatt per VBA: Spalten ein- und ausblenden 2250

Datenzugriff programmieren

DAO: Datenbank im Griff mit dem Database-Objekt 453
DAO: Recordsets öffnen, lesen und bearbeiten 474
DAO: Recordsets filtern, durchsuchen und sortieren 486
DAO: Querydefs und Parameterabfragen unter VBA 748
Tabellen bearbeiten mit dem TableDef-Objekt 1199
Bilder in Access, Teil II: Bilder in OLE-Feldern 953
Datensätze per VBA kopieren 1026
Verknüpfte Daten kopieren 1031
Tabellen und Abfragen über DAO verwalten 1266
Kodierte Texte mit VBA umwandeln 1310
ADODB als Alternative zu DAO 1405
Recordset-Typen 1446
DAO-Objekte und -Auflistungen 1452
DAO-Objekte und -Auflistungen, Teil II 1484
Zugriff auf Server über ODBC, Teil II 1655
Transaktionen in Access 1859
Autowert festlegen 2079

Access-Benutzeroberfläche

Datenbankfenster und Navigationsbereich im Detail 67
Der Ausdrucks-Generator 243
Formular- und Berichtsvorlagen 399
Datenblattansicht im Griff 438
Bilder in Access, Teil I 924
Kontextmenüs 957
Filtern und Suchen 1184
Navigationsbereiche gestalten 1292
Ribbon-Programmierung, Teil I 1651
Ribbons, Teil III: Interaktion im Ribbon 1794
Access-Add-Ins programmieren 1880
Access-Objekt-Assistenten programmieren, Teil I 1888
Ribbon für die Handyverwaltung 2056
Daten im Ribbon anzeigen 2069
Zoomfenster: Einsatz und Neuerungen 2420

Daten importieren

Daten aus CSV-Dateien importieren 114
Vom Textfeld zur Lookuptabelle 137
Ex- und Import zu Microsoft Word 1214
Textdateien exportieren und einlesen 1304
XML-Import und -Export 1432
Feldinhalte aufteilen, Teil I 2422

Interaktiv

CSV-Export für Adressdaten 349
Im- und Export-Spezifikationen definieren und verwenden 357
Access und Excel: Import und Export 653
Access und Excel: Verknüpfungen 677
E-Mails mit Outlook versenden 857
Berichte per E-Mail versenden, Teil II 863
E-Mails sicher versenden 872
Daten ins Web exportieren 1345
Google- und Bing-Maps im Formular 1562
Zugriff auf Server über ODBC 1641
Erfahrungen mit LibreOffice Base 1722
Outlook-E-Mails importieren 1730
Ribbons, Teil II: Bilder im Ribbon 1741
Der Webbrowser als Bildanzeige 1812
Access und Excel: Makros aufzeichnen 2082
Datenaustausch mit Excel per VBA 2119
Excel fernsteuern, Teil 1: Initialisierung 2126
Exceldaten per SQL und DAO 2131
Excel fernsteuern, Teil 2: Workbooks und -sheets 2155
XML: Austausch mit Access-Bordmitteln 2346
Kategorie-XML-Export mit Transformation 2362
XML exportieren per VBA 2355
XML-Dokumente transformieren mit XSL 2367
XML-Dokumente mit VBA lesen 2380
XML mit VBA lesen: Nodes selektieren 2452

Lösungen

Projektzeiterfassung - Aufgaben und Tätigkeiten verwalten 275
Kunden per Bezeichnung verwalten 496
Kommunikationsdaten im Griff 526
Internetmarke mit Access nutzen 706
Internetmarke mit Access nutzen, Teil II 732
Reihenfolge individuell einstellen 766
Feiertage verwalten, Teil 1 774
Feiertage verwalten, Teil 2 796
Änderungen protokollieren, Teil I 836
Berichte per E-Mail verschicken, Teil I 843
Platzhalter in Texten ersetzen 901
Berichte per E-Mail verschicken, Teil III 906
Zippen mit Bordmitteln 1284
Access als MP3-Player 1296
Reimen leicht gemacht 1356
Frontends aktualisieren 1416
Horoskopgenerator 1527
Lokaler Webshop 1665
Rezepte verwalten 1685
Nährwertdatenbank 1690
Lokaler Webshop, Teil II 1701
Platzhalter in Textdateien füllen 1800
Lokaler Webshop, Teil III 1843
Handyverwaltung I: Datenmodell 1916
Handyverwaltung II: Personen verwalten 1925
Handyverwaltung III: Provider verwalten 1940
Handyverwaltung IV: Verträge und SIM-Karten 1951
Aufgabenplaner 2038
Handyverwaltung: Personen, Verträge, Handys 2045
Lookup-Daten löschen 2084
Ligaverwaltung, Teil 1: Datenmodell 2409
Ligaverwaltung, Teil 2: Formulare 2439

Tools

Beispieldaten-Assistent 682

Tipps und Tricks

Fixierte Formulare 1620
Access-Systemtabellen 1243
Datenbanken von Altlasten säubern 1629
Tipps und Tricks für Memofelder 1970
Tipps und Tricks: Standardverzeichnis einstellen 432
Tipps und Tricks: Objekte ausblenden 432
Tipps und Tricks: Ausgeblendete Objekte anzeigen 433
Tipps und Tricks: Aktuelles Datum und Zeit blitzschnell 433
Tipps und Tricks: Wert aus vorherigem Datensatz übernehmen 432
Tipps und Tricks: Datum und Zeit im Formulartitel 434
Tipps und Tricks: Memofeld als Sortierungs- und Gruppierungsfeld 435
Tipps und Tricks: Datenbankkennwort vergeben 435
Tipps und Tricks: Textbausteine verwenden 436
Tipps und Tricks: Ereignisprozeduren schneller anlegen 801
Tipps und Tricks: Monatsnamen und Wochentage 802
Tipps und Tricks: Texte in leeren Feldern vorblenden 802

Vorwort

Access [basics] ist ein Magazin im PDF-Format, das erstmalig 2010 erschienen ist und seitdem zunächst zwölf Mal im Jahr mit jeweils ca. 20 Seiten und seit 2017 sechs Mal im Jahr mit ca. 40 Seiten Umfang erschienen ist. Dieses Buch enthält alle Ausgaben von 2010 bis 2019 und damit eine umfangreiche Know-how-Sammlung rund um die Datenbankentwicklung mit Microsoft Access.

Aufbau des Buchs

Das PDF-Buch enthält alle Ausgaben von **Access [basics]** von 2010 bis 2019. Dabei wurden die Magazine in der Reihenfolge des Erscheinens aneinandergereiht. Wir haben das Originallayout, das sich über die Jahre ein paar Mal geändert hat, beibehalten, da es gerade in den ersten Ausgaben aufwendig gestaltet war – eine Überführung in das aktuelle Layout wäre zu aufwendig gewesen.

Auch die Titelseiten, Editorials und das Impressum haben wir jeweils beibehalten, damit Sie die Magazine nach Wunsch einzeln und komplett ausdrucken können.

Wie Sie das Buch lesen können

Die ersten Ausgaben des Magazins **Access [basics]** sind so konzipiert, dass der Einsteiger diese lesen und damit die Datenbankentwicklung mit Access erlernen kann. Die erste Ausgabe liefert beispielsweise grundlegende Informationen darüber, was Access überhaupt ist. Die folgenden Ausgaben zeigen, wie man eine einfache Kundenverwaltung erstellt. Außerdem liefern zahlreiche Grundlagenartikel Informationen darüber, wie Sie Tabellen, Abfragen, Formulare, Berichte und Makros anlegen und wie Sie mit VBA programmieren. Spätere Ausgaben gehen dann in die Tiefe oder erlauben einen Blick über den Tellerrand und zeigen, wie Sie etwa Daten mit Excel oder Outlook austauschen.

Wenn Sie Einsteiger sind, beginnen Sie also einfach von vorn und erlernen den Umgang mit Microsoft Access. Wenn Sie bereits mit Access arbeiten, suchen Sie sich die Themen heraus, die Sie interessieren und nutzen Sie das Know-how für Ihre eigenen Anwendungen.

Beispieldatenbanken

Fast zu jedem Artikel dieses Magazins gibt es eine Beispieldatenbank. Die Beispieldatenbanken finden Sie im Verzeichnis Beispieldatenbanken des Downloads zu diesem Buch. Darunter sehen Sie Unterverzeichnisse für die Jahrgänge und darunter für die einzelnen Ausgaben.

Danke!

Bedanken muss ich mich vor allem bei meinen Abonnenten, die es überhaupt erst ermöglicht haben, ein Magazin in Zeiten, in denen es offensichtlich jede Information kostenlos an irgendeiner Stelle im Internet gibt, zu verlegen. Einige Abonnenten halten **Access [basics]** von Anfang an die Treue!

Ebenfalls bedanke ich ich bei meinen beiden Co-Autoren Sascha Trowitzsch, der vor allem in den neueren Ausgaben vielen Artikel beigesteuert hat und Lorenz Hölscher, der sich ebenfalls zwischendurch mit einigen Artikeln in die Autorenliste eingetragen hat.

Nun aber viel Spaß beim Lesen in diesem Kompendium zur Datenbankentwicklung mit Microsoft Access!

Duisburg, im Mai 2020

André Minhorst

Ausgabe 1/2010
1. April 2010

Access [basics]

Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

Access - eine Einführung

Access wirft viele Fragen auf. Wie ordnet es sich in den Kontext der übrigen Office-Anwendungen ein? Welche Aufgabe übernimmt es grundsätzlich, und wo kann es den übrigen Anwendungen wie Excel, Word und Co. Arbeit abnehmen, weil es einfach besser dafür geeignet ist? Wozu dienen die verschiedenen Objekte in Access wie Tabellen, Abfragen, Formulare, Berichte und Module? Gibt es überhaupt Grenzen für die Anwendbarkeit von Access als Datenbank-Management-System? Diese und weitere Fragen beantworten wir in der ersten Ausgabe von *Access im Unternehmen [basic]*.

Wen interessiert das alles?

Wenn Sie schon mit Access gearbeitet haben und eine oder mehrere der Fragen aus der Einleitung beantworten konnten, haben Sie einen großen Vorteil: Im Gegensatz zu anderen, die Ihre Kunden, Projekte und Artikel noch mit Excel verwalten, sind Sie bereits auf dem richtigen Weg und haben schon die Erkenntnis gewonnen und verdaut, dass Sie möglicherweise jahrelang ein wichtiges und mächtiges Werkzeug zur Verarbeitung von Daten ignoriert und sich stattdessen mit anderer, nicht wirklich geeigneter Software herumgeärgert haben.

... mehr zu den Unterschieden zwischen Access und Excel gegen Ende dieser Ausgabe!

Wenn Sie noch nicht wissen, was Sie mit Access anfangen sollen und ob es Ihnen vielleicht eine Menge Probleme und Sorgen vom Hals schaffen kann, lesen Sie einfach weiter – Sie erhalten eine Menge Informationen, die Ihnen dabei helfen wird, bisher eingeschlagene Wege zu überdenken und gegebenenfalls die Richtung zu korrigieren.

Das ist alles kalter Kaffee für Sie? Kein Problem: Wir sind sicher, dass es um Sie herum genügend Menschen gibt, die nicht wissen, was Sie verpassen, wenn Sie Access einfach ignorieren. Speziell für diese erste Ausgabe unseres Magazins gilt: Es steht kostenfrei zum Download bereit und Sie dürfen es bedenkenlos an alle verteilen, denen Sie etwas Gutes tun möchten.

Zielgruppen

Unter Entwicklern spricht man in Zusammenhang mit Access von drei Zielgruppen:

- **Benutzer**
- **Poweruser**
- **Entwickler**

Wenn Sie noch nicht wissen, was Datenmodellierung, Abfragen, Formulare et cetera sind, lesen Sie einfach weiter ... kommt alles noch!

Der **Benutzer** kümmert sich weder um Datenmodellierung noch um die Definition von Abfragen noch um das Design von

Herzlich Willkommen zur ersten Ausgabe von Access [basics]!

Es ist soweit: Ein neues Magazin erblickt die [Access]-Welt. Für den Start nicht als Druckwerk, sondern im PDF-Format und somit schneller, günstiger und flexibler. Flexibler? Aber wenn ich es nun lesen möchte, wo andere ... - Nun: Es ist PDF, es sind nur ein paar Blättchen, also steht es Ihnen frei, es auszudrucken und mitzunehmen, wohin auch immer Sie mögen.

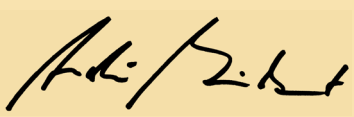
Warum nun ein neues Access-Magazin? Die Antwort ist einfach: Weil es einfach noch so viele Lücken zu schließen gibt, und weil vieles anderswo viel zu oberflächlich und lustlos beschrieben wird. Und weil ich glaube, dass das Internet Ihnen auch jetzt noch immer nicht alle Lösungen so präsentiert, dass Sie diese schnell umsetzen und leicht verstehen können.

Denn das ist es, was einen Softwareentwickler ausmacht: Er soll nicht einfach nur reproduzieren, er soll auch Know-how in sich aufsaugen, das Gelernte vertiefen, es mit Fakten untermauern und auf diese Weise immer sicherer werden, dass er jederzeit die passende Architektur, den richtigen Entwurf und die optimalen Techniken verwendet.

Access [Basics] begleitet Sie vom ersten Schritt Ihrer Laufbahn als Access-Entwickler an und entwickelt sich mit Ihnen zusammen weiter. Wenn Sie diese Ausgabe als Access-Einsteiger in Händen halten, haben Sie die richtige Wahl getroffen: Sie erfahren auf anspruchsvolle und manchmal auch unorthodoxe Weise, wie Sie Datenbankanwendungen mit Access entwickeln. Und falls Sie schon ein wenig weiter sind, seien Sie versichert: Dieses Magazin wird Ihnen auch zu bekannten Themen noch Informationen liefern, mit denen Sie Ihr Know-how erweitern können.

Und nun: Viel Spaß beim Lesen der ersten Ausgabe von Access [Basics]!

Herzlichst, Ihr



[André Minhorst]

Formularen und Berichten und erst recht nicht um die Programmierung mit VBA. Er soll einfach nur eine von einem Entwickler oder Poweruser erstellte Access-Anwendung, also eine weitgehend fertige und fehlerfreie Software auf Basis von Microsoft Access, für seine tägliche Arbeit oder auch für sein Hobby benutzen.

Der **Poweruser** ist unter Freiberuflern und Selbstständigen sowie in den EDV-Abteilungen verschiedenster Unternehmen anzutreffen. Er macht aus der Not eine Tugend und verwendet Access, um die sonst in Excel-Tabellen verwalteten Daten zuverlässig in Access-Tabellen zu speichern, Abfragen für die Ermittlung bestimmter Daten zu definieren, diese mit Formularen anzuzeigen und zu bearbeiten und in Berichten für die Ausgabe auf Papier vorzubereiten.

Er arbeitet entweder selbst mit seinen Access-Datenbanken oder stellt diese seinen Kollegen zur Verfügung, und hier und da greift er zu Makros oder VBA, um Aufgaben zu automatisieren. Der **Poweruser** nimmt auch gern Beispieldatenbanken oder fertige Lösungen und passt sie nach bestem Gutdünken entsprechend der gewünschten Funktionalität an.

Der **Entwickler** ist selbständig oder angestellt und verbringt den größten Teil seiner Zeit damit, Datenbankanwendungen auf Basis von Microsoft Access zu erstellen. Dies geschieht entweder nach den Vorgaben des Auftraggebers (Kunde oder Vorgesetzter) oder nach den eigenen Vorstellungen, wenn ein Tool oder auch eine Standardsoftware entwickelt wird. Der **Entwickler** kennt das Objektmodell von Access, Formularen, Berichten und Steuerelementen aus dem FF und kann die VBA-Befehle blind anwenden. DAO-Schleifen zum Durchlaufen von Datensatzgruppen programmiert er bereits vor dem ersten Kaffee. Der **Entwickler** ist nur ein richtiger **Entwickler**, wenn er sagt: »Mit Access kann man alles machen!«

Sind Sie Access-Anwender, Poweruser oder sogar schon Access-Entwickler? Als Anwender oder Poweruser sind Sie hier genau richtig - Sie werden schon bald eine höhere Ebene erreichen!

Warum diese Einordnung? Nun: Wenn Sie sich auf dieses Magazin einlassen, sollten Sie wissen, ob es auch das Richtige für Sie ist. Wenn Sie Access-Benutzer sind und Ihre Anwendung, die jemand für Sie programmiert hat, anpassen möchten und somit zum Poweruser werden möchten, sind Sie hier richtig. Aber nur, wenn Sie ein wenig Zeit und Experimentierfreude mitbringen.

Wenn Ihr eigentlicher Job Sie allzusehr einspannt, denken Sie besser darüber nach, jemanden für die Anpassung Ihrer Datenbankanwendung nach Ihren Wünschen zu bezahlen.

Wenn Sie Poweruser sind, also eigentlich schon ein wenig mit Access herumexperimentiert haben, sind Sie hier auf jeden Fall richtig.

Sie können schon alles und wollen perfekt werden? Na, dann lesen Sie mal unseren Tipp links ...

Falls Sie sich in die Gruppe der Access-Entwickler einordnen, wissen Sie wahrscheinlich schon alles, was wir in diesem Magazin erklären – zumindest in den ersten Ausgaben. Wenn Sie dennoch Ihre Fertigkeiten verbessern, Ihren Programmierstil perfektionieren oder von coolen Lösungen profitieren möchten, schauen Sie sich doch einmal unser Schwestermagazin **Access im Unternehmen** an. Mehr Informationen hierzu finden Sie unter **www.access-im-unternehmen.de**.

Was ist Access eigentlich?

Für viele, die private oder berufliche Büroaufgaben mit Office erledigen, stellt sich irgendwann einmal die Frage: Was hat mir Microsoft da eigentlich so alles ins Office-Paket gesteckt?

Klar, mit Outlook betreiben wir Kommunikation, mit Word schreibt man Briefe, Angebote und auch längere Abhandlungen, mit Excel verwalten wir Tabellen mit Kunden, Artikeln oder Projekten (tun wir das wirklich? Hoffentlich nicht

mehr lange!) und in PowerPoint binden wir das alles ein, um es unseren Kunden, Familienmitgliedern und allen anderen, die nicht bei drei auf den Bäumen sind, zu präsentieren. Wie aber passt Access nun ins Bild?

Definieren wir zunächst ein paar Begriffe:

- Eine **Datenbank** speichert Daten in Tabellen und verwaltet die Beziehungen zwischen den Tabellen. Es gibt noch viele andere Systeme außer Access, die allein auf diese Aufgabe spezialisiert sind – etwa der Microsoft SQL Server, MySQL oder Oracle.

Datenbanken wie Access, MySQL, Microsoft SQL Server oder Oracle speichern Daten in Tabellen. Unter Access liegen diese Tabellen in der mdb- oder accdb-Datei - aber nicht nur die ....

- Ein **Datenbank-Management-System** (**DBMS**) ist eine Software, die das Speichern der in der Datenbank enthaltenen Daten sowie den Zugriff darauf organisiert.

- Eine **Datenbankanwendung** ist eine Software, mit deren Hilfe Sie die Daten einer Datenbank anzeigen, eingeben, bearbeiten oder löschen können. Die Datenbankanwendung steuert die Elemente der Benutzeroberfläche (Menüs, Formulare und Berichte) und die dahinter steckende Technik bei, die Datenbank liefert die Daten, die in der Datenbankanwendung angezeigt werden.

... sondern auch noch Abfragen, Formulare, Berichte und VBA-Code - also die komplette Anwendung!

- **Microsoft Access** ist nicht nur ein Datenbank-Management-System (wenn auch kein vollwertiges, wie Sie später erfahren werden), sondern Sie können mit diesem auch die Tabellen einer Datenbank erstellen, über Abfragen auf die enthaltenen Daten zugreifen oder die Benutzeroberfläche definieren, über die der Benutzer die Daten einsehen und ändern kann. Access dient aber nicht nur zur Erstellung der Datenbankanwendung und der Datenbank, sondern ist auch für deren Ausführung zwingend erforderlich (genaueres hierzu erfahren Sie gleich).

- Eine mit Microsoft Access entwickelte Datenbankanwendung landet samt Datenbank meist in einer einzigen Datei, die eine der Dateiendungen **.mdb** oder **.accdb** (ab Access 2007) besitzt. Bei einfachen Datenbankanwendungen brauchen Sie also nur eine einzige Datei weiterzugeben, wenn jemand anderes damit arbeiten soll – vorausgesetzt, die entsprechende Version von Microsoft Access ist auf dem Zielrechner installiert (man neigt dazu, das Verhältnis von Access und der Datenbankdatei mit Word und einem Word-Dokument zu vergleichen, aber dieser Vergleich hinkt).

Tabellen, Abfragen, Formulare, Berichte, Makros, VBA-Module, Definitionen von Menüs oder Ribbons und auch die Daten selbst landen alle in einer einzigen Datei - außer natürlich, Sie quartieren Objekte wie etwa Tabellen in eine andere Datenbank aus (das wäre dann ein sogenanntes Backend).

Diese Definitionen sind für den reinen Benutzer einer Datenbankanwendung unter Access interessant: Es gibt Daten, die in Tabellen gespeichert und mit der Benutzeroberfläche der Datenbankanwendung angezeigt werden, und die Bearbeitung und Ansicht der Daten geschieht mit dem Microsoft-Produkt Access.

Access für den Endanwender

Vermutlich haben Sie bereits mit einer ganzen Reihe von Access-Datenbankanwendungen gearbeitet, vielleicht sogar ohne dies zu bemerken – Access-Datenbankanwendungen können wie herkömmliche Softwareprodukte wie etwa Warenwirtschaftssysteme, Adressverwaltungen oder auch Mailprogramme aussehen.

Für den Anwender ist es aber meist völlig uninteressant, mit welchem System eine Software erstellt wurde und welches Datenbanksystem dahintersteckt – für ihn ist allein wichtig, dass sie die gewünschte Funktion liefert und seine Aufgaben in akzeptabler Zeit erledigt.

Für den Benutzer ist die hinter einer Anwendung steckende Software egal - hauptsache, sie funktioniert!

Wenn ein Anwender mit einer Datenbankanwendung auf Basis von Microsoft Access arbeiten möchte, muss er nur sicherstellen, dass die benötigte Version auf seinem Rechner installiert ist. Das kann er auf zwei Arten erledigen:

- Der Anwender hat ohnehin die Vollversion von Access auf dem Rechner installiert, weil er beispielsweise eines der größeren Pakete von Microsoft Office verwendet.
- Der Anwender hat keine Access-Installation auf dem Rechner. Auch das ist kein Problem: Es gibt eine sogenannten Runtime-Version von Access, die alle für den Betrieb einer fertig entwickelten Access-Datenbankanwendung notwendigen Funktionen liefert. Diese kann sich jeder Anwender seit der Version Access 2007 aus dem Internet herunterladen; für die Runtime-Variante früherer Access-Versionen muss der Entwickler eine spezielle Developer-Erweiterung für ein paar Hundert Euro kaufen. Damit kann er aber ohne weitere Kosten beliebig viele Access-Datenbankanwendungen ausstatten und diese weitergeben. Wenn ein Entwickler diese Runtime-Version besitzt, braucht der Benutzer einer damit ausgestatten Access-Datenbankanwendung eigentlich nur noch einen Rechner mit einem entsprechenden Betriebssystem.

Access aus Poweruser- und Entwicklersicht

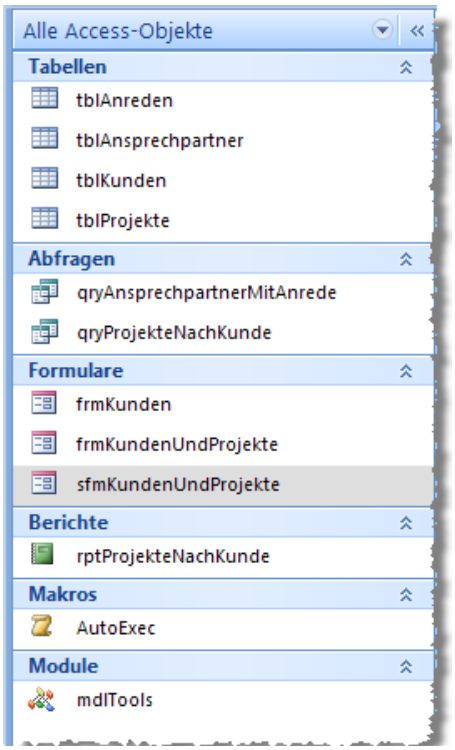
Aus welche Elementen besteht eine Access-Datenbank eigentlich? Hier erfahren Sie mehr über den Aufbau einer Access-Datenbank und lernen die verschiedenen Objekte kennen.

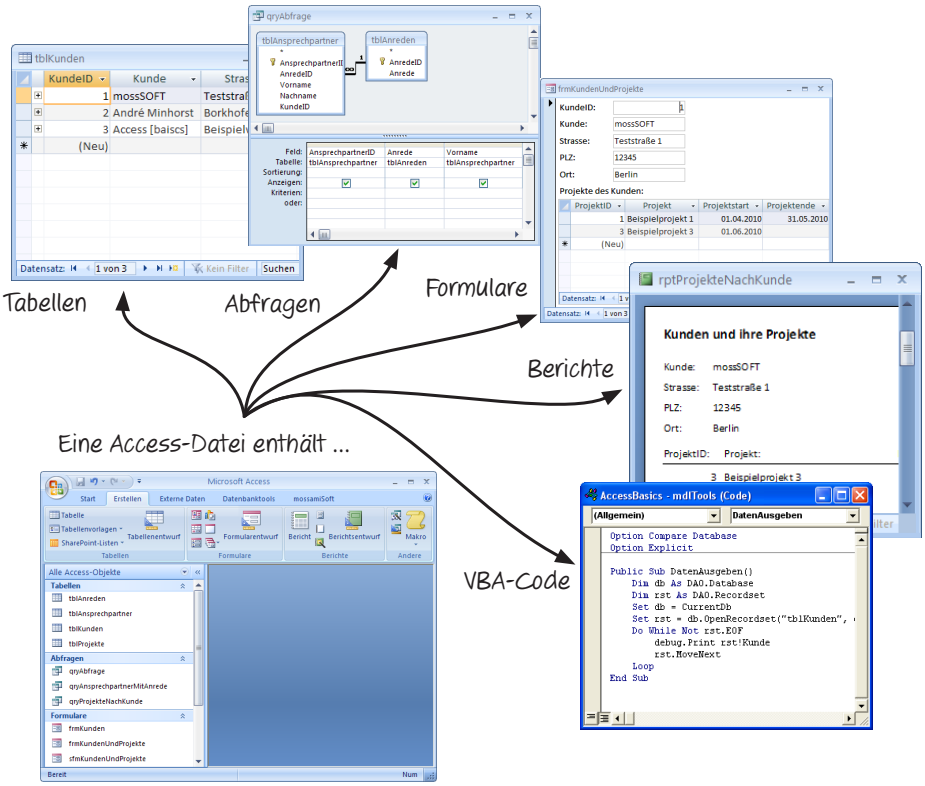
Während der Benutzer einer Datenbankanwendung nur Menüs, Formular und Berichte zu sehen bekommt, muss sich der Entwickler um einiges mehr kümmern. Für einen ersten Überblick reicht eine Auflistung wichtiger Begriffe, die wir später genauer erläutern werden:

Eine **Datenbank** ist im Kontext dieses Magazins die Gesamtheit aller in einer **.mdb**- oder **.accdb**-Datei gespeicherten Objekte.

Ein **Objekt** einer Datenbank kann eine **Tabelle**, eine **Abfrage**, ein **Formular**, ein **Bericht**, ein **Makro** oder ein **Modul** sein. Es gibt noch weitere Objekte, die jedoch zu diesem Zeitpunkt nicht wichtig sind. Die meisten der für die Arbeit mit einer Access-Datenbank wichtigen Objekte finden Sie im **Datenbankfenster** (bis Access 2003) beziehungsweise im **Navigationsbereich** (ab Access 2007).

Wir stellen die verschiedenen Datenbankobjekte in der folgenden Liste erstmal kurz vor. Weiter unten erklären wir dann genauer, wie die einzelnen Objekte aussehen - und in den folgenden Ausgaben lernen Sie dann, diese selbst zu erstellen und damit zu arbeiten.





Eine **Tabelle** speichert Daten in Datensätzen, wobei jeder Datensatz einer Tabelle die gleichen Felder zum Eingeben bestimmter Daten bietet. Ein Datensatz einer Tabelle repräsentiert in den meisten Fällen ein reales Objekt, wobei die Werte der Felder des Datensatzes den Eigenschaften des betroffenen Objekts entsprechen.

Da reale Objekt in der Regel sehr viele Eigenschaften besitzen, bilden die Felder einer Tabelle nur die für die jeweilige Anwendung wichtigen Eigenschaften ab. Eigentlich sieht dies so ähnlich wie in einer Excel-Tabelle aus:

tblKunden

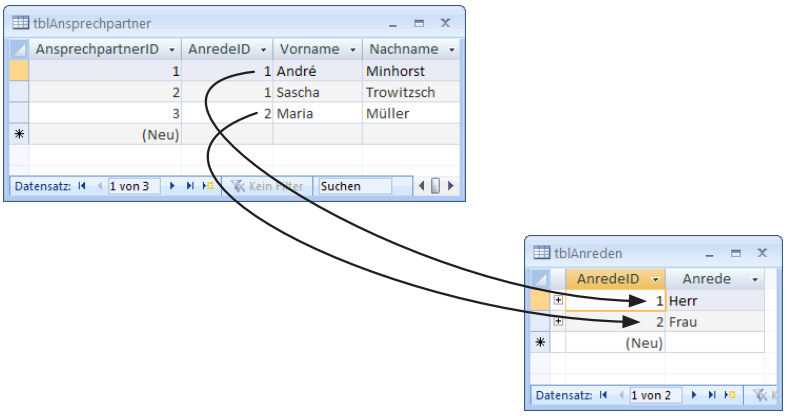
KundeID	Kunde	Strasse	PLZ	Ort
1	mossSOFT	Teststraße 1	12345	Berlin
2	André Minhorst	Borkhofer Str. 17	47137	Duisburg
(Neu)				

Datensatz: 1 von 2 | Kein Filter | Suchen

Das Aussehen einer solchen Tabelle mit seinen Feldern (also Spalten) wird allerdings genau definiert, und zwar in der Entwurfsansicht – dazu später mehr.

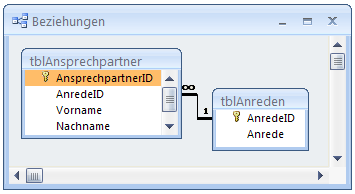
Beziehungen, Primärschlüssel und Fremdschlüssel

Zwischen Tabellen kann es **Beziehungen** geben. Wenn man von einer Beziehung zwischen zwei Tabellen spricht, dann meint man, dass die eine Tabelle ein Feld mit Werten enthält, die in einer anderen Tabelle wieder auftauchen. Im folgenden Beispiel haben wir mögliche Anreden für Ansprechpartner in eine eigene Tabelle namens **tblAnreden** geschrieben und jeder Anrede eine eigene Nummer gegeben. Im Feld der Tabelle **tblAnsprechpartner**, in dem man eigentlich Anreden wie **Herr** oder **Frau** erwarten wurde, stehen nur noch Zahlenwerte:



Aber nicht irgendwelche Zahlenwerte, sondern genau diejenigen, die zu den passenden Anreden aus der Tabelle **tblAnreden** gehören.

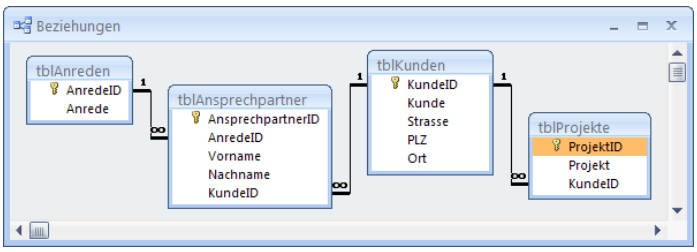
Eine Beziehung wie diese hier legt man im Beziehungen-Fenster von Access fest. Dazu fügt man dort die betroffenen Tabellen hinzu und verbindet die an der Beziehung beteiligten Felder miteinander:



Anhand einer einfachen Beziehung wie dieser hier lässt sich prima der Begriff **Schlüssel** beziehungsweise **Index** erläutern. Das Feld **AnredeID** der Tabelle **tblAnreden** ist ein **Primärschlüssel**. Damit geht vor allem einher, dass dieses Feld eindeutige Werte enthält. Damit sorgen Sie gleichzeitig dafür, dass jeder Datensatz der Tabelle **tblAnreden** über den Wert des Feldes **AnredeID** eindeutig identifiziert werden kann.

Im vorliegenden Fall ist das vor allem für die Beziehung zwischen den Tabellen **tblAnsprechpartner** und **tblAnreden** wichtig. Wenn Sie im Feld **AnredeID** der Ansprechpartner-Tabelle einen der Zahlenwerte des gleichnamigen Feldes der Tabelle **tblAnsprechpartner** eintragen, möchten Sie sich auch konkret auf die hinter dieser Zahl verborgenen Anrede beziehen. Das wäre unmöglich, wenn es beispielsweise zwei Anreden in der Tabelle **tblAnreden** gäbe, die den Wert **1** im Feld **AnredeID** enthielten. Das Feld **AnredeID** der Tabelle **AnsprechpartnerID** ist übrigens ein **Fremdschlüssel**. Das bedeutet, dass Sie dort Werte eintragen, die aus dem Primärschlüsselfeld einer anderen Tabelle stammen – und zwar genau aus der Tabelle, die mit der Tabelle **tblAnsprechpartner** in einer Beziehung steht (hier die Tabelle **tblAnreden**). Die hier verwendete Beziehung heißt übrigens 1:n-Beziehung: Sie können jedem Datensatz der 1-Seite der Beziehung (einer Anrede) beliebig viele Datensätze der n-Seite der Beziehung (einem Ansprechpartner) zuweisen.

Meist sind alle Tabellen einer Datenbank irgendwie durch Beziehungen miteinander verbunden. Diese Beziehungen fasst das **Datenmodell** als die Gesamtheit aus Tabellen und Beziehungen zusammen.Es stellt die Zusammenhänge zwischen den einzelnen Tabellen einer Datenbank anschaulich dar.



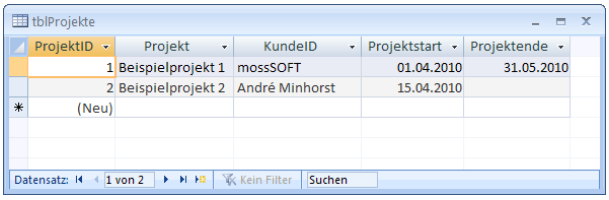
Sie können hier gut weitere Beispiele für Beziehungen zwischen Tabellen erkennen: So lassen sich jedem Kunden beliebig viele Ansprechpartner zuordnen, gleiches gilt für die Beziehung zwischen Projekten und Kunden.

Es gibt neben der 1:n-Beziehung noch weitere Beziehungstypen, die wir in einer späteren Ausgabe von **Access [basics]** beleuchten werden.

Abfragen

Eine **Abfrage** erlaubt es, Daten aus einzelnen Tabellen zu sortieren oder nach bestimmten Kriterien zu filtern. Sie können so beispielsweise alle Adressdatensätze ausgeben, deren Vorname mit A beginnt oder die Datensätze nach dem Geburtstag sortieren. Sie können damit aber auch Daten aus mehreren Tabellen zusammenführen.

So lassen sich damit beispielsweise die Daten zu einem Projekt zusammen mit der Adresse des entsprechenden Kunden darstellen. Mit einer Abfrage können Sie aber auch Änderungen an Daten mit bestimmten Kriterien durchführen. Wichtig ist an dieser Stelle, dass Sie verstehen, dass das Ergebnis einer Abfrage immer nur eine bestimmte Sicht der gewünschten Daten liefert, diese aber nicht speichert.


tblProjekte

ProjektID	Projekt	KundeID	Projektstart	Projektende
1	Beispielprojekt 1	mossSOFT	01.04.2010	31.05.2010
2	Beispielprojekt 2	André Minhorst	15.04.2010	
(Neu)				

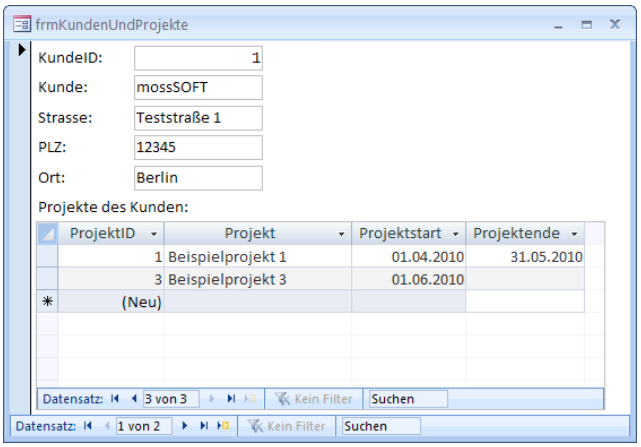
Datensatz: 1 von 2 | Kein Filter | Suchen

Diese Abfrage liefert alle Projekt mit Start ab dem 1.4.2010.

Formulare

Ein **Formular** stellt die Daten aus Tabellen oder Abfragen strukturiert dar. Klar, die tabellarische Ansicht von Tabellen oder Abfragen liefert auch eine strukturierte Darstellung, aber eine sehr unflexible. Ein Formular kann jedes Feld der anzuzeigenden Tabelle oder Abfrage in einem entsprechenden Steuerelement wie einem Textfeld, einem Kombinationsfeld, einem Kontrollkästchen oder auch in einer Optionsgruppe darstellen.

Zusätzlich können Sie weitere Steuerelemente wie Schaltflächen zum Aufrufen bestimmter Funktionen auf einem Formular platzieren. Ein weiterer Clou ist, dass Formulare Daten aus mehreren Tabellen und deren Beziehung auf verschiedene Arten darstellen können.


frmKundenUndProjekte

KundeID: 1
Kunde: mossSOFT
Strasse: Teststraße 1
PLZ: 12345
Ort: Berlin

Projekte des Kunden:

ProjektID	Projekt	Projektstart	Projektende
1	Beispielprojekt 1	01.04.2010	31.05.2010
3	Beispielprojekt 3	01.06.2010	
(Neu)			

Datensatz: 3 von 3 | Kein Filter | Suchen

Datensatz: 1 von 2 | Kein Filter | Suchen

Dieses Formular zeigt Kundendaten und in einem Unterformular die Projekte des Kunden.

Ein **Bericht** kann prinzipiell wie ein Formular gestaltet werden und seine Daten tabellarisch oder auf andere Weise darstellen.

Im Gegensatz zum Formular erlaubt er keine Bearbeitung der enthaltenen Daten, sondern dient ausschließlich deren Ausgabe.

Er bietet eine Vorschauansicht, mit der Sie sich ansehen können, wie ein Ausdruck auf Papier oder die Ausgabe in ein PDF-Dokument aussehen wird. Die Möglichkeiten der Ausgabe sind jedoch noch ausgefeilter als bei Formularen: Sie können damit Gruppierungen definieren oder die Daten in mehreren Spalten ausgeben.

rptProjekteNachKunde

Kunden und ihre Projekte

Kunde: mossSOFT
Strasse: Teststraße 1
PLZ: 12345
Ort: Berlin

ProjektID:	Projekt:	Projektstart:	Projektende:
3	Beispielprojekt 3	01.06.2010	
1	Beispielprojekt 1	01.04.2010	31.05.2(
3	Beispielprojekt 3	01.06.2010	

Die gleichen Daten wie oben im Formular können Sie natürlich auch in einem Bericht ausgeben, der als Grundlage für einen Ausdruck oder ein PDF-Dokument dienen kann.

Access automatisieren

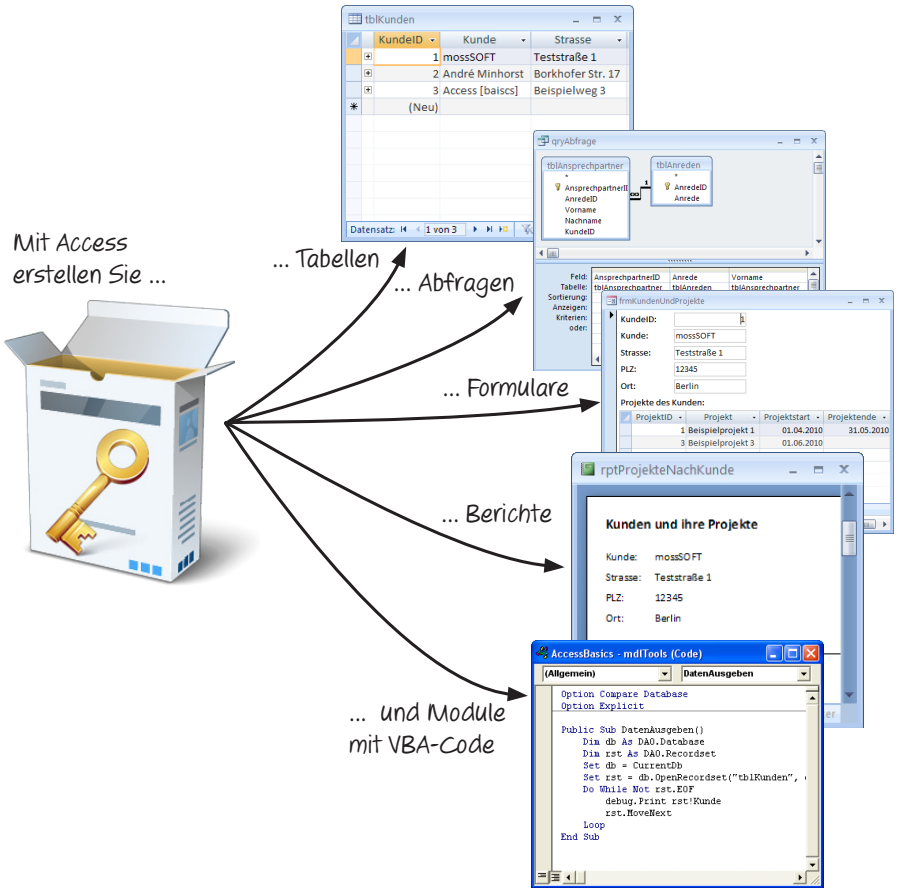
Ein **Makro** bietet die Möglichkeit, einfache Vorgänge schnell zu automatisieren. So einfach dies ist, so eingeschränkt ist der Leistungsumfang eines Makros.

Die Programmiersprache **VBA** bietet ungleich mächtigere Möglichkeiten als ein Makro. Sie können damit ausgefeilte Funktionen und Prozeduren erstellen, die in **Modulen** gespeichert werden (den Unterschied zwischen Funktionen und Prozeduren erfahren Sie später).

Benutzerfreundliche Entwicklungsumgebung

Die soeben vorgestellten Objekte haben einen entscheiden Nachteil: Sie müssen durch den **Poweruser** oder **Entwickler** zunächst erstellt werden – also durch Sie! Zum Glück steht Ihnen Access mit einer Reihe mächtiger Tools zur Seite.

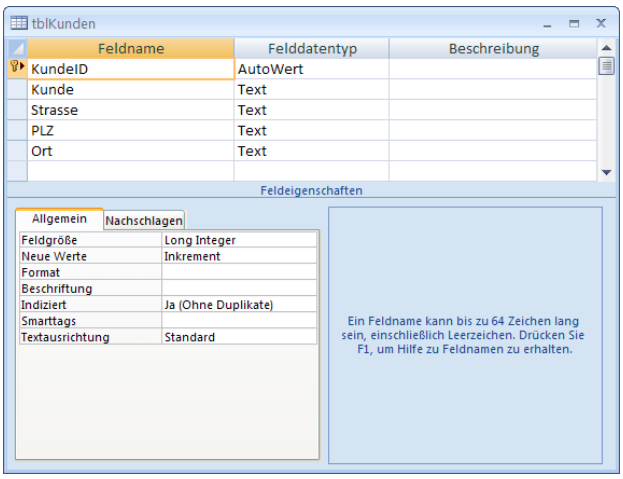
Access ist nur das Rahmenwerk für Ihre Anwendung. Tabellen, Abfragen, Formulare, Berichte, Makros und Module müssen Sie komplett selbst definieren (oder aus bestehenden Datenbanken zusammenkopieren - das heißt dann Wiederverwendung).



Eigentlich finden Sie sogar zwei Entwicklungsumgebungen vor, wenn Sie Datenbankanwendungen mit Access erstellen: Die erste ist Access selbst, die zweite ist die VBA-Entwicklungsumgebung, mit der Sie Prozeduren erstellen können, um die übrigen Objekte mit Leben zu füllen.

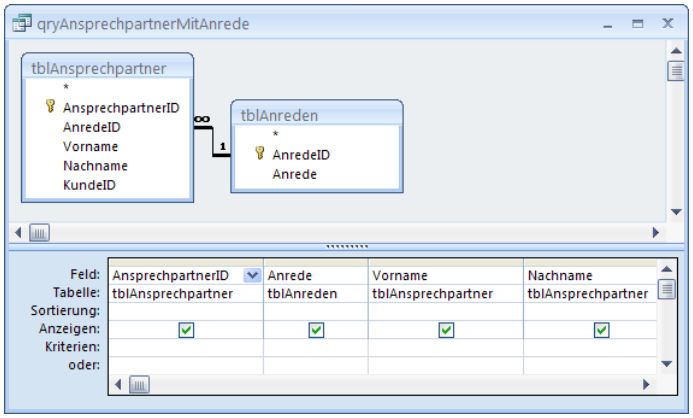
Access stellt Entwurfsansichten für alle wichtigen Objekte zur Verfügung:

- Im Tabellenentwurf legen Sie fest, aus welchen Feldern eine Tabelle besteht, welchen Datentyp die einzelnen Felder besitzen und welche Felder einer Tabelle mit einem Index versehen werden.



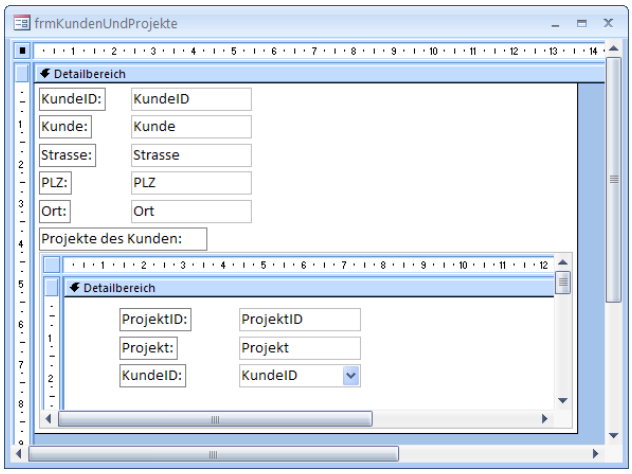
Der Datentyp Autowert beim Feld KundeID sorgt dafür, dass für jeden neuen Datensatz ein eindeutiger Wert in dieses Feld geschrieben wird. So können Sie jeden Datensatz immer eindeutig identifizieren - das ist zum Beispiel beim Verknüpfen von Datensätzen zweier Tabellen hilfreich.

- Im Abfrageentwurf wählen Sie die Tabellen aus, deren Daten die Abfrage anzeigen soll. Außerdem bestimmen Sie, welche Felder das Abfrageergebnis liefern soll und welche Kriterien und welche Sortierung Access bei seiner Ermittlung berücksichtigen soll.



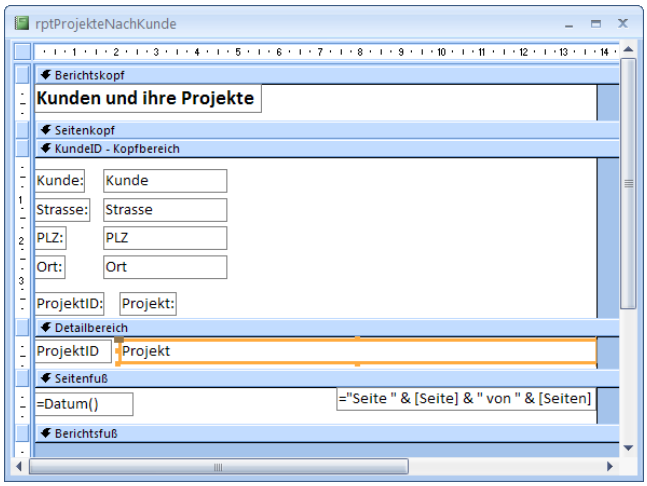
Der Abfrageentwurf sieht auf den ersten Blick verwirrend aus. Sie werden aber bald erfahren, wie einfach es ist, die Daten aus verschiedenen Tabellen zusammenzuführen oder einfach nur nach bestimmten Kriterien auszuwählen.

- Der Formularentwurf hilft dabei, benutzerfreundliche Formulare für das Betrachten und das Bearbeiten von Daten zu erstellen. Dort legen Sie fest, aus welcher Tabelle oder Abfrage die anzuzeigenden Daten stammen und ordnen die Steuerelemente zur Darstellung der Feldwerte der Datensätze an. Außerdem bestimmen Sie, ob das Formular die Daten detailliert mit einem Datensatz pro Formular oder tabellarisch anzeigen soll.



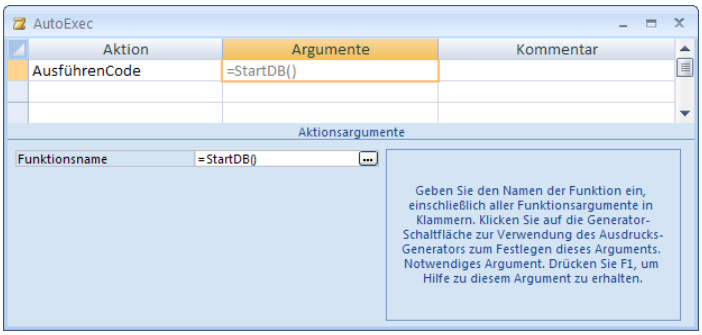
Es gibt kaum leichtere Tätigkeiten, als das Grundgerüst eines Formulars mit der entsprechenden Entwurfsansicht von Access zu erstellen. Der Teufel steckt allerdings im Detail - bis erstmal alles ordentlich ausgerichtet und mit den nötigen Funktionen versehen ist, kann es schon etwas dauern.

- Der Berichtsentwurf bietet noch ein wenig mehr, denn dort definieren Sie auch noch Gruppierungen für die anzuzeigenden Daten und legen den Inhalt spezieller Bereiche wie Seitenkopf und -fuß oder Berichtskopf und -fuß fest.



Berichte bieten mitunter eine verwirrende Menge verschiedener Bereiche an. Aber wenn Sie alle erst einmal ausprobiert haben, steht einer ordentlichen Präsentation Ihrer Daten nichts mehr im Wege.

- Und schließlich hilft der Makroentwurf beim Zusammenklicken von Befehlen zu einfachen Abläufen. Um Einsteigern das Leben möglichst einfach zu machen, gibt es ab Access 2010 mit den Data Makros eine wesentliche Erweiterung der Makros. Damit können Sie wesentlich mehr erreichen als in den Vorgängerversionen, aber so flexibel wie VBA sind Makros immer noch nicht (und werden es auch wahrscheinlich niemals sein). Wir werden im wesentlichen Spezialmakros verwenden, die automatisch beim Öffnen einer Datenbank ausgeführt werden oder die zum Festlegen von Tastenkombinationen dienen.



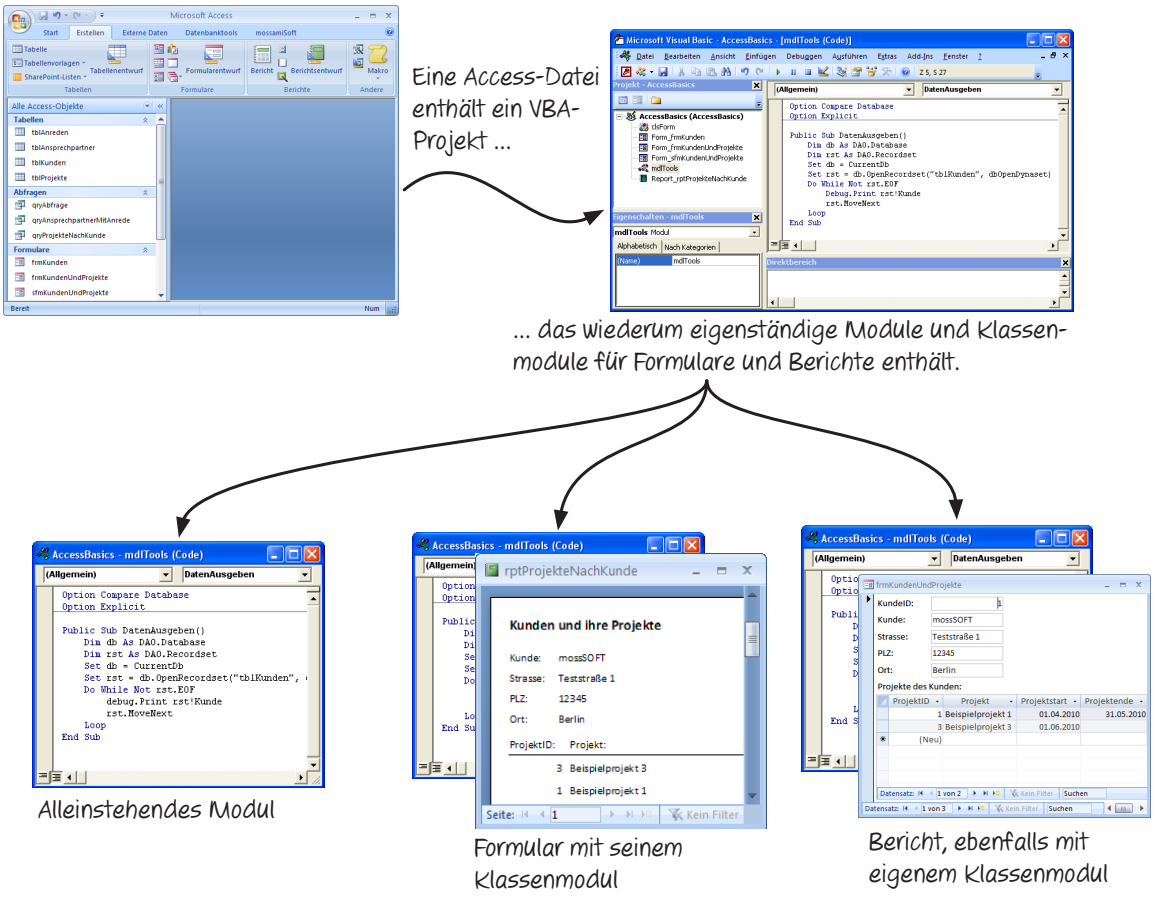
Das Makro namens AutoExec wird automatisch beim Öffnen einer Datenbankdatei ausgeführt. Das ist praktisch, wenn Sie beispielsweise gleich zu Beginn eine VBA-Prozedur ausführen möchten oder etwas Ähnliches erledigen wollen.

VBA in Access

Primär wird VBA Ihnen helfen, die Anwendungslogik hinter Formularen zur Eingabe und Bearbeitung und Berichten zur Ausgabe der Daten zu optimieren.

Der Code landet dabei in sogenannten Modulen, von denen es unterschiedliche Typen gibt. Momenten reicht es, wenn Sie wissen, dass es alleinstehende Module (Standardmodule) und Klassenmodule gibt. Klassenmodule gehören meist zu einem Formular oder Bericht und enthalten all den Code, der beispielsweise beim Öffnen eines Formulars oder Berichts, beim Schließen eines solchen Objekts oder auch durch die verschiedenen Ereignisse eines Steuerelements wie etwa dem Anklicken einer Schaltfläche ausgelöst werden.

Die Module befinden sich genau wie die übrigen Datenbankobjekte in der **.mdb**- oder **.accdb**-Datei, und dort genau genommen im VBA-Projekt dieser Datenbankanwendung. Dieses VBA-Projekt und seine Module mit dem Code bearbeiten Sie im VBA-Editor.



Wozu mit VBA programmieren?

Zwar bietet Access schon eine ganze Menge Funktionen, mit denen Sie ohne eine Zeile Code Daten in Formularen anzeigen und bearbeiten können. Aber schon eine einfache **OK**-Schaltfläche zum Schließen eines Formulars braucht zumindest eine Zeile Code, um ihre Aufgabe auszuführen. Und wenn Sie Ihre Formulare mit professionellen Features ausstatten möchten, kommen Sie um die VBA-Programmierung früher oder später nicht herum. Damit können Sie beispielsweise folgendes tun:

- Durchführen einer Validierung nach der Eingabe von Werten in einzelne Felder oder nach Abschluss der Eingabe des Datensatzes
- Öffnen weiterer Formulare etwa zur Eingabe detaillierterer Informationen
- Öffnen von Berichten zur Ausgabe von Daten
- Durchführen einer Suche nach Datensätzen mit bestimmten Kriterien

Fernsteuern mit VBA

Neben Access können Sie auch die wichtigsten anderen Office-Produkte wie Word, Excel, Outlook und PowerPoint mit VBA steuern – und deshalb gibt es für alle Office-Anwendungen eine einheitliche (Code-)Entwicklungsumgebung. Und damit kommen wir gleich auf einen ganz wichtigen Vorteil von Access gegenüber anderen Datenbank-Management-Systemen zu sprechen: Es gliedert sich sehr gut in die übrigen Produkte der Microsoft Office-Reihe ein und erlaubt eine fast grenzenlose Interaktion mit diesen. Beispiele gefällig?

- Sie können Mails, Termine, Aufgaben und Kontakte mit Outlook austauschen und synchronisieren.

Access [basics]

Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

Ausgabe 2/2010
10. Mai 2010

Vom Excel-Chaos zur Access-Projektzeiterfassung

Teil I: Kunden verwalten

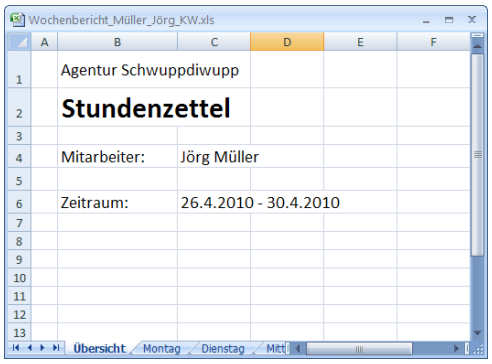
Der Chef der kleinen Agentur Schwuppdiwupp in Duisburg-Meiderich ist ein richtiger Erbsenzähler. Jeder Mitarbeiter muss penibel in einer Excel-Tabelle aufschreiben, wann er was für welches Projekt für welchen Kunden getan hat. Und an jedem Wochenende sitzt der Chef im Büro und wertet die Daten aus, um die Kosten zu berechnen (und am Montag die Kollegen zu ermahnen, die zu langsam waren ...).

Die Auswertung der Daten macht dem Chef viel Spaß – immerhin hat er am Wochenende viel Zeit, weil seine Frau und seine Freunde ihn schon vor langer Zeit sitzen lassen haben. Und das nur, weil er immer so fleißig war und alles im Griff hatte! Naja, so ein Chef muss sich halt voll auf den Job konzentrieren.

Gut, dass er seinen Mitarbeitern nach fünf Jahren jetzt endgültig eingeimpft hat, dass sie immer ordentlich ihre Stunden aufschreiben sollen - okay, bei neuen Mitarbeitern kostet das immer noch jede Menge Energie, aber es lohnt sich ja.

Jeder Kollege bekommt jede Woche eine neue Excel-Datei mit sechs Tabellenblättern (fünf für die Wochentage und eines für die Mitarbeiterdaten). Jede der fünf Excel-Tabellen hat einige Spalten wie Projektnummer, Projektbezeichnung, Kundennummer, Kundenbezeichnung, Startzeit, Endzeit und Tätigkeitsbeschreibung.

Der Chef hat sieben Mitarbeiter, und seine eigenen Tätigkeiten zeichnet er selbstverständlich ebenso akribisch auf wie seine Mitarbeiter. Das heißt also, dass er am Wochenende acht mal fünf Excel-Tabellen auswertet. Dabei fasst er erst die fünf Excel-Tabellen für jeden Mitarbeiter zusammen und fügt am Ende eine Summenzeile hinzu. So erfährt er auf einfache Weise, wieviele Stunden jeder Mitarbeiter in dieser Woche gearbeitet hat.



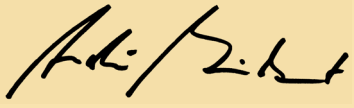
In dieser Aufgabe:

S. 1: Vom Excel-Chaos zur Access-Projektzeiterfassung
S. 5: Tabellen entwerfen
Teil I: Entwurf einer einfachen Kundentabelle
S. 9: Formulare für die Dateneingabe
Teil I: Formulare für einfache Tabellen
S. 16: Die Schaltzentrale von Access
Datenbankfenster und Navigationsbereich im Detail

Im Großen und im Kleinen ...

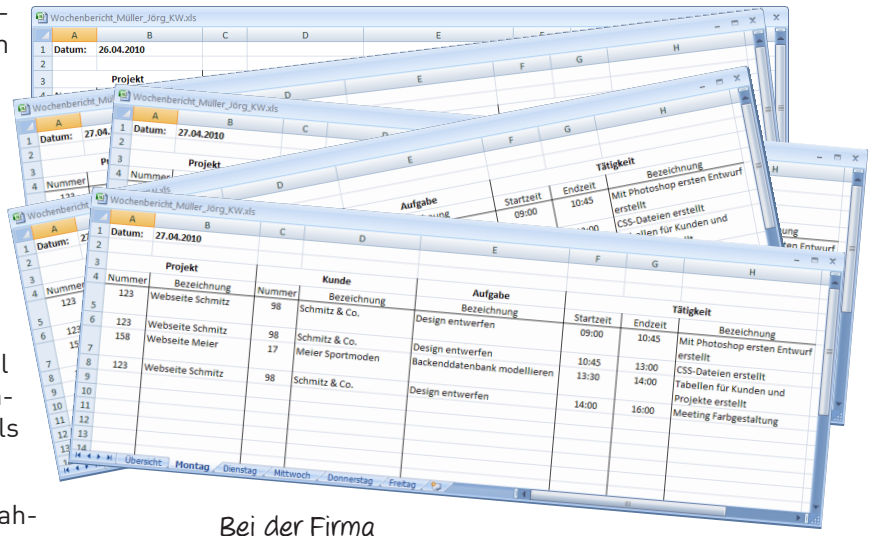
Es gibt eine große Menge Einsteigerbücher für angehende Access-Entwickler. Ein paar davon sind bestimmt gut geeignet, um die ersten Schritte zu machen, andere weniger. Tatsache ist: Die meisten verwenden irgendein Beispiel, um dem Leser zu zeigen, wie er mit Access arbeiten soll. In das Beispiel flechten sie dann die verschiedenen Techniken ein, die für das Erstellen von Tabellen, Abfragen, Formularen, Berichten und so weiter nötig sind. Ich möchte das ein wenig anders gestalten: Natürlich gibt es auch mehr oder weniger realistische Beispiele. Möglichst eingebettet in eine Story, denn Sie sollen ja auch ein wenig Spaß beim Lesen haben. Die Story enthält dann Informationen zu der jeweiligen Anwendung, die wir zusammen erstellen. Aber die Techniken, die landen woanders: Nämlich in der [basics]-Ecke. Unter [basics] verstehe ich Anleitungen zu den grundlegenden Schritten beim Erstellen einer Access-Anwendung. Zum Beispiel das Erstellen einer Tabelle. Oder das Hinzufügen eines Primärschlüsselfeldes, eines Textfeldes oder eines Datumsfeldes. Und auch das Anlegen von Beziehungen, das Erzeugen von Formularen und das Hinzufügen von Steuerelementen. Die [basics] kommen in ein eigenes PDF-Dokument, dass mit jeder Ausgabe wächst - und damit wächst auch Ihr Grundstock an Access-Techniken. Die brauchen Sie sich nicht alle zu merken, denn Sie finden sie ja, alphabetisch sortiert, im [basics]-Dokument. Und damit Sie von den rund 20 Seiten einer jeden Ausgabe von Access [basics] auch etwas haben, verweisen wir natürlich immer wieder auf das [basics]-Dokument, anstatt alles zehn Mal zu beschreiben. In dieser Ausgabe beginnen wir übrigens mit dem Beispiel einer kleinen Firma, die Ihre Projektzeiterfassung mit Excel erledigt. Wenn Sie wissen möchten, was dabei alles schiefläuft und wie der Chef der Firma mit einer passenden Access-Anwendung demnächst viel Zeit spart, lesen Sie am besten gleich los!

Viel Spaß!



[André Minhorst]

Dann erstellt er je Projekt eine neue Excel-Tabelle und kopiert die Spalten, die zu einem Projekt gehören, aus den 40 Excel-Tabellen dort hinein. So kann er auch direkt sehen, wieviele Stunden in dieser Woche in die einzelnen Projekte geflossen sind. Schließlich kopiert er die neue Wochenbilanz je Projekt in eine weitere Excel-Tabelle, die alle in den vorherigen Wochen für ein Projekt geleisteten Tätigkeiten zusammenfasst. Auch hier kommt – zack! – schnell eine Summezeile hinzu, sodass der Chef schnell die Summe der Stunden für einen bestimmten Zeitraum erfassen und gegebenenfalls eine Rechnung an den Kunden stellen kann.



Bei der Firma Schwuppdiwupp werden am Wochenende Stundenzettel in Form von Excel-Blättern ausgewertet. Von vielen Excel-Blättern!

Apropos Rechnung: Die wird natürlich – Sie ahnen es bereits – in einer weiteren Excel-Datei erstellt. Der Chef hat eine coole Idee gehabt, wie er eine Menge Zeit sparen kann: Er füllt die Rechnungstabelle nicht mehr komplett neu aus, sondern er kopiert einfach die vorherige Rechnung für einen Kunden. So braucht er nur noch die Rechnungspositionen zu ändern und die Summe neu zu bilden!

Der Neue

Eines Tages geschah etwas, wodurch sich das Leben in der Firma Schwuppdiwupp entscheidend änderte: Der Chef stellt mal wieder einen neuen Mitarbeiter ein, aber dieser war anders als alle Mitarbeiter, die jemals zuvor in der Firma gearbeitet haben: Er hatte zuvor als Datenbankprogrammierer mit Microsoft Access gearbeitet! Er wusste, dass er behutsam, aber bestimmt vorgehen musste, um gewissen Dinge im organisatorischen Ablauf zu ändern – vor allem bei der Projektzeiterfassung ...

Die übliche zweistündige Einführung in die Vorgehensweise bei der Zeiterfassung ließ er ohne Widerworte über sich ergehen. Gleich am ersten Tag Abläufe zu kritisieren, die durch den Chef eingeführt wurden und von denen möglicherweise auch die anderen Mitarbeiter mittlerweile überzeugt waren – das wäre keine gute Idee.

Im stillen Kämmerlein durchdachte er jedoch, was er gesehen hatte, und überlegte sich, wie man das Thema Projektzeiterfassung anders angehen könnte. Dabei fielen ihm zunächst einige Probleme auf:

- Der Chef musste am Wochenende Überstunden arbeiten, um den Aufwand der einzelnen Mitarbeiter für die verschiedenen Projekte zu ermitteln.
- Die Mitarbeiter mussten für jede Tätigkeit viel mehr Daten eingeben, als eigentlich nötig: Normalerweise sollte es doch reichen, wenn man die Aufgabe aus einer Liste auswählt und dazu eine neue Tätigkeit anlegt. Und dabei sollte man nicht mehr erledigen, als schlicht eine kurze Beschreibung und die Start- und Endzeit einzutragen (vielleicht geht das auch noch einfacher ...).
- Außerdem ist die Dateneingabe fehleranfällig: Wenn ein Mitarbeiter für jede Tätigkeit zum Beispiel die Projektnummer, den Projektnamen, die Kundennummer und den Kundennamen einträgt, passiert früher oder später mal ein Fehler: Dann wird vielleicht eine Zeile angelegt, die eine Projektnummer enthält, die zu einem falschen Kunden gehört, oder die es gar nicht gibt. Wenn der Chef sich dann am Wochenende alle Tätigkeiten

Tabellen entwerfen

Teil I: Entwurf einer einfachen Kundentabelle

Normalerweise sollte man sich vor dem Anlegen der ersten Tabelle einer Datenbankanwendung nicht nur Gedanken über die einzelnen Tabellen und deren Felder, sondern auch über deren Beziehungen gemacht haben. Aber Sie fangen ja möglicherweise gerade erst an, sich mit Access zu beschäftigen, und daher wollen wir Sie nicht gleich mit Theorie überladen, sondern zwischendurch auch was Praktisches erledigen – zum Beispiel das Erstellen der einen oder anderen Tabelle.

Welche Tabellen wir voraussichtlich benötigen, um die Projektzeiterfassung abzubilden, haben wir ja bereits ermittelt (es werden noch mehr hinzukommen, aber vorerst reichen diese). Die erste Tabelle, die wir anlegen, soll die Kunden der Firma Schwuppdiwupp speichern. Das sind zwar nicht besonders viele, aber ... ach, lassen wir das. Mit dieser ersten Tabelle können wir dann schon eine Menge erledigen – zum Beispiel eine Abfrage erstellen, welche die Kunden nach dem Namen sortiert oder ein Formular, mit dem Sie die Kunden bearbeiten.

Beispieldatenbank

Die Beispieldatenbank zu diesem Artikel finden Sie im Onlinearchiv unter

http://www.access-basics.de/index.php/Heftarchiv.html.

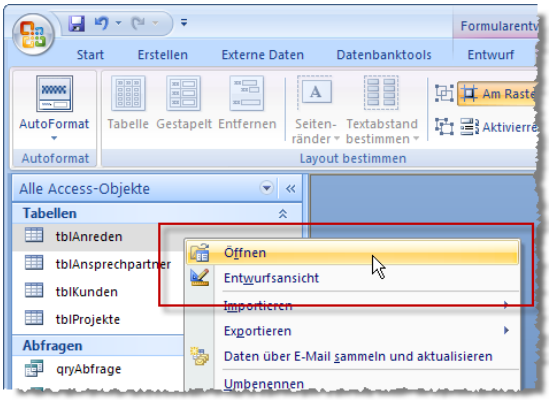
Tabellenansichten

Bevor wir uns auf das Erstellen der ersten Tabelle stürzen, sollten Sie noch schnell die beiden Ansichten von Tabellen kennenlernen:

- Die Entwurfsansicht dient der Definition der Felder der Tabelle, der Eigenschaften der Tabelle und der Eigenschaften der einzelnen Felder.
- Die Datenblattansicht kann zum Betrachten des Tabelleninhalts und zum Anlegen neuer Datensätze und zum Löschen und Bearbeiten vorhandener Datensätze dienen.

Tabelleninhalte kann man zwar in der Datenblattansicht der Tabelle bearbeiten ... man tut es aber nicht! Für das Bearbeiten von Daten gibt es schließlich Formulare!

Wenn Sie eine Tabelle in der einen oder anderen Ansicht öffnen möchten, haben Sie mehrere Möglichkeiten. Die Datenblattansicht erhalten Sie am schnellsten, indem Sie einfach doppelt auf den Tabellennamen im Datenbankfenster (Access 2003 und älter) oder im Navigationsbereich (Access 2007 und jünger) klicken. Beide Ansichten können Sie anzeigen, indem Sie mit der rechten Maus auf die gewünschte Tabelle klicken und einen der Einträge **Öffnen** (öffnet in der Datenblattansicht) oder **Entwurfsansicht** auswählen (in der Abbildung für Access 2007).



Kundentabelle entwerfen

Die erste Tabelle soll **tblKunden** heißen. Nach dem Anlegen dieser Tabelle (→*[basics]* **Tabelle erstellen**) fügen Sie die folgenden Felder hinzu:

- **KundeID**: Das Feld enthält einen eindeutigen Wert, damit jeder Kunde ein eindeutiges Merkmal besitzt. Wenn Sie den Datentyp auf Autowert einstellen, fügt Access für jeden neuen Datensatz automatisch einen Zahlenwert ein, der noch nicht verwendet wurde (→*[basics]* **Autowertfeld anlegen**). Damit Access darauf achtet, dass hier auf gar keinen Fall doppelte Werte vorkommen, legen Sie dieses Feld außerdem als Primärschlüsselfeld der Tabelle fest (→*[basics]* **Primärschlüsselfeld festlegen**).
- **Kunde**: Ein Textfeld zum Speichern des Firmennamens (→*[basics]* **Textfeld anlegen**)
- **Anrede**: Anrede des Kunden (ist vorerst ein Textfeld, später werden wir dieses in ein Nachschlagefeld umwandeln)

Formulare für die Dateneingabe

Teil I: Formulare für einfache Tabellen

Wenn Sie ein paar Datensätze über die Datenblattansicht der Tabelle eingeben und sich vorstellen, wie sich das wohl in einer Tabelle mit noch mehr Feldern anfühlt, werden Sie feststellen, dass die einzeilige Ansicht möglicherweise nicht der Weisheit letzter Schluss ist. Viel schöner wäre es doch, wenn man die Eingabefelder anders anordnen könnte – beispielsweise untereinander, wie es in den meisten Eingabeformularen von Webanwendungen der Fall ist.

Hierfür gibt es unter Access Formulare. Formulare erlauben die Bearbeitung der Inhalte von Tabellen, wobei Sie die Eingabefelder beliebig platzieren können – nebeneinander, untereinander, in einer oder mehreren Spalten. Der Fantasie sind hier keine Grenzen gesetzt – außer durch die Gesetze der Ergonomie. Und die sagen: Ordnen Sie die Steuerelemente so an, dass der Benutzer beim Eingeben von Daten größtes Vergnügen empfindet. ;-)

Beispieldatenbank

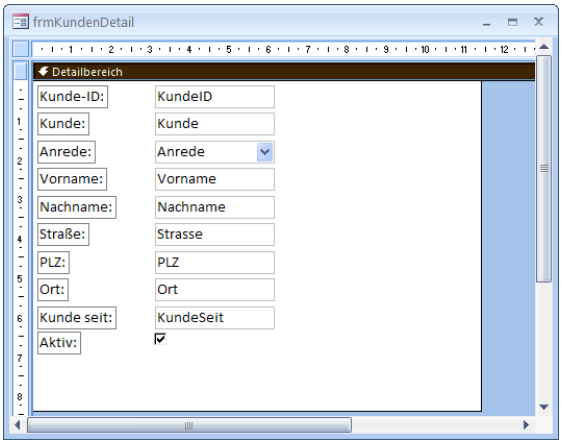
Die Beispieldatenbank zu diesem Artikel finden Sie im Onlinearchiv unter

http://www.access-basics.de/index.php/Heftarchiv.html.

Formular erstellen

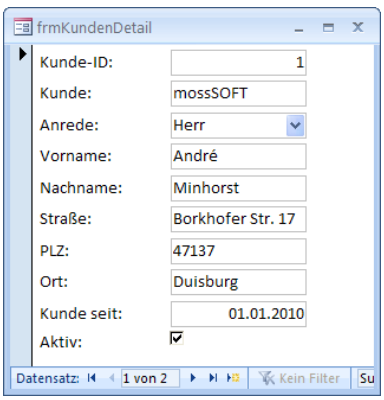
Unser Formular soll das Anlegen neuer Kunden, das Bearbeiten vorhandener Kunden und das Löschen von Kunden erlauben. Dafür brauchen Sie gar nicht so viel zu tun: Sie müssen einfach nur ein neues, leeres Formular erstellen, seine Eigenschaft **Datenherkunft** auf den Namen der Tabelle einzustellen, deren Daten das Formular anzeigen soll und die Steuerelemente zu erstellen, welche die Daten der Tabellen anzeigen.

Unser Formular soll den Namen **frmKundenDetail** erhalten, an die Tabelle **tblKunden** gebunden sein und in der Entwurfsansicht so aussehen wie in der Abbildung. Die Schritte zu dieser Ansicht, die Sie später immer wieder beim Erstellen anderer Formulare durchführen werden, können Sie unter →*[basics]* **Gebundenes Formular anlegen**nachlesen.



Wenn das Formular erstellt ist und die Steuerelemente nach Ihren Wünschen ausgerichtet sind (→*[basics]* **Steuerelemente ausrichten und skalieren**), könnte es so wie in der Abbildung rechts aussehen.

Nun ist der Zeitpunkt gekommen, sich das Resultat in der Formularansicht anzusehen – das ist die Ansicht, in der Sie mit dem Formular Daten in die zugrunde liegende Tabelle eingeben können. Wechseln Sie gleich einmal in die Formularansicht und bewundern das Ergebnis (am schnellsten gelangen Sie mit der Tastenkombination **Alt + F5** in die Formularansicht, mehr über die verschiedenen Formularansichten erfahren Sie unter →*[basics]* **Formularansichten**). Das Formular zeigt gleich den ersten Datensatz an, den Sie bereits zuvor in die Datenblattansicht der Tabelle eingegeben haben. Die in der Tabelle festgelegten Nachschlagefeld-Eigenschaften des Feldes **Anrede** wurden in das Formular übertragen, sodass dort statt eines Textfeldes ein Kombinationsfeld die Auswahl der Anrede erleichtert.



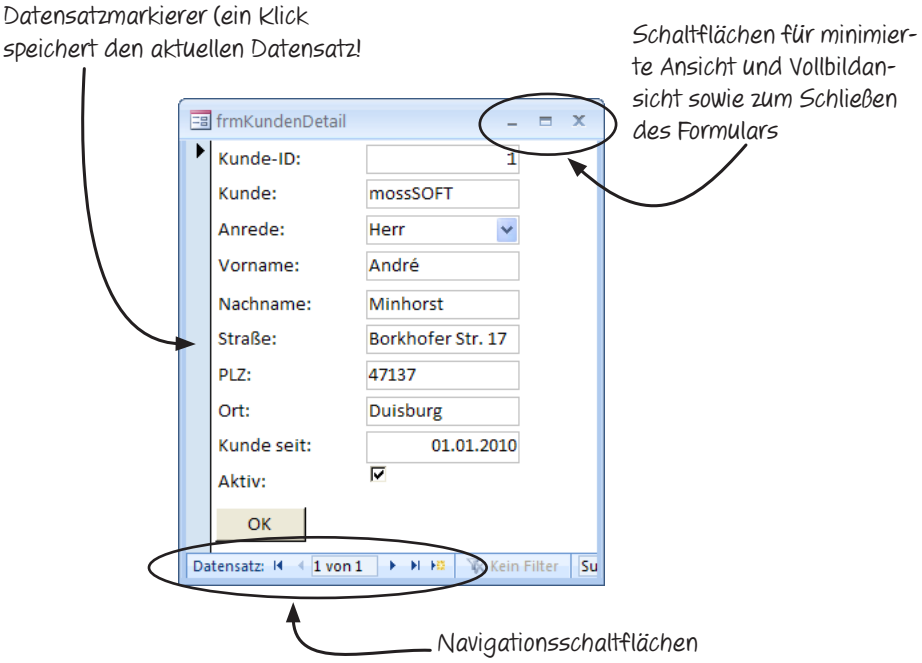
Funktionen gebundener Formulare

Nebenbei liefert Access mit dem gebundenen Formular eine ganze Reihe wichtiger Funktionen, um die Sie sich überhaupt nicht kümmern mussten:

Es fügt unten im Formular Navigationsschaltflächen ein, mit denen Sie zum ersten, vorherigen, nächsten oder letzten Datensatz springen, die absolute Position eines Datensatz eingeben und diesen dadurch anzeigen und einen neuen Datensatz anlegen können. Außerdem zeigt ein Textfeld zwischen den Navigationsschaltflächen an, welcher Datensatz aktuell angezeigt wird und wieviele Datensätze die Datenherkunft insgesamt enthält.

Es zeigt im linken Bereich den sogenannten Datensatzmarkierer an, der verschiedene Aufgaben hat:

- Anzeigen, in welchen Zustand der aktuelle Datensatz sich befindet (etwa durch Anzeige eines Stiftes für einen aktuell bearbeiteten, aber noch nicht gespeicherten Datensatzes, oder ein Gesperrt-Zeichen für einen Datensatz, der gerade anderweitig bearbeitet wird)
- Speichern der Änderungen an einem Datensatz per Mausklick auf den Datensatzmarkierer
- Markieren des Datensatzes etwa zum Löschen, Kopieren oder Ausschneiden, ebenfalls durch Anklicken des Datensatzmarkierers



Außerdem besitzen gebundene Formulare eine eingebaute Speichern-Funktion. Diese speichert automatisch Änderungen an einem Datensatz, wenn

- Sie den Datensatz wechseln,
- Sie das Formular schließen oder
- Sie zu einem eventuell vorhandenen Unterformular wechseln (dies nur der Vollständigkeit halber – später mehr zu diesem Thema).

Sie können das Speichern außerdem forcieren, indem Sie

- die Tastenkombination **Strg + S** betätigen,

- den entsprechenden Menübefehl unter Access 2003 und älter (Datensätze|Datensatz speichern) beziehungsweise den Ribbon-Eintrag **Start|Datensätze|Speichern** betätigen (Access 2007 und jünger) oder
- mit der Maus auf den Datensatzmarkierer klicken.

Wenn Sie Änderungen verwerfen möchten, können Sie dies standardmäßig nur für den aktuell angezeigten und gegebenenfalls geänderten Datensatz durchführen – und das auch nur rückwirkend bis zur letzten Speicherung des Datensatzes. Änderungen verwerfen Sie mit der **Escape**-Schaltfläche.

Viel Arbeit durch wenig Drag and Drop

Werfen wir einen Blick darauf, was beim Hinzufügen der Felder zum Detailbereich des Formulars geschehen ist: Das Einstellen von Datenherkunft auf **tblKunden** sorgt dafür, dass das Formular an die Tabelle **tblKunden** gebunden ist und deren Daten anzeigen, bearbeiten oder löschen kann. Wenn Sie, wie unter →*[basics]* **Gebundenes Formular anlegen** beschrieben, ein Feld aus der Feldliste in das Formular ziehen, geschehen gleich mehrere Schritte, die Sie auch auf andere Weise durchführen können und durch die Sie besser verstehen, warum ein Textfeld überhaupt die Daten aus der zugrunde liegenden Tabelle anzeigt. Dazu gehen Sie wie folgt vor:

- Erstellen Sie nochmals ein leeres Formular (nur für diese kleine Lektion, anschließend können Sie es wieder löschen) und öffnen es in der Entwurfsansicht.
- Stellen Sie die Eigenschaft **Datenherkunft** auf **tblKunden** ein.
- Fügen Sie ein Textfeld zum Detailbereich hinzu:

  2003 und älter: Klicken Sie in der Toolbox (Menüeintrag **Ansicht|Toolbox**) auf die Schaltfläche **Textfeld** und anschließend auf die Stelle im Detailbereich des Formulars, an der Sie das Steuerelement platzieren möchten.

  2007 und jünger: Klicken Sie im Ribbon auf den Eintrag **Entwurf|Steuerelemente|Textfeld** und anschließend auf die Stelle im Detailbereich des Formulars, an der Sie das Steuerelement einfügen möchten.

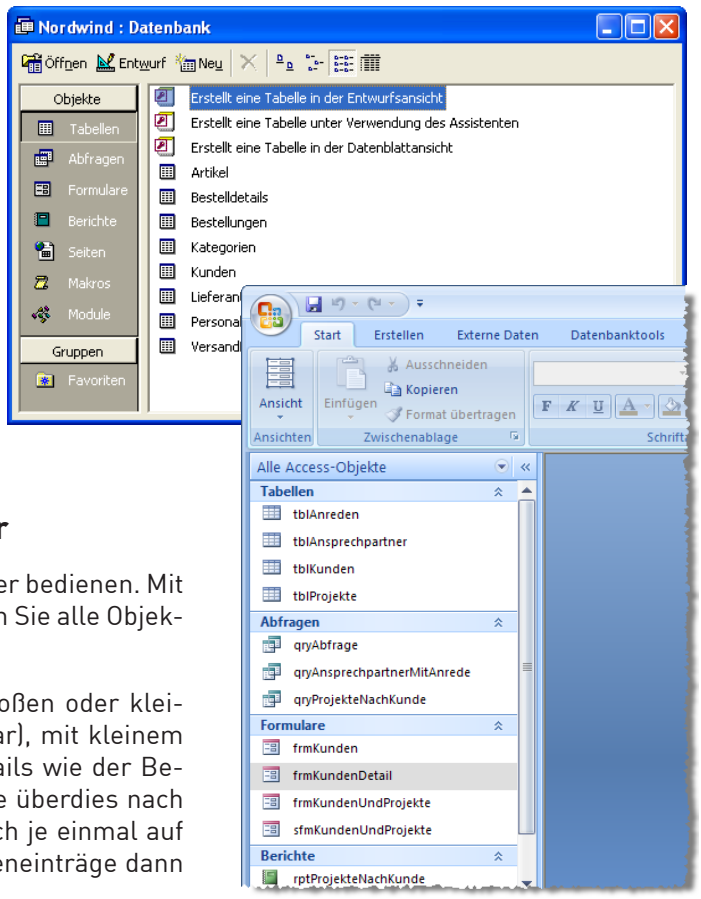
- Da Sie eine Datenherkunft für das Formular festgelegt haben, können Sie nun über die Eigenschaft **Steuerelementinhalt** das Feld der Datenherkunft auswählen, welches das Textfeld anzeigen soll – zum Beispiel **Vorname**.
- Außerdem hat Access beim Hinzufügen des Textfelds automatisch ein bezeichnungsfeld hinzugefügt, dessen Text Sie noch einstellen müssen (den Namen dieses Steuerelements können Sie belassen – er interessiert in 99% der Fälle nicht.
- Schließlich – und das ist der entscheidende Unterschied zum Anlegen eines Feldes per Drag and Drop aus der Feldliste – stellen Sie den Namen des Textfeldes ein. Und zwar nicht, wie bei obiger Methode üblich, auf den Namen des zugrunde liegenden Feldes der Tabelle, sondern auf ein dem Steuerelementtyp entsprechendes Präfix (hier **txt** für **Textbox**) und den Namen des Feldes, also beispielsweise **txtVorname**.

Die Schaltzentrale von Access

Datenbankfenster und Navigationsbereich im Detail

In einer fertigen Access-Anwendung tummeln sich eine ganze Reihe verschiedener Objekte: Tabellen, Abfragen, Formulare, Berichte, Module und Makros. Damit Sie optimal darauf zugreifen können, schauen wir uns die Schaltzentrale für Entwickler einmal genau an.

Einer der größten Unterschiede der Access-Versionen bis Access 2003 und ab Access 2007 ist die Verwaltung der Datenbankobjekte:



2003 und älter: Hier verwalten Sie alle Access-Objekte im Datenbankfenster. Dieses erscheint gleich beim Öffnen oder Anlegen einer Datenbank und bietet über verschiedene Registerseiten Zugriff auf die verschiedenen Arten von Access-Objekten. Sie können das Datenbankfenster frei positionieren und seine Größe anpassen, und es verschwindet gegebenenfalls auch hinter anderen Datenbankobjekten, wenn diese den Fokus haben.

2007 und jünger: In Access 2007 wurde das Datenbankfenster durch den Navigationsbereich ersetzt. Dieser ist am linken Rand des Access-Fensters verankert und zeigt alle Objekte untereinander an, wobei Sie die Anzeige durch verschiedene Einstellungen anpassen können. Der Navigationsbereich kann minimiert werden.

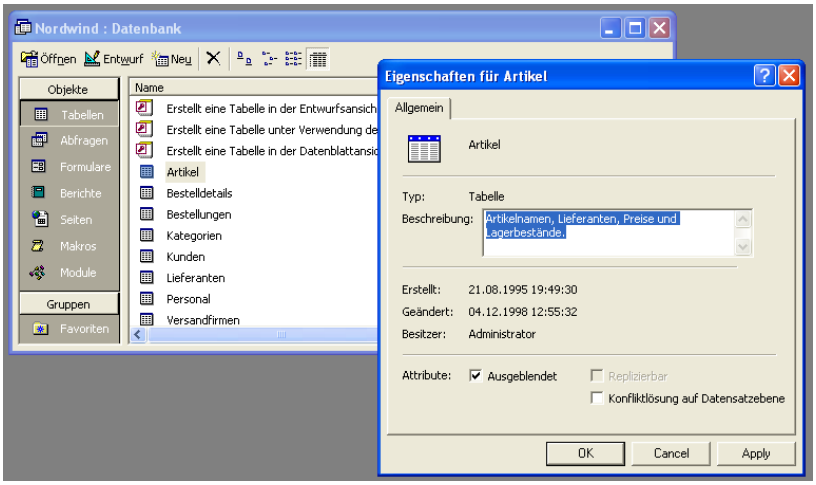
Das Datenbankfenster unter Access 2003 und älter

Das Datenbankfenster lässt sich scheinbar wesentlich intuitiver bedienen. Mit einem Klick auf eine der Objektgruppen am linken Rand zeigen Sie alle Objekte der jeweiligen Objektart im Datenbankfenster an.

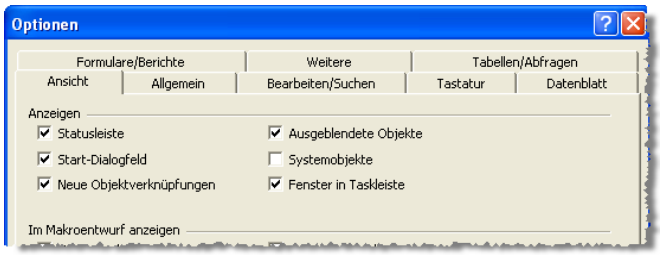
Diese können Sie auf verschiedene Arten darstellen: mit großen oder kleinen Symbolen und Name (alle Objekte sind frei positionierbar), mit kleinem Symbol und Name in Listenform oder in einer Liste mit Details wie der Beschreibung oder dem Erstellungsdatum. Die Liste können Sie überdies nach den verschiedenen Spalten sortieren. Dazu klicken Sie einfach je einmal auf den Kopf der zu sortierenden Spalte. Access sortiert die Listeneinträge dann abwechselnd auf- oder absteigend.

Objekte ein- und ausblenden

Wenn Sie möchten, können Sie einzelne Objekte vor den Augen des Benutzers verbergen. Dazu öffnen Sie per rechtem Mausklick und Auswahl des Eintrags **Eigenschaften** den Eigenschaften-Dialog für das betroffene Objekt und aktivieren die Option **Ausgeblendet**. Das Datenbankfenster zeigt das Objekt fortan nicht mehr an.



Wenn Sie möchten, dass das Datenbankfenster auch die ausgeblendeten Objekte anzeigt, können Sie dies in den Optionen von Access einstellen. Den Optionen-Dialog öffnen Sie mit dem Menüeintrag **Extras|Optionen**.



Dort finden Sie auf der Registerseite **Ansicht** die Option **Ausgeblendete Objekte**. Aktivieren Sie diese Option, erscheinen ausgeblendet Objekte mit einem leicht veränderten Symbol im Datenbankfenster. So können Sie erneut auf die Eigenschaften eines ausgeblendeten Objekt zugreifen und es wieder einblenden.

Systemobjekte ein- und ausblenden

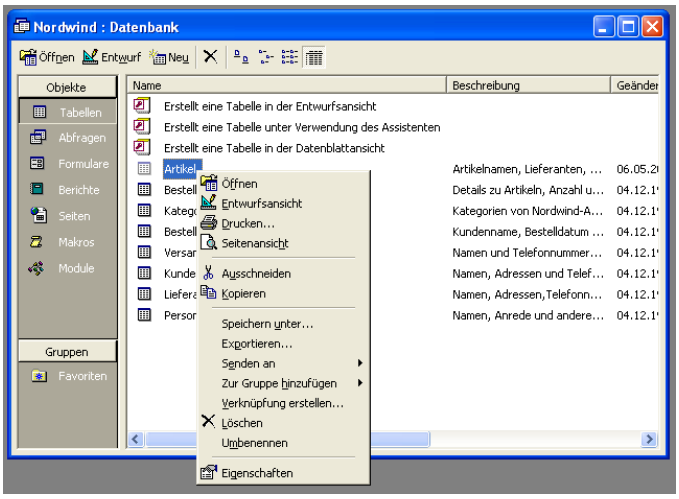
Access speichert einige interne Informationen, zum Beispiel über die enthaltenen Objekte, in Systemtabellen. Diese sind normalerweise nicht sichtbar. Wenn Sie jedoch die Option **Systemobjekte** in obigem Dialog aktivieren, finden Sie im Datenbankfenster auch die Systemtabellen vor. Aktuell können Sie mit diesen sicher nicht viel anfangen, aber später zeigen wir Ihnen interessante Anwendungsfälle.

Aktionen im Datenbankfenster

Das Datenbankfenster stellt verschiedene Möglichkeiten für den Zugriff auf die angezeigten Objekte bereit. Mit einem Doppelklick

- öffnen Sie eine Tabelle oder Abfrage in der Datenblattansicht,
- öffnen ein Formular oder einen Bericht in der unter Standardansicht eingestellten Ansicht,
- führen ein Makro aus oder
- öffnen ein VBA-Modul im VBA-Editor.

Im oberen Bereich des Datenbankfensters finden Sie weitere Aktionen. So können Sie das jeweils markierte Objekt in der Entwurfsansicht öffnen oder ein neues Objekt des Typs erstellen, dessen Registerseite gerade angezeigt wird. Außerdem finden Sie dort eine Schaltfläche zum Löschen des aktuell markierten Objekts.



All diese Möglichkeiten und mehr bietet auch das Kontextmenü eines der Einträge des Datenbankfensters an, wenn Sie mit der rechten Maustaste auf einen der enthaltenen Einträge klicken.

Hilfreich sind auch die Einträge, die Microsoft den Listen der Datenbankobjekte hinzugefügt hat: Im Bereich **Tabellen** können Sie so beispielsweise eine neue Tabelle in der Entwurfsansicht erstellen oder den Dialog zum Auswählen eines Assistenten zum Erstellen einer Tabelle öffnen.

Access [basics]
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen
Ausgabe 3/2010
4. Juni 2010

Tabellen entwerfen

Teil II: Kunden und Projekte verwalten

Im ersten Teil dieser Artikelserie haben wir eine Kundentabelle entworfen, mit der Sie auf einfache Weise Kundendaten verwalten können. Auf ähnliche Weise gestalten wir nun eine Projekte-Tabelle, und dann wird es richtig spannend: Wir bringen die beiden Tabellen so zusammen, dass Sie jedes Projekt einem Kunden zuordnen können.

Was Sie in diesem Artikel lernen

» Verknüpfen von Tabellen
» Herstellen einer 1:n-Beziehung
» Umgang mit dem Beziehungen-Fenster
» Grundlagen der referentiellen Integrität
» Erstellen eines Nachschlagefeldes

Dazu brauchen wir eine Projekte-Tabelle, die alle notwendigen Daten eines Projekts speichern soll. Aber welche Daten sind dies? Im Kontext unserer Beispielanwendung möchten wir erfassen, welcher Mitarbeiter wann mit welchen Tätigkeiten zur Durchführung eines Projekts beigetragen hat.

Dabei soll ein Projekt aus mehreren Aufgaben bestehen, und eine Aufgabe aus mehreren Tätigkeiten, die durch jeweils einen Mitarbeiter durchgeführt werden.

Dies landet aber nicht alles in der Projekte-Tabelle – sonst könnten wir ja gleich so weitermachen, wie im Artikel **Vom Excel-Chaos zur Access-Projektzeiterfassung** in Ausgabe 2/2010 beschrieben. Nein, diese Daten werden fein säuberlich aufgeteilt: Die Tätigkeiten landen in einer Tätigkeiten-Tabelle, die Aufgaben in einer Aufgaben-Tabelle und allein die projektspezifischen Daten werden in der Projekte-Tabelle gespeichert.

Und da wir mit der Projektzeiterfassung wirklich erstmal nur die Zeiten für die durchgeführten Arbeiten aufnehmen möchten, brauchen wir für jedes Projekt nur ganz wenige Daten zu speichern.

Die Projekte-Tabelle soll, den Vorgaben aus →*[basics]* **Konventionen** folgend, den Namen **tblProjekte** erhalten. Sie soll einige Felder mit den grundlegenden Projektinformationen enthalten, um das Projekt soweit wie hier nötig zu abstrahieren. Im einzelnen sind dies die folgenden Felder:

ProjektID: Dieses Feld soll eine eindeutige ID für jedes Projekt enthalten. Damit wir uns nicht um die Vergabe dieser ID kümmern müssen, überlassen wir Access die Arbeit und stellen den Feldda-

In dieser Aufgabe:

S. 1: Tabellen entwerfen
Teil II: Kunden und Projekte verwalten
S. 10: Abfragen für die Datenauswahl
Teil I: Einfache Auswahlabfragen kennenlernen
S. 17: Abfragen für die Datenauswahl
Teil II: Darstellung von 1:n-Beziehungen
S. 20: Formulare für die Dateneingabe
Teil II: Formulare für Daten aus 1:n-Beziehungen

Auf zur ersten Anwendung!

Nachdem wir uns in der ersten Ausgabe von Access [basics] mit den Grundlagen zum Einsatz von Microsoft Access und seinen Möglichkeiten beschäftigt haben und die zweite Ausgabe einen ersten Einblick in die Gestaltung von Tabellen und Formularen und ihrer Zusammenarbeit lieferte, gehen wir nun einen großen Schritt weiter.

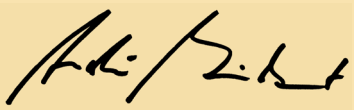
Sie erfahren, wie Sie Projekt- und Kundendaten auf zwei Tabellen aufteilen. Warum? Weil Kunden und Projekte in einer einzigen Tabelle spätestens dann Redundanzen (also doppelt gehaltene Daten) nach sich ziehen, wenn Sie mehr als nur ein Projekt für einen Kunden verwalten möchten.

Da Kunden und Projekte aber nun einmal zusammengehören, zeigen wir Ihnen, wie Sie die Tabellen zum Speichern dieser Daten derart verknüpfen, dass Sie jederzeit nachvollziehen können, welches Projekt zu welchem Kunden gehört. Und noch mehr: Sie lernen auch noch, wie Sie beim Anlegen eines neuen Projekts ganz leicht den Kunden auswählen, für den Sie dieses Projekt durchführen.

Außerdem beschäftigt sich diese Ausgabe von Access [basics] erstmalig mit dem Thema Abfragen, und zwar in gleich zwei Artikeln. Die erste zeigt, wie Sie einfache Abfragen auf Basis einer einzigen Tabelle ausführen und damit etwa nur bestimmte Spalten anzeigen, Datensätze mit speziellen Eigenschaften herausfiltern oder diese nach den gewünschten Kriterien sortieren. Und natürlich lassen sich Abfragen auch dazu nutzen, Daten aus verschiedenen Tabellen zusammenzuführen - und genau das ist Thema des zweiten Artikels über Abfragen. Hier lernen Sie, wie Sie Kunden- und Projektdaten wieder zusammenführen.

Und zu guter Letzt erfahren Sie auch noch, wie Sie Kunden- und Projektdaten im Zusammenhang in Formularen darstellen, um diese komfortabel zu bearbeiten.

Viel Spaß!



[André Minhorst]

tentyp auf Autowert ein →[basics] **Autowertfeld anlegen**. Außerdem soll dieses Feld als Primärschlüsselfeld dienen →[basics] **Primärschlüsselfeld festlegen**.

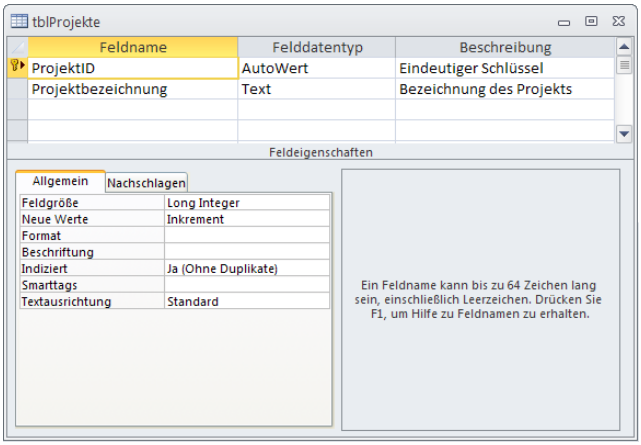
Projektbezeichnung: Eine Bezeichnung für das Projekt, zum Beispiel **Access [basics] Ausgabe 3**. Das Feld soll den Datentyp **Textfeld** erhalten →[basics] **Textfeld anlegen**.

Beispieldatenbank

Die Beispieldatenbank zu diesem Artikel finden Sie im Onlinearchiv unter

http://www.access-basics.de/index.php/Heftarchiv.html.

Zum Anlegen der Tabelle gehen Sie wie in →[basics] **Tabelle erstellen** vor. Danach sollten Sie in der Entwurfsansicht der Tabelle etwa den Entwurf aus der Abbildung rechts sehen.



Diese Tabelle enthält schon alle wesentlichen Projektdaten. Bis auf eine Kleinigkeit ...

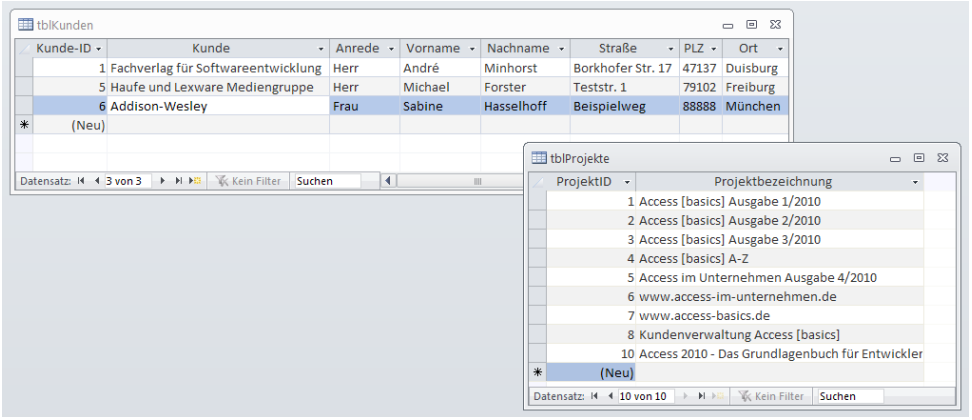
Später, wenn Sie die Projektzeiterfassung erweitern möchten, um beispielsweise auch die Projektplanung zu berücksichtigen, können Sie die Projekttabelle erweitern. Vorerst fehlt uns allerdings nur eine Kleinigkeit, denn: Woher wissen wir eigentlich, für welchen Kunden wir dieses Projekt durchführen?

In der Excel-Tabelle, die wir im Artikel **Access im Netzwerk-/Mehrbenutzerbetrieb** in Ausgabe 2/2010 vorgestellt haben und mit der dort die Projektzeiterfassung durchgeführt wurde, erfasste jede Zeile alle Informationen vom Kunden über das Projekt bis hin zur Aufgabe und den einzelnen Tätigkeiten.

Dadurch wurden aber zum Beispiel die Kundendaten für jede einzelne Projekttätigkeit erneut aufgeführt, was zu doppelten Daten führte und sicher irgendwann in Inkonsistenzen im Datenbestand enden wird (Das heißt, dass der Kunde XY einmal XY und ein zweites Mal etwa durch einen Schreibfehler YX heißt. Eine Suche nach allen Tätigkeiten für den Kunden XY liefert so ein fehlerhaftes Ergebnis, weil nicht alle Tätigkeiten aufgeführt werden.)

Eines der Hauptziele war daher, jeden Kunden nur einmal zu erfassen. Das Gleiche gilt für die Projekte: Auch diese sollten jeweils nur einmal in die Datenbank aufgenommen werden. Deshalb haben wir bereits zwei Tabellen erzeugt, die ausschließlich Kunden und Projekte speichern – und zwar jeden Kunden und jedes Projekt nur einmal!

Kunden und Projekte verknüpfen



Die Tabellen zum Speichern von Kunden und Projekten liegen schon mal vor. Aber woher wissen wir, welches Projekt zu welchem Kunden gehört?

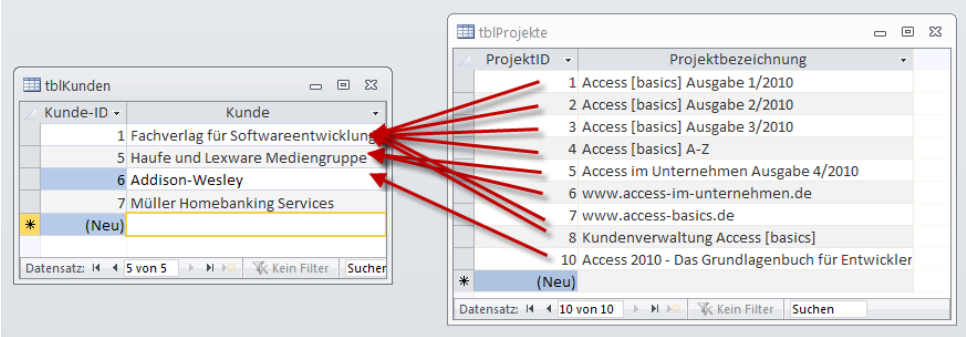
Und damit kommen wir zu dem Punkt, der eine Datenbank überhaupt zu einer relationalen Datenbank macht: Die Verknüpfung von Daten aus verschiedenen Tabellen. In der Beispieldatenbank zu diesem Artikel finden Sie aktuell zwei Tabellen vor, die wie in der Abbildung rechts aussehen. Die Tabelle **tblKunden** enthält einige Kundendatensätze, die Tabelle **tblProjekte** wurde mit einigen Projekten gefüllt. Welches Projekt gehört aber nun zu welchem Kunden? Und müssten wir diese Information nicht auf irgendeine Art speichern?

Schauen wir uns erst einmal an, wie die Datensätze der Tabelle **tblProjekte** überhaupt mit denen der Tabelle **tblKunden** verknüpft sind. Die folgende Abbildung deutet die Beziehungen zwischen den einzelnen Datensätzen durch die eingezeichneten Pfeile an. Dort sind eine ganze Reihe Projekte mit dem ersten Kunden der Tabelle **tblKunden** verknüpft, zwei mit dem zweiten

Kunden und eines mit dem dritten Kunden. Dem vierten Kunden der Tabelle **tblKunden** wurde gar überhaupt kein Projekt der Tabelle **tblProjekte** zugewiesen.

Wir halten also fest: Wir speichern zwar mehrere Projekte für einen einzigen Kunden in der Tabelle **tblKunden**, aber jedes der Projekte gehört zu genau einem Kunden. Aus Kundensicht bedeutet dies: Jeder Kunde kann keines, eines oder auch mehrere Projekte besitzen.

Und können wir sicher ausschließen, dass kein Projekt einmal für mehr als einem Kunden durchgeführt wird? Ja, im Rahmen der geplanten Projektzeiterfassung gehen wir davon aus.



Zu jedem Kunden gehören keins, eines oder viele Projekte. Jedes Projekt gehört zu genau einem Kunden.

Feld zum Speichern verknüpfter Projekte und Kunden

Wie oben erwähnt, müssen wir die Information über die Zuordnung von Projekten zu Kunden in irgendeiner Form in einer der Tabellen speichern (es gibt auch noch die Möglichkeit, dies in einer zusätzlichen Tabelle zu erledigen – dazu kommen wir später).

Nun haben wir jeder Tabelle ein eindeutiges Indexfeld hinzugefügt: der Tabelle **tblKunden** das Feld **KundeID** und der Tabelle **tblProjekte** das Feld **ProjektID**.

Damit lassen sich auf Basis der obigen Abbildung Aussagen treffen wie „Projekte **1**, **2**, **3**, **4**, **7** und **8** gehören zu Kunde **1**, die Projekte **5** und **6** gehören zu Kunde **5** und Projekt **10** gehört zu Kunde **6**.

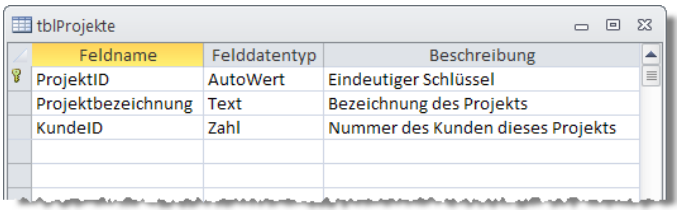
Diese Beziehungen müssen wir an irgendeiner Stelle speichern, in diesem Falle in einer der beiden Tabellen. Wir legen also entweder in der Tabelle **tblKunden** ein Feld namens **ProjektID** an, dass die **ProjektID** enthält, das zu dem Kunden gehört, oder wir legen in der Tabelle **tblProjekte** ein Feld namens **KundeID** an, dass die **KundeID** enthält, für den das Projekt durchgeführt wird.

Das würde bedeuten, dass das mögliche Feld **ProjektID** in der Tabelle **tblKunden** durchaus auch einmal mehrere Werte annehmen müsste, weil einem Kunden ja kein, ein oder mehrere Projekte zugeordnet werden können. Andersherum brauchen wir in das andere mögliche Feld **KundeID** in der Tabelle **tblProjekte** immer nur einen Wert des Feldes **KundeID** der Tabelle **tblKunden** einzutragen, da ja jedes Projekt ganz sicher nur im Auftrag eines einzigen Kunden durchgeführt wird.

Und hier kommen bestimmte Gesetze ins Spiel, die beim Modellieren von Tabellen und deren Beziehungen gelten – und die wir uns später im Detail ansehen. Für uns ist hier und jetzt wichtig: Ein Feld darf nicht mehr als eine gleichartige Information aufnehmen. Das bedeutet: Wir können kein Feld namens **ProjektID** in der Tabelle **tblKunden** einfügen, dass eine Auflistung aller Projekte dieses Kunden enthält.

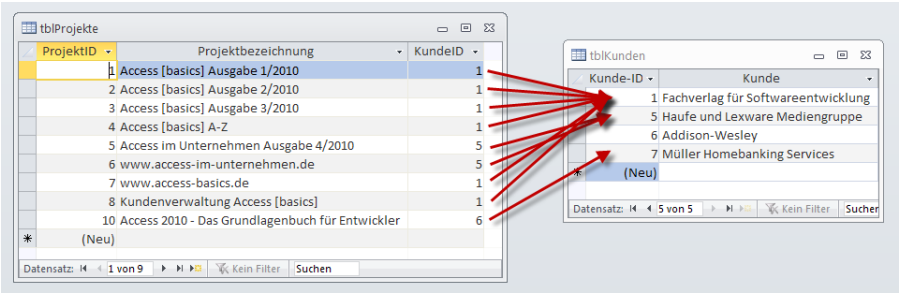
Also bleibt nur die Möglichkeit, die Tabelle **tblProjekte** um ein Feld namens **KundeID** zu erweitern, in das wir die **KundeID** des Kunden eintragen, für den das entsprechende Projekt durchgeführt wird.

Nach dem Hinzufügen eines Zahlenfeldes mit der Bezeichnung **KundeID** (→*[basics]* **Zahlenfeld anlegen**) sieht die Tabelle in der Entwurfsansicht nun wie in der Abbildung rechts aus. Die Eigenschaft **Feldgröße** belassen Sie auf dem Wert **Long Integer** (hier nicht dargestellt).



Die 1:n-Beziehung

Fein: Damit können wir nun in das Feld **KundeID** der Tabelle **tblProjekte** jeweils die **KundeID** aus der Tabelle **tblKunden** eintragen. Eigentlich sind wir damit schon am Ziel angelangt: Kunden und Projekte werden in eigenen Tabellen verwaltet, und wir haben eine Möglichkeit geschaffen, um anzugeben, welches Projekt zu welchem Kunden gehört.



Wenn wir hiermit schon zufrieden wären, hätten wir allerdings nicht Access anschmeißen müssen, sondern hätten uns mit Excel zufrieden geben können. Mit dem aktuellen Zustand gehen nämlich einige Probleme einher:

- Wir müssen jedesmal manuell die **KundeID** des Kunden in das entsprechende Feld der Tabelle **tblProjekte** eintragen. Das mag an dieser Stelle noch nicht so schwerwiegend sein, weil wir nicht so oft neue Kunden und Projekte anlegen, aber später beim Anlegen der Tabellen für die Aufgaben und Tätigkeiten wollen wir ja auf ähnliche Weise vorgehen – und dort würde durch die manuelle Eingabe in großer Mehraufwand entstehen.

- Wir können alle möglichen Zahlen in das Feld **KundeID** der Tabelle **tblProjekte** eintragen, also auch solche, die gar nicht im Feld **KundeID** der Tabelle **tblKunden** vorkommen, oder dieses Feld einfach leer lassen – was bedeuten würde, dass wir die Tätigkeiten zu einem solchen Projekt keinem Kunden in Rechnung stellen könnten.

Daraus folgern gleich die Anforderungen für die nächste Version unserer Tabellen:

- Access soll für uns kontrollieren, ob überhaupt ein Wert in das Feld **KundeID** der Tabelle **tblProjekte** eingegeben wurde und ob dieser Wert einem der vorhandenen Werte des Feldes **KundeID** der Tabelle **tblKunden** entspricht.

- Access sollte uns auf irgendeine Weise dabei helfen, einen der gültigen Werte in das Feld **KundeID** einzutragen.

- Und, was noch schöner wäre: Vielleicht könnte man im Feld **KundeID** der Tabelle **tblProjekte** gleich noch erkennen, um welchen Kunden es sich handelt, denn die von Access vergebenen Zahlen für das Feld **KundeID** der Tabelle **tblKunden** sind ja nun nicht besonders aussagekräftig.

Herstellen einer Beziehung zwischen den Tabellen tblProjekte und tblKunden

Beginnen wir mit der Kontrollfunktion, die sicherstellt, dass das Feld **KundeID** der Tabelle **tblKunden** immer einen gültigen Wert enthält. Auch wenn wir schon wissen, dass **KundeID** in der Tabelle **tblProjekte** immer einen Wert des Feldes **KundeID** der Tabelle **tblKunden** enthalten soll, kann Access dies mangels hellseherischer Fähigkeiten nicht erkennen.

Abfragen für die Datenauswahl

Teil I: Einfache Auswahlabfragen kennenlernen

Abfragen dienen, wie der Name schon sagt, zum Abfragen – und zwar von Daten der Tabellen Ihrer Access-Anwendung. Sie können damit die Daten einer einzigen Tabelle nach bestimmten Kriterien sortieren oder filtern, und auch das Zusammenführen von Daten aus mehreren Tabellen ist möglich.

Was Sie in diesem Artikel lernen

» Erstellen einfacher Abfragen
» Abfragen bestimmter Felder einer Tabelle
» Abfragen von Datensätzen mit speziellen Eigenschaften
» Sortieren von Daten nach verschiedenen Kriterien

Abfragen fragen Daten ab - das sagt ja schon der Name. Zur besseren Unterscheidung von anderen Abfragetypen reden wir auch von Auswahlabfragen.

Darüberhinaus bieten Abfragen, soviel dürfen wir vorab verraten, auch dem Editieren von Daten: Mit sogenannten Aktionsabfragen können Sie Datensätze zu einer Tabelle hinzufügen, bestehende Datensätze löschen oder die in den Datensätzen einer Tabelle enthaltenen Daten ändern.

Abfragen können aber auch Datensätze erstellen, löschen und manupilieren. Solche Abfragen heißen Aktionsabfragen.

Und die Möglichkeiten von Abfragen gehen noch weiter: Sie können damit sogar ganze Tabellen erstellen, Felder hinzufügen, ändern oder löschen und Beziehungen zwischen den Tabellen definieren.

Und sogar Tabellen und Beziehungen können Sie mit Abfragen erstellen, löschen und verändern!

Beides, sowohl das Ändern von Daten als auch das Bearbeiten des Tabellenentwurfs mithilfe von Abfragen, liegt noch ein ganzes Stück weit weg. Bis dahin reicht es, wenn Sie diese Aufgaben über die Elemente der Benutzeroberfläche von Access durchführen.

Grundsätzliches zu Auswahlabfragen

Der wesentliche Unterschied zwischen Tabellen und Auswahlabfragen ist, dass Tabellen Daten speichern und Auswahlabfragen die in den Tabellen gespeicherten Daten abfragen und anzeigen. Im einfachsten Fall zeigt eine Abfrage genau die gleichen Felder und die gleichen Datensätze an, die Sie auch vorfinden würden, wenn Sie die Tabelle in der Datenblattansicht öffnen. In der Regel werden Sie durch eine Auswahlabfrage aber eine spezielle Sicht auf die in den Tabellen gespeicherten Daten erhalten. So können Sie

- nur bestimmte Felder einer Tabelle über eine Abfrage ausgeben,
- nur bestimmte Datensätze einer Tabelle anzeigen,
- die Datensätze nach dem Inhalt eines oder mehrerer Felder sortieren,
- die Daten nach bestimmten Kriterien gruppieren oder
- zusätzlich zu den ausgewählten Feldern weitere Felder anzeigen, die beispielsweise Berechnungsergebnisse auf Basis der übrigen Felder liefern.

Ein paar Felder hier, ein paar Datensätze da, fertig ist das Abfrageergebnis.

Das Ergebnis einer Abfrage sieht genauso aus wie eine Tabelle in der Datenblattansicht. Und in den meisten Fällen können Sie auch genauso damit arbeiten: Sie können darin Datensätze anlegen und bestehende Daten bearbeiten oder löschen. Die Änderungen schlagen sich dann auf die Daten der Tabellen nieder, auf denen die Abfrage basiert.

Die Datenblattansicht von Tabellen und Abfragen sieht genau gleich aus.

Eine Abfrage liefert aber immer nur eine Sicht auf den aktuellen Zustand der ausgewählten Daten! Das Ergebnis der Abfrage wird in dieser Form

Abfrageergebnisse sind immer aktuell!

nirgends gespeichert (es sei denn, Sie wünschen dies explizit). Der Vorteil ist: Eine Abfrage liefert immer eine aktuelle Sicht auf die Daten.

Eine der wichtigsten Funktionen von Abfragen ist es jedoch, die Daten aus verschiedenen Tabellen zusammenzuführen. Access ist zwar ein relationales Datenbanksystem, in dem Daten möglichst auf mehrere Tabellen aufgeteilt werden sollen, damit diese nicht doppelt vorkommen. Das heißt aber nicht, dass der Benutzer die auf die unterschiedlichen Tabellen aufgeteilten Daten nicht doch für bestimmte Zwecke im Zusammenhang betrachten möchte.

Das ist schon schräg, oder? Da pflücken wir Kunden, Projekte, Aufgaben und Tätigkeiten auseinander und teilen sie auf verschiedene Tabellen auf, nur um sie anschließend in Abfragen wieder zusammenzuführen ...

Abfragen liefern genau das Extrakt der Daten, das Sie für einen bestimmten Zweck brauchen - etwa für die Anzeige im Formular oder Bericht.

Wozu Auswahlabfragen?

Was Sie mit Auswahlabfragen anstellen können, wissen Sie jetzt – aber wozu das Ganze? Nun: Die Access-Welt besteht nicht nur aus Tabellen und Abfragen, sondern auch noch aus Formularen, Berichten und VBA-Prozeduren. Formulare zeigen Daten an, und Berichte auch. Beide können ihre Daten direkt aus Tabellen beziehen, aber in den allermeisten Fällen werden nicht alle Felder oder nicht alle Datensätze benötigt. Und hier kommen die Abfragen ins Spiel, mit denen Sie aus den vielen in Tabellen gespeicherten Daten genau die rauspicken, die der Benutzer in seinem Formular bearbeiten oder per Bericht ausdrucken möchte. Und auch in VBA-Prozeduren werden Sie Abfragen verwenden, um die dort zu bearbeitenden Daten vorzusortieren oder zu filtern.

Abfrage anlegen

Kommen wir zum eigentlichen Thema dieses Artikels: Das Erstellen und Ausführen einfacher Auswahlabfragen. »Einfach« deshalb, weil wir uns auf die Anzeige von Daten aus einer einzigen Tabelle beschränken werden. In einem weiteren Artikel, den Sie unter dem Titel **Abfragen für die Datenauswahl - Teil II: Darstellung von 1:n-Beziehungen** finden, erweitern wir die hier erworbenen Kenntnisse um die Anwendung auf Abfragen mit zwei Tabellen.

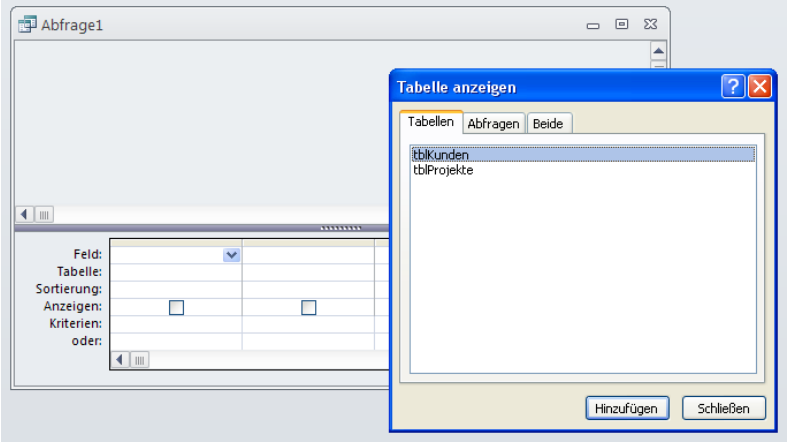
In diesem Artikel beschäftigen wir uns zum Warmwerden mit einfachen Auswahlabfragen. Weiter unten steigern wir dann den Schwierigkeitsgrad ...

Eine Abfrage legen Sie je nach Access-Version auf verschiedene Arten an:

2003 und älter: Aktivieren Sie das Datenbankfenster (falls nicht sichtbar: Taste **F11** drücken) und wechseln Sie dort zum Register **Abfragen**. Klicken Sie auf die Schaltfläche **Neu**, um eine neue, leere Abfrage in der Entwurfsansicht zu erstellen.

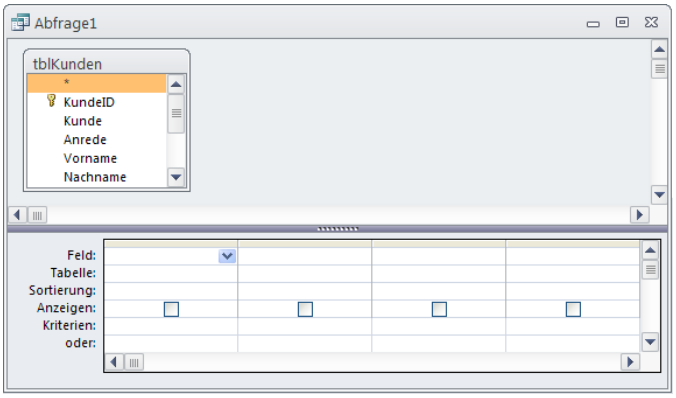
2007 und jünger: Klicken Sie im Ribbon auf den Eintrag **Erstellen|Andere|Abfrageentwurf** (Access 2007) beziehungsweise Eintrag **Erstellen|Abfragen|Abfrageentwurf** (Access 2010) .

In beiden Fällen erscheint eine unberührte Abfrage in der Entwurfsansicht, für die Sie im Dialog **Tabelle anzeigen** zunächst die Tabelle(n) auswählen müssen, deren Daten Sie anzeigen möchten. Diese Entwurfsansicht ist ein sehr mächtiges Werkzeug, das Ihnen viel Arbeit abnehmen kann. Warum? Weil sich hinter jeder Abfrage, die Sie damit mit nur wenigen Mausklicks zusammenstellen, ein Ausdruck in der Abfragesprache **SQL** (**Structured Query Language**, **Strukturierte Abfragesprache**) verbirgt. Dieser kann je nach der Anzahl der beteiligten Tabellen und der festgelegten Kriterien und weiterer Einstellungen sehr kompliziert werden.



Vorerst jedoch brauchen Sie sich nicht um SQL zu kümmern, sondern erfahren erstmal, wie Sie eine einfache Abfrage auf Basis der Tabelle **tblKunden** erstellen.

Im Dialog **Tabelle anzeigen** wählen Sie dazu einfach die Tabelle **tblKunden** aus, klicken auf **Hinzufügen** und schließen den Dialog dann per Mausklick auf die Schaltfläche **Schließen**. Was übrig bleibt, ist die Entwurfsansicht der Abfrage und eine Feldliste der Felder der ausgewählten Tabelle.



Hier wird ersichtlich, dass der Begriff der Auswahlabfrage doppeldeutig ist: Sie können nämlich erstmal auswählen, aus welchen Feldern die anzuzeigenden Daten stammen, und später legen Sie über entsprechende Kriterien fest, für welche Datensätze die Inhalte der gewählten Felder ausgegeben werden sollen.

Wichtig ist, dass Sie die Bereiche der Entwurfsansicht für Abfragen richtig einordnen: Der obere Bereich zeigt die Feldliste aller Tabellen an, deren Felder für die Ausgabe im Abfrageergebnis zur Verfügung stehen. Der untere Bereich legt fest, aus welchen Feldern der Tabellen die anzuzeigenden Daten stammen. Außerdem stellen Sie dort die Sortierungen, Filterkriterien und weitere Informationen wie Gruppierungen und Gruppierungsfunktionen ein, auf die wir später eingehen. Diesen unteren Bereich nennen wir im Folgenden **Entwurfsraster**.

Alle Felder im Abfrageergebnis anzeigen

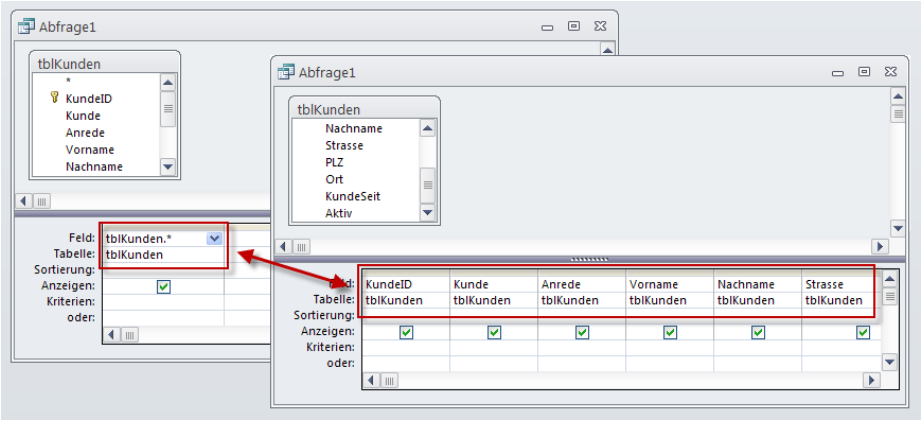
Wenn das Abfrageergebnis alle Felder der Datenherkunft enthalten soll, haben Sie zwei Möglichkeiten:

- Fügen Sie das Sternchen(*) ganz oben aus der Feldliste der Datenherkunft zum Entwurfsraster des Abfrageentwurfs hinzu. Dies erledigen Sie entweder per Drag and Drop oder durch einen Doppelklick auf den Sternchen-Eintrag. Access zeigt gleich im Abfrageergebnis automatisch die Daten für alle Felder der Datenherkunft der Abfrage an.

- Alternativ können Sie auch einfach alle Felder der Datenherkunft aus der Feldliste zum Entwurfsraster hinzufügen. Dazu klicken Sie entweder doppelt auf jeden einzelnen Eintrag, ziehen diese einzeln per Drag and Drop in das Entwurfsraster oder markieren alle Einträge und ziehen diese dann gemeinsam in den unteren Teil der Entwurfsansicht. Beim Markieren können Sie die **Umschalt**- und die **Strg**-Taste einsetzen: Bei gedrückter **Umschalt**-Taste werden alle Einträge der Feldliste vom ersten bis zum letzten angeklickten Eintrag markiert, und wenn Sie die **Strg**-Taste gedrückt halten, können Sie Einträge einzeln aus- und wieder abwählen.

Die Tabelle, deren Daten die Abfrage ausgeben soll, nennen wir Datenherkunft. Erinnern Sie sich? Auch ein Formular hat eine Eigenschaft namens Datenherkunft, für die Sie eine Tabelle (oder auch eine Abfrage) angeben können. Dies gilt übrigens auch für Berichte, wie Sie noch lernen werden.

Die Abbildung rechts zeigt die Entwurfsansicht für beide Varianten. Egal, ob Sie alle Felder aus der Feldliste mit dem Sternchen-Feld oder durch Hinzufügen der einzelnen Felder im Entwurfsraster platzieren – das Ergebnis wird später den Inhalt aller Felder wiedergeben.



Wo ist aber nun der Unterschied? Der wichtigste ist die größere Flexibilität beim Hinzufügen jedes einzelnen Feldes. Sie können damit vorher fest-

Abfragen für die Datenauswahl

Teil II: Darstellung von 1:n-Beziehungen

Im Artikel Tabellen entwerfen - Teil II: Kunden und Projekte verwalten haben Sie erfahren, wie Sie Tabellen verknüpfen. Nun wollen wir auch wissen, welche Folgen dies für die Gestaltung von Abfragen hat und wie Sie die Daten aus zwei miteinander verknüpften Tabellen sinnvoll abfragen können.

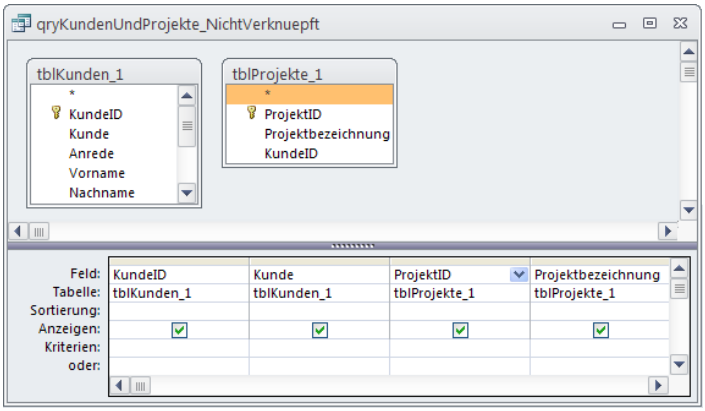
Was Sie in diesem Artikel lernen

- Zusammenführen der Daten zweier Tabellen per Abfrage
- Nur verknüpfte Datensätze anzeigen
- Beziehungen per Abfrage schaffen
- Beziehung durch Verwendung eines Kriteriums realisieren
- Bedeutung von Beziehungen für das Abfrageergebnis

Prinzipiell erstellen Sie eine Abfrage auf Basis zweier Tabellen wie solche mit nur einer Tabelle and Datenherkunft – Sie müssen lediglich eine Tabelle mehr in den Entwurfsbereich des Abfrageentwurfs ziehen.

Kombination zweier Tabellen ohne Beziehung

Um die Auswirkung des Hinzufügens einer zweiten Tabelle zum Abfrageentwurf zu beobachten, verwenden wir zunächst die beiden Tabellen aus dem oben genannten Artikel ohne Vorhandensein einer Beziehung zwischen den beiden Tabellen.



Das bedeutet, dass Sie eine neue Abfrage erstellen (→*[basics]* **Abfrage erstellen**) und die beiden Tabellen **tblKunden** und **tblProjekte** zum Entwurf der Abfrage hinzufügen. Zu diesem Zweck liefert die Beispieldatenbank Kopien dieser beiden Tabellen namens **tblKunden_1** und **tblProjekte_1**. Ziehen Sie einige Felder dieser beiden Tabellen in das Entwurfsraster.

Überlegen Sie sich, wie das Ergebnis dieser Abfrage aussehen könnte. Wird es die Kundendaten kombiniert mit den Daten ihrer Projekte liefern? Liefert es überhaupt Daten? Ein Wechseln in die Datenblattansicht der Abfrage offenbart, dass dies nicht die richtige Formulierung der Abfrage ist: Diese liefert nämlich alle überhaupt möglichen Kombinationen aus Kunden und Projekten.

Kunde-ID	Kunde	ProjektID	Projektbezeichnung
1	Fachverlag für Softwareentwicklung	1	Access [basics] Ausgabe 1/2010
5	Haufe und Lexware Mediengruppe	1	Access [basics] Ausgabe 1/2010
6	Addison-Wesley	1	Access [basics] Ausgabe 1/2010
7	Müller Homebanking Services	1	Access [basics] Ausgabe 1/2010
8	Klaus Müller GmbH	1	Access [basics] Ausgabe 1/2010
9	E. Müller & Co.	1	Access [basics] Ausgabe 1/2010
1	Fachverlag für Softwareentwicklung	2	Access [basics] Ausgabe 2/2010
5	Haufe und Lexware Mediengruppe	2	Access [basics] Ausgabe 2/2010
6	Addison-Wesley	2	Access [basics] Ausgabe 2/2010
7	Müller Homebanking Services	2	Access [basics] Ausgabe 2/2010
8	Klaus Müller GmbH	2	Access [basics] Ausgabe 2/2010
9	E. Müller & Co.	2	Access [basics] Ausgabe 2/2010
1	Fachverlag für Softwareentwicklung	3	Access [basics] Ausgabe 3/2010
5	Haufe und Lexware Mediengruppe	3	Access [basics] Ausgabe 3/2010
6	Addison-Wesley	3	Access [basics] Ausgabe 3/2010
7	Müller Homebanking Services	3	Access [basics] Ausgabe 3/2010
8	Klaus Müller GmbH	3	Access [basics] Ausgabe 3/2010
9	E. Müller & Co.	3	Access [basics] Ausgabe 3/2010
1	Fachverlag für Softwareentwicklung	4	Access [basics] A-Z
5	Haufe und Lexware Mediengruppe	4	Access [basics] A-Z
6	Addison-Wesley	4	Access [basics] A-Z
7	Müller Homebanking Services	4	Access [basics] A-Z
8	Klaus Müller GmbH	4	Access [basics] A-Z
9	E. Müller & Co.	4	Access [basics] A-Z

Erfahrenen Access-Benutzern war dies klar: Dadurch, dass wir für die beiden Tabellen keine Beziehung definiert haben, an der Access erkennt, welche Datensätze der Tabelle **tblKunden** zu denen der Tabelle **tblProjekte** gehören, kann es nur alle möglichen Kombinationen der Datensätze beider Tabellen ausgeben.

Kombination zweier Tabellen mit 1:n-Beziehung

Erstellen wir also eine neue Abfrage und ziehen wir die beiden Tabellen **tblKunden** und **tblProjekte** der Beispieldatenbank in den Entwurf. Für diese beiden Tabellen existiert eine Beziehung über das Fremdschlüsselfeld **KundeID** der Tabelle **tblProjekte** und dem Primärschlüsselfeld **KundeID** der Tabelle **tblKunden**.

Formulare für die Dateneingabe

Teil II: Formulare für Daten aus 1:n-Beziehungen

Im ersten Teil dieser Artikelreihe haben Sie erfahren, wie Sie einfache Tabellen in der Detailansicht eines Formulars anzeigen. In dieser Folge gehen wir einen Schritt weiter und kümmern uns um Daten, die aus zwei per 1:n-Beziehung verknüpfte Tabellen stammen.

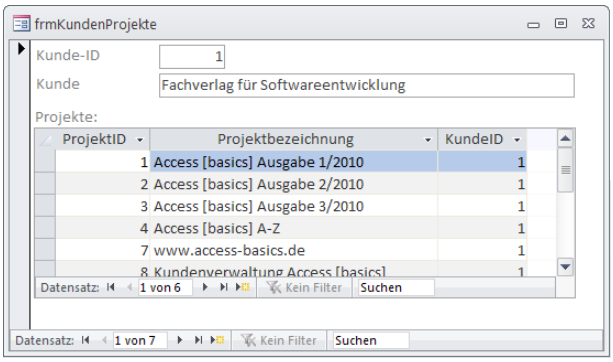
Was Sie in diesem Artikel lernen

- Darstellen der Daten einer 1:n-Beziehung mit Kombinationsfeld
- Darstellen der Daten einer 1:n-Beziehung in Haupt- und Unterformular
- Einrichten von Haupt- und Unterformularen

Genau genommen beziehen wir uns auf die beiden Tabellen **tblKunden** und **tblProjekte** und deren Beziehung, die wir im Artikel **Tabellen entwerfen - Teil II: Kunden und Projekte verwalten** genauer beleuchten.

Für die Darstellung von Daten aus einer solchen Beziehung gibt es mindestens zwei Möglichkeiten, und zwar aus Sicht der jeweiligen Tabelle:

- Vermeintlich schnell erledigt ist die Ansicht aus Sicht der Tabelle **tblProjekte**. Das entsprechende Formular würde die wesentlichen Daten der Tabelle **tblProjekte** anbieten und zusätzlich ein Auswahlfeld, mit dem man den Kunden dieses Projekts aus der Tabelle **tblKunden** auswählen kann.
- Bei der Darstellung aus der Sicht der Tabelle **tblKunden** wird es schon ein wenig komplizierter: Während wir für jedes Projekt nur einen Kunden festlegen, kann ein Kunde eines, keines oder auch mehrere Projekte besitzen. Wie sollen wir diese sinnvoll in einem Formular darstellen? Die Antwort lautet: Gar nicht. Wir verwenden zwei Formulare zur Abbildung dieses Sachverhalts. Dabei zeigt ein sogenanntes Hauptformular die Daten jeweils eines ausgewählten Kunden an, und ein weiteres Formular, das als Unterformular in das Hauptformular eingebettet wird, steuert die zum Kunden im Hauptformular gehörenden Projekte aus der Tabelle **tblProjekte** bei.



Projekte und Kundenauswahl in einem Formular

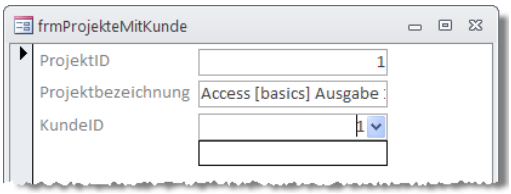
Die einfachere Variante der Darstellung zeigt die Projektdaten plus dem Kunden des Projekts an. Um dies zu realisieren, erstellen Sie einfach wie in →*[basics]* **Gebundenes Formular anlegen** beschrieben ein neues Formular an, das als Datenherkunft die Tabelle **tblProjekte** verwendet und alle Felder dieser Tabelle im Detailbereich anzeigt. Speichern Sie dieses Formular unter dem Namen **frmProjekteMitKunde**.

Prinzipiell sind Sie damit schon fertig – wenn da nicht die ergonomisch unbefriedigende Situation wäre, dass Sie den Kunden über seine **KundeID** festlegen müssen. Das bedeutet eigentlich, dass Sie entweder die Werte des Feldes **KundeID** der Tabelle **tblKunden** auswendig kennen (was nach einiger Arbeit damit sicher möglich ist) oder diese immer in der Tabelle nachschlagen müssen.

Kombinationsfeld zur Kundenauswahl

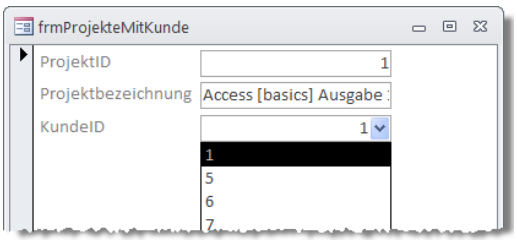
Die Lösung ist, das Feld **KundeID** durch ein Kombinationsfeld zu ersetzen, dass alle Kunden zur Auswahl anbietet. Wir zeigen zuerst, wie Sie dies umsetzen und gehen dann darauf ein, was im Hintergrund geschieht: Klicken Sie mit der rechten Maustaste auf das Textfeld **KundeID** und wählen Sie den Eintrag **Ändern zu|Kombinationsfeld** aus. Damit haben Sie aus dem Textfeld zumindest schon einmal ein Kombinationsfeld gemacht.

Wenn Sie nun in die Formularansicht wechseln, zeigt das Formular zwar ein Kombinationsfeld an. Wenn Sie dieses aber durch einen Klick auf die Schaltfläche am rechten Rand des Kombinationsfelds aufklappen, finden Sie nur gähnende Leere vor.



Ein Kombinationsfeld ohne Daten? Nicht gut ...

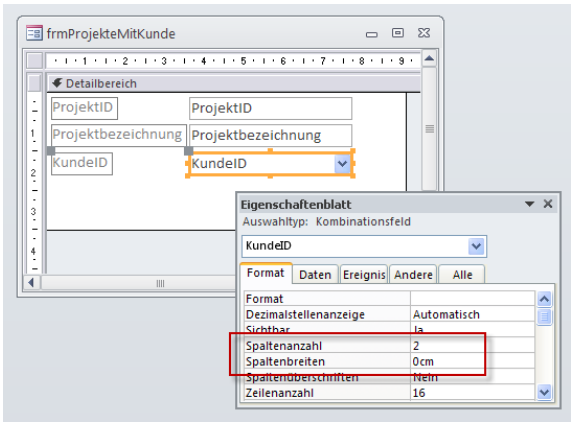
Um dies zu ändern, stellen Sie einige Eigenschaften des Kombinationsfelds ein. Markieren Sie dieses dazu in der Entwurfsansicht und ändern Sie dann im Eigenschaftsfenster den Wert von **Datensatzherkunft** auf **tblKunden**.



Die Auswahl der Zahlen ist schon gut - aber noch längst nicht perfekt ...

Das neue Zwischenergebnis in der Formularansicht sieht besser aus: Immerhin bietet das Kombinationsfeld nun schon einmal alle Werte des Feldes **KundeID** der Tabelle **tblKunden** an.

Noch schöner wäre es, wenn das Kombinationsfeld nicht nur den Inhalt des Feldes **KundeID**, sondern auch noch den Kundennamen anzeigen würde – oder am besten nur den Kundennamen, denn eigentlich interessiert uns das Feld **KundeID** nicht besonders.

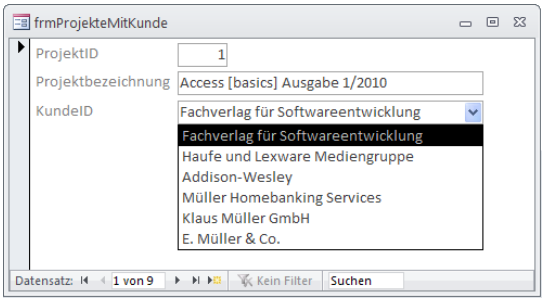


Mal sehen, was diese Änderung bewirkt.

Um dieses nächste Ziel zu erreichen, brauchen Sie nur zwei weitere Eigenschaften zu ändern: Stellen Sie einfach auf der Registerseite Format die Eigenschaft **Spaltenanzahl** auf **2** und **Spaltenbreiten** auf **0cm** ein. Wie meinen? Zwei Spalten, aber nur eine Spaltenbreite angeben – und die auch noch mit der Breite **0cm**?

Der Zusammenhang ist schnell erklärt: Wir haben als Datensatzherkunft des Kombinationsfelds die Tabelle **tblKunden** eingestellt, was bedeutet: Es würde, wenn Sie nur genügend Spalten in der entsprechenden Spaltenbreite festlegen, alle Daten der Tabelle **tblKunden** anzeigen. Das soll es aber gar nicht: Wir möchten nicht das Feld **KundeID** sehen, aber das Feld **Kunde**, und die übrigen Felder sowieso nicht.

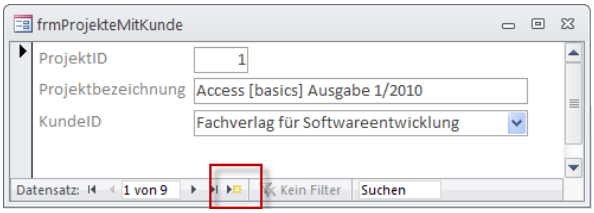
Unser Blick beschränkt sich also maximal auf die ersten beiden Felder der Datensatzherkunft, weshalb wir **Spaltenanzahl** auf **2** einstellen. Was aber hat es mit dem Wert **0cm** für **Spaltenbreite** auf sich? Dies ist schlicht eine Abkürzung für: »Zeige mir die erste Spalte mit der Breite **0cm** an, und die zweite Spalte soll den Rest des Platzes einnehmen.« Was schlicht bedeutet: Zeige einfach den Inhalt des zweiten Feldes der Datenherkunft in der kompletten Breite des Kombinationsfelds an.



So muss es sein: Keine lästigen Kundennummern, sondern nur die Kundennamen stehen zur Auswahl (okay, hier und da wären Kundennummern hilfreich - aber es ist ja nur ein Beispiel ...).

Und warum brauchen wir überhaupt die erste Spalte mit dem Feld **KundeID** und zeigen nicht ausschließlich den Kundennamen an? Auch dies ist schnell erklärt: Das Fremdschlüssfeld **KundeID** der Tabelle **tblProjekte** speichert die Zahl, die dem Primärschlüsselfeld **KundeID** des verknüpften Datensatzes der Tabelle **tblKunden** entspricht. Diese Zahl muss schlicht und einfach in der Datensatzherkunft des Formulars enthalten sein und auch angezeigt werden, und zwar in einer speziellen Spalte, die durch die Eigenschaft **Gebundene Spalte** festgelegt wird – normalerweise ist dies die erste Spalte. Aber der Wert von **KundeID** wird ja im Kombinationsfeld gar nicht angezeigt? Doch, wird er: Allerdings mit der Breite von **0cm**, was ein kleiner, aber sehr weit verbreiteter Trick ist.

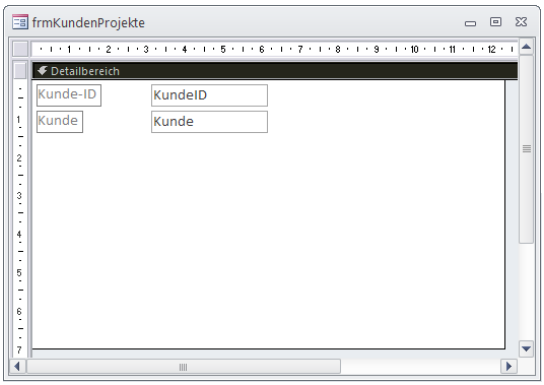
Mit dem Formular frmProjekteMitKunde arbeiten



Der schnellste Weg zum neuen Datensatz geht hier entlang!

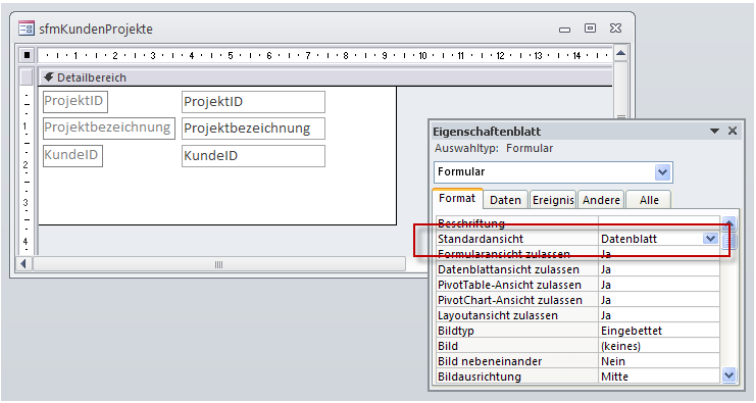
Damit wäre das Formular bereits fertig: Sie können damit bestehende Projekte bearbeiten, indem Sie die Bezeichnung ändern oder einen anderen Kunden auswählen. Außerdem können Sie neue Projekte anlegen und ihren Namen und Kunden festlegen. Dazu klicken Sie einfach auf die Schaltfläche **Neuer (leerer) Datensatz** der Navigationsleiste unten im Formular. Sie können auch bestehende Projekte löschen: Dazu markieren Sie einfach den kompletten Datensatz durch einen Klick auf den Datensatzmarkierer am linken Rand des Formulars und betätigen dann die **Entfernen**-Taste. Keine Sorge: Der Kunde, für den dieses Projekt bestimmt war, wird nicht automatisch mitgelöscht.

Kunden und Projekte in Haupt- und Unterformular



Wenn wir Kunden und Projekte in zwei verschachtelten Formularen anzeigen möchten, brauchen wir zunächst ein einfaches Formular, welches die Daten der Tabelle **tblKunden** in der Detailansicht anzeigt – ähnlich wie in →*[basics]* **Gebundenes Formular anlegen** beschrieben. In diesem Fall soll das Formular die Tabelle **tblKunden** als Datenherkunft verwenden und lediglich die beiden Felder **KundeID** und **Kunde** anzeigen – die Reduzierung der angezeigten Felder dient allerdings nur der Übersichtlichkeit des Beispiels. Dieses Formular, dass wir als Hauptformular verwenden werden, speichern Sie unter dem Namen **frmKundenProjekte**. Es sieht im Entwurf schließlich wie in der Abbildung auf der rechten Seite aus.

Das Hauptformular ist damit schon fast fertig – es fehlt nur noch eine Kleinigkeit, die wir im Anschluss an die Erstellung des Unterformulars besprechen.



In der Datenblattansicht sieht ein Formular genau wie eine Tabelle aus.

Das Unterformular erhält den Namen **sfmKundenProjekte** und verwendet die Tabelle **tblProjekte** als Datenherkunft. Fügen Sie dem Detailbereich der Entwurfsansicht alle drei Felder dieser Tabelle, also **ProjektID**, **Projekt** und **KundeID** hinzu. Die Daten des Unterformulars sollen als Liste angezeigt werden, was wir am einfachsten durch die Verwendung des Wertes **Datenblatt** für die Eigenschaft **Standardansicht** des Formulars erreichen. Die Entwurfsansicht des Formulars sieht nun wie in der Abbildung aus.

Nun können Sie zur Tat schreiten und aus den beiden Formularen ein aus Haupt- und Unterformular bestehendes Gespann erzeugen. Dazu können Sie eine von mehreren möglichen Techniken durchführen, die in →*[basics]* **Unterformular zu Formular hinzufügen** genau erläutert werden.

Wir verwenden die einfachere der dort vorgestellten Möglichkeiten und gehen wie folgt vor:

- Öffnen Sie das angehende Hauptformular **frmKundenProjekte** in der Entwurfsansicht. Das zukünftige Unterformular **sfmKundenProjekte** hingegen schließen Sie komplett.
- Sorgen Sie dafür, dass der Name des Unterformulars im Datenbankfenster oder im Navigationsbereich zu sehen ist und das Hauptformular nicht verdeckt wird.

Access [basics]
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen
Ausgabe 4/2010
14. Juli 2010

Anreden und Co. mit Wertlisten oder Lookup-Tabellen verwalten

Es gibt in Access eine ganze Reihe unterschiedlicher Tabellentypen. Am weitesten verbreitet sind wohl solche wie die Kunden- oder die Projekte-Tabelle, die Sie bereits kennengelernt haben. Die enthaltenen Felder beschreiben Objekte der realen Welt anhand ihrer Eigenschaften. Daneben gibt es Tabellen, die ausschließlich Werte für solche Felder enthalten: Diese heißen Lookup-Tabellen. Eine weitere wichtige Art sind die Verknüpfungstabellen zur Herstellung von m:n-Beziehungen – auf diese gehen wir jedoch erst in einer der folgenden Ausgaben ein.

Was Sie in diesem Artikel lernen

» Nachschlagefelder anwenden
» Nachschlagefelder mit Wertliste
» Nachschlagefelder mit Lookup-Tabelle

Für die Arbeit mit Wertlisten oder Lookup-Tabellen fügen wir zunächst eine neue Tabelle zu unserer Beispieldatenbank hinzu. Neben den Projekten und Kunden wollen wir nun auch noch Mitarbeiterdaten speichern, und zwar in einer Tabelle namens **tblMitarbeiter**. Diese Tabelle enthält wie üblich zunächst einmal ein Primärschlüsselfeld namens **MitarbeiterID**. Außerdem speichern wir zu einem Mitarbeiter Informationen wie die folgenden (den Entwurf dieser Tabelle können Sie in der Beispieldatenbank unter dem Tabellennamen **tblMitarbeiter** einsehen):

Anrede, Vorname, Nachname, Straße, PLZ, Wohnort, Telefon, E-Mail und Geburtsdatum.

Das sind natürlich noch lange nicht alle Daten, die wir in einer richtigen Mitarbeitertabelle aufnehmen würden, aber für Beispielzwecke reichen diese völlig aus.

Die meisten dieser Felder werden sehr individuelle Werte aufnehmen: Es gibt Vornamen und Nachnamen wie Sand am Meer, und auch Straßennamen, Ortsnamen und Postleitzahlen lassen sich nicht gerade an einer Hand abzählen. Gleiches gilt für Telefonnummern oder E-Mail-Adressen.

Eines der genannten Felder wird jedoch voraussichtlich nur ganz wenige Werte aufnehmen. Bevor Sie weiterlesen: Schauen Sie sich die obige Liste der genannten Felder an und überlegen Sie, welches Feld im Gegensatz zu den übrigen Feldern vergleichsweise wenige Werte aufnehmen wird.

In dieser Aufgabe:

Anreden und Co. mit Wertlisten oder Lookup-Tabellen verwalten

Reporting mit Berichten -
Teil I: Einfache Tabelle in der Detailansicht

Aktionsabfragen - Teil I: Anfügeabfragen

Daten importieren - Teil I: Daten aus CSV-Dateien

Datendetails aus der Tabelle!

In Access [basics] verwenden wir eine Projektzeiterfassung als Beispielanwendung zum Erläutern der grundlegenden Techniken von Microsoft Access. Dabei rollen wir das Feld von hinten auf: Wir erlauben erst einmal das Anlegen von Projekten und Kunden, und später arbeiten wir uns über die Mitarbeiter, die Aufgaben und Tätigkeiten zur eigentlichen Zeiterfassung vor.

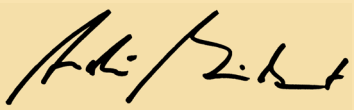
In der letzten Ausgabe haben wir uns um die Kunden und Projekte gekümmert und dabei eine 1:n-Beziehung und verschiedene Möglichkeiten der Darstellung der in den an dieser Beziehung beteiligten Tabellen vorgestellt. In dieser Ausgabe widmen wir uns zunächst einem speziellen Typ der 1:n-Beziehung, nämlich der sogenannten Lookup-Tabelle und ihrer Darstellung in Formularen.

Darüberhinaus wollen wir das erste Mal ein wenig in die Reporting-Techniken von Microsoft Access hineinschnuppern: Wir schauen uns einen ersten einfachen Bericht an, der die Kunden unserer Datenbank auflistet.

Außerdem kümmern wir darum, wie überhaupt die Daten in Ihre Datenbank gelangen. Klar: In vielen Fällen geben die Benutzer diese Daten ein. Kaum eine Anwendung startet jedoch bei Null: Oft gibt es bereits Daten wie beispielsweise eine Kundentabelle. Diese kann als Excel-Datei, als CSV-Datei oder auch als herkömmliche Textdatei daherkommen. Deshalb beleuchten wir einige grundlegende Techniken zum Überführen bestehender Daten in eine Access-Tabelle. Keine Frage: Dabei gibt es die eine oder andere Hürde zu überwinden, vor allem, weil die Daten oft nicht im gewünschten Format vorliegen. Wir werden aber alle gängigen Probleme beim Import von Daten darstellen und eine passende Lösung bieten.

Und schließlich schauen wir uns an, wie Sie mit Abfragen Daten aus einer Tabelle in eine andere kopieren können.

Viel Spaß!



[André Minhorst]

Richtig: Es ist das Feld **Anrede**. Bei der Anrede gibt es meist nur die beiden Varianten **Herr** und **Frau** (zumindest in der Mitarbeitertabelle – in einer Kundentabelle könnte man unter Umständen noch den Wert **Firma** in das Feld aufnehmen).

Wenn ein Feld nur bestimmte Werte enthalten darf, kann man dies auf verschiedene Arten erreichen. Die einfachste ist jedoch, dem Benutzer einfach ein Nachschlagefeld anzubieten, mit dem er einen der zur Verfügung stehenden Werte auswählen kann. Im Artikel **Tabellen entwerfen - Teil II: Kunden und Projekte verwalten** haben Sie bereits ein Beispiel für den Einsatz eines Nachschlagefeldes kennengelernt.

Beispieldatenbank

Die Beispieldatenbank zu diesem Artikel finden Sie im Onlinearchiv unter

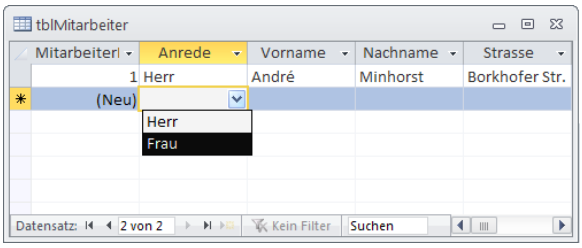
http://www.access-basics.de/index.php/Heftarchiv.html.

Nachschlagefelder bieten gegenüber einem herkömmlichen Textfeld aber noch zwei weitere wichtige Vorteile:

- Der Benutzer muss den Wert des Feldes nicht eintippen, sondern kann diesen aus einer Liste auswählen. Und noch besser: Die Liste zeigt nach dem Eintippen der ersten paar Zeichen gleich den nächsten passenden Eintrag an.

- Daraus, dass der Benutzer die Werte nicht selbst eintippen muss (zumindest nicht vollständig), resultiert der zweite Vorteil: Beim Auswählen eines Wertes aus einer Liste geschehen wesentlich weniger Tippfehler als beim Eintippen des Wertes von Hand.

Powertipp: Wenn Sie das Nachschlagefeld nicht per Maus aufklappen möchten, können Sie das auch mit der Tastatur erledigen - und zwar mit der Taste F4!



Wertlisten und Tabellen als Datenquelle von Nachschlagefeldern

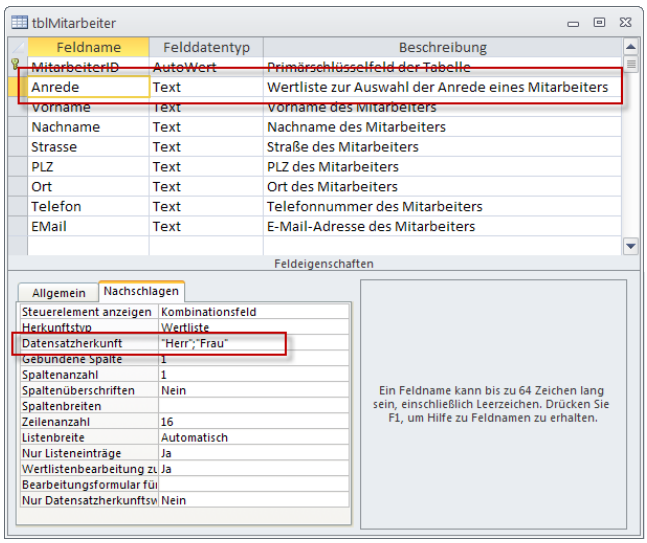
Es gibt zwei interessante Möglichkeiten, ein Nachschlagefeld mit Daten zu füllen: Über eine Tabelle (diese Variante haben wir Ihnen im oben genannten Artikel schon am Beispiel der Kundenauswahl in der Projektetabelle vorgestellt) und über eine Wertliste.

Eine Wertliste ist nichts weiter als eine Liste von Werten, die durch Semikola getrennt werden. Eine Wertliste für ein **Anrede**-Feld könnte so aussehen:

„Herr";"Frau"

Wenn Sie noch mehr Einträge benötigen, fügen Sie diese jeweils durch ein Semikolon vom vorherigen Eintrag getrennt an die Liste an. Wie aber richten wir ein Feld einer Tabelle so ein, dass es die gewünschten Listenelemente in einem Nachschlagefeld anzeigt? Dies erfahren Sie im Detail unter →*[basics]* **Nachschlagefeld für Wertliste einrichten**.

Der Nachteil einer Wertliste als Datenquelle ist, dass Sie diese ausschließlich in der Entwurfsansicht der Tabelle bearbeiten können! Sie müssen in die rechts abgebildete Entwurfsansicht der Tabelle wechseln, um die Wertliste mit den Anreden zu bearbeiten.



Daraus leitet sich eine goldene Regel ab, die Sie sich jetzt merken müssen und niemals vergessen dürfen: Wenn sich die Datenherkunft eines Nachschlagefeldes beziehungsweise eines Kombinationsfeldes irgendwann einmal ändern könnte, verwenden Sie keine Wertliste, sondern eine Tabelle als Datenquelle – selbst wenn die Wahrscheinlichkeit noch so gering ist!

In diesem Fall sollten Sie eine Tabelle anlegen, die nur eine Aufgabe hat: Die möglichen Werte für das Feld bereitzuhalten und als Datenquelle für das entsprechende Nachschlagefeld zu dienen.

Reporting mit Berichten

Teil I: Einfache Tabelle in der Detailansicht

Was Sie in diesem Artikel lernen

» Unterschiede zwischen Berichten und Formularen
» Datensatz im Detail ausdrucken
» Steuerelemente zum Bericht hinzufügen
» Überschrift anzeigen
» Datum und Seitenzahl unterbringen
» Seitenränder verstehen

Berichte bereiten die Daten in den Tabellen einer Datenbank für die Ausgabe etwa auf Papier oder in PDF-Dokumente vor. Dabei unterscheiden Sie sich deutlich von Formularen: Formulare dienen allein der Steuerung der Anwendung und der Eingabe, dem Bearbeiten oder dem Löschen von Daten. Berichte geben die Daten in statischer Form wieder – ohne dem Benutzer auch nur die Chance zu geben, die Daten damit zu bearbeiten.

Mit Erscheinen dieser vierten Aufgabe von **Access [basics]** haben Sie bereits erfahren, wie Sie Daten aus einfachen Tabellen in Formularen anzeigen und bearbeiten können, und auch die Behandlung von Daten aus Tabellen, die per 1:n-Beziehung verknüpft sind, haben Sie schon kennengelernt. Später kommen noch weitere Beziehungstypen hinzu wie etwa die m:n-Beziehung.

Bei den Berichten beginnen wir wieder bei Null – genau genommen bei der Darstellung einfacher Daten aus einer einzigen Tabelle. Für den Start ist die Kundentabelle prima geeignet: Also legen wir in den folgenden Abschnitten einen Bericht an, mit dem Sie die Kunden unserer Beispieldatenbank formschön auf Papier oder in Form eines PDF-Dokuments ausgeben können (letzteres ist mit einem speziellen Add-In erst ab Access 2007 möglich, ab Access 2010 gehört der PDF-Export zu den eingebauten Funktionen).

Sie können auch Formulare ausdrucken, diese sind jedoch nicht für den Ausdruck etwa im DIN-A4-Format optimiert. Tragen Sie also in Ihre Liste der goldenen Access-Regeln ein: Formulare dienen der Datenbearbeitung, Berichte der Datenausgabe.

Berichte und Formulare

Dennoch gibt es eine überraschende Menge an Gemeinsamkeiten zwischen Formularen und Berichten. Beide sind in der Basisversion ähnlich aufgebaut, denn sowohl Formulare als auch Berichte besitzen einen Detailbereich, einen Formularkopf und -fuß (beziehungsweise Berichtskopf und -fuß) sowie einen Seitenkopf und -fuß. Den letzteren benötigen Formulare übrigens nur zum Zwecke des Ausdrucks, und wie soeben gelernt, wollen wir Formulare nicht für das Reporting zweckentfremden.

Sowohl Formulare als auch Berichte enthalten eine Eigenschaft namens **Datenherkunft**, mit der Sie die Tabelle oder Abfrage festlegen können, deren Daten das Formular oder der Bericht anzeigen soll. Außerdem können Sie fast alle Steuerelemente, die Sie in Formularen einsetzen, auch in Berichten verwenden – mit dem Unterschied, dass in Berichten nur wenige Steuerelemente wirklich sinnvoll sind. Dazu gehören beispielsweise das Bezeichnungsfeld und das Textfeld. Oft setzt man in Berichten auch das Linie- oder das Rechteck-Steuerelement ein – zum Beispiel, um die Überschriften einer Tabelle zu unterstreichen oder um Summen zu kennzeichnen.

Kundendetails per Bericht ausdrucken

Nun wollen wir nicht lange fackeln und den ersten Bericht erstellen. Dazu gehen Sie wie in →[basics] **Bericht erstellen** vor und erstellen so einen Bericht mit dem Namen **rptKundenDetails** und der Tabelle **tblKunden** als Datenherkunft.

Beispieldatenbank

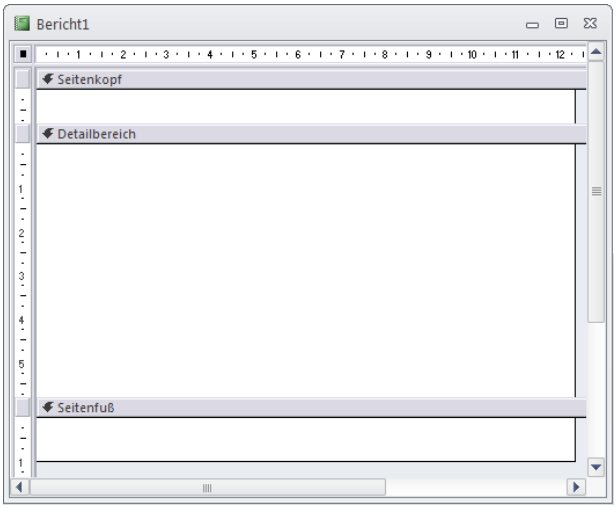
Die Beispieldatenbank zu diesem Artikel finden Sie im Onlinearchiv unter

http://www.access-basics.de/index.php/Heftarchiv.html.

Powertipp: Formulare dienen nur zur Datenbearbeitung und Berichte nur zur Ausgabe von Daten auf Papier, in PDF-Dateien et cetera!

Der resultierende Bericht erscheint in der Entwurfsansicht. Informationen über alle Berichtsansichten finden Sie unter →*[basics]* **Berichtsansichten**.

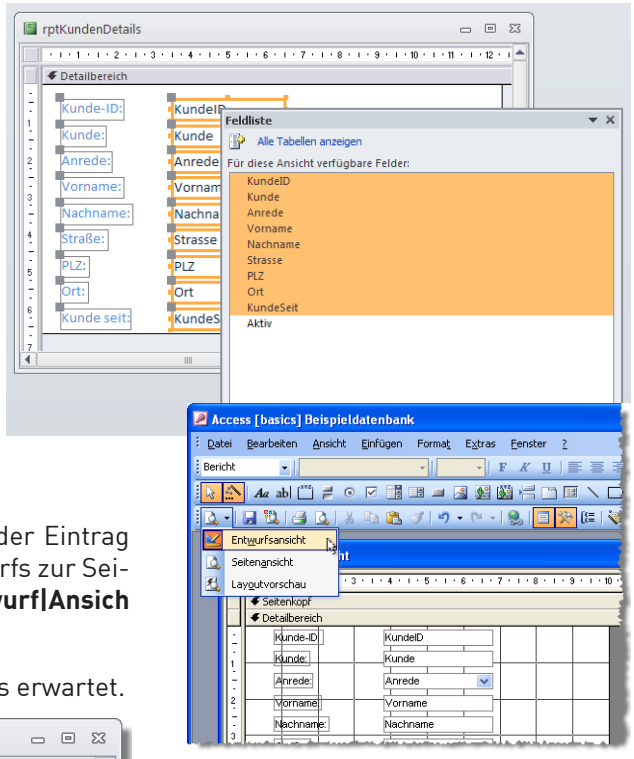
Der Bericht sieht nach dem Anlegen wie in der Abbildung rechts aus. Er enthält noch keine Steuerelemente, aber drei Bereiche: **Seitenkopf**, **Detailbereich** und **Seitenfuß**. Mit diesen wollen wir uns vorerst zufrieden geben – genau genommen mit noch weniger: Blenden Sie einfach mit dem Kontextmenü-Eintrag **Seitenkopf/-fuß** alle Bereiche mit Ausnahme des Detailbereichs aus (Sie können auch einfach die Höhe diese Bereiche mit der Maus minimieren).



Steuerelemente zum Berichtsentwurf hinzufügen

Die Datenherkunft ist bereits auf die Tabelle **tblKunden** eingestellt. Die einfachste Möglichkeit, die Felder dieser Tabelle zum Bericht hinzuzufügen, ist das Ziehen der Felder aus der Feldliste direkt in den Detailbereich des Berichts (→*[basics]* **Gebundene Steuerelemente zu Formularen und Berichten hinzufügen**).

Das Ergebnis sieht wie in der Abbildung rechts aus. In diesem Fall haben wir gleich mehrere Felder der Datenherkunft **tblKunden** markiert und diese auf einen Rutsch in den Detailbereich des Berichtsentwurfs gezogen. Access ordnet diese sorgfältig untereinander an.

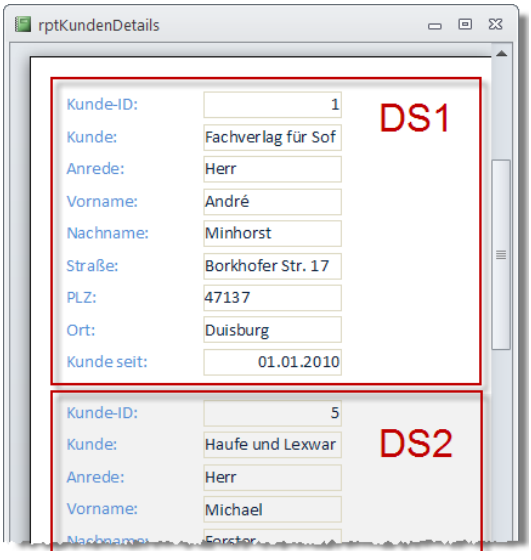


Sie können sich nun gleich einmal das Zwischenergebnis ansehen, indem Sie in die Seitenvorschau des Berichts wechseln. Dazu gibt es verschiedene Möglichkeiten:

2003 und älter: Wählen Sie aus dem Kontextmenü der Titelleiste des Berichtsentwurfs den Eintrag **Seitenansicht** aus oder führen Sie den Menübefehl **Ansicht|Seitenansicht** aus. Alternativ klicken Sie auf die Schaltfläche **Ansicht** oder auf den dahinter verborgenen Popup-Eintrag **Entwurfsansicht** der Symbolleiste aus der Abbildung rechts.

2007 und jünger: Auch bei den neueren Access-Versionen führt der Eintrag Seitenansicht des Kontextmenüs der Titelleiste des Berichtsentwurfs zur Seitenansicht. Alternativ verwenden Sie hier den Ribbon-Eintrag **Entwurf|Ansichten|Ansicht|Entwurfsansicht**.

Schauen wir uns nun an, was uns in der Seitenansicht des Berichts erwartet.



Diese sieht wie in der Abbildung rechts aus. Der Bericht enthält untereinander angeordnet einige Blöcke mit Daten, die genau wie in der Entwurfsansicht aufgebaut sind: links ein Bezeichnungsfeld mit dem Feldnamen, rechts ein Steuerelement mit dem Feldwert.

Die Bezeichnungsfelder enthalten übrigens normalerweise die Feldnamen, wenn Sie die Felder aus der Feldliste in den Entwurf des Berichts

Aktionsabfragen

Teil I: Anfügeabfragen

Was Sie in diesem Artikel lernen

» Verschiedene Arten von Aktionsabfragen
» Anfügeabfragen erstellen
» Anfügeabfragen ausführen
» Meldungen unterbinden
» Anfügeabfragen ohne Quelltabelle einsetzen

Die Grundlagen zum Thema Abfragen haben Sie bereits in einem anderen Artikel kennengelernt. Diese Artikelreihe enthält eine Einführung in die Welt der Aktionsabfragen, die sich grundlegend von den bereits bekannten Auswahlabfragen unterscheiden: Sie dienen nämlich dem Ändern oder Löschen vorhandener Daten oder dem Anlegen neuer Datensätze. Dabei greifen diese Aktionsabfragen auch gern mal auf die Mittel der Auswahlabfragen zurück – schließlich müssen wir ja auch irgendwie festlegen, welche Datensätze geändert werden sollen! Im ersten Teil der Artikelreihe beschäftigen wir uns mit den Anfügeabfragen.

Anfügeabfragen haben die Aufgabe, neue Datensätze in Tabellen anzulegen und diese mit entsprechenden Werten zu füllen. Zunächst jedoch wollen wir grundlegend erklären, welche Arten von Aktionsabfragen es gibt.

- **Aktualisierungsabfragen** ändern bestehende Datensätze in einer Tabelle. Eine Aktualisierungsabfrage enthält als Parameter den Namen der Tabelle, deren Daten sie ändern soll, die Namen der zu ändernden Felder und die neuen Werte dieser Felder sowie gegebenenfalls einen Kriteriumsausdruck, der festlegt, welche Datensätze der Tabelle überhaupt geändert werden sollen. Eine solche Abfrage könnte beispielsweise alle Preise einer Artikeltabelle um 30% reduzieren, deren Bestand kleiner als 10 ist.

- **Anfügeabfragen** fügen neue Datensätze zu bestehenden Tabellen hinzu und füllen Felder der neuen Datensätze mit den entsprechenden Werten. Eine Anfügeabfrage erwartet als Parameter den Namen der Zieltabelle sowie Informationen über den beziehungsweise die einzufügenden Datensätze. Eine Anfügeabfrage kann einen einzelnen Datensatz anfügen, wobei Sie die einzutragenden Werte einzeln übergeben, oder Sie fügt einen oder mehrere Datensätze auf Basis der Datensätze einer anderen Tabelle oder Abfrage an (was bedeutet, dass die Daten auch aus mehreren Tabellen stammen können). Dieser Fall kommt beispielsweise im Artikel **Daten importieren** vor, wo wir die Adressen einer verknüpften Tabelle in eine Kundentabelle übertragen.

- **Löschabfragen** entfernen Datensätze aus einer Tabelle. Die Löschabfrage muss wissen, aus welcher Tabelle die Datensätze gelöscht werden sollen. Außerdem können Sie mit einem Kriterium festlegen, welche Datensätze gelöscht werden sollen – also zum Beispiel alle Bestellungen, deren Bestelldatum älter als zehn Jahre ist.

- **Tabellenerstellungsabfragen** arbeiten im Prinzip wie Anfügeabfragen – mit dem kleinen Unterschied, dass sie die Tabelle, der sie die Daten hinzufügen sollen, zunächst erstellt. Die Feldnamen und Felddatentypen werden üblicherweise von der Quelltabelle übernommen, wobei Sie die Feldnamen noch anpassen können. Eine Tabellenerstellungsabfrage könnten Sie beispielsweise einmal im Monat aufrufen, um veraltete Datensätze einer Tabelle mit Logdaten in eine Archivtabelle zu schreiben und damit den Zugriff und die Suche in den aktuellen Daten beschleunigen.

Beispieldatenbank

Die Beispieldatenbank zu diesem Artikel finden Sie im Onlinearchiv unter

http://www.access-basics.de/index.php/Heftarchiv.html.

Aktualisierungsabfragen aktualisieren bestehende Daten, Löschabfragen löschen Datensätze, Anfügeabfragen fügen Datensätze zu bestehenden und Tabellenerstellungsabfragen Datensätze zu neuen Tabellen hinzu.

Anfügeabfragen erstellen

In diesem ersten Teil der Artikelreihe beginnen wir mit den Anfügeabfragen. Grundsätzlich gibt es zwei Möglichkeiten, eine solche Aktionsabfrage zu definieren:

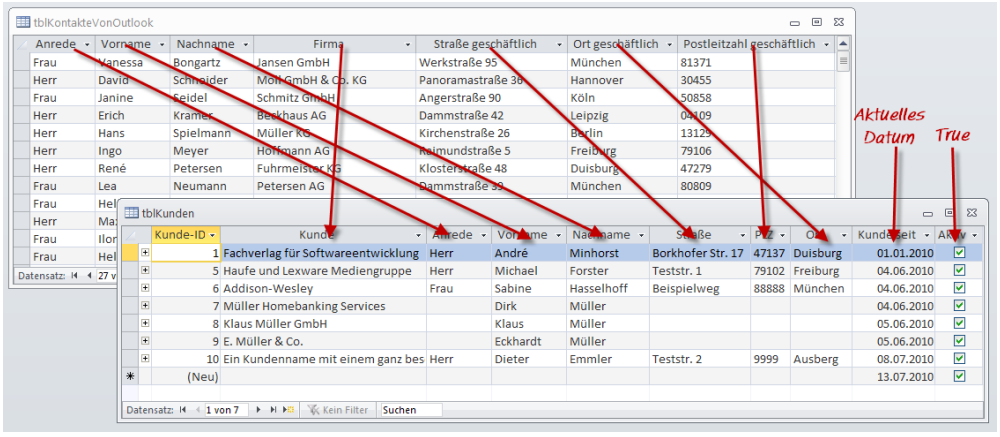
- Die Abfrage liest die anzufügenden Daten aus einer oder mehreren anderen Tabellen oder Abfragen ein und schreibt diese in die Zieltabelle.
- Die Abfrage schreibt nur einen Datensatz in die Zieltabelle, deren Werte Sie manuell vorgeben – also ohne Verwendung einer Quelltabelle.

In beiden Fällen gehen Sie, abhängig von der Access-Version, grundsätzlich wie folgt vor:

- Erstellen Sie eine neue Abfrage wie in →*[basics]* **Abfrage erstellen** beschrieben.
- Fügen Sie die Quelltabelle(n) in die Entwurfsansicht und ziehen Sie die Felder, mit denen die Zieltabelle gefüllt werden soll, zum Entwurfsraster hinzu.
- Wandeln Sie die Tabelle in eine Anfügeabfrage um und wählen Sie die Zieltabelle aus.
- Legen Sie fest, in welchen Feldern der Zieltabelle die jeweiligen Inhalte der Quelltabelle(n) landen sollen.
- Testen Sie die Anfügeabfrage.
- Wenn das Ergebnis korrekt ist, führen Sie die Abfrage aus.

Anfügeabfrage am Beispiel von Adressdaten

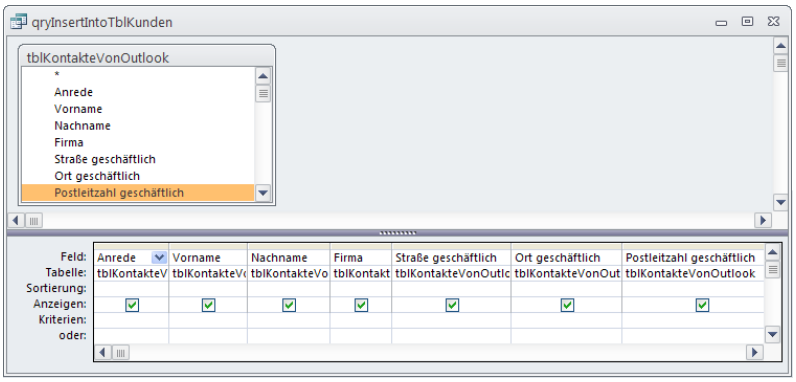
Im Artikel **Daten importieren - Teil I: Daten aus CSV-Dateien** erfahren Sie, wie Sie Kontakte aus Outlook exportieren und diese in eine Access-Tabelle importieren. Für unser Beispiel wollen wir diese Daten aus der Tabelle **tblKontakteVonOutlook**, die Sie in der Beispieldatenbank finden, als Datenquelle für das Befüllen der Tabelle **tblKunden** unserer Beispiellösung verwenden.



Dabei sollen die Inhalte der Felder der Tabelle **tblKontakteVonOutlook** so wie in der Abbildung in der Tabelle **tblKunden** landen. Die beiden letzten Felder wollen mir mit selbst ausgewählten Werten füllen – dies sparen wir uns allerdings für den zweiten Schritt auf. Erst einmal schauen wir uns an, wie wir die Daten von A nach B schaufeln können.

Als erstes erstellen Sie, wie oben schon erwähnt, auf Basis der Anleitung aus →*[basics]* **Abfrage erstellen** eine Auswahlabfrage mit der Tabelle **tblKontakteVonOutlook** als Datenquelle.

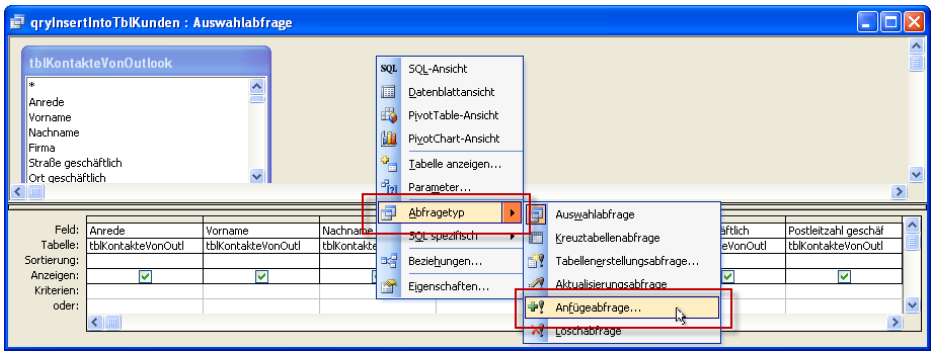
Diese Abfrage sieht nun in der Entwurfsansicht wie in der Abbildung rechts aus. Sie zeigt alle Felder der Tabelle **tblKontakteVonOutlook** an, deren Inhalte Sie später in die entsprechenden Felder der Tabelle **tblKunden** importieren werden. Speichern Sie diese Abfrage unter dem Namen **qryInsertIntoTblKunden**. Das Präfix kennen Sie bereits aus **Konventionen**, der Rest setzt sich aus der Art der Aktionsabfrage und dem Namen der Zieltabelle zusammen.



Nun folgt der entscheidende Schritt: Die Umwandlung der Auswahlabfrage in eine Anfügeabfrage.

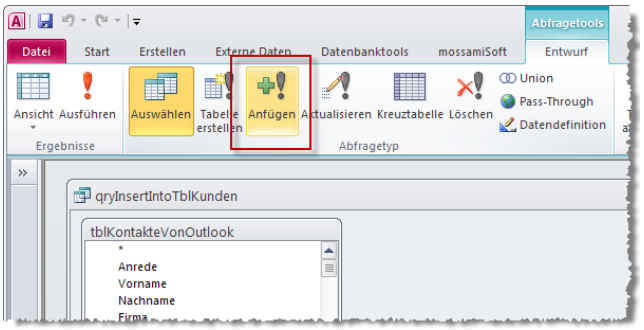
2003 und älter

Hier haben Sie mehrere Möglichkeiten, die Abfrage in eine Anfügeabfrage umzuwandeln. Die einfachste ist wohl die über das Kontextmenü: Klicken Sie mit der rechten Maustaste auf die Titelleiste des Abfragefensters und wählen Sie den Eintrag **Abfragetyp|Anfügeabfrage...** aus.



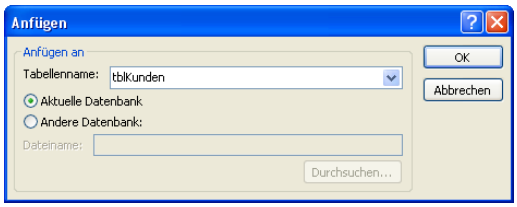
Alternativ verwenden Sie den Menüeintrag **Abfrage|Anfügeabfrage...** oder den entsprechenden Eintrag des Symbolleiste.

2007 und jünger

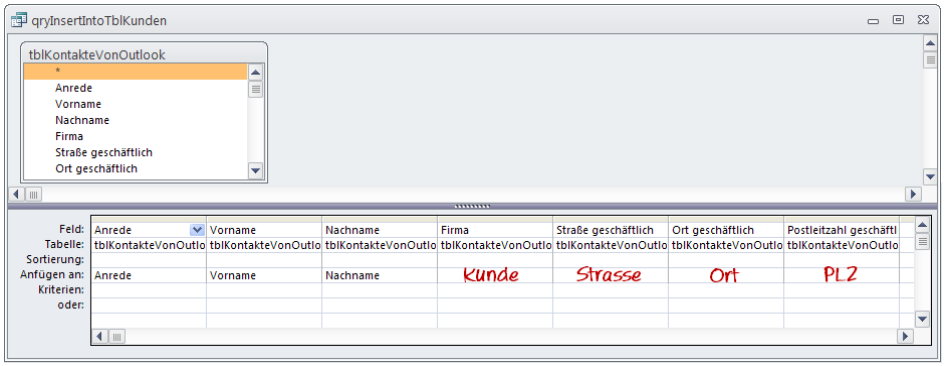


Bei den neuen Access-Versionen können Sie ebenfalls den Kontextmenüeintrag **Abfragetyp|Anfügeabfrage...** verwenden. Außerdem steht der Ribbon-Eintrag **Entwurf|Abfragetyp|Anfügen** dafür zur Verfügung.

In allen Access-Versionen erscheint der Dialog **Anfügen**, der einige weitere Informationen erwartet. Die erste uns wichtigste Information ist die Angabe der Zieltabelle der Abfrage. Diese können Sie bequem aus dem Kombinationsfeld auswählen. Mit der zweiten Option legen Sie fest, in welcher Tabelle sich die Zieltabelle befindet. Sie können mit einer Anfügeabfrage nämlich auch Daten in Tabellen anderer Datenbanken schreiben. Dies ist besonders praktisch, wenn Sie eine Anfügeabfrage zum Archivieren aktuell nicht mehr benötigter Daten verwenden: Sie können solche Daten per Doppelklick in die Archivtabelle in einer anderen Datenbankdatei speichern und sparen damit eine Menge Platz in der aktuellen Datenbank.



Im Moment wollen wir die Daten jedoch in eine Tabelle der aktuellen Datenbank schreiben, und zwar **tblKunden**. Nach dem Bestätigen dieser Auswahl erscheint die Abfrage in leicht veränderter Form: Sie weist im Entwurfsraster eine neue Zeile mit der Zeilenüberschrift **Anfügen an:** auf.



Daten importieren

Teil I: Daten aus CSV-Dateien

Was Sie in diesem Artikel lernen

» Was sind CSV-Dateien?
» Daten in eine neue Tabelle importieren
» Daten zu einer bestehenden Tabelle hinzufügen
» Verknüpfung zu einer CSV-Datei erstellen
» Verknüpfte Datei zu einer Tabelle hinzufügen

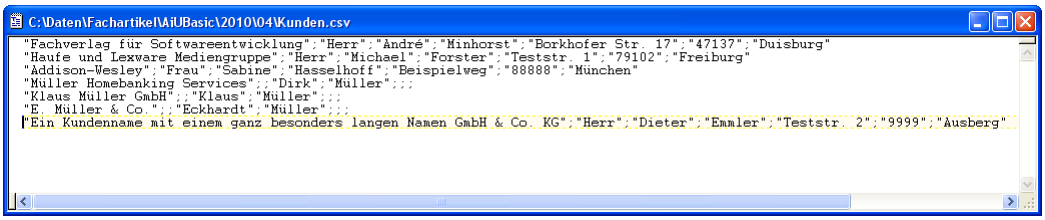
Beispieldatenbank

Die Beispieldatenbank zu diesem Artikel finden Sie im Onlinearchiv unter

http://www.access-basics.de/index.php/Heftarchiv.html.

Nur ein Bruchteil aller Access-Anwendungen erledigt eine komplett neue Aufgabe und startet mit leeren Tabellen. In den meisten Fällen gibt es entweder eine Anwendung auf Basis einer anderen Software oder die zu bearbeitenden Daten wurden bisher in Excel-Tabellen, CSV-Dateien oder Textdateien gespeichert. Für diesen Fall gibt es gute Nachrichten: Strukturierte Daten lassen sich meist leicht in Access-Tabellen einlesen. Und auch für die schwereren Fälle liefern wir Ihnen Lösungsansätze.

CSV-Dateien sind normalerweise recht unverfänglich: CSV heißt Comma Separated Values, und das bedeutet: Die Datei enthält eine oder mehrere Zeilen mit Daten, die eine bestimmte Struktur haben und somit durch ein spezielles Zeichen – meist ein Semikolon oder ein Tabulator-Zeichen – in eine Art Spalten aufgeteilt sind.

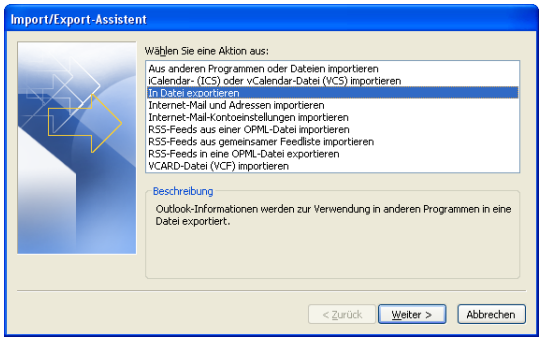


Die Abbildung zeigt ein Beispiel für eine recht einfache und – wie Sie gleich sehen werden – einfach zu importierende CSV-Datei. Die Datei weist eine konsistente Struktur auf: Sie speichert die Daten eines Kunden jeweils in einer Zeile und die einzelnen Informationen sind durch Semikola voneinander getrennt. Hinzu kommt, dass die Daten auch unvollständiger Datensätze nicht einfach hintereinander geschrieben sind. Stattdessen finden Sie dort zwei oder mehr Semikola hintereinander, die als Trennzeichen für die einzelnen Spalten dienen und so dafür sorgen, dass sich etwa der Nachname immer in der richtigen Spalte befindet.

Warum CSV?

Der Grund, warum die erste Datei, deren Import wir beschreiben wollen, eine CSV-Datei ist, ist die weite Verbreitung dieses Formats. Annähernd jede Anwendung bietet die Möglichkeit, ihre Daten im CSV-Format zu speichern. Dies gelingt mit Access, Excel oder Outlook genauso wie mit den Anwendungen anderer Anbieter als Microsoft. Mittlerweile etabliert sich zwar auch XML als Datenaustauschformat und es bietet viel mehr Möglichkeiten – allerdings ist die Handhabung auch ein wenig komplizierter.

Wenn Sie eine CSV-Datei zum Experimentieren erstellen möchten, exportieren Sie doch einfach die Liste Ihrer Kontakte aus Outlook. Dazu wählen Sie zum Beispiel unter Outlook 2007 zunächst den Menü-Eintrag **Datei|Importieren/Exportieren...** aus.



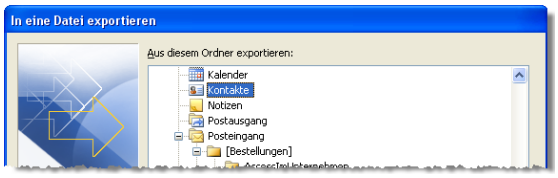
Wählen Sie im Dialog **Import/Export-Assistent** den Eintrag **In Datei exportieren** aus und im nächsten Dialog den Eintrag **Kommagetrennte Werte** oder **Tabulatorgetrennte Werte** aus (außerdem sollten Sie die Windows-Variante verwenden – auf diese Weise verhindern Sie später Probleme mit Sonderzeichen wie beispielsweie Umlauten). Der Export mit tabulatorgetrennten Werten ist vorzuziehen, wenn Sie nicht aus-

schließen können, dass irgendeiner der zu exportierenden Werte seinerseits ein Komma-Zeichen enthält. In diesem Fall würde dies praktisch eine zusätzliche Zeile in der Exportdatei erzeugen und beim anschließenden Einlesen in Access möglicherweise zu Fehlern führen.

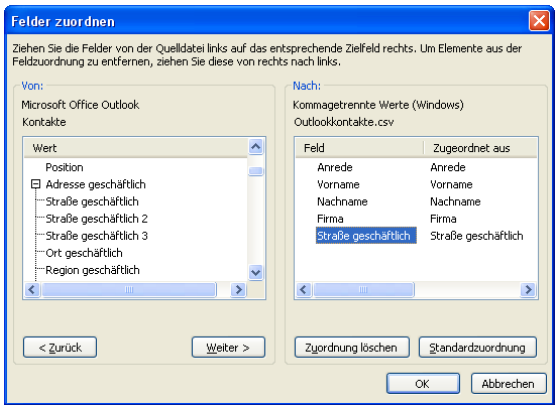
Im deutschsprachigen Raum (und auch anderswo) wird außerdem das Komma als Trennzeichen für Dezimalzahlen verwendet. Die weiter unten vorgestellten Assistenten für das Importieren und Verknüpfen externer Daten meckern übrigens von selbst, wenn das Trennzeichen und das Dezimalzeichen identisch sind. Sorgen Sie also nach Möglichkeit dafür, dass Datenlieferanten entweder das Tabulatorzeichen oder ein anderes unverfängliches Zeichen wie das Semikolon als Trennzeichen verwenden.

Achtung: Wenn die zu importierende Datei das Komma als Trennzeichen verwendet, könnte es Probleme geben, weil dieses Zeichen in vielen Ländern auch als Dezimaltrennzeichen verwendet wird!

Im nächsten Dialog legen Sie dann fest, welche Daten Sie überhaupt exportieren möchten. Wir wollen beispielhaft unsere Kontakte in die Beispieldatenbank importieren, also wählen wir in dem entsprechenden Dialog den Eintrag **Kontakte** aus.



Anschließend geben Sie noch den Dateinamen an und legen fest, welche Felder überhaupt im Export landen sollen. Da Outlook standardmäßig alle vorhandenen Felder exportiert, sollten Sie hier eingreifen und nur die benötigten Felder festlegen. Dazu klicken Sie einmal auf die Schaltfläche **Zuordnung löschen**, was sämtliche Einträge aus der rechten Liste entfernt (wenn Sie nur einige Einträge entfernen möchten, können Sie diese einfach per anklicken und bei gedrückter linker Maustaste aus der Liste herausziehen und irgendwo außerhalb loslassen). Anschließend können Sie die benötigten Felder auf die gleiche Weise per Drag and Drop von der linken in die rechte Liste ziehen. Wenn Sie alle Felder zusammengestellt haben, schließen Sie den Dialog mit der **OK**-Schaltfläche.



Mit einem Klick auf die Schaltfläche **Fertigstellen** sorgen Sie dafür, dass Outlook die gewünschten Kontakt-Informationen in eine CSV-Datei schreibt.

Achtung:

Kontakte importieren

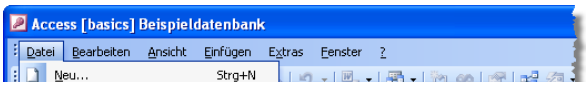
Schauen wir uns nun die verschiedenen Möglichkeiten an, Daten aus einer CSV-Datei zu importieren. Im Wesentlichen gibt es diese Varianten:

- Direkter Import in eine neue Tabelle
- Direkter Import in eine bestehende Tabelle
- Verknüpfung mit der Quelldatei und anschließender Import per Aktionsabfrage
- Verknüpfung mit der Quelldatei und anschließender Import per VBA/Abfrage (für kompliziertere Fälle)

Import in eine neue Tabelle

Die einfachste Variante ist die erste. Hier wählen Sie zunächst die Quelldatei aus uns stellen einige weitere Parameter ein. Access erstellt dann eine neue Tabelle und füllt diese automatisch mit den Daten der Quelldatei. Die Vorgehensweise unter Access 2003 und älter und Access 2007 und jünger unterscheidet sich geringfügig.

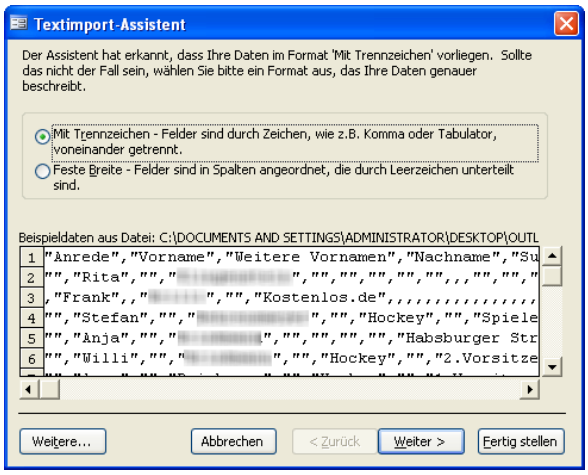
2003 und älter:



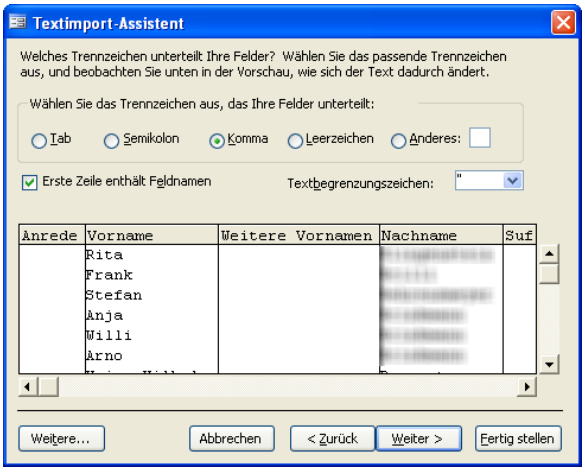
- Führen Sie den Menüeintrag **Datei|Externe Daten|Importieren** aus.

- Wählen Sie dann im **Importieren**-Dialog die gewünschte Datei aus.

Der folgende Dialog bildet den ersten Schritt des **Textimport-Assistenten** ab. Sie können hier bereits den Inhalt der Quelldatei erkennen. Der Assistent fragt in diesem ersten Schritt ab, ob die Felder der einzelnen Zeilen durch ein spezielles Zeichen wie ein Komma oder ein Tabulatorzeichen voneinander getrennt werden oder ob diese gleich entsprechend untereinander angeordnet sind.

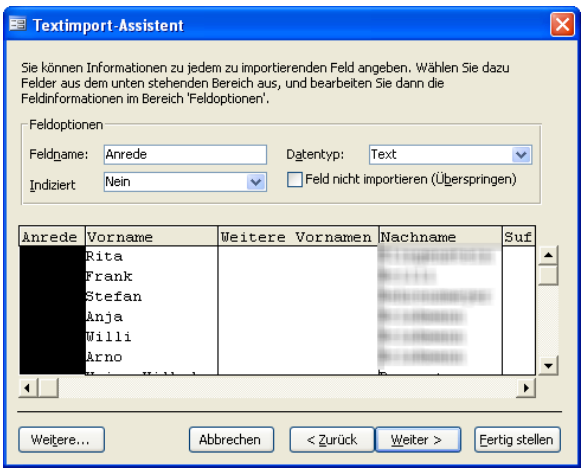


- Im nächsten Schritt legen Sie fest, welches Trennzeichen die Datei genau verwendet, um den Text einer Zeile auf die verschiedenen Felder aufzuteilen. Access findet dieses Zeichen dabei meist schon vorher heraus und stellt dieses entsprechend ein. Eine weitere wichtige Einstellung dieses Dialogs betrifft die Spaltenüberschriften. Wenn die Datei eine solche enthält, aktivieren Sie die passende Option. Dies bringt zwei Vorteile: Erstens wird diese erste Zeile nicht mit importiert, und zweitens können Sie die Spaltenüberschriften als Voreinstellung für die Feldnamen der zu erstellen Datenbank verwenden.

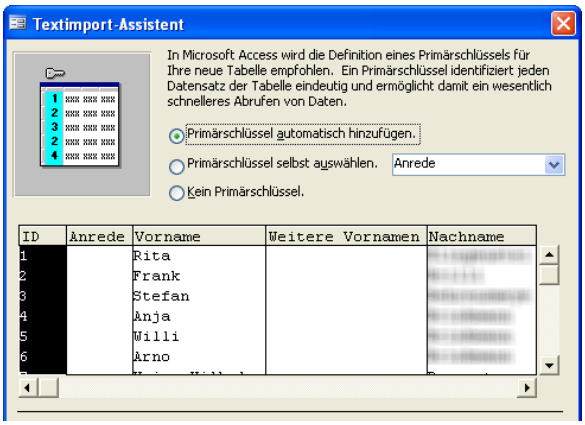


- Der folgende Dialog fragt eine sehr wichtige Information ab: Möchten Sie die Daten in eine neue oder in eine existierende Tabelle importieren? In diesem Fall gehen wir den einfachen Weg und erstellen eine neue Tabelle. Weiter unten gehen wir darauf ein, wie Sie Daten in eine vorhandene Tabelle importieren.

- Danach legen Sie die Eigenschaften der Felder der Zieltabelle für jede Spalte der Quelldatei fest. Dabei klicken Sie jeweils die zu bearbeitende Spalte an und stellen die vier Eigenschaften ein. Den **Feldnamen** sollten Sie auf jeden Fall so anpassen, dass er den gängigen Konventionen entspricht (s. →*[basics]* **Konventionen**). Mit der Eigenschaft **Indiziert** legen Sie fest, ob das Feld einen eindeutigen, einen einfachen oder gar keinen Index erhalten soll (mehr zur Bedeutung eines Indexes erfahren Sie unter →*[basics]* **Index für ein Tabellenfeld festlegen**). Auch den Datentyp des Feldes können Sie in diesem Schritt einstellen. Hier ist Vorsicht angezeigt: Möglicherweise enthält etwa eine Spalte der Quelldatei Datumsangaben, aber vielleicht nicht in einem gültigen Datumsformat. Das kommt beispielsweise vor, wenn Bearbeiter von Daten in Excel-Dateien zusätzliche Informationen in ein Feld eintragen (etwa in einem Feld namens Geburtstag: **1.2.1971 (noch prüfen)**). Diesen Inhalt kann Access natürlich nicht problemlos in ein Feld des Datentyps **Datum/Uhrzeit** importieren. Stattdessen meldet Access einen nicht näher spezifierten Fehler beim Import. In einem solchen Fall kommen Sie nicht um eine manuelle Suche nach dem Verursacher des Fehlers herum. Techniken zum Lösen solcher Probleme liefern wir später nach. Mit der letzten Eigenschaft können Sie schließlich noch festlegen, ob ein Feld überhaupt importiert werden soll.



- Anschließend geht um die Bestimmung eines Primärschlüsselfeldes. Hier gibt es drei Möglichkeiten: Erstens können Sie einfach einen neuen Primärschlüssel hinzufügen. Im vorliegenden Fall ist das sinnvoll, weil die von Outlook exportierten Daten kein Feld mit eindeutigem Wert mitliefern. Wäre dies der Fall, könnten Sie dieses Feld als Primärschlüsselfeld festlegen. Vielleicht

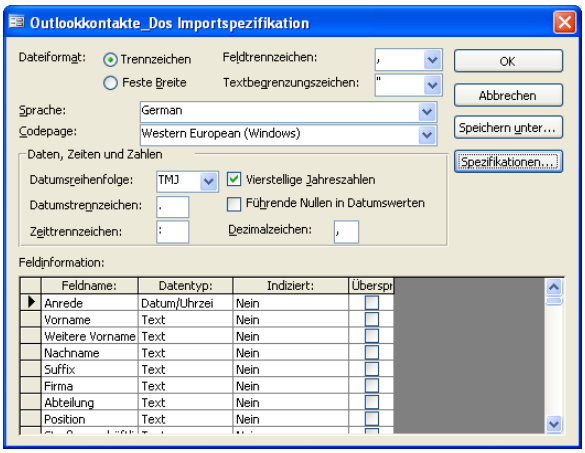


möchten Sie aber auch gar kein Primärschlüsselfeld verwenden, weil die Daten beispielsweise noch nachbearbeitet und dann erst in die eigentliche Zieltabelle geschrieben werden sollen.

- Schließlich geben Sie im letzten Schritt des Assistenten noch den Namen der zu erstellenden Tabelle an.

Fertig! Durch einen Klick auf die Schaltfläche **Fertigstellen** starten Sie den Import der Daten.

Die Tabelle ist nun unter dem angegebenen Namen im Datenbankfenster zu finden und sieht im Entwurf genau wie geplant aus – und auch die Datenblattansicht liefert die soeben aus Outlook exportierten Daten. Wenn dies die endgültige Zieltabelle der importierten Daten ist, können Sie sich nun an das Finetuning begeben: Zum Beispiel möchten Sie vielleicht aus dem Textfeld Anrede ein Fremdschlüsselfeld namens **AnredeID** machen, das auf eine Tabelle **tblAnrede** verweist. Wie dies funktioniert, lesen Sie unter **Anreden und Co. mit Wertlisten oder Lookup-Tabellen verwalten**.



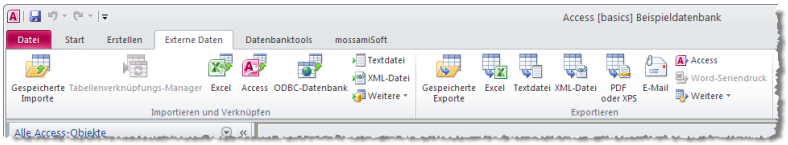
Möglicherweise möchten Sie diese Daten später nochmals genau mit den gleichen Parametern durchführen. In diesem Fall können Sie die Einstellungen als Spezifikation speichern. Klicken Sie dazu einfach nach der Einstellung aller Parameter auf die Schaltfläche **Weitere...** und dann auf **Speichern unter...** – nach Eingabe einer Bezeichnung speichert Access die für eine Wiederholung des Imports nötigen Einstellungen.

Bereits gespeicherte Importspezifikationen finden Sie, wenn Sie auf die Schaltfläche **Spezifikationen...** klicken. Sie können diese dann über einen weiteren Dialog laden und gegebenenfalls anpassen oder auch vorhandene Spezifikationen löschen.

In diesem Dialog können Sie auch die meisten der bereits getätigten Eingaben nochmals (und teilweise auch einfacher) anpassen.

2007 und jünger

Mit Access 2007 haben sich einige wenige Schritte etwas geändert – vor allem der Aufruf des Import-Assistenten. Generell sind die Import- und Export-Funktionen etwas prominenter platziert, nämlich in einem eigenen Ribbon-Tab namens **Externe Daten**. Dort finden Sie in zwei Untergruppen wiederum den Bereich **Importieren und Verknüpfen** sowie **Exportieren**. Im Falle unserer CSV-Datei verwenden Sie die Schaltfläche **Textdatei**, um den Import zu starten.



Hier wählen Sie die Datei aus, welche die zu importierenden Daten enthält und entscheiden, ob Sie die Daten in eine neue Tabelle oder in eine bestehende Tabelle importieren sollen oder ob Sie nur eine Verknüpfung herstellen möchten.

Die folgenden Schritte erfolgen prinzipiell mit dem Assistenten wie in den älteren Access-Versionen – lediglich das Design des Assistenten hat sich ein wenig geändert.

Access [basics]
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen
Ausgabe 5/2010
17. August 2010

Tabellen entwerfen

Teil III: Mitarbeiter per m:n-Beziehung zu Projektteams zusammenstellen

Die Zuordnung von Kunden und Projekten ist klar geregelt: Sie führen jedes Projekt für genau einen Kunden durch. Andersherum kann jeder Kunde Auftraggeber für kein, ein oder mehrere Projekte sein. Beim Zusammenstellen von Projektteams sieht es anders aus: Jeder Mitarbeiter kann kein, ein oder mehrere Projekte bearbeiten. Andersherum benötigt jedes Projekt einen oder mehrere Bearbeiter. Das ist der Stoff, aus dem m:n-Beziehungen sind!

Datenbank-Neulinge, die sich gerade mit 1:n-Beziehungen beschäftigt haben, stehen ehrfurchtsvoll vor der nächsten Hürde: der m:n-Beziehung. Dabei ist jede Scheu unnötig: Eine m:n-Beziehung ist nämlich nichts anderes als die Kombination zweier 1:n-Beziehungen (oder, in diesem Fall, einer 1:n und einer 1:m-Beziehung). Eine Bestandsaufnahme vor dem Erstellen der m:n-Beziehung zwischen Mitarbeitern und Projekten liefert Folgendes zu Tage:

- die Tabelle **tblMitarbeiter** mit Feldern wie **MitarbeiterID**, **Vorname** und **Nachname** und
- die Tabelle **tblProjekte** mit Feldern wie **ProjektID** und **Projektbezeichnung**.

Es gibt hier keine Möglichkeit, einem Mitarbeiter ein Projekt zuzuweisen – also beispielsweise ein Nachschlagefeld, mit dem wir dem Mitarbeiter ein Projekt zuweisen können. Andersherum gilt das Gleiche: Die Projekte-Tabelle enthält lediglich ein Feld zur Auswahl des Kunden, aber die Mitarbeiter eines Projekts lassen sich nicht festlegen.

Das wäre auch recht unpraktisch: Wenn Sie der Tabelle **tblProjekte** ein Fremdschlüsselfeld namens **MitarbeiterID** hinzufügen würden, mit dem Sie einen Eintrag der Tabelle **tblMitarbeiter** auswählen könn-

Was Sie in diesem Artikel lernen

» Wozu braucht man überhaupt m:n-Beziehungen?
» Wie ist eine m:n-Beziehung aufgebaut?
» Wie stelle ich sicher, dass jede Kombination von Datensätzen nur einmal auftaucht?
» Welche Anwendungsfälle für m:n-Beziehungen gibt es noch?

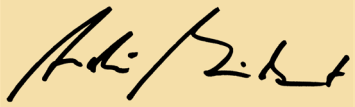
Finetuning

Nach den ersten vier Ausgaben von **Access [basics]**, die großes Interesse auslösten und viele Anregungen brachten, ist es Zeit für ein wenig Optimierung. Sie werden feststellen, dass diese Ausgabe nicht mehr einspaltig mit Screenshots und Kommentaren am rechten Rand, sondern zweispaltig daherkommt - und die Screenshots sind nun mit Bildnummern ausgestattet. Der Grund ist ganz einfach: Manchmal bietet eine Seite einfach keine Gelegenheit, am rechten Rand etwas zur Auflockerung unterzubringen. Eine komplett leere rechte Seite sieht aber erstens nicht schön aus. Zweitens könnte dies den Eindruck erwecken, Sie würden nicht genug Inhalt für Ihr Geld erhalten. Und bevor ich irgendwelche Screenshots oder Kommentare unterbringe, nur um den Platz zu füllen, schreibe ich diesen lieber mit nützlichen Inhalten voll!

Außerdem kommt ab Ausgabe 5/2010 jede Ausgabe als eigenes PDF-Dokument. Kopieren Sie einfach alle Ausgaben in ein Verzeichnis, dann klappt's auch mit dem Querverweisen.

Und nun arbeiten wir weiter an unserer Projektzeiterfassung! In der aktuellen Ausgabe erfahren Sie, wie Sie Projekte und Mitarbeiter einander über eine m:n-Beziehung zuteilen und so Projektteams zusammenstellen.

Viel Spaß beim Lesen!



[André Minhorst]

In dieser Ausgabe:

Tabellen entwerfen - Teil III: Mitarbeiter per m:n-Beziehung zu Projektteams zusammenstellen
Erfahren Sie, wozu Sie m:n-Beziehungen zwischen Tabellen benötigen und wie Sie diese realisieren.

Formulare für die Dateneingabe - Teil III: m:n-Beziehung per Unterformular
Erstellen Sie Haupt- und Unterformular zur Eingabe der Daten einer m:n-Beziehung.

Fremdschlüsselfelder mit Nachschlage-Funktion: Pro und Kontra Wo helfen Fremdschlüsselfelder und wo schaden Sie?

Daten importieren – Teil II: Vom Textfeld zur Lookuptabelle
Daten aus einer Tabelle gleich auf mehrere aufteilen

ISSN: 2190-8761

ten, hätten Sie Folgendes erreicht: Sie könnten für jedes Projekt nur einen Projektmitarbeiter festlegen. Auf diese Weise könnten Sie zwar einen Mitarbeiter verschiedenen Projekten zuweisen, aber nicht umgekehrt. Andersherum könnten Sie auch der Tabelle **tblMitarbeiter** ein Fremdschlüsselfeld namens **ProjektID** zuweisen, mit dem Sie wiederum den Mitarbeitern Projekte zuweisen. Wenn Sie dann zwei oder mehr Mitarbeitern das gleiche Projekt zuweisen, hätten Sie ein Projektteam zusammengestellt. Aber Sie können natürlich jedem Mitarbeiter nur ein Projekt zuweisen.

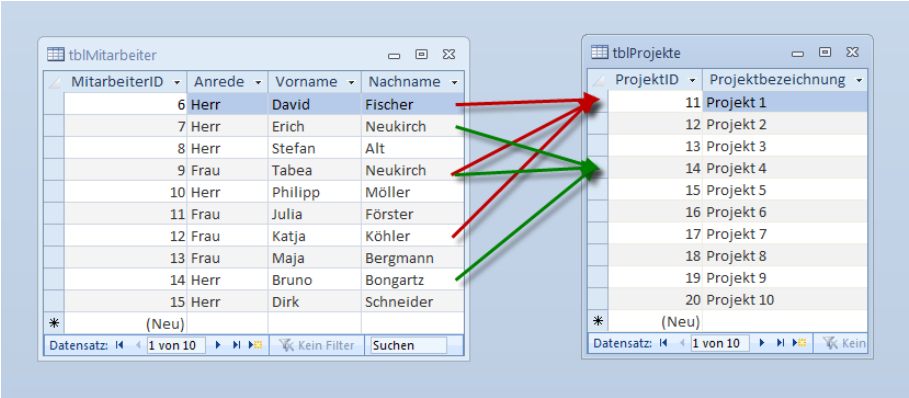


Bild 1: Zuordnung von Projekten zu Mitarbeitern und umgekehrt

Mit der Kombination beider Techniken hätten wir unser Ziel doch erreicht, oder? Mitnichten: Bereits wenn wir Mitarbeiter **A** und **B** den Projekten **C** und **D** zuweisen möchten, wobei jeder Mitarbeiter an beiden Projekten beteiligt sein soll, scheitert diese Vorgehensweise: Sie könnten zwar in der Mitarbeitertabelle für beide Mitarbeiter **A** und **B** das Projekt **C** auswählen, aber es bleiben ja zwei Kombinationen offen: Mitarbeiter **A**/Projekt **D** und Mitarbeiter **B**/Projekt **D**.

Dies müssten wir im Fremdschlüsselfeld der Tabelle **tblProjekte** einstellen, was aber nicht funktioniert: Dort könnten wir einem Projekt nur maximal einen Mitarbeiter zuweisen.

Jeder Mitarbeiter mit jedem Projekt

Bild 1 verdeutlicht, dass es noch kompliziertere Konstellationen geben kann. Wie nun bekommen wir die Menge an Kombinationen in den Griff? Es gibt nur eine einzige Möglichkeit: Wir erstellen eine weitere Tabelle, mit der wir alle gewünschten Kombinationen aus Mitarbeitern und Projekten darstellen können. Der Clou ist, dass wir darin für jede Kombination einen eigenen Datensatz anlegen. Ein solcher Datensatz muss also das eindeutige Merkmal eines Projekts, also den Wert des Feldes **ProjektID** der Tabelle **tblProjekte** und den Wert des entsprechenden Feldes der Tabelle **tblMitarbeiter**, hier **MitarbeiterID**, aufnehmen.

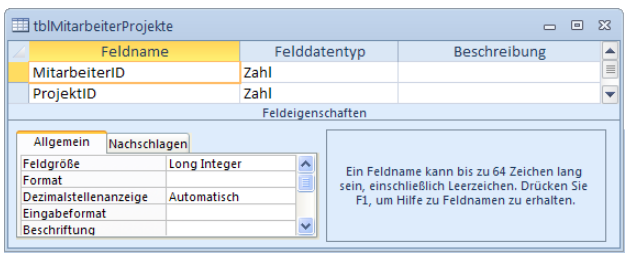


Bild 2: Zuordnung von Projekten zu Mitarbeitern und umgekehrt

Erstellen Sie also eine neue Tabelle namens **tblMitarbeiterProjekte** und fügen Sie dieser Tabelle zwei Felder namens **MitarbeiterID** und **ProjektID** hinzu – beide sollen den Datentyp **Zahl** aufweisen (→*[basics]* **Tabelle erstellen**, →*[basics]* **Zahlenfeld anlegen**). Im Entwurf sieht die Tabelle nun wie in Bild 2 aus.

Zum Verknüpfen von Datensätzen der Tabellen **tblMitarbeiter** und **tblProjekte** über diese Hilfstabelle ist nun ein wenig Handarbeit nötig. Um etwa die in Bild 1 dargestellten Zuordnungen nachzubilden, müssen Sie einige Datensätze wie in Bild 3 anlegen. Diesen Aufwand sollen Sie natürlich nicht für jede Kombination aus Mitarbeiter und Projekt betreiben. Daher greifen wir erneut in den Entwurf dieser Tabelle ein und legen zwei Nachschlagefelder an (→*[basics]* **Nachschlagefeld für eine 1:n-Beziehung einrichten**).

Dabei soll das Feld **MitarbeiterID** als Fremdschlüsselfeld zur Auswahl eines Datensatzes der Tabelle **tblMitarbeiter** und das Feld **ProjektID** zur Auswahl der Datensätze der Tabelle **tblProjekte** dienen. In der Datenblattansicht sieht die Tabelle **tblMitarbeiterProjekte** nun wie in Bild 4 aus.

Die Mitarbeiter und die Projekte lassen sich in der neuen Version bequem per Nachschlagefeld auswählen (Informationen zum Pro und Kontra von Nachschlagefeldern erhalten Sie übrigens im Beitrag **Fremdschlüsselfelder mit Nachschlage-Funktion: Pro und Kontra**).

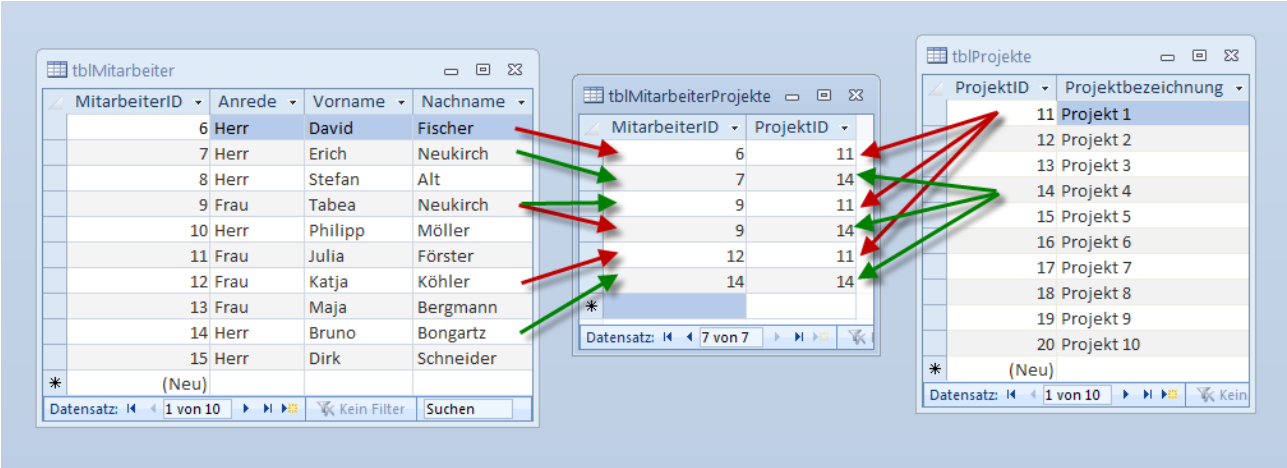


Bild 3: Zuordnung von Projekten zu Mitarbeitern und umgekehrt

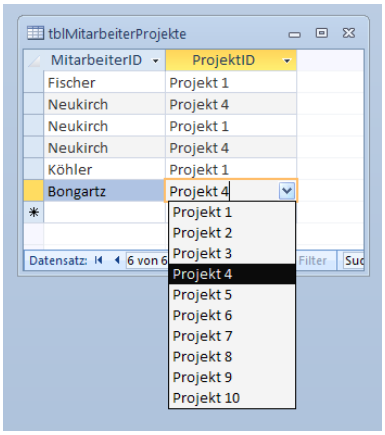


Bild 4: Nach der Einrichtung von Nachschlagefeldern fällt das Zusammenstellen von Projektteams bereits viel leichter.

Eindeutigkeit der Verknüpfungen

Nun stehen noch zwei wichtige Überlegungen an:

- Brauchen wir noch weitere Felder in dieser Verknüpfungstabelle?
- Wie stellen wir sicher, dass jede Kombination aus Mitarbeiter und Projekt nur einmal ausgewählt werden kann?

Die beiden Fragen stehen in engem Zusammenhang. Normalerweise sollte jede Tabelle ein Primärschlüsselfeld aufweisen, das Sie jedoch technisch auch aus zwei oder mehr Feldern zusammensetzen können.

Dies würde bedeuten, dass Sie die beiden einzigen Felder der Tabelle **tblMitarbeiterProjekte** markieren und auf die Schaltfläche zum Anlegen eines Primärschlüssels klicken.

Access markiert dann alle beteiligten Felder mit dem Primärschlüsselsymbol, was wie in Bild 5 aussieht.

Der zusammengesetzte Primärschlüssel wirkt sich so aus, dass Sie auf jeden Fall für beide Felder eines jeden Datensatzes einen Wert auswählen müssen und dass jede Kombination nur einmal vorkommen darf. Wenn Sie also beispielsweise den Mitarbeiter **Michael Müller** zweimal mit **Projekt 1** kombinieren, meldet Access einen Fehler. Grundsätzlich ist dieses Verhalten wünschenswert. Allerdings hat die Verwendung eines aus mehr als einem Feld zusammengesetzten Primärschlüssels einen entscheidenden Nachteil: Sie können nicht mit einem einfachen Fremdschlüsselfeld aus einer anderen Tabelle heraus auf die Datensätze dieser Tabelle verweisen.

Das ist oftmals gar nicht nötig und deshalb auch nicht entscheidend – wieso sollten Sie auch eine Kombination aus Mitarbeiter und Projekt von einer anderen Tabelle aus referenzieren?

Gegebenenfalls erlaubt der Anwendungsfall aber auch das mehrfache Auftreten der gleichen Kombination, und dann ist ein weiterer eindeutiger Schlüssel unumgänglich – in diesem Falls wohl ein Primärschlüsselfeld, so wie Sie es in →[basics] **Primärschlüsselfeld festlegen** anlegen. Grundsätzlich kann ein wenig Konsistenz im Aufbau der Tabellen auch nie schaden: Ein eigenständiger Primärschlüssel

Bild 5: Ein aus zwei Feldern zusammengesetzter Primärschlüssel

Formulare für die Dateneingabe

Teil III: m:n-Beziehung per Unterformular

Für die Darstellung von m:n-Beziehungen gibt es verschiedene Möglichkeiten. Die einfachste ist wohl die Darstellung mithilfe eines Unterformulars. Dabei zeigt das Hauptformular einen Datensatz der m-Seite der Beziehung im Hauptformular an und das Unterformular die damit verknüpften Datensätze der n-Seite. Ein Wechsel des Datensatzes im Hauptformular soll die Daten im Unterformular automatisch aktualisieren. Wie dies funktioniert und warum dies die einfachste Variante ist, erfahren Sie in diesem Artikel.

Als Beispiel für diesen Artikel dient die m:n-Beziehung zur Zusammenstellung von Projektteams. Dabei verknüpft die Tabelle **tblMitarbeiter-Projekte** die Datensätze der Tabellen **tblMitarbeiter** und **tblProjekte** und fügt zusätzlich die Funktion eines jeden Mitarbeiters im entsprechenden Projekts hinzu. Details über diese m:n-Beziehung lesen Sie im Artikel **Tabellen entwerfen - Teil III: Mitarbeiter per m:n-Beziehung zu Projektteams zusammenstellen** nach. Das Beispielformular soll später wie in Bild 1 aussehen und die einfache Auswahl von Projektmitgliedern aus der Liste der Mitarbeiter ermöglichen.

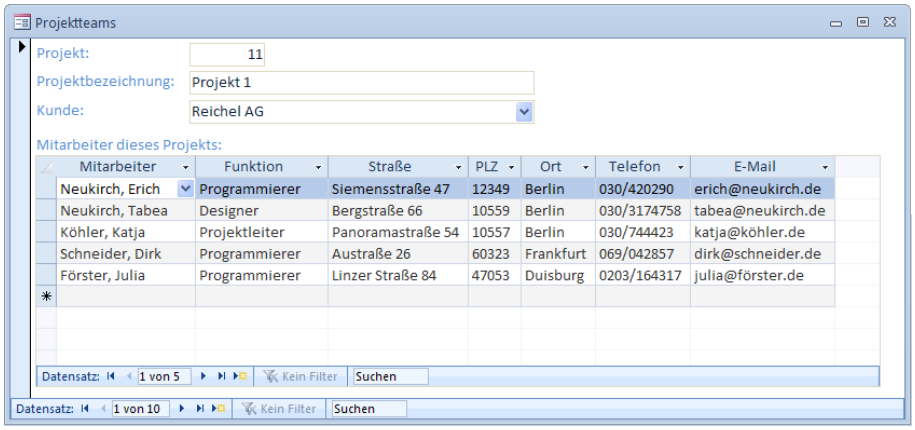


Bild 1: So soll das Formular zur Verwaltung von Daten aus zwei per m:n-Beziehung verknüpften Tabellen später aussehen.

Haupt- und Unterformular

Grundlegende Informationen über die Aufteilung von Daten in Haupt- und Unterformular liefert der Artikel **Formulare für die Dateneingabe - Teil II: Formulare für Daten aus 1:n-Beziehungen**. Der grundsätzliche Aufbau für die Darstellung von m:n-Beziehungen sieht genauso aus wie bei 1:n-Beziehungen: Das Hauptformular zeigt die Daten der einen Seite der Beziehung an, also beispielsweise die Projekte. Das Unterformular steuert die Daten der anderen Seite bei, in diesem Fall die dem Projekt zugewiesenen Mitarbeiter.

Der Unterschied zwischen diesen beiden Konstellationen ist, dass bei der 1:n-Beziehung eine Datenherkunft im Unterformular verwendet wird, die direkt mit der Datenherkunft im Hauptformular verknüpft ist. Im Beispiel des oben genannten Artikels zeigt das Hauptformular die Datensätze der Tabelle **tblKunden** an. Das Unterformular enthält die Daten der Tabelle **tblProjekte**. Diese Tabelle stellt über das Feld **KundeID** die Beziehung zu den Daten im Hauptformular her.

Bei einer m:n-Beziehung haben wir nun nicht mehr zwei Tabellen, von denen die eine die andere per Fremdschlüsselfeld referenziert, sondern wir müssen uns gleich mit drei Tabellen auseinandersetzen. Keine der beiden Tabellen mit den relevanten Daten besitzt ein Fremdschlüsselfeld, mit dem es auf die Daten der anderen Tabelle verweisen kann. Stattdessen stellt eine Verknüpfungstabelle die Beziehung zwischen den Tabellen her.

Dennoch müssen wir, wenn wir die Daten der m:n-Beziehung ähnlich wie die einer 1:n-Beziehung darstellen wollen, eine passende Konstellation schaffen. Dazu müssen wir dem Hauptformular eine Datenherkunft zuweisen, die ihre Daten unabhängig von Unterformular anzeigen kann. Das Unterformular wiederum enthält Daten, die über ein Fremdschlüsselfeld mit den Daten des Hauptformulars verknüpft sind.

Der erste Teil der Lösung ist ganz einfach: Das Hauptformular zeigt die Daten der Tabelle **tblProjek-**

Was Sie in diesem Artikel lernen

» Darstellung von Daten aus m:n-Beziehungen in Haupt- und Unterformular
» Schnelles Hinzufügen verknüpfter Daten per Kombinationsfeld
» Bearbeitung der Daten der m:n-Verknüpfungstabelle

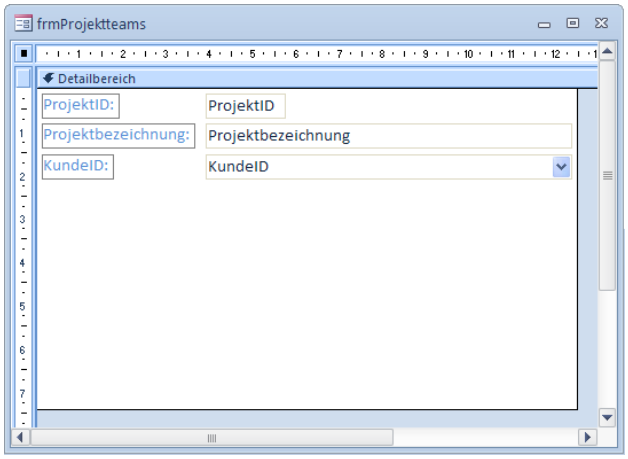


Bild 2: Das Hauptformular mit den Feldern der Tabelle **tblProjekte**

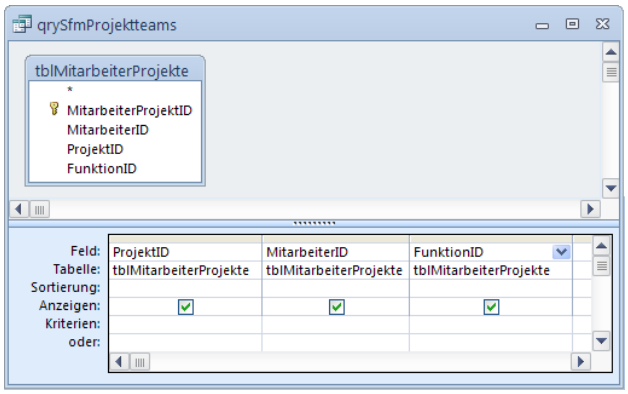


Bild 3: Dies ist die Datenherkunft der einfachsten Variante des Unterformulars auf Basis der Tabelle **tblMitarbeiterProjekte**.

te an. Bleiben zwei Tabellen: Die Tabelle **tblMitarbeiter** sowie die Verknüpfungstabelle **tblMitarbeiterProjekte**.

Aus diesen beiden Tabellen müssen wir irgendwie eine Datenherkunft bauen, die ein Fremdschlüsselfeld zur Tabelle **tblProjekte** aus dem Hauptformular enthält. Wie Sie gleich sehen werden, ist dies ganz einfach: Wir brauchen die beiden Tabellen **tblMitarbeiter** und **tblMitarbeiterProjekte** einfach nur per Abfrage zusammenzuführen, die benötigten Felder zum Abfrageergebnis hinzuzufügen und diese Abfrage als Datenherkunft des Unterformulars einzustellen. Damit erhalten wir prinzipiell genau den gleichen Aufbau wie bei der Darstellung der 1:n-Beziehung in Haupt- und Unterformular, wenngleich der Teufel im Detail steckt und einige Anpassungen für ein ergonomisches Handling nötig sind.

Hauptformular zur Anzeige der Projekte

Das Hauptformular erstellen wir nach der Beschreibung in →*[basics]* **Gebundenes Formular anlegen**. Stellen Sie als Datenherkunft die Tabelle **tblProjekte** ein und legen Sie die Felder etwa wie in Bild 2 an. Speichern Sie das Formular unter dem Namen **frmProjektteams**. Stellen Sie die Eigenschaft **Beschriftung** auf **Projektteams verwalten** ein, damit der Benutzer gleich weiß, welche Aufgaben er mit diesem Formular durchführen kann.

Damit sind die Arbeiten an diesem Formular selbst bereits weitgehend erledigt – Sie brauchen nur noch das Unterformular zu erstellen, einzufügen und anzupassen.

Unterformular mit Mitarbeitern füllen

Legen Sie anschließend ein neues Formular an, das später als Unterformular dienen wird und speichern Sie es unter dem Namen **sfmProjektteams**. Das Präfix **sfm** weist darauf hin, dass dieses Formular als Unterformular eines anderen Formulars eingesetzt wird.

Das Formular soll die Projektmitglieder übersichtlich in der Datenblattansicht anzeigen, daher stellen Sie seine Eigenschaft **Standardansicht** entsprechend ein.

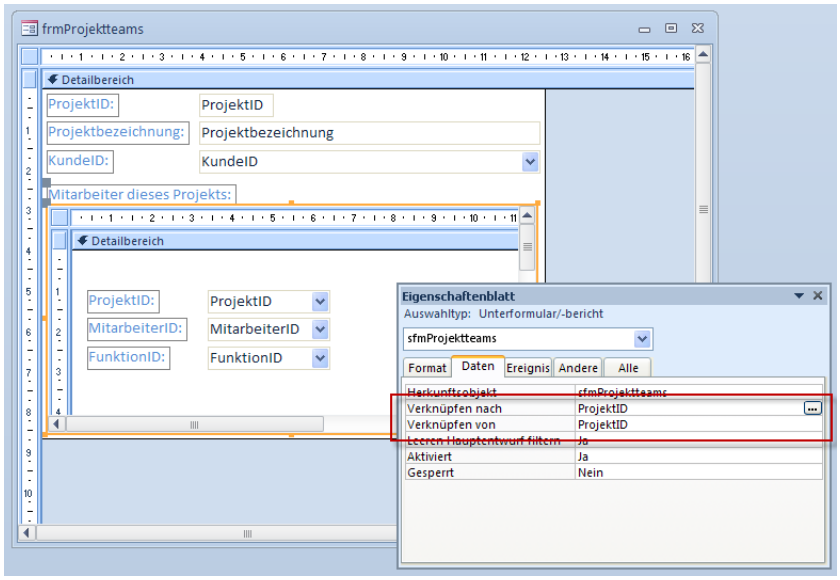


Bild 4: Damit das Unterformular die zum Datensatz im Hauptformular passenden Daten anzeigt, müssen Sie diese Eigenschaften entsprechend einstellen.

Fremdschlüsselfelder mit Nachschlage-Funktion: Pro und Kontra

Wenn Sie schnell für ein Fremdschlüsselfeld Daten aus verknüpften Tabellen auswählen möchten, legen Sie das Fremdschlüsselfeld als Nachschlagefeld aus. In der Entwicklergemeinde scheiden sich die Geister, ob dies nun sinnvoll ist oder eher negative Folgen hat. Access [basics] zeigt, wie Sie Nachschlagefelder einsetzen und dabei mögliche Probleme umgehen.

Wo auch immer eine 1:n-Beziehung auftaucht, sind Nachschlagefelder nicht weit: Es ist aber auch allzu bequem, diese nützlichen Dinger gleich in Entwurf einer Tabelle anzulegen und davon in Abfragen und Formularen zu profitieren. Wie das genau funktioniert, zeigen wir in →*[basics]* **Nachschlagefeld für eine 1:n-Beziehung einrichten**.

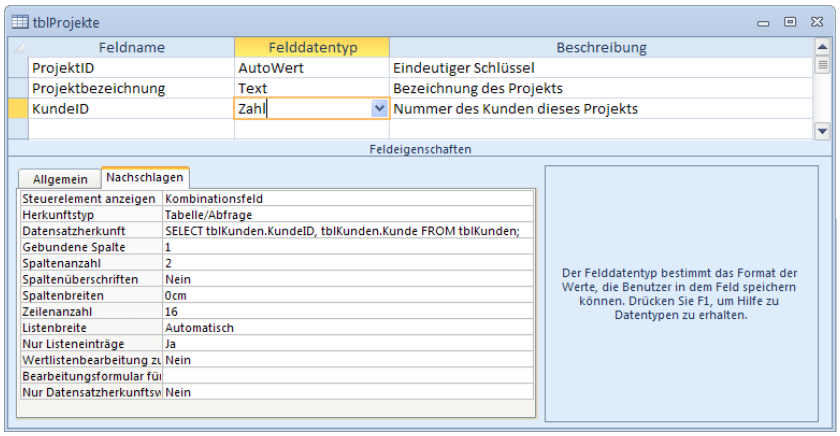


Bild 1: Das Feld **KundeID** ist als Nachschlagefeld ausgelegt, was die Eigenschaften der Registerseite **Nachschlagen** der Feldeigenschaften verdeutlichen.

Prinzipiell hat so etwas wie ein Nachschlagefeld in einer Tabelle nichts zu suchen, weil eine Tabelle schlicht aus Felddefinitionen bestehen und Datensätze speichern soll – und keine Features liefern, die den Zugriff auf die enthaltenen Daten vereinfachen.

Der Hintergrund ist auch ganz einfach: Der Benutzer soll niemals direkt in den Tabellen arbeiten, sondern die enthaltenen Daten ausschließlich über Formulare verwalten. Formulare bieten dazu viel mehr Möglichkeiten – zum Beispiel die Auswahl verknüpfter Daten über Kombinationsfelder.

Womit wir wieder beim Nachschlagefeld angekommen wären: Dabei handelt es sich nämlich um nichts anderes als um ein Kombinationsfeld – es heißt nur anders. Wenn der Benutzer die Daten aber ohnehin nicht direkt in den Tabellen bearbeiten soll, wozu brauchen Sie dann ein Kombinationsfeld gleich in der Tabelle?

Der Grund ist ganz einfach: Weil Sie vielleicht genauso faul sind wie der Autor dieser Zeilen. Der Nachschlagefeld-Assistent spart nämlich nicht nur verdammt viel Zeit beim Anlegen einer Beziehung, weil er gleich noch einen Index für das Fremdschlüsselfeld erstellt. Wie viele andere Eigenschaften von Tabellenfeldern werden die Nachschlagefeld-Eigenschaften nämlich in den Entwurf von Formularen und Berichten übernommen – und natürlich zeigen auch Abfragen ein Nachschlagefeld an, wenn das zugrunde liegende Feld als solches definiert ist.

Vorneweg gleich die Information, dass Kombinationsfelder in Berichten natürlich überhaupt nichts verloren haben, da Berichte nun einmal keine Möglichkeit für den Zugriff auf die angezeigten Daten bieten und dynamische Steuerelemente wie Kombinationsfelder dort nichts verloren haben.

Vom Nachschlagefeld zum Kombinationsfeld

Wenn ein Formular die Daten einer Tabelle anzeigen soll, die ein Nachschlagefeld besitzt, wird das entsprechende Feld gleich nach dem Hinzufügen zum Entwurf des Formulars als Kombinationsfeld ausgelegt. Schauen wir uns das Beispiel der Tabelle **tblProjekte** an – es besitzt ein Nachschlagefeld namens **KundeID** zum Auswählen der Datensätze der Tabelle **tblKunden** (siehe Bild 1).

Wenn Sie nun ein Formular auf Basis dieser Tabelle erstellen, wie es beispielsweise im Formular **frmProjektteams** der Fall ist, und die Felder aus der Feldliste in den Formularentwurf ziehen, wird ein als Nachschlagefeld ausgelegtes Tabellenfeld direkt als Kombinationsfeld hinzugefügt. Bild 2 zeigt dies am Beispiel des Feldes **KundeID**. Dort können Sie direkt einige der im Tabellenentwurf festgelegten Nachschlagefeld-Eigenschaften im Eigenschaftsfenster des Kombinationsfelds **KundeID** erkennen.

Der Clou ist, dass die Nachschlagefeld-Eigenschaften in jedes neue Kombinationsfeld auf Basis des Feldes übernommen werden. Wenn Sie das Feld nicht als Nachschlagefeld definiert hätten, müssen Sie jedesmal erstens das Textfeld in ein Kombinationsfeld umwandeln und zweitens dessen datenbezogenen Eigenschaften einstellen.

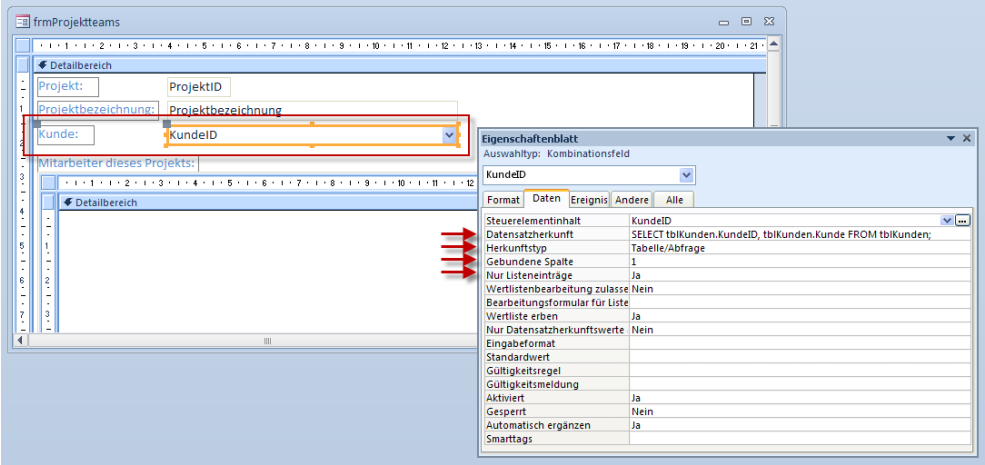


Bild 2: Einige der vom Nachschlagefeld übernommenen Eigenschaften finden Sie auf der Registerseite **Daten** des Kombinationsfeldes **KundeID** wieder.

Probleme durch Nachschlagefelder und ihre Lösung

Natürlich gibt es tatsächlich auch Nachteile durch die Verwendung von Nachschlagefeldern. Diese schauen wir uns in diesem Abschnitt an und stellen Lösungen für die Probleme vor. Wir betrachten die folgenden Situationen, wobei wir die Tabellen **tblProjekte** und **tblKunden** und das Nachschlagefeld **KundeID** der Tabelle **tblProjekte** als Beispiel verwenden:

- Reihenfolge-Problem in Abfragen: Sie möchten die Projekte nach dem Namen des Kunden sortieren, was aber schiefgeht, weil die Namen der Kunden ja nur in der Tabelle **tblProjekte** angezeigt werden, aber nicht tatsächlich darin enthalten sind.

- Sichtbarkeit der „echten" Daten in der Tabelle: Sie möchten als Entwickler zu Testzwecken Einblick in die tatsächlichen Werte eines Nachschlagefeldes erhalten, was jedoch das Umwandeln des Nachschlagefeldes in ein Textfeld und umgekehrt erfordern würde.

- Nachschlagefelder in Berichten: Sie möchten in einer Liste der Projekte die Kundennamen wie im Nachschlagefeld veröffentlichen, was aber nicht ohne die Anzeige des Kombinationsfeldes zu funktionieren scheint.

- Index und Beziehung: Das Anlegen eines Nachschlagefeldes mit einer weiteren Tabelle als Datensatzherkunft legt automatisch einen Index für das Fremdschlüsselfeld und eine Beziehung zwischen Fremdschlüsselfeld und Primärschlüsselfeld der Datensatz herkunft an. Manche befürchten dadurch Performance-Verschlechterungen.

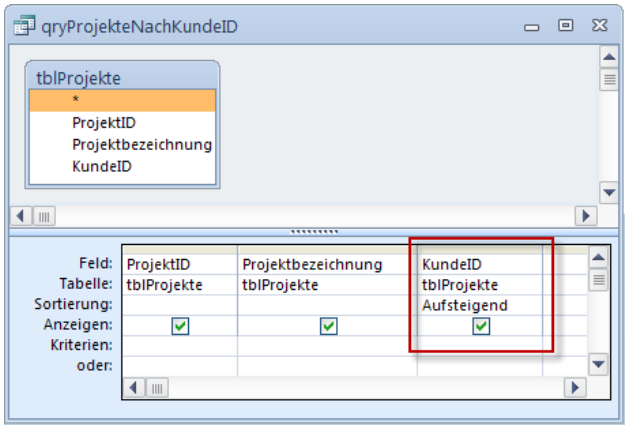


Bild 3: Diese Einstellung sollte die Datensätze der Tabelle **tblProjekte** nach dem Kundennamen sortieren.

Reihenfolge-Problem in Abfragen

Wenn Sie eine Abfrage auf Basis der Tabelle **tblProjekte** erstellen und die enthaltenen Datensätze aufsteigend nach dem Inhalt des Felds **KundeID** sortieren möchten (siehe Bild 3), erhalten Sie ein auf den ersten Blick überraschendes Ergebnis: Die Kunden werden gar nicht nach dem Alphabet, sondern in scheinbar willkürlicher Reihenfolge sortiert (siehe Bild 4).

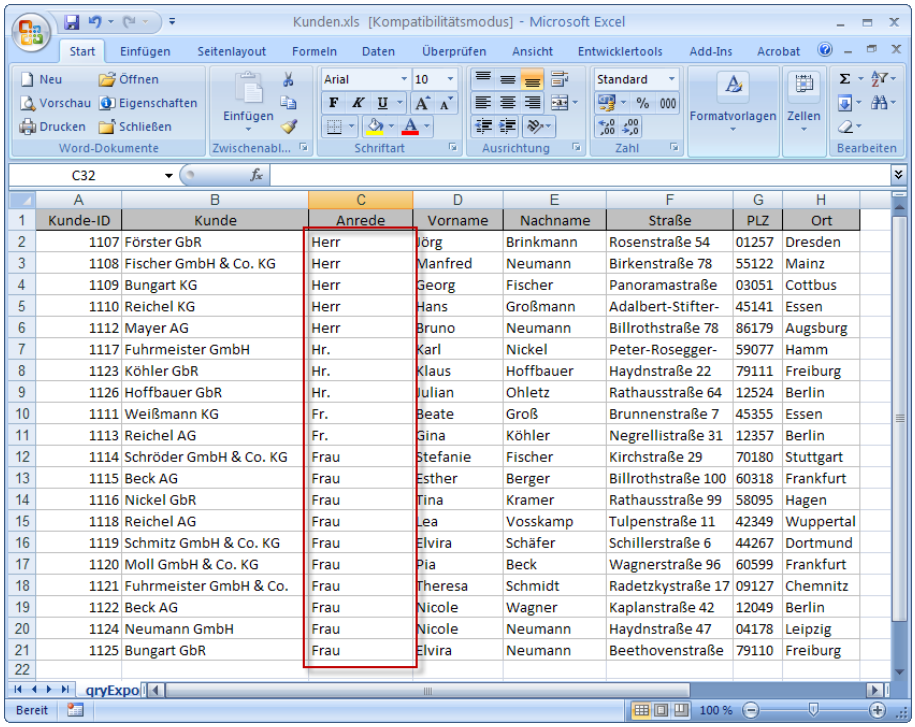
Des Rätsels Lösung ist, dass Access schlicht nach dem eigentlichen Wert des Feldes **KundeID** sortiert. Wenn Sie tatsächlich nach dem Kundennamen sortieren möchten, müssen Sie der Abfrage die verknüpfte Tabelle **tblKunden** hinzufügen und das Feld mit dem eigentlichen Kunden namen als Sortierkriterium verwenden (siehe Bild 5).

Daten importieren

Teil II: Vom Textfeld zur Lookuptabelle

Die wenigsten in Excel- oder sonstigen Tabellen gespeicherten Daten liegen so vor, dass man sie problemlos in verknüpfte Tabellen übernehmen kann. In dieser Folge der Artikelreihe zum Datenimport schauen wir uns an, wie Sie die Daten aus Spalten mit Anrede oder sonstigen Detaildaten gleich in eine entsprechende Lookuptabelle überführen.

Wenn Sie beispielsweise Kundendaten aus einer Excel-Tabelle importieren möchten, enthält diese etwa ein Feld namens Anrede. Dieses ist wiederum mit Ausdrücken wie Herr und Frau gefüllt – wenn Sie Glück haben! Gegebenenfalls finden sich dort auch noch ein paar Tippfehler und sorgen dafür, dass Sie die Daten normalerweise auch noch korrigieren müssen. Bild 1 vermittelt einen Eindruck davon, wie solche Daten aussehen können – die Spalte mit den kritischen Werten ist markiert. Mit den Werten **Herr** und **Frau** hätten wir noch leben können, aber zwischendurch war auch noch ein Abkürzungsfetischist mit der Dateneingabe beschäftigt, sodass noch die Werte **Hr.** und **Fr.** hinzugekommen sind. Normalerweise sollte man den Verursacher die Werte der Spalte händisch korrigieren lassen, aber möglicherweise ist der Übeltäter ja gar nicht mehr greifbar ... Wenn wir davon ausgehen, dass die Excel-Tabelle nicht eine Bildschirmseite voller Adressen, sondern vielleicht ein paar Tausend enthält, können wir von einer manuellen Anpassung ohnehin absehen.



	A	B	C	D	E	F	G	H
1	Kunde-ID	Kunde	Anrede	Vorname	Nachname	Straße	PLZ	Ort
2	1107	Förster GbR	Herr	Jörg	Brinkmann	Rosenstraße 54	01257	Dresden
3	1108	Fischer GmbH & Co. KG	Herr	Manfred	Neumann	Birkenstraße 78	55122	Mainz
4	1109	Bungart KG	Herr	Georg	Fischer	Panoramastraße	03051	Cottbus
5	1110	Reichel KG	Herr	Hans	Großmann	Adalbert-Stifter-	45141	Essen
6	1112	Mayer AG	Herr	Bruno	Neumann	Billrothstraße 78	86179	Augsburg
7	1117	Fuhrmeister GmbH	Hr.	Karl	Nickel	Peter-Rosegger-	59077	Hamm
8	1123	Köhler GbR	Hr.	Klaus	Hoffbauer	Haydnstraße 22	79111	Freiburg
9	1126	Hoffbauer GbR	Hr.	Julian	Ohletz	Rathausstraße 64	12524	Berlin
10	1111	Weißmann KG	Fr.	Beate	Groß	Brunnenstraße 7	45355	Essen
11	1113	Reichel AG	Fr.	Gina	Köhler	Negrellistraße 31	12357	Berlin
12	1114	Schröder GmbH & Co. KG	Frau	Stefanie	Fischer	Kirchstraße 29	70180	Stuttgart
13	1115	Beck AG	Frau	Esther	Berger	Billrothstraße 100	60318	Frankfurt
14	1116	Nickel GbR	Frau	Tina	Kramer	Rathausstraße 99	58095	Hagen
15	1118	Reichel AG	Frau	Lea	Vosskamp	Tulpenstraße 11	42349	Wuppertal
16	1119	Schmitz GmbH & Co. KG	Frau	Elvira	Schäfer	Schillerstraße 6	44267	Dortmund
17	1120	Moll GmbH & Co. KG	Frau	Pia	Beck	Wagnerstraße 96	60599	Frankfurt
18	1121	Fuhrmeister GmbH & Co.	Frau	Theresa	Schmidt	Radetzkystraße 17	09127	Chemnitz
19	1122	Beck AG	Frau	Nicole	Wagner	Kaplanstraße 42	12049	Berlin
20	1124	Neumann GmbH	Frau	Nicole	Neumann	Haydnstraße 47	04178	Leipzig
21	1125	Bungart GbR	Frau	Elvira	Neumann	Beethovenstraße	79110	Freiburg
22								

Bild 1: Inkonsistente Anrede-Bezeichnungen

Ziel der folgenden Operationen ist es, die Daten aus der Excel-Tabelle in den Ausschnitt des Datenmodells aus Bild 2 zu importieren. Dabei sollen die Einträge der Spalte **Anrede** der Excel-Tabelle gleich mit den Werten der Tabelle **tblAnreden** abgeglichen und der entsprechende Wert des Feldes **AnredeID** dieser Tabelle in das gleichnamige Feld der Tabelle **tblKunden** eingetragen werden. Wenn der erste Datensatz der zu importierenden Daten also den Wert **Herr** enthält, ermitteln wir zunächst den Wert von **AnredeID** für den Datensatz, der den Wert **Herr** enthält, in diesem Fall **1**. Tragen wir diesen für das Feld **AnredeID** des neu importierten Kundendatensatzes ein, haben wir die Anrede korrekt übernommen.

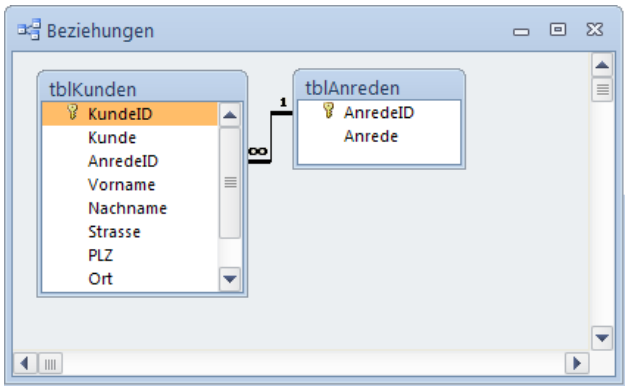


Bild 2: Ziel des Kundenimports: Die Tabelle **tblKunden**

Den Zusammenhang verdeutlicht Bild 3. Alle Felder der Excel-Tabelle können direkt in die Tabelle **tblKunden** importiert werden, nur das Feld **Anrede**

nicht. Hier machen wir einen Umweg über die Tabelle **tblAnreden**, wo wie die **AnredeID** für den Wert aus der Excel-Tabelle ermitteln und diesen schließlich in die Tabelle **tblKunden** eintragen.

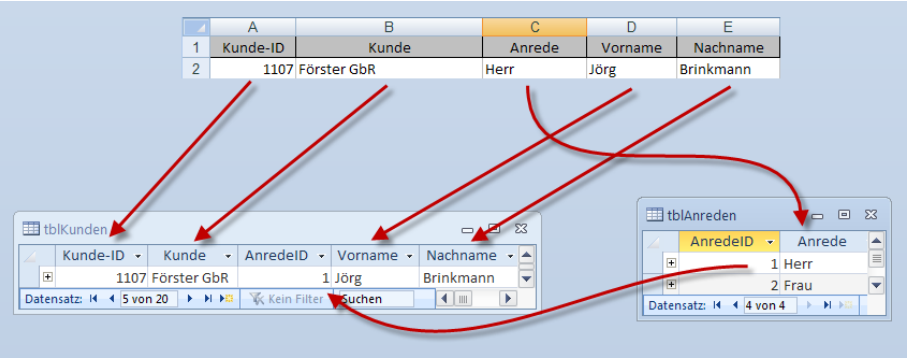


Bild 3: Die meisten Felder landen ohne Umweg in der Tabelle **tblKunden**. Nur die Anrede macht einen Umweg über die Tabelle **tblAnreden**.

Technische Umsetzung

Die grundlegenden Techniken haben wir bereits im Artikel **Daten importieren - Teil I: Daten aus CSV-Dateien** beschrieben. Wir wollen die im hinteren Bereich verwendete Technik verwenden, um zunächst eine Verknüpfung namens **tblImportKunden** auf die Excel-Tabelle zu erstellen und dann mit geeigneten Abfragen die Daten in die Zieltabelle überführen. Die Verknüpfung mit einer Excel-Datei anstelle einer CSV-Datei wie im Artikel beschrieben funktioniert prinzipiell genauso, nur dass Sie an geeigneter Stelle den Pfad in Richtung Excel einschlagen müssen (unter Access 2010 beispielsweise gleich durch einen Klick auf die Ribbon-Schaltfläche **Externe Daten|Importieren und Verknüpfen|Excel**).

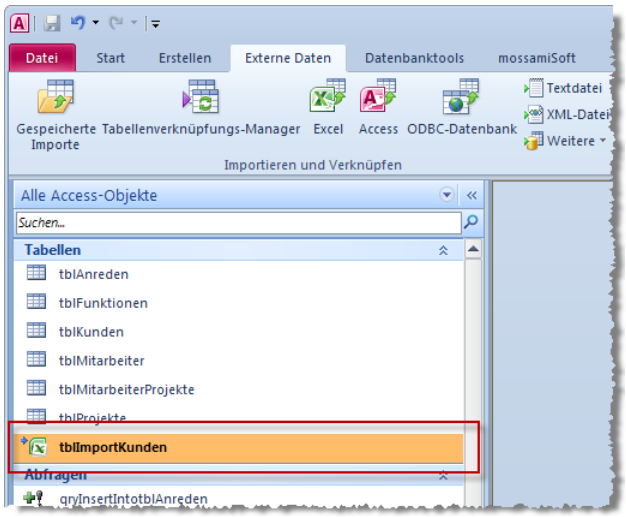


Bild 4: Liegt erst einmal eine Verknüpfung auf die Daten vor, können wir diese einfach mit Bordmitteln in die Zieltabellen überführen.

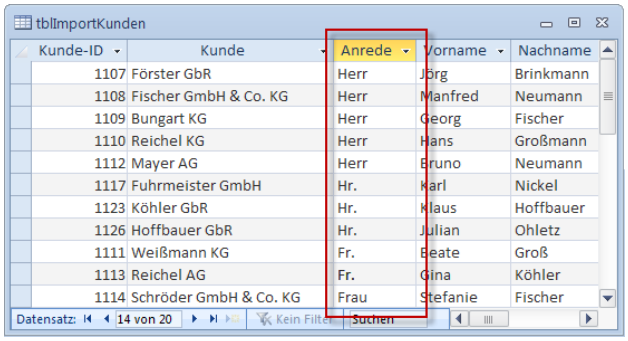


Bild 5: Der Inhalt der Verknüpfung sieht nun schon fast gut aus – wenn man vom Feld Anrede absieht ...

Herr und Frau auflösen

Den entscheidenden Schritt müssen wir jedoch noch gehen: Wir müssen irgendwie dafür sorgen, dass das Feld **Anrede** durch **AnredeID** und entsprechend die Werte **Herr** und **Frau** beziehungsweise **Hr.** und **Fr.** durch die Zahlenwerte **1** und **2** ersetzt werden.

Das Resultat sehen Sie am Beispiel von Access 2010 in Bild 4. Die Herkunft der Daten lässt sich unschwer am Excel-Symbol der Verknüpfung erkennen. Auch der Inhalt der Verknüpfung ist nicht mehr weit vom Ziel entfernt (siehe Bild 5).

Wir nähern uns der Lösung an, indem wir zunächst so tun, als würden **Hr.** und **Fr.** nicht existieren. Da wir in **Access [basics]** keine Techniken verwenden möchten, die nicht schon vorgestellt wurden, können wir hier nicht mit VBA arbeiten, um die Daten so anzupassen, dass wir diese in die Zieltabelle importieren können. Also bleibt uns nur übrig, die Daten mithilfe einer Abfrage so umzuformen, dass wir das gewünschte Ergebnis erhalten und diese Abfrage anschließend als Ausgangspunkt für eine Anfügeanfrage zu verwenden, mit der wir die Daten letztendlich in die Zieltabelle schreiben (Informationen über Anfügeabfragen finden Sie in **Aktionsabfragen - Teil I: Anfügeabfragen**).

Also öffnen wir eine neue, leere Abfrage in der Entwurfsansicht und fügen alles hinzu, was wir im Abfragergebnis benötigen. Die meisten Daten liefert die Verknüpfungstabelle **tblImportKunden**, den Rest die

Access [basics]
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

Formulare mit VBA programmieren

Teil I: Formularereignisse nutzen

Formulare können zwar ohne eine Zeile VBA-Code dazu dienen, Daten anzulegen, zu bearbeiten und wieder zu löschen – das haben Sie in einigen Artikeln bereits erfahren. Die notwendige Flexibilität erhalten Sie allerdings erst durch die Programmierung von Formularen. Die Programmiersprache für diesen Zweck heißt VBA. In diesem Artikel erhalten Sie grundlegende Informationen über die Programmierung von Formularen mit dieser Programmiersprache.

Formulare und ihre Steuerelemente bringen bereits einiges mit, damit Sie diese ohne auch nur eine Zeile VBA sinnvoll einsetzen können. Wenn Sie ein Formular mit einer Tabelle als Datenherkunft versehen und die Felder dieser Datenherkunft aus der Feldliste in den Detailbereich des Formulars gezogen haben, erhalten Sie bereits eine beachtliche Menge Funktionen:

- Sie finden Steuerelemente wie Textfelder, Kombinationsfelder oder Kontrollkästchen vor, welche die Daten des jeweils aktuellen Datensatzes anzeigen.
- Sie können mit den Navigationsschaltflächen durch die Datensätze blättern.
- Änderungen an den Inhalten der Steuerelemente werden nach dem Speichern automatisch in die zugrunde liegende Tabelle geschrieben.
- Das Formular bietet Schaltflächen an, mit denen Sie es schließen oder seine Anzeige verändern können.

Normalerweise werden Sie sich aber nicht auf diese Grundfunktionen beschränken wollen. Sie möchten

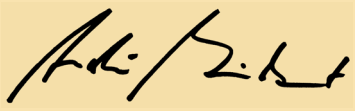
Was Sie in diesem Artikel lernen

» Wozu dienen Ereignisprozeduren?
» Wie legt man Ereignisprozeduren an?
» Welche Alternativen gibt es zu Ereignisprozeduren?
» Wie verwendet man den Cancel-Parameter einer Ereignisprozedur?

Schlagzahl erhöhen

Nachdem Sie nun fünf Ausgaben von **Access [basics]** erhalten haben, die bis auf eine Ausnahme ohne die Beschreibung von VBA-Techniken ausgekommen sind, wollen wir nun endlich beginnen, diese Programmiersprache zu lernen. Sie ahnen ja nicht, welche Fähigkeiten Sie Ihrer Access-Anwendung damit hinzufügen können! Klar, dass Sie Formulare und den Rest der Benutzeroberfläche damit automatisieren können. Aber später werden Sie damit auch andere Anwendungen wie Word oder Excel steuern, über das Internet auf Anwendungen wie eBay, Google oder Twitter zugreifen und vieles mehr. Zunächst jedoch konzentrieren wir uns auf unsere Beispielanwendung, die Projektzeiterfassung. Ihre Kenntnisse bezüglich VBA und Ereignisprozeduren, die Sie in dieser Ausgabe erwerben, können Sie nämlich gleich in die Praxis umsetzen. Beispielsweise im Artikel **Formulare für die Dateneingabe – Teil IV: Formulare öffnen und schließen**. Wichtige Grundlagen dafür liefert der Artikel **Formulare mit VBA programmieren – Teil I: Formularereignisse nutzen**. Dort erfahren Sie, wie Sie VBA-Code schreiben, der beispielsweise automatisch beim Öffnen eines Formulars ausgelöst werden soll. Außerdem beginnen wir unsere Reihe zum Thema **Steuerelemente**. Dort stellen wir das Schaltflächen-Steuerelement vor. VBA-Basics erhalten Sie vor allem im Beitrag **VBA-Grundlagen – Teil I: Einführung**.

Viel Spaß beim Lesen!



[André Minhorst]

In dieser Ausgabe:

Formulare mit VBA programmieren – Teil I: Formularereignisse nutzen

Steuerelemente – Teil I: Die Schaltfläche

Formulare für die Dateneingabe – Teil IV: Formulare öffnen und schließen

VBA-Grundlagen – Teil I: Einführung

Abfragen für die Datenauswahl – Teil II: Daten aus verknüpften Tabellen darstellen

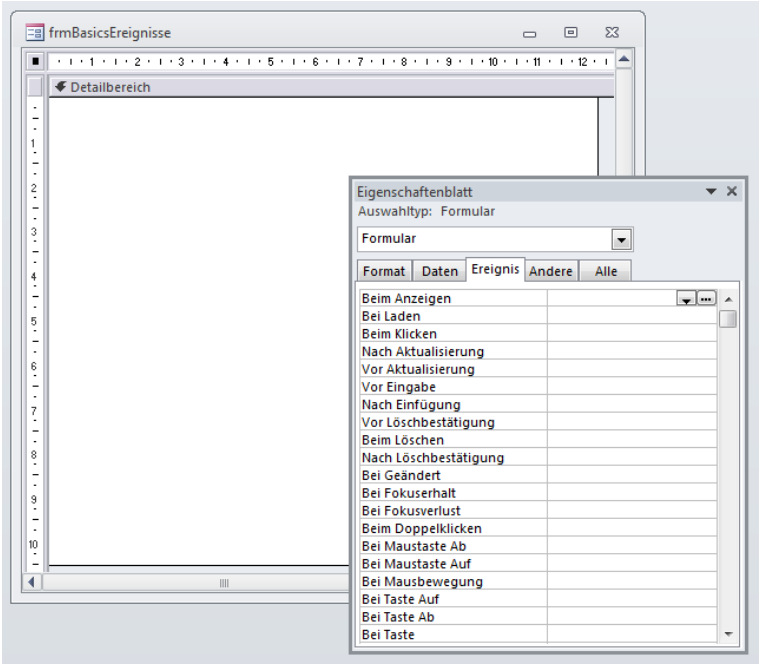


Bild 1: Das Eigenschaftsfenster mit den Ereigniseigenschaften

einem Formular vielleicht eine eigene Schaltfläche zum Schließen oder zum Abbrechen der Eingabe mitgeben, ein anderes Formular zur Eingabe von Detaildaten öffnen, eine eigene Meldung anzeigen, wenn ein Benutzer eine ungültige Eingabe durchgeführt hat, oder den Inhalt eines Kombinationsfeldes in Abhängigkeit von einem anderen Wert anpassen möchten. Dazu müssen Sie auf den Klick auf eine Schaltfläche, das Abschließen einer Eingabe oder auch das Speichern eines Datensatzes reagieren.

Damit dies möglich ist, liefert Access die sogenannten Ereigniseigenschaften. Diese finden Sie in der Entwurfsansicht eines Formulars auf der Seite **Ereignis** des Eigenschaftsfensters. Das Eigenschaftsfenster aus Bild 1 zeigt beispielsweise die Ereignisse für das Formular selbst an. Dies erkennen Sie daran, dass das Kombinationsfeld oben im Eigenschaftsfenster den Eintrag **Formular** anzeigt.

Wenn Sie nun im Formular auf den Bereich unterhalb des Textes **Detailbereich** klicken, zeigt das Eigenschaftsfenster die Ereignisse dieses speziellen Bereichs an. Und wenn Sie ein Steuerelement, wie beispielsweise eine Schaltfläche, hinzufügen und dieses markieren, erhalten Sie im Eigenschaftsfenster die Ereignisse der Schaltfläche.

Die Namen all dieser Ereigniseigenschaften bezeichnen den Zeitpunkt, zu dem die entsprechenden Ereignisse ausgelöst werden. Wenn Sie etwa ein Formular öffnen, wird das Ereignis **Beim Öffnen** ausgelöst. Wenn Sie das Formular schließen, feuert das Ereignis **Beim Schließen**. Manche Bezeichnungen sind nicht so intuitiv. Das Ereignis **Beim Anzeigen** beispielsweise erfolgt immer beim Anzeigen eines neuen Datensatzes, also prinzipiell bei jedem Datensatzwechsel.

Für Schaltflächen ist das Ereignis **Beim Klicken** besonders wichtig. Es wird jedes Mal ausgelöst, wenn der Benutzer mit der Maus auf die Schaltfläche klickt (und auch, wenn die Schaltfläche den Fokus besitzt und die Leer- oder Eingabetaste betätigt wird).

Bei den Steuerelementen, die Daten anzeigen – also beispielsweise Text- oder Kombinationsfelder –, sind andere Ereignisse wichtig: Beispielsweise das Ereignis **Vor Aktualisierung**, das kurz vor dem Speichern einer Änderung am Inhalt des jeweiligen Steuerelements ausgelöst wird.

Schauen Sie sich nun an, wie Sie von den Ereignissen profitieren können.

Von der Ereigniseigenschaft zur Ereignisprozedur

Wenn Sie Redewendungen wie „das Ereignis feuert“ oder „das Ereignis wird ausgelöst“ hören, bedeutet dies, dass ein Ereignis eintritt. Das Ereignis **Beim**

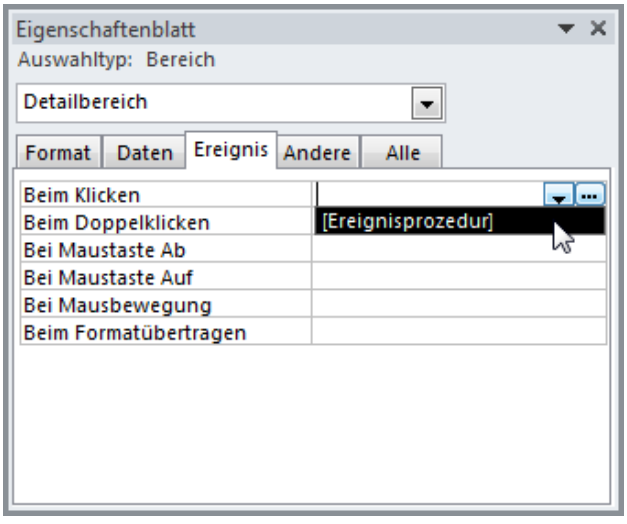


Bild 2: Einstellen des Wertes **[Ereignisprozedur]** für eine Ereigniseigenschaft

Abfragen für die Datenauswahl

Teil II: Daten aus verknüpften Tabellen darstellen

Im ersten Teil dieser Artikelreihe haben Sie erfahren, wie Sie die Daten einer einfachen Tabelle abfragen. Diesmal nehmen Sie eine zweite Tabelle hinzu und schauen sich an, wie Sie das Abfrageergebnis durch den geschickten Einsatz von Verknüpfungen beeinflussen können.

Abfragen dienen nicht nur dazu, die Felder einer einzigen Tabelle auszuwählen und deren Inhalte anzuzeigen oder die Datensätze nach verschiedenen Kriterien zu filtern. Eine wesentliche Aufgabe ist es, Daten aus mehr als einer Tabelle zusammenzuführen.

Dabei spielt es gar keine Rolle, ob die betroffenen Tabellen verknüpfte Daten enthalten. Dies ist zwar in der Regel der Fall, aber manchmal verwendet man gerade Tabellen ohne Verknüpfung, um spezielle Effekte zu erreichen. Schauen Sie sich doch zunächst einmal die herkömmliche Art an, zwei Tabellen in einer Abfrage zusammenzufassen.

Tabelle und Lookup-Tabelle zusammenführen

Der einfachste Grund zum Zusammenführen zweier Tabellen per Abfrage ist es, den in eine Lookup-Tabelle ausgelagerten Wert zusammen mit den übrigen Daten einer Tabelle anzuzeigen.

In der Beispieldatenbank finden Sie etwa die beiden Tabellen **tblKunden** und **tblAnreden**, die über das Feld **AnredeID** der Tabelle **tblKunden** miteinander verknüpft sind (siehe Bild 1).

Wenn Sie nun wie in →*[basics]* **Abfrage erstellen** beschrieben eine neue Abfrage anlegen und die beiden Tabellen **tblKunden** und **tblAnreden** in die Entwurfsansicht ziehen, übernimmt Access den Beziehungspfeil in die Abfrage.

Mit Beziehung oder ohne?

Es spielt eine gravierende Rolle, ob dieser unscheinbare Pfeil vorhanden ist oder nicht. Schauen Sie sich zunächst einmal das Ergebnis der Abfrage an, wie sie in Bild 2 dargestellt ist.

Sie zeigt genau die erwarteten Werte an: Sie enthält einige Felder eines jeden Datensatzes der Tabelle **tblKunden** und zusätzlich das Feld **Anrede** aus der Tabelle **tblAnreden** – und zwar genau entsprechend der Verknüpfung zwischen den beiden Tabellen (s. Bild 2).

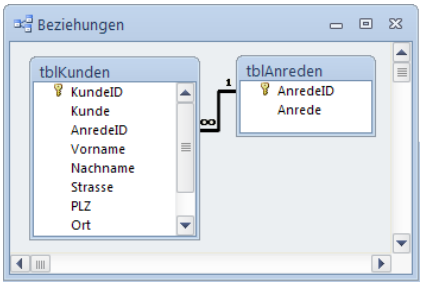


Bild 1: Die beiden Tabellen **tblKunden** und **tblAnreden** sind über das Feld **AnredeID** miteinander verknüpft.

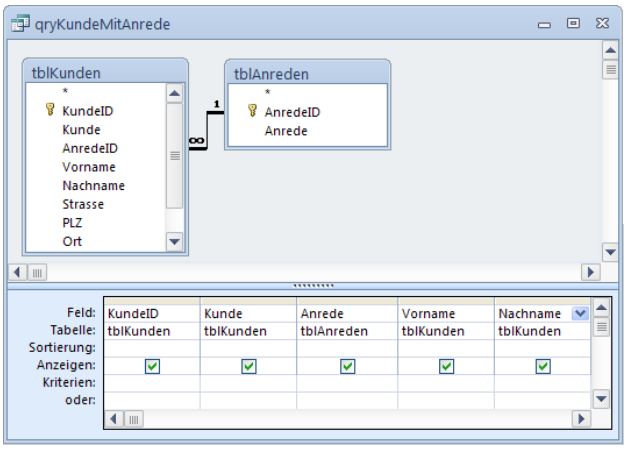


Bild 2: Beim Hinzufügen dieser beiden Tabellen wird die Beziehung aus dem Datenbankfenster in die Abfrage übernommen.

Bild 3: Das Ergebnis der Abfrage **qryKundeMitAnrede** sieht wie erwartet aus.

Was Sie in diesem Artikel lernen

» Tabellen und Lookup-Tabellen mit Abfragen zusammenführen
» Welche Rolle spielen Beziehungspfeile in einer Abfrage?
» Wie kombiniert man alle Datensätze zweier Tabellen?
» Warum Beziehungspfeile eigentlich Bedingungen sind

Beispieldatenbank

Die Beispieldatenbank zu diesem Artikel finden Sie im Onlinearchiv unter **http://www.access-basics.de/index.php/Heftarchiv.html**.

Wenn Sie nun zurück zur Entwurfsansicht der Abfrage wechseln, den Beziehungspfeil zwischen den beiden Tabellen markieren und löschen und dann wieder die Datenblattansicht der Abfrage anzeigen, liefert diese die Datensätze aus Bild 4. Dort erscheint jeder Datensatz dreimal und außerdem in Kombination mit allen drei möglichen Einträgen der Tabelle **tblAnreden**!

Hinweis: Beziehungspfeile in Abfragen

Sie können die Beziehungen in Abfragen setzen und entfernen, wie Sie möchten – die im Datenmodell festgelegten Beziehungen bleiben auf jeden Fall erhalten.

Jeder mit jedem

Um dies zu verstehen, müssen Sie wissen, wie Access Abfragen auswertet. Wenn eine Abfrage eine Tabelle ohne Kriterium enthält, liefert diese alle Datensätze der Tabelle zurück. Wenn eine Abfrage, wie im ersten Beispiel, zwei Tabellen mit Beziehungspfeil enthält, dann wertet Access dies als Kriterium.

Dieses besagt: Liefere mir alle Kombinationen der Datensätze der beiden Tabellen, für die gilt: Der Wert des Feldes **AnredeID** der Tabelle **tblKunden** ist gleich dem Wert des gleichnamigen Feldes der Tabelle **tblAnreden**. Der Beziehungspfeil ist also prinzipiell ein Kriterium, das die eigentliche Menge der zu liefernden Datensätze einschränkt. Wenn Sie dieses Kriterium weglassen, indem Sie den Beziehungspfeil löschen, lautet die Vorgabe für die Abfrage: Liefere mir alle Kombinationen der Datensätze der beiden Tabellen **tblKunden** und **tblAnreden**.

Und da die Tabelle **tblKunden** in diesem Beispiel 20 und die Tabelle **tblAnreden** drei Datensätze enthält, ergibt dies 20 x 3 = 60 Datensätze.

Das Ergebnis an sich ist sicher unsinnig, denn wozu benötigen Sie schon alle Kombinationen aus Kunden und Anreden? Immerhin wissen Sie aber nun, wie Sie alle Kombinationen der Datensätze zweier Tabellen erhalten – früher oder später werden Sie das einmal benötigen.

Autoverknüpfung

Es gibt in den Access-Optionen übrigens eine Option für Abfragen namens **AutoVerknüpfung aktivie-**

qryKundeMitAnrede

Kunde-ID	Kunde	Anrede	Vorname	Nachname
1107	Förster GbR	Herr	Jörg	Brinkmann
1107	Förster GbR	Frau	Jörg	Brinkmann
1107	Förster GbR	Firma	Jörg	Brinkmann
1108	Fischer GmbH &	Herr	Manfred	Neumann
1108	Fischer GmbH &	Frau	Manfred	Neumann
1108	Fischer GmbH &	Firma	Manfred	Neumann

Datensatz: 1 von 60 | Kein Filter | Suchen

Bild 4: Ohne Beziehungspfeile liefert die Abfrage ein ganz anderes Ergebnis.

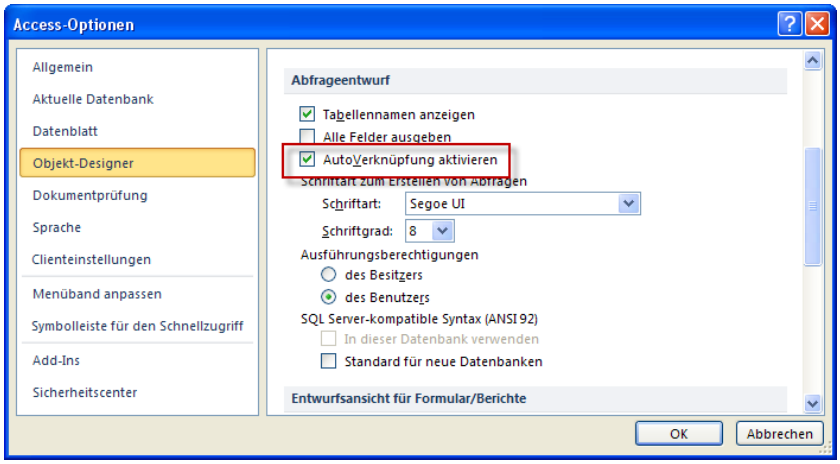


Bild 5: Diese Option sorgt mitnichten dafür, dass Beziehungen zwischen zwei Tabellen in den Abfrageentwurf übernommen werden.

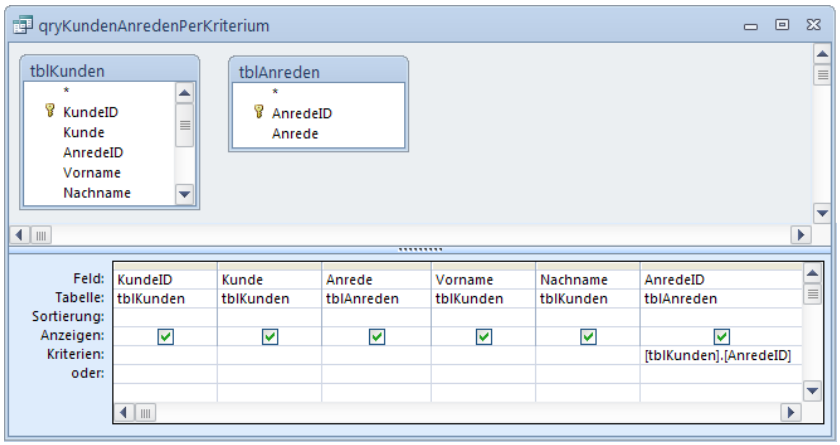


Bild 6: Die Funktion des Beziehungspfeils in Abfragen kann auch durch ein Kriterium nachgestellt werden.

VBA-Grundlagen

Teil I: Einführung

Mit Tabellen speichern Sie Daten, mit Abfragen wählen Sie diese aus, mit Formularen zeigen Sie diese an und Berichte sorgen für die Druckaufbereitung. Wer mehr Interaktion als die pure Eingabe, Ausgabe und Bearbeitung seiner Daten will, kommt um den Einsatz von VBA nicht herum: VBA liefert den wesentlichen Teil der Anwendungslogik und professionelle Access-Anwendungen kommen ohne sowieso nicht aus. Im ersten Teil unserer VBA-Grundlagen lernen Sie die Grundbegriffe kennen und erfahren, warum VBA so wichtig ist.

In den bisherigen Beispielen sind Sie ganz gut ohne VBA ausgekommen. Immerhin haben Sie schon Formulare erstellt, mit denen Sie Daten aus einfachen Tabellen und sogar aus verknüpften Tabellen angezeigt und bearbeitet haben. Dass dies alles funktioniert, liegt aber auch nur daran, dass Access seine Formulare standardmäßig mit einigen Grundfunktionen für die Datenverarbeitung ausstattet: Sie legen damit über verschiedene Eigenschaften beispielsweise fest, aus welcher Tabelle oder Abfrage die anzuzeigenden Daten stammen und welches Steuerelement welches Feld der Datenherkunft repräsentiert. Access synchronisiert automatisch die in Haupt- und Unterformular angezeigten Daten und sorgt dafür, dass die Formulare beim Öffnen mit Daten gefüllt werden. Außerdem blättern Sie mit den Elementen der Navigationsleiste einfach durch die einzelnen Datensätze und das Formular sorgt dabei automatisch dafür, dass geänderte Datensätze beim Datensatzwechseln gespeichert werden – genau wie beim Schließen des Formulars.

Dahinter steckt eine Menge Code, den Sie nicht mehr selbst schreiben müssen: Fast alle Aufgaben, die mit der Anbindung von Formularen und Berichten an die Daten aus Tabellen und Abfragen zusammenhängen, nimmt Access uns ab.

Access weiß aber nicht, was Sie sonst noch so mit den Formularen und Berichten anstellen möchten. Es weiß beispielsweise nicht, ob ein Klick auf eine **OK**-Schaltfläche das Formular schließen und den Datensatz speichern soll, während ein Klick auf eine **Abbrechen**-Schaltfläche die durchgeführten Änderungen vor dem Schließen verwerfen soll. Und Access kann auch nicht wissen, dass ein Doppelklick auf einen Datensatz in einem Listenfeld ein weiteres Formular mit der Detailansicht des angeklickten Datensatzes anzeigen soll.

Es gibt noch viel mehr Beispiele, was Sie in Access alles mit Ihren Daten anstellen können, ohne dass Access dafür eine vorgefertigte Funktion liefern würde – aber das macht Access gerade interessant: Es bietet einen soliden Grundstock an Funktionen, die speziell für die Datenbindung ausgelegt sind. Darüber hinaus können Sie es aber beliebig erweitern!

Im Gegensatz zum Zusammenklicken von Tabellen, Abfragen, Formularen und Berichten, was nach kurzer Zeit im Handumdrehen funktioniert, liegt die Latte bei all den kleinen Feinheiten ein wenig höher: Da Access, wie soeben erwähnt, zwar eine Menge eingebauter Funktionen besitzt, aber eben nicht alle vorstellbaren, müssen Sie Ihrer Datenbank-Anwendung selbst beibringen, was diese noch leisten soll.

VBA und Makros

Wenn Sie mit Word oder Excel arbeiten, kennen Sie möglicherweise die Funktion zum Aufzeichnen von Makros. Dabei starten Sie die Aufnahme, führen dann bestimmte Aktionen mit dem aktuell geöffneten Word- oder Excel-Dokument durch und beenden die Aufnahme. Danach können Sie die aufgezeichneten Schritte per Knopfdruck erneut ausführen lassen.

Im Hintergrund speichern diese Anwendungen die durchgeführten Schritte in Form einiger Codezeilen, sie erstellen also praktisch ein kleines Programm oder auch eine Routine in der Programmiersprache VBA (Visual Basic for Applications).

VBA-Routine

Unter einer VBA-Routine verstehen wir alle über einen Namen ansprechbaren Abfolgen von VBA-Befehlen. Es gibt verschiedene Arten von VBA-Routinen wie zum Beispiel Prozeduren und Funktionen, die wir nachfolgend vorstellen. Ein alternativer Begriff wäre Programm. Dieser Begriff ist jedoch schon synonym für Computeranwendungen vergeben, sodass wir besse den Begriff **Routine** verwenden, wenn wir allgemein über Prozeduren, Funktionen und Co. sprechen.

Diese können Sie sogar noch anpassen, um die Routine flexibler zu gestalten oder zu erweitern. Eine solche Routine hat immer die folgende Form:

```
Sub <Makroname>()
    <Befehle>
End Sub
```

<Makroname> ist der Name der Word- oder Excel-Routine. Über diesen Namen können Sie die Routine erneut aufrufen. **Sub** und **End Sub** markieren die erste und letzte Zeile der Routine, dazwischen befinden sich die Anweisungen, welche die Routine ausführen soll. Solche Routinen nennt man unter Word und Excel Makros.

Namen für VBA-Routinen

Sie können den Namen für eine VBA-Routine in den meisten Fällen frei wählen (die Ausnahme sind Ereignisprozeduren – dazu später mehr). Wichtig ist, dass Sie nur Buchstaben, Zahlen und Unterstriche verwenden. Das erste Zeichen darf keine Zahl sein, die Länge darf 200 nicht überschreiten und der Name darf noch nicht für eine andere VBA-Routine verwendet werden – weder für eine benutzerdefinierte noch für eine eingebaute Routine.

Sie sollten außerdem darauf achten, dass der Name möglichst genau beschreibt, was die VBA-Routine erledigt. Wenn Sie dazu mehr als ein Wort benötigen, beginnen Sie neue Wörter einfach mit einen Großbuchstaben (zum Beispiel **LoescheKunde**).

Wenn Sie den Namen einer VBA-Routine ändern, bedenken Sie, dass es bereits von anderen Stellen innerhalb der Anwendung Aufrufe dieser Routine geben kann, und passen Sie diese dann entsprechend an.

Nun sind Word und Excel auf der einen und Access auf der anderen Seite zwei verschiedene Welten: Word und Excel dienen vorrangig der Anzeige und Bearbeitung von Dokumenten, während Access gleichzeitig zur Entwicklung von Datenbankanwendungen und zu deren Ausführung verwendet wird. Ein weiterer wichtiger Unterschied ist die Benennung von Routinen: Während diese unter Word und Excel **Makro** heißen, verwendet man unter Access normalerweise die Bezeichnung **Prozedur**. Warum nennen Access-Entwickler ihre Programme nicht auch einfach Makro? Ganz einfach: Weil es dort noch einen ganz anderen Objekttyp zur Automatisierung von Vorgängen gibt, der bereits diese Bezeichnung innehat. Dieser erwartet nicht das Zusammenschreiben der Befehle in Form einer Routine, sondern bietet einen eigenen Editor, um die durchzuführenden Schritte zu notieren. Bevor wir eine feinere Einteilung vornehmen, fassen wir zusammen:

- Unter Word und Excel heißen VBA-Routinen Makros.
- Unter Access heißen VBA-Routinen Prozeduren oder Funktionen.
- Unter Access gibt es auch Makros, die aber nichts mit der Programmiersprache VBA gemeinsam haben.

Die Gemeinsamkeit von VBA-Routinen und Makros unter Access ist, dass Sie damit Befehlsfolgen auslösen können, wobei VBA deutlich flexiblere Möglichkeiten bietet.

VBA-Routinenarten

Es gibt verschiedene Arten von VBA-Routinen, wobei die Unterschiede je nach Betrachtungsweise leicht verwässern.

Vorsicht, Theorie!

Möglicherweise vermissen Sie in diesem Artikel zunächst den praktischen Teil. Sie sollten jedoch unbedingt einige grundlegende Begriffe kennenlernen, bevor Sie die VBA-Entwicklungsumgebung starten und dort loslegen.

Die einfachste Art einer VBA-Routine entspricht vom Aufbau her dem, was auch beim Aufzeichnen eines Makros unter Word oder Excel herauskommt:

```
Sub <Routinenname>()
    <Routinenschritte>
End Sub
```

Solch eine Routine rufen Sie ganz einfach über ihren Namen auf (wie das technisch geschieht, erfahren Sie gleich). Die Routine führt dann die enthaltenen Schritte aus. Wenn eine Routine nicht immer genau die gleichen Schritte ausführen soll, müssen Sie ihr die veränderlichen Parameter irgendwie mitteilen.

Dazu füllen Sie die im Beispiel oben noch leeren Klammern mit der Liste der möglichen Parameter:

```
Sub <Routinenname>(<Parameter1>, <Parameter2>, ...)
    <Routinenschritte abhängig von den Parametern>
End Sub
```

Die Routine kann die Parameter auswerten und die Schritte in Abhängigkeit von den Parametern gestalten. Der Aufruf solcher Routinen ist nicht so einfach wie der von Routinen ohne Parameter: Sie können sie nicht einfach über den Namen starten, sondern müssen auch noch die Parameter übergeben. Routinen wie diese heißen schlicht Prozedur.

VBA-Prozedur

Eine VBA-Routine, die in der Kopfzeile das Schlüsselwort **Sub** enthält, heißt VBA-Prozedur.

Die folgende Routinenart heißt Funktion und besitzt gegenüber einer einfachen Prozedur eine wichtige Erweiterung: Sie führt nicht nur die gewünschten Schritte aus, sondern gibt auch noch einen Wert zurück.

Später erfahren Sie, dass dies auch mit einfachen Prozeduren möglich ist – aber schauen Sie sich erstmal den Aufbau des Routinentyps Funktion an:

```
Function <Routinenname>()
    <Routinenschritte>
    <Routinenname> = <Ergebnis>
End Function
```

Funktionen zeichnen sich durch die Verwendung des Schlüsselworts **Function** statt **Sub** im Routinenkopf aus. Gleiches gilt für das Routinenende. Außerdem können Sie hier eine Zuweisung verwenden, um das Ergebnis der Routine an die aufrufende Instanz zurückzugeben (mit „aufrufender Instanz“ ist beispielsweise eine andere VBA-Routine gemeint).

VBA-Funktion

VBA-Routinen, die in der Kopfzeile das Schlüsselwort **Function** enthalten, heißen VBA-Funktion. Ihr Name dient beim Aufrufen quasi als Variable für den Rückgabewert der VBA-Funktion.

Funktionen können genau wie Prozeduren mit Parametern gefüttert werden:

```
Function <Programmname>(<Parameter1>, ...)
    <Programmschritte abhängig von den Parametern>
    <Programmname> = <Ergebnis>
End Function
```

Parameter können Sie aber nicht nur von der aufrufenden Instanz an eine VBA-Routine übergeben, sondern Sie können die Parameter unter bestimmten Umständen auch in der Routine ändern und die geänderte Fassung in der aufrufenden Instanz weiterverarbeiten. An dieser Stelle wird es Zeit für den Übergang vom theoretischen zum praktischen Teil. Sie erfahren hier, wie Sie Funktionen und Prozeduren aufrufen, wie die verschiedenen Parameter verwendet werden, wie Sie diese ändern und wie Sie die Rückgabewerte von Funktionen angeben und auswerten.

Entwicklungsumgebung

Den Schritt von der Theorie zur Praxis erledigen Sie durch das Öffnen der Beispieldatenbank **AccessBasics_2010_06.mdb**. Dort starten Sie den VBA-Editor wie in →*[basics]* **VBA-Editor öffnen** angegeben.

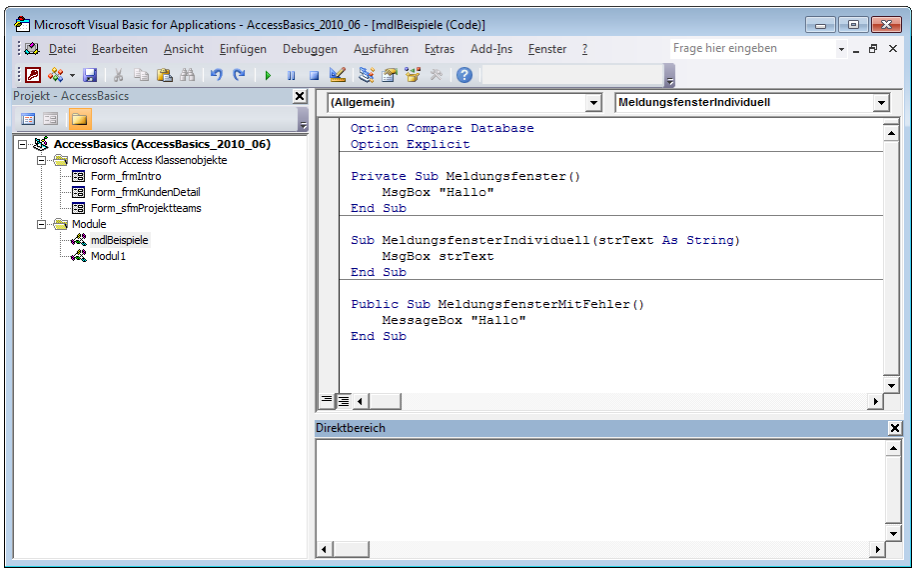


Bild 1: Eine durch einen Syntaxfehler verursachte Fehlermeldung

Im Gegensatz zu Tabellen, Abfragen, Formularen, Berichten und Makros, die Sie über die im Access-Fenster integrierten Entwicklungswerkzeuge entwerfen können, bearbeiten Sie die in den Modulen des VBA-Projekts enthaltenen Programme im VBA-Editor.

Nach dem Öffnen des VBA-Editors (s. Bild 1) brauchen Sie noch einen Ort, wo Sie Ihre ersten Beispielprogramme anlegen können. In diesem Fall benötigen Sie ein Standardmodul (es gibt noch andere Modultypen, die

Formulare für die Dateneingabe

Teil IV: Formulare öffnen und schließen

Bevor der Benutzer mit der Dateneingabe mittels Formular beginnen kann, muss Access dieses erst einmal anzeigen. Und das ist gar nicht so einfach, denn vielleicht möchten Sie das Formular ja dazu bringen, gleich einen bestimmten Datensatz anzuzeigen oder gar gleich einen neuen Datensatz zu präsentieren. Dieser Artikel zeigt, wie das Öffnen eines Formulars grundsätzlich funktioniert und wie Sie ein Formular von einem anderen Formular aus unter Verwendung verschiedener Einstellungen aufrufen können.

Die einfachste Art, ein Access-Formular für die Dateneingabe zu öffnen, ist der Doppelklick auf den entsprechenden Eintrag im Datenbankfenster (Access 2003 und älter) oder im Navigationsbereich (Access 2007 und jünger).

Diese Möglichkeit sollte jedoch dem Entwickler vorbehalten bleiben. Wer eine professionelle Software auf Basis von Microsoft Access entwickeln möchte, wird das Datenbankfenster genauso wenig einblenden wie er den Benutzer direkt in Tabellen arbeiten lässt (→*[basics]* **Keine Dateneingabe via Tabelle oder Abfrage**).

Wie sonst aber soll der Benutzer an die Formulare zur Datenbearbeitung gelangen? Dazu gibt es grundsätzlich folgende Möglichkeiten:

- Sie erstellen ein Übersichtsformular mit einer Reihe Schaltflächen, mit denen der Benutzer Formulare zur Erledigung der gewünschten Aufgaben öffnen kann. Diese Formulare werden automatisch beim Öffnen der Anwendung angezeigt.

- Sie stellen eine Menü- oder Symbolleiste (Access 2003 und älter) oder ein Ribbon (Access 2007 und jünger) bereit, das Schaltflächen zum Öffnen der benötigten Formulare anzeigt.

Beide Varianten sind weit verbreitet und es gibt sicher tausende Access-Anwendungen, die ihre Formulare per Datenbankfenster oder Navigationsbereich anbieten.

Jede Variante hat Vor- und Nachteile: Es ist zum Beispiel schwierig, das Übersichtsformular immer so zu platzieren, dass es ständig sichbar ist. Das ist aber sinnvoll, damit der Benutzer jederzeit auf die Funktionen der Anwendung zugreifen kann. Dafür ist es blitzschnell erstellt: Sie brauchen nur ein Formular anzulegen und ein paar Schaltflächen hinzuzufügen, welche die benötigten Formulare zur Datenbearbeitung öffnen.

Menü- und Symbolleisten sowie in den neueren Access-Versionen das Ribbon sind garantiert immer sichtbar, da sie normalerweise nicht durch andere Elemente der Benutzeroberfläche verdeckt werden. Außerdem kennen Benutzer anderer Office-Anwendungen sich bereits mit der Benutzung dieser Elemente aus.

Der Nachteil ist, dass es zumindest unter Access 2007 und jünger relativ aufwendig ist, ein benutzerdefiniertes und auf die jeweilige Anwendung abgestimmtes Ribbon zu definieren und mit den entsprechenden Funktionen zu füllen.

Die Beispieldatenbank **AccessBasics_2010_06.mdb** soll daher zunächst einfach nur ein Formular öffnen, von dem aus der Benutzer dann weitere Formulare anzeigen kann.

Grundsätzlich schaffen Sie damit auch den Grundstein für das Aufrufen der Formulare per Menü-/Symbolleiste oder über das Ribbon: Schließlich benötigen Sie überall die gleichen VBA-Befehle zum Anzeigen von Formularen. Den Umgang mit Menüs und dem Ribbon lernen Sie in einem späteren Artikel kennen.

Öffnen eines Formulars beim Starten einer Anwendung

Da das Beispiel vorerst ohne Menüs oder Ribbon auskommen soll, soll dieses gleich beim Start der Anwendung ein Formular anzeigen, mit dem der Benutzer die gewünschten Aktionen durchführen kann. Dazu verwenden Sie das Formular **frmKundenProjekte** aus unserer Beispieldatenbank.

Dieses stellen Sie einfach wie in →*[basics]* **Startformular einer Anwendung festlegen** beschrieben für die Eigenschaft **Startformular** beziehungsweise **Formular anzeigen** der aktuell geöffneten Datenbank ein. Beim nächsten Öffnen wird Access gleich zu Beginn das gewünschte Formular einblenden.

Formular zentriert anzeigen

Normalerweise wird das Formular einfach an der Position angezeigt, an der es zuletzt gespeichert wurde. Wenn Sie möchten, dass es bezogen auf das Access-Fenster zentriert angezeigt wird, wechseln Sie in die Entwurfsansicht des Formulars und stellen seine Eigenschaft **Automatisch zentrieren** auf **Ja** ein.

Die OpenForm-Methode

Zum Öffnen eines Formulars benötigen Sie VBA-Code. In diesem Falle handelt es sich um die Methode **OpenForm** eines speziellen Objekts namens **DoCmd**, das eine ganze Reihe Methoden zur Verfügung stellt (was eine Methode ist, erfahren Sie unter **VBA-Grundlagen – Teil I: Einführung** – dort lernen Sie auch, was es mit dem nachfolgend erwähnten Direktfenster auf sich hat).

Zum Testen der **DoCmd.OpenForm**-Methode öffnen Sie den VBA-Editor und aktivieren dort das Direktfenster zur Eingabe von VBA-Anweisungen. Beides gelingt mit der Tastenkombination **Strg + G** in einem einzigen Schritt. Wenn Sie das Access-Fenster und das VBA-Fenster nebeneinander oder untereinander platzieren, können Sie die Auswirkungen der folgenden Befehle direkt beobachten. Sie verwenden nun die einfachste Variante der **DoCmd.OpenForm**-Anweisung, um das Formular **frmKunden-Projekte zu** öffnen. Geben Sie dazu den folgenden Befehl in das Direktfenster ein und betätigen Sie die Eingabetaste:

```
DoCmd.OpenForm "frmKundenProjekte"
```

Beim Eingeben zeigt der VBA-Editor gleich die Liste aller Parameter dieser Methode an (s. Bild 1). Den Namen des zu öffnenden Formulars übergeben Sie gleich mit dem ersten Parameter. Alle übrigen Parameter sind optional.

Parameter der OpenForm-Methode

OpenForm besitzt die folgenden Parameter:

- **View**: Legt die Ansicht des Formulars fest, zum Beispiel Standardansicht, Datenblattansicht oder Entwurfsansicht.
- **FilterName**: Stellt einen Filter ein, um nur bestimmte Datensätze anzuzeigen.
- **WhereCondition**: Gibt eine Bedingung an, um die anzuzeigenden Datensätze einzuschränken.
- **DataMode**: Gibt an, ob das Formular zum Bearbeiten eines vorhandenen oder zum Anlegen eines neuen Datensatzes geöffnet werden soll.
- **WindowMode**: Legt fest, ob das Formular normal, versteckt, minimiert oder als modaler Dialog angezeigt werden soll.
- **OpenArgs**: Dient zur Übergabe einer Zeichenkette, die vom Formular ausgewertet werden kann.

In den folgenden Beispielen lernen Sie die Einsatzzwecke der verschiedenen Werte für diese Parameter kennen.

Formularansicht einstellen

Mit dem ersten Parameter **View** stellen Sie die Ansicht ein, in der Access das Formular anzeigen soll.

Sie können diesen Parameter jedoch in der Regel weglassen, denn das Formular soll in der unter der Eigenschaft **Standardansicht** des Formulars festgelegten Ansicht angezeigt werden – hier **Formularansicht**.

Von Formular zu Formular

Die Erläuterung der übrigen Parameter macht in Zusammenhang mit dem Öffnen eines Formulars von einem anderen Formular aus mehr Sinn als am Beispiel eines einzelnen Formulars.

Im folgenden Beispiel soll das Formular **frmProjektMitKunde** auf verschiedene Arten vom Formular **frmKundenProjekte** aus geöffnet werden.

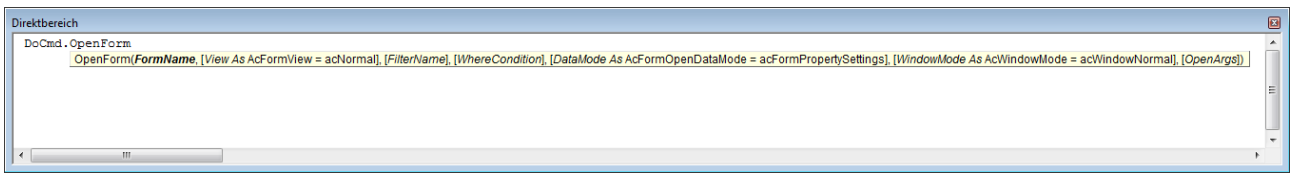


Bild 1: Die Parameterliste der DoCmd.OpenForm-Anweisung

Als Erstes lernen Sie dabei den Parameter **WhereCondition** kennen. Das Formular **frmProjektMitKunde** soll so geöffnet werden, dass es genau das Projekt anzeigt, das gerade im Unterformular **sfmKundenProjekte** ausgewählt ist.

Dazu benötigen Sie erst einmal eine entsprechende Schaltfläche. Also öffnen Sie das Formular **frmKundenProjekte** in der Detailansicht, vergrößern den Detailbereich durch Ziehen am unteren Rand ein wenig und fügen dann die neue Schaltfläche hinzu (weitere Informationen unter **Steuerelemente – Teil I: Die Schaltfläche**). Stellen Sie als Beschriftung **Projekt öffnen** und als Steuerelementname **cmdProjektOeffnen** ein (s. Bild 2).

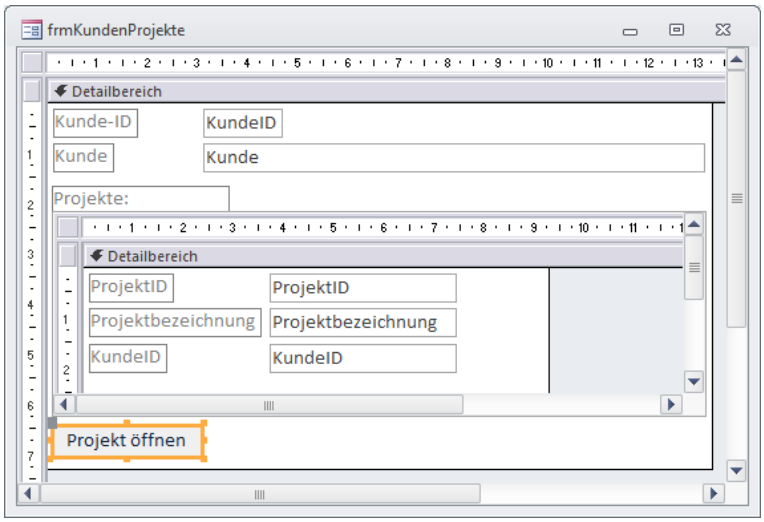


Bild 2: Hinzufügen einer Schaltfläche zum Anzeigen der Details eines Datensatzes

Nun legen Sie eine Ereignisprozedur für diese Schaltfläche an (grundlegende Informationen zu Ereignisprozeduren liefert der Artikel **Formulare mit VBA programmieren – Teil I: Formularereignisse nutzen**). Die Ereignisprozedur soll Folgendes für uns erledigen:

- Ermitteln, welcher Datensatz des Unterformulars aktuell ausgewählt ist.
- Das Formular **frmProjekteMitKunde** öffnen und dort gleich den im Unterformular ausgewählten Kunden anzeigen.

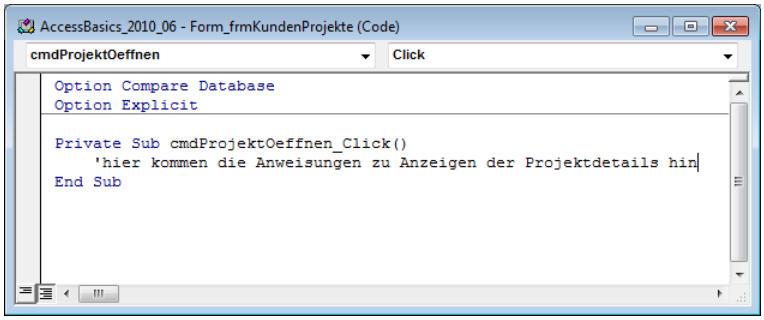


Bild 3: Diese Prozedur wird beim Klick auf die Schaltfläche **cmdProjektOeffnen** ausgelöst.

Dazu markieren Sie die neue Schaltfläche, zeigen das Eigenschaftsfenster an (Taste **F4**), wechseln zur Registerseite **Ereignis** und wählen dort für die Eigenschaft **Beim Klicken** den Wert **[Ereignisprozedur]** aus. Klicken Sie anschließend auf die Schaltfläche mit den drei Punkten (...), öffnet Access den VBA-Editor. Dieser hat bereits ein Klassenmodul namens **Form_frmKundenProjekte** für das Formular **frmKundenProjekte** erstellt und den Rahmen der Prozedur **cmdProjektOeffnen_Click** angelegt (s. Bild 3).

OpenForm mit Bedingung

Grundsätzlich verwenden Sie wieder die **DoCmd.OpenForm**-Methode, um das Formular mit den Projektdetails zu öffnen. Diesmal geben Sie dem aufgerufenen Formular jedoch noch einen Wert für den Parameter **WhereCondition** mit auf den Weg. Diese **WhereCondition** soll dem Formular eine Information darüber liefern, welchen Datensatz es beim Öffnen anzeigen soll (genau genommen soll es überhaupt nur diesen einen Datensatz anzeigen).

Diese Bedingung besteht im einfachsten Fall aus dem Namen des Feldes, einem Gleichheitszeichen und dem Wert, den das Feld haben soll.

In diesem Fall soll die Prozedur den Wert des Feldes **ProjektID** des aktuell markierten Datensatzes aus dem Unterformular ermitteln und dafür sorgen, dass das neu geöffnete Formular genau den Datensatz mit dieser **ProjektID** anzeigt. Der Ausdruck, der dazu übergeben werden soll, sieht beispielsweise so aus:

```
ProjektID = 123
```

Der Wert für den Parameter **WhereCondition** wird in Form einer Zeichenkette übergeben und muss daher in Anführungszeichen eingefasst werden:

```
"ProjektID = 123"
```

Steuerelemente

Teil I: Die Schaltfläche

Eine Schaltfläche ist eines der einfachsten Steuerelemente: Sie klicken darauf und es geschieht etwas. Man kann Schaltflächen vielleicht noch aktivieren und deaktivieren und die Beschriftung anpassen, aber viel mehr Möglichkeiten bieten sie nicht. Gute Voraussetzungen also, um unsere Reihe zum Thema Steuerelemente zu beginnen.

In diesem Artikel erfahren Sie, wie Sie Schaltflächen anlegen, lernen die wichtigsten Eigenschaften kennen und fügen den Schaltflächen VBA-Code hinzu, damit diese eine Funktion erhalten.

Schaltfläche anlegen

Eine Schaltfläche fügen Sie folgendermaßen zu einem Formular hinzu:

- Öffnen Sie das Formular in der Entwurfsansicht.
- Fügen Sie über die Schaltfläche über die Toolbox (**Ansicht|Toolbox**, Access 2003 und älter) oder über den Ribbon-Eintrag **Entwurf|Steuerelemente|Schaltfläche** (Access 2007 und jünger) zur Entwurfsansicht hinzu.

Fertig! Bringen Sie die Schaltfläche nun in die richtige Größe und an die richtige Position (mehr zu diesem Thema erfahren Sie unter →*[basics]* **Steuerelemente ausrichten und skalieren**).

Möglicherweise meldet sich beim Anlegen der Schaltfläche der Befehlsschaltflächen-Assistent von Access. Dieser unterstützt Sie beim Anlegen einer Schaltfläche inklusive der Zuweisung einer von mehreren Aktionen, der Benennung und der Beschriftung. Natürlich können Sie diesen Assistenten nutzen, aber leider verrät er Ihnen anschließend nicht, wo er welche Einstellungen vorgenommen hat.

Sie sollten einen Assistenten aber nicht verwenden, ohne genau zu wissen, was dieser tut – sonst werden Sie später nur schwer nachträgliche Änderungen vornehmen können.

Beispieldatenbank

Die Beispiele zu diesem Artikel finden Sie in der Beispieldatenbank zu dieser Ausgabe (**Access-Basics_2010_06.mdb**) im Formular **frmBasics-Schaltflaechen**.

Wenn das Erscheinen des Assistenten beim Anlegen von Steuerelementen stört, können Sie diesen über die Schaltfläche **Steuerelement-Assistenten** der Toolbox (Access 2003 und älter) oder den Ribbon-Eintrag **Entwurf|Steuerelemente|Steuerelement-Assistenten verwenden** (Access 2007 und jünger) deaktivieren. Diese Einstellung gilt für alle Steuerelement-Assistenten.

Beschriftung der Schaltfläche

Danach stellen Sie die Beschriftung ein, was auf zwei Arten funktioniert:

- Am einfachsten ist es, wenn Sie die Schaltfläche erst markieren und diese dann erneut anklicken. Die Beschriftung beginnt zu blinken und kann geändert werden.
- Alternativ aktivieren Sie die Schaltfläche und anschließend das Eigenschaftsfenster (Taste **F4**) und passen auf der Registerseite **Format** den Eintrag **Beschriftung** an.

Sie können auch die gewünschte Schaltfläche aus dem Kombinationsfeld des Eigenschaftsfensters auswählen und dann die entsprechende Eigenschaft ändern.

Name der Schaltfläche

In diesem Fall wird offensichtlich, dass die Schaltfläche mit **Befehl0** (oder ähnlich) noch den von Access beim Anlegen vergebenen Namen trägt.

Dies müssen Sie dringend ändern, wenn Sie später einmal per VBA auf diese Schaltfläche zugreifen möchten oder die für diese Schaltfläche angelegten Ereignisprozeduren (mehr dazu weiter unten) anhand eines aussagekräftigen Namens identifizieren wollen.

Wie unter →*[basics]* **Konventionen** vorgeschlagen, verwenden Sie für die Benennung von Schaltflächen das Präfix **cmd** und einen Namen, der die durch die Schaltfläche ausgelöste Funktion beschreibt. In diesem Fall soll ein Klick auf die Schaltfläche zunächst

Access [basics]
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

Steuerelemente

Teil II: Das Listenfeld

Eines der praktischsten und vielseitigsten Steuerelemente auf Formularen ist ein Listenfeld. Es zeigt ähnlich wie ein Unterformular komplette Datensätze oder auch nur einzelne Felder in Listenform an und kann beispielsweise als Ersatz für ein Kombinationsfeld zur Auswahl eines verknüpften Datensatzes zum Einsatz kommen.

Zum Nachvollziehen der auf den folgenden Seiten dieses Artikels beschriebenen Beispieltechniken erstellen Sie zunächst ein leeres Formular in der Entwurfsansicht und speichern es unter dem Namen **frmListenfeld**. Wir wollen uns zunächst auf das Listenfeld selbst konzentrieren, daher binden wir das Formular selbst an keine Tabelle oder Abfrage als Datenquelle.

Die Anzeige von Daten soll in diesem Fall das Listenfeld übernehmen, was auch völlig unabhängig von einer eventuell dem Formular zugewiesenen Datenherkunft möglich ist. In der Praxis geschieht dies oft: Das Listenfeld dient dann beispielsweise schlicht zum Auswählen eines der darin enthaltenen Datensätze. Ein Doppelklick auf diesen öffnet dann beispielsweise ein Detailformular. Oder Sie markieren einen Datensatz im Listenfeld, der dann beim Klick auf eine bestimmte Schaltfläche gelöscht werden soll. Die Alternative ist, dass das Formular selbst Daten wie beispielsweise die Eigenschaften eines Fahrzeugs anzeigt und das Listenfeld die zu diesem Fahrzeug gehörenden Sonderausstattungsmerkmale auflistet.

Doch nun zurück zur Vorbereitung des Beispielformulars. Dort legen Sie folgendermaßen ein Listenfeld an:

- **2003 und älter:** Klicken Sie in der Toolbox (die Sie mit **Ansicht|Toolbox** sichtbar machen) auf das Symbol **Listenfeld**.

Was Sie in diesem Artikel lernen

» Daten einer Tabelle oder Abfrage im Listenfeld darstellen
» Eigenschaften wie Spaltenanzahl oder Spaltenbreiten verwenden
» Reihenfolge und Spaltenüberschriften definieren
» Auf die einzelnen Spalten und Zeilen zugreifen

Access [basics] mit Verstärkung

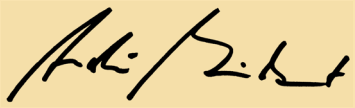
Nach sechs Ausgaben und einem halben Jahr **Access [basics]** ist es Zeit für Veränderungen. Das Magazin soll noch professioneller werden, und dafür haben wir mit Lorenz Hölscher einen weiteren Autor gewonnen. Er hat bereits einige Bücher über Access für Microsoft Press geschrieben und außerdem Artikel in Magazinen wie inside Access veröffentlicht. Von Ausgabe 7/2010 an werden wir also mindestens zu zweit am Inhalt des Magazins feilen.

Inhaltlich machen wir weiter wie geplant. In dieser Ausgabe kümmern wir uns zunächst um eines der interessantesten der in Access eingebauten Steuerelemente: das Listenfeld. Sie können dieses direkt an Tabellen oder Abfragen binden und somit beispielsweise eine Kundenliste anzeigen. Aber nicht nur das: Sie können das Listenfeld natürlich auch zur Auswahl eines der enthaltenen Einträge nutzen und beispielsweise ein Formular mit den Details zu diesem Datensatz öffnen.

Außerdem schauen wir uns die Löschabfragen an, mit denen Sie auf einen Rutsch Datensätze mit bestimmten Eigenschaften aus einer Tabelle löschen. Damit dies nicht in schwerwiegenden Datenverlusten oder inkonsistente Datenbeständen endet, klären wir Sie über Folgen und Nebenwirkungen auf.

Die letzten beiden Artikel schließlich beackern das in der vorherigen Ausgabe eröffnete Thema VBA-Programmierung. Dort dreht sich alles um das Anlegen eigener Prozeduren und das Definieren von Bedingungen mit If...Then und Select Case.

Viel Spaß beim Lesen!



[André Minhorst]

In dieser Ausgabe:

Steuerelemente - Teil II: Das Listenfeld

Aktionsabfragen - Teil II: Löschabfragen

VBA-Grundlagen - Teil II: Eigene Routinen schreiben

VBA-Grundlagen - Teil III: If...Then- und Select Case-Bedingungen

- **2007 und jünger:** Klicken Sie im Ribbon auf den Eintrag **Entwurf|Steuerelemente|Listenfeld** (Formularsteuerelement).

Danach klicken Sie einfach auf die Stelle im Formularentwurf, an der Sie das Listenfeld hinzufügen möchten. Gegebenenfalls mischt sich noch der Listenfeld-Assistent ein, den Sie aber mit einem Klick auf die **Abbrechen**-Schaltfläche beenden – schließlich wollen Sie genau wissen, wie sich welche Einstellung der Eigenschaften auswirkt. Grundsätzlich deaktivieren Sie die Steuerelement-Assistenten durch einen Klick auf die Schaltfläche mit dem Zauberstab-Symbol.

Nach dem Hinzufügen passen Sie Größe und Position des Listenfeldes durch Ziehen mit der Maus oder eine der übrigen Techniken an (→*[basics]* **Steuerelemente ausrichten und skalieren**). Stellen Sie nun den Namen des Listenfeldes auf **lstKunden**, wobei **lst** das Präfix für Listenfeld-Steuerelemente ist.

Im Moment ist das Listenfeld noch leer, wenn Sie in die Formularansicht umschalten, denn es hat keine Datensatzherkunft. Dort steht typischerweise eine Tabelle oder Abfrage, die Sie ganz bequem auch aus einer Liste auswählen können, die für diese Eigenschaft angezeigt wird. Wählen Sie hier zunächst die Tabelle **tblKunden** aus und schauen sich das Ergebnis in der Formularansicht an (s. Bild 1).

Die Darstellung der Kunden-IDs ist zwar korrekt, aber nur wenig hilfreich. So ist zwar sichergestellt, dass nur gültige Kunden-IDs auswählbar sind, aber wie heißen denn die Kunden? Eine kleine Anpassung der Listenfeld-Eigenschaften führt gleich zu einem besseren Ergebnis. Stellen Sie einfach nur die Eigenschaft **Spaltenanzahl** auf den Wert **2** ein, sieht das Ergebnis wie in Bild 2 aus.

Na, wenn das so einfach ist, können wir eigentlich auch gleich die ganze Adresse des Kunden anzeigen lassen! Wie ein Blick in die Tabelle **tblKunden** zeigt, steht diese in den Spalten sechs bis acht. Sie müssten also jetzt mindestens acht Spalten anzeigen lassen.

Das ist technisch möglich, macht die Anzeige aber wirklich nicht übersichtlicher. Dazwischen stehen einige uninteressante Spalten, die sich immerhin über die Angabe der Spaltenbreiten ausblenden lassen. Da acht Spalten angezeigt werden, sollten in der **Spaltenbreiten**-Eigenschaft auch acht cm-Werte stehen. Tatsächlich reichen sieben, denn die letzte Spalte erhält dann einfach den restlichen Platz. Um eine Spalte auszublenden, setzen Sie deren Breite auf **0cm**, hier also die erste (**KundeID**) und die dritte bis fünfte (**AnredeID** bis **Nachname**):

```
0cm;4cm;0cm;0cm;0cm;3cm;1,2cm
```

Die übrigen **cm**-Werte sind vor allem von der Gesamtbreite des Listenfeldes und der gewählten Schriftart abhängig, damit die Einträge gut zu lesen sind. Wie zu sehen ist, sind auch Nachkommawerte erlaubt, oft werden diese aber von Access noch ein wenig nachgerundet (aus 1,5cm werden so 1,501cm). Anschließend müssen Sie auch die Gesamtbreite des Listenfeldes oben im Lineal in der Entwurfsansicht überprüfen. Sie sollte Platz für alle Spalten bieten. Etwas irritierend wirken auf den Benutzer nun vermutlich die Datensatzmarkierer und Navigationselemente – zumal deren Betätigung sich aktuell nicht bemerkbar macht. Also blenden Sie diese einfach aus, und zwar mit den folgenden Eigenschaftswerten (s. Bild 3):

- **Datensatzmarkierer**: **Nein**
- **Navigationsschaltflächen**: **Nein**

Sie wären damit so zufrieden? Na, dann sollten Sie mal den Chef unserer Beispielfirma Schwuppdiwupp schimpfen hören: „Die langen Firmennamen kann ich nicht vollständig lesen und die Hausnummer ist auch nicht immer zu sehen! Und überhaupt: das ist ja völlig unsortiert!“

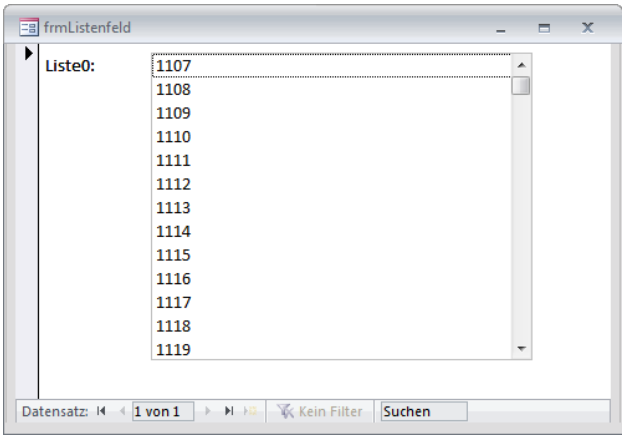


Bild 2: Die Liste zeigt die Kunden-IDs an.

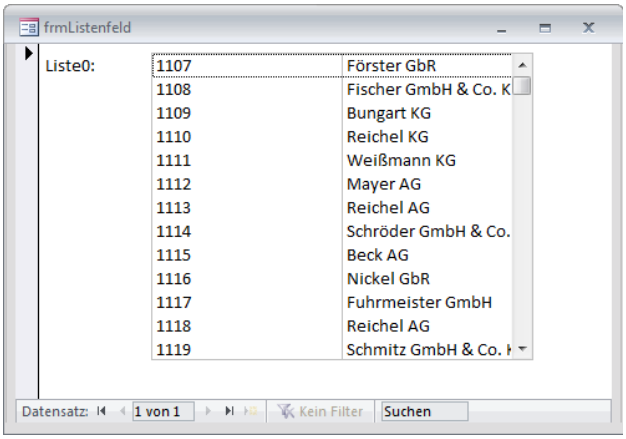


Bild 1: Mit der zweiten (Spalte) sieht man besser...

Bild 3: Jetzt kann man alle gewünschten Angaben im Listenfeld sehen und auch die unnützen Formularelemente sind verschwunden.

Jetzt haben Sie zwei Möglichkeiten (mal abgesehen von einer sofortigen Kündigung): Sie flicken weiter an den Spalten herum oder Sie machen es gleich richtig. Anstatt also die Spalten dieser Tabelle ein- und auszublenden, erstellen Sie lieber eine Abfrage, die nur die benötigten Felder liefert. Diese kann bei der Gelegenheit auch gleich sortieren und Texte bündeln.

Klicken Sie in der Listenfeld-Eigenschaft **Datensatzherkunft** neben dem Eintrag **tblKunden** auf die Schaltfläche mit den drei Pünktchen. Nach dem Bestätigen der nun erscheinenden Meldung öffnet sich der Abfrage-Generator und Sie können gezielt nur die Spalten auswählen, die später auch im Listenfeld erscheinen sollen. Erste und vor allem wichtigste Spalte ist **KundeID**, denn sie identifiziert den markierten Datensatz. Auch das Feld **Kunde** können Sie noch ungeändert übernehmen und dort auch gleich die gewünschte Sortierung aktivieren.

Anstatt nun die Spalten **AnredeID** bis **Nachname** später über die Spaltenbreiten des Listenfelds auszublenden, fügen Sie diese erst gar nicht zum Abfrageentwurf hinzu. Weil die Adresse mit sehr unterschiedlichen Textlängen sehr schlecht zu lesen war, fassen Sie diese am besten in einem berechneten Feld zusammen. Dazu fügen Sie den folgenden Ausdruck in die oberste Zeile ein, wo sonst der Feldname steht:

```
Adresse: [PLZ] & " " [Ort] & ", " & [Strasse]
```

Diesen Ausdruck können Sie übrigens auch komfortabler eingeben als direkt im Abfrageentwurf eingeben. Dazu platzieren Sie die Einfügemarke im zu bearbeitenden Text und betätigen dann die Tastenkombination **Umschalt + F2**. Nach der Eingabe und einem Klick auf die **OK**-Schaltfläche wird der Ausdruck in den Abfrageentwurf übernommen.

Danach schließen Sie diesen Abfrage-Generator und nicken die folgende Access-Meldung ab, um die Abfrage als Datensatzherkunft des Listenfelds zu übernehmen.

Statt des Namens der Tabelle finden Sie nun in der **Datensatzherkunft**-Eigenschaft ein komplettes SQL-Statement, das den Inhalt des Abfrageentwurf repräsentiert. Diese Abfrage wird übrigens nicht wie herkömmliche Abfragen separat gespeichert und erscheint somit nicht im Datenbankfenster beziehungsweise Navigationsbereich.

Da die Abfrage allerdings inzwischen nur noch drei Spalten zurückgibt, sollten Sie für das Listenfeld die entsprechenden Eigenschaften anpassen:

- **Spaltenanzahl**: **3**
- **Spaltenbreiten**: **0cm;5cm**

Mit dieser Gestaltung sollte auch der Chef zufrieden sein, denn das Listenfeld ist sortiert und viel besser lesbar (s. Bild 4).

Spaltenüberschriften anzeigen

Gelegentlich werden Sie Daten in Listenfeldern anzeigen, deren Spalteninhalt erklärungsbedürftig ist – beispielsweise Preise von Artikeln mit und ohne Mehrwertsteuer. Für diesen Fall sieht Access die Anzeige von Spaltenüberschriften in Listenfeldern vor. Diese aktivieren Sie schlicht durch Einstellen der Eigenschaft **Spaltenüberschriften** in den Eigenschaften des Listenfeldes.

Typischerweise enthalten die Spaltenüberschriften die in der als Datenherkunft hinterlegten Tabelle oder Abfrage angegebenen Feldnamen. Wegen der Konventionen für Feldnamen taugen diese jedoch nicht immer als Beschriftungen in der Benutzeroberfläche – oder möchten Sie den Benutzern Ihrer Datenbank Beschriftungen wie **PreisMitMwSt** vorsetzen?

Es ist jedoch kein Problem, dies zu ändern. Die Felder von Tabellen und Abfragen besitzen in der Entwurfsansicht eine Eigenschaft namens **Beschriftung**. Dort legen Sie den Ausdruck fest, der in Formularen angezeigt werden soll (mehr dazu unter →*[basics]* **Beschriftungen von Tabellenfeldern einstellen**).

Bild 4: Überarbeitetes Listenfeld mit ausreichender Spaltenbreite

Die Sache hat nur einen kleinen Haken: In diesem Beispiel steht die Datensatzherkunft des Listenfeldes bislang direkt als SQL-Ausdruck in der entsprechenden Eigenschaft. Dort werden Informationen wie etwa alternative Beschriftungen der einzelnen Felder nicht gespeichert. Sie müssen also eine eigene gespeicherte Abfrage auf Basis des für die Eigenschaft angegebenen SQL-Ausdrucks anlegen. Das geht zum Glück ganz einfach: Sie brauchen nur den Entwurf dieser Abfrage zu öffnen (per Klick auf die Schaltfläche mit den drei Punkten) und anschließend die Tastenkombination **Strg + S** zu betätigen. Als Name für die Abfrage verwenden Sie **qryFrmListenfeldLstKunden**, also die Kombination aus dem Präfix **qry**, dem Namen des Formulars und dem Namen des Listenfelds. So können Sie diese Abfrage später genau zuordnen. Praktischerweise ersetzt Access den SQL-Ausdruck in der Eigenschaft **Datensatzherkunft** nach dem Speichern gleich durch den Namen der gespeicherten Abfrage.

Hier können Sie nun für jede einzelne Spalte eine individuelle Beschriftung festlegen (s. Bild 5). Das Listenfeld mit Spaltenüberschriften sieht dann etwa wie in Bild 6 aus.

Reihenfolge der Spalten

Auch wenn Sie dies vermutlich schon ahnen: Das Einstellen der Reihenfolge der Felder der Datensatzherkunft für die Darstellung im Listenfeld geschieht ausschließlich über die Datensatzherkunft selbst. Wenn Sie die Reihenfolge beeinflussen möchten, brauchen Sie also zwingend eine Abfrage, mit der Sie die Felder der zugrunde liegende(n) Tabelle(n) sortieren können.

Zusatzinformationen anzeigen

Hätten Sie gedacht, dass unser Chef nun einfach so zufrieden ist? Da kennen Sie ihn aber schlecht! Kaum haben wir alle seine Wünsche erfüllt, fallen ihm ein paar neue ein: „Ich will zu der ausgewählten Firma sehen, wie der Ansprechpartner heißt! Und für manche Gelegenheiten brauche ich nur die Firmen in der Liste, die auch aktiv sind!“

Fangen wir mit der Anzeige des Ansprechpartners an. Der soll ja nicht in der Liste selber auftauchen, weil das so viel Platz braucht. Nur für den markierten Kunden will der Chef den Ansprechpartner sehen können. Legen Sie also zuerst ein neues Textfeld unterhalb des Listenfeldes an:

- **2003 und älter:** Klicken Sie in der Toolbox auf das Symbol **Textfeld**.
- **2007 und jünger:** Klicken Sie im Ribbon auf Eintrag **Entwurf|Steuerelemente|Textfeld**.

Anschließend klicken Sie unterhalb des Listenfeldes in den Detailbereich und benennen das Textfeld gleich als **txtAnsprechpartner**. Es bleibt übrigens ungebunden, denn mangels Datensatzquelle des Formulars könnte es ja ohnehin an kein Datenfeld gebunden werden.

Wie erhält es aber seine Daten zum Anzeigen? Es nutzt eine Funktion für seine Eigenschaft **Steuerelementinhalt**, die zu den so genannten Domänen-Funktionen gehört. Diese sind in der Lage, aus Datensatzgruppen (eben jenen „Domänen“) Werte zu ermitteln. Sie können diese als Mini-Abfragen betrachten (mehr dazu in einer der folgenden Ausgaben). Tragen Sie für den Steuerelementinhalt also keinen Feldnamen ein wie sonst, sondern die folgende Formel:

```
=DomWert("Nachname";"tblKunden";"KundeID = " &
Me!lstKunden)
```

Wichtig ist dabei nicht nur, dass die Formel in einer Zeile notiert wird, sondern vor allem, dass der Verweis auf **lstKunden** nicht innerhalb der Zeichenkette steht. Die **DomWert**-Funktion erwartet als erstes Argument den Namen des Feldes, dessen Wert Sie wissen wollen. Als zweites folgt die Angabe der Tabelle oder Abfrage, ein **SELECT**-Statement ist hier übrigens nicht erlaubt. Das dritte Argument ist optional und wird bei Bedarf als Filter eingesetzt. Hier wollen wir genau den Datensatz des markierten Kunden auslesen, daher wird die Kunden-ID so gefiltert, dass sie zur Auswahl im Listenfeld passt.

So wird allerdings nur der Nachname angezeigt. Die Formel muss sozusagen verdoppelt werden, damit Vor- und Nachname sichtbar sind:

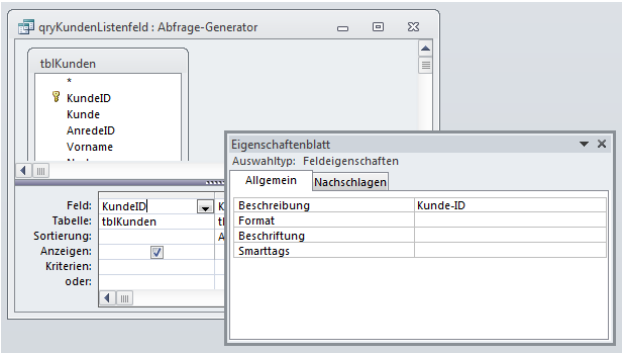


Bild 5: Festlegen von Texten für Beschriftungen und Spaltenüberschriften

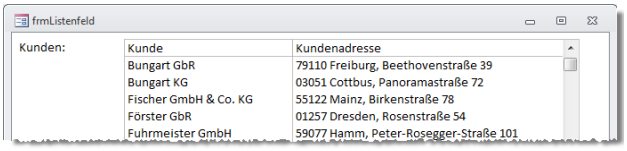


Bild 6: Listenfeld mit Spaltenüberschriften

Aktionsabfragen

Teil II: Löschabfragen

Das ganze Wochenende war verregnet und der Chef unserer Beispielfirma Schwuppdiwupp hat vor lauter Langeweile schon angefangen, zu Hause alten Krimskrams auszuräumen. Jetzt kommt er voller Schwung ins Büro und beschließt, dort auch gleich mal richtig auszumisten. Vor allem die Datenbank hat er auf dem Kieker, da ist doch bestimmt ganz viel überflüssiges Zeugs drin. So ganz überzeugt ist er ja doch nicht von den vielen Tabellen, bei Excel hat schließlich auch immer eine Tabelle ausgereicht.

Bevor er jetzt schon wieder von den guten alten Zeiten erzählt, in denen sowieso alles besser war, machen wir ihm ein Friedensangebot: es gibt in Access extra Löschabfragen, damit geht das Aufräumen rasend schnell.

So besänftigt verschwindet er in seinem Büro und nervt andere Kollegen. Diese Verschnaufpause müssen wir ausnutzen. Was löschen wir denn mal auf die Schnelle?

Die erste Idee klingt super: Kunden können aktiv sein oder nicht, dafür gibt es ja extra ein **Ja/Nein**-Feld. Inaktive Kunden sind unnütze Kunden, also weg damit.

Wie machen wir das? Dazu gibt es mindestens zwei Möglichkeiten. Die erste ist die, einfach eine Tabelle, Abfrage oder ein Formular mit Kunden zu öffnen, dort den betroffenen Projekt-Datensatz per Klick auf den Datensatzmarkierer links auszuwählen und diesen dann etwa durch Betätigen der Entfernen-Taste zu löschen. Wer regelmäßig mehrere Datensätze löscht, möchte dies aber vielleicht schneller erledigen. Mit ein wenig Glück gelingt dies auf einfache Weise: Wenn die zu löschenden Datensätze nämlich direkt untereinander angezeigt werden, markieren Sie den obersten Datensatz, halten die Umschalttaste gedrückt und klicken dann auf den letzten zu löschenden Datensatz. Access markiert dann alle Datensätze vom ersten bis zum letzten angeklickten Datensatz. Zum Löschen betätigen Sie dann ebenfalls einfach die Entfernen-Taste.

Leider liegen nicht immer alle zu löschenden Datensätze so untereinander, dass man diese so leicht gleichzeitig markieren und löschen kann. Bevor Sie diese nun doch einzeln entfernen, machen Sie sich die Fähigkeiten einer Auswahlabfrage zu Nutze: Erstellen Sie eine neue Abfrage, fügen Sie die Tabelle mit den zu löschenden Daten hinzu und ziehen Sie die Felder, die als Kriterium für die zu löschenden Datensätze dienen, in das Entwurfsraster.

In diesem Fall reicht das Feld **Aktiv** aus. Zur Sicherheit fügen Sie noch ein weiteres Feld mit dem Kundennamen hinzu – Sie wollen vor Löschen schließlich sehen, welche Datensätze Sie letztendlich entfernen.

Für das Feld **Aktiv** stellen Sie in der Zeile Kriterien des Abfrageentwurfs den Wert **Falsch** ein (s. Bild 1). Nach einem Wechsel in die Datenblattansicht (am schnellsten mit der Tastenkombinatin **Strg + .**) finden Sie tatsächlich nur noch die Datensätze vor, die nicht mehr aktiv sind und folglich gelöscht werden können. Hier brauchen Sie nun tatsächlich nur noch blind alle Datensätze zu markieren und diese zu löschen – Sie werden sicher nicht versehentlich einen noch aktiven Kunden löschen (wenn Sie übrigens mal eine Tabelle komplett leeren wollen, lassen Sie die Bedingung einfach weg).

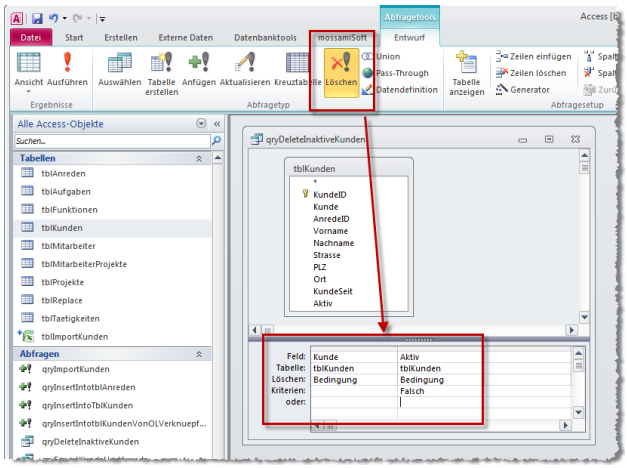


Bild 1: Diese Abfrage liefert alle inaktiven Kunden.

Was Sie in diesem Artikel lernen

» Definieren von Abfragen zum Löschen eines oder mehrerer Datensätze
» Auswirkungen von Löschvorgängen
» Referentielle Integrität und Löschweitergabe
» Aktivieren von Meldungen beim Löschen von Daten per Löschabfrage

VBA-Grundlagen

Teil II: Eigene Routinen schreiben

Im ersten Teil dieser Artikelreihe haben wir grundlegende Begriffe zum Thema VBA-Programmierung vorgestellt. Diesmal gehen wie ein wenig ins Detail: Wir zeigen Ihnen, wie Sie Ihre erste eigene VBA-Routine erstellen und ausführen.

In **VBA-Grundlagen - Teil I: Einführung** haben Sie grundlegende Begriffe zum Thema VBA-Programmierung kennengelernt, und unter **Formulare mit VBA programmieren - Teil I: Formularereignisse nutzen** haben Sie erfahren, wie Sie VBA-Code einsetzen, um Formulare zu automatisieren. Heute möchten wir den Einstieg in die Programmierung ein wenig allgemeiner gestalten.

Ein Blick auf die Objekt-Kategorien von Access fördert Erstaunliches zu Tage: Access ist das einzige Office-Programm mit zwei Programmiersprachen! Sowohl Makros als auch Module haben mit der Programmierung zu tun.

Eine schlechte Nachricht gibt es in diesem Zusammenhang allerdings auch: Access ist ebenfalls das einzige Office-Programm, in dem es keine Makro-Aufzeichnung gibt. Das ist allerdings so schlimm nun auch wieder nicht, denn selbst in den anderen Programmen lässt sich kaum die Hälfte der möglichen Befehle überhaupt aufzeichnen (wie soll man eine selbst erzeugte Meldung schließlich aufzeichnen, bevor sie erzeugt wurde?). Auch dort ist also trotzdem Lesen und Probieren angesagt. Zurück zu Access und seiner Programmierung. Es gibt wie schon erwähnt zwei Programmiersprachen:

- Makros sind deutschsprachig und bieten die datenbankspezifischen Grundfunktionen an.

- Module enthalten die englische Programmiersprache VBA, mit der Sie völlig unbegrenzte Möglichkeiten haben.

Mit den Makros werden wir uns in einem eigenen Beitrag ausführlich beschäftigen, hier soll es um die eigentliche Programmiersprache VBA gehen. VBA ist eine Abkürzung für **Visual Basic for Applications**, also in Deutsch etwa **Grafisch orientiertes BASIC für Anwendungen**. Wie der Name schon erkennen lässt, ist es eine Variante von **BASIC**, einer Programmiersprache, die in den 1960er Jahren entwickelt wurde. **BASIC** (**Beginner's All-Purpose Instruction Set**, etwa **Sammlung von Anfänger-Befehlen für alle Zwecke**) war eine Sprache, mit der die Programmierung einfach erlernt werden sollte. Darüber ist das heutige VBA meilenweit hinausgewachsen. Es ist keineswegs eine auf Anfänger beschränkte Sprache, sondern bietet den vollen Zugriff auf alle Fähigkeiten Ihres Computers.

Trotzdem hat sie sich die Einfachheit erhalten, die den Code übersichtlich und lesefreundlich belässt. Na gut, die Sprache ist Englisch, aber so schlimm ist das auch wieder nicht. Englisch ist nämlich wesentlich kürzer als die entsprechend Wörter im Deutschen und hat keine problematischen Sonderzeichen wie Umlaute. Kurz gesagt: es macht Spaß, mit VBA zu programmieren! Uns selber sowieso, aber Ihnen sicherlich auch bald.

Die Programmierung mit VBA erfolgt mit einem eigenen Programm, dem VBA-Editor. Das ist wieder eine gute Nachricht, denn dieser VBA-Editor ist für alle Office-Programme gleich. Auch VBA ist für die unterschiedlichen Programme weitgehend identisch, sodass Sie nach dem Erlernen von Access-VBA praktisch sofort in Word-VBA oder Excel-VBA weiterprogrammieren können. Den VBA-Editor starten Sie in allen Access-Versionen mit **Alt + F11** und damit können Sie auch jeweils wieder zu Access zurückwechseln. Der VBA-Editor besteht typischerweise aus drei Fenstern (s. Bild 1):

- Der Projekt-Explorer oben links zeigt alle offenen Datenbanken und die darin enthaltenen (Klassen-) Module an. Er lässt sich mit **Ansicht|Projekt-Explorer** anzeigen, falls er nicht sichtbar ist.

- Das Eigenschaften-Fenster unten links (zu aktivieren mit **Ansicht|Eigenschaftenfenster**) zeigt in Access lediglich den jeweiligen Modulnamen an.

Was Sie in diesem Artikel lernen

» VBA-Editor: Grundlagen
» Variablendeklaration erzwingen
» Erstellen und Benennen benutzerdefinierter Routinen
» Programmieren eines Meldungsfenster-Aufrufs
» Aufrufen benutzerdefinierter Routinen

VBA-Grundlagen

Teil III: If...Then- und Select Case-Bedingungen

VBA-Routinen durchlaufen standardmäßig alle Zeilen von oben nach unten. Manchmal soll eine Routine jedoch abhängig von einer Bedingung nur einen Teil der Zeilen abarbeiten. Dieser Artikel erläutert, wie Sie diese Bedingungen programmieren und welche Ausprägung für welchen Einsatzzweck geeignet ist.

Jedes Routine muss irgendwann mal Entscheidungen fällen, zum Beispiel:

- Wenn dieser Wert größer als 100 ist, wird diese Prozedur ausgeführt, ansonsten jene.
- Wenn die Rechnungssumme des Kunden mehr als 50,- Euro beträgt, dann wird kein Mindermengenzuschlag berechnet.
- Wenn der Benutzer die Frage mit Nein beantwortet hat, soll die genannte Datei nicht gelöscht werden.

Dahinter verbirgt sich eine immer wieder gleiche Konstruktion mit dem Schlüsselwort-Paar **If** und **End If**. Sie hatten schon mal einen ersten Blick in **Steuerelemente - Teil I: Die Schaltfläche** darauf geworfen, weil dort eine Schaltfläche in Abhängigkeit vom Zustand des Datensatzes geändert werden sollte.

Hier werden wir uns das **If...Then** mit allen seinen Möglichkeiten aber nicht im Formular-Klassenmodul, sondern in einem ganz „normalen" Standard-Modul ansehen. Der Unterschied zwischen diesen beiden Arten von Modulen ist, um dies nochmals in Erinnerung zu rufen:

- Standard-Module enthalten Prozeduren, die Sie jederzeit von überall aus aufrufen können.
- Klassen-Module (die gibt es einzeln oder als Teil von Formularen/Berichten) müssen erst aktiviert werden, bevor deren Funktionen nutzbar sind. Das passiert zum Beispiel automatisch beim Anzeigen von Formularen/Berichten.

Obwohl das vorherige Aufrufen von Klassenmodulen (wir werden später von Instanzieren sprechen, weil eine neue Instanz angelegt wird) eher mühsam erscheint, bieten sie durchaus viele Vorteile. Damit werden wir uns ausführlich beim Thema Klassen-Programmierung beschäftigen.

Zum Testen und Kennenlernen sind allerdings die Standardmodule praktischer, weil Sie darin erstellte Prozeduren sofort ausführen können. Im einfachsten Fall verwenden Sie einfach das Modul, dass Sie im Artikel **VBA-Grundlagen - Teil II: Eigene Routinen schreiben** erstellt haben.

Die Aufgabe besteht darin, eine Benutzerentscheidung auszuwerten. Ihr neues Makro soll den Benutzer etwas fragen und je nach Antwort unterschiedlich reagieren. Das sieht als Prozedur-Rumpf erst einmal ganz ähnlich aus wie für die Meldung:

```
Sub BenutzerFragen()

End Sub
```

Praktischerweise kann der **MsgBox**-Befehl nicht nur einfache Meldungen erzeugen, sondern auch verschiedene Symbole und Schaltflächen anzeigen, zum Beispiel so:

```
Sub BenutzerFragen()
   MsgBox „Gefällt Ihnen das?“, vbYesNo
End Sub
```

Sie können das bereits mit **F5** starten und es sieht genau wie geplant aus, aber: wie erfährt Ihr Programm eigentlich, was der Benutzer angeklickt hat? Nun: Genaugenommen ist die MsgBox-Routine eine Funktion, die nur immer wie eine einfache Prozedur aufgerufen wird – also ohne das Abfragen eines Rückgabewertes.

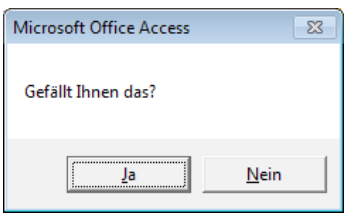


Bild 1: So sieht diese Meldung mit zwei Schaltflächen aus.

Was Sie in diesem Artikel lernen

» Verwenden der If...Then-Bedingung
» Auswerten der Rückgabewerte der MsgBox-Funktion
» Verwenden der Select Case-Verzweigung
» Vor- und Nachteile von If...Then und Select Case

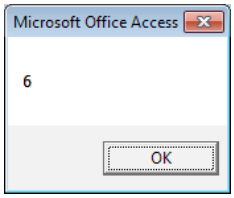


Bild 2: Beim Anklicken der **Ja**-Schaltfläche liefert die folgende **MsgBox** den Wert **6**.

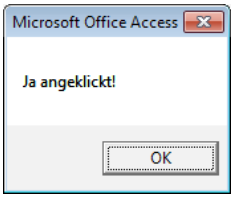


Bild 3: Diese Meldung erscheint nur, wenn vorher **Ja** angeklickt wurde.

Wenn Sie erfahren wollen, welchen Rückgabewert die **MsgBox**-Funktion liefert, brauchen Sie erstens eine Variable, die den Rückgabewert speichert und müssen die Funktion zweitens wie in folgendem Beispielcode formulieren:

```
Sub BenutzerFragen()
    Dim intAntwort As Integer
    intAntwort = MsgBox(„Gefällt Ihnen das?“, 7
                                          vbYesNo)
End Sub
```

Die **Dim**-Anweisung deklariert **intAntwort** als Variable des Typs **Integer**. Das bedeutet, dass diese Variable Zahlenwerte von -32.768 bis 32.767 aufnehmen kann. Auf die verschiedenen Variablentypen gehen wir in einer der folgenden Ausgaben ein.

Wenn Sie diese Routine starten, sehen Sie noch keinen Unterschied zur Variante ohne Rückgabewert. VBA weiß nun zwar schon, wie sich der Benutzer beim Anklicken entschieden hat, aber die Prozedur teilt es Ihnen nicht mit. Sie müssen es sich irgendwie anzeigen lassen, beispielsweise mit einem weiteren **MsgBox**-Aufruf, diesmal ohne Rückgabewert:

```
Sub BenutzerFragen()
    Dim intAntwort As Integer
    intAntwort = MsgBox(„Gefällt Ihnen das?“, 7
                                          vbYesNo)
    MsgBox intAntwort
End Sub
```

Je nachdem, was Sie oder ein zukünftiger Benutzer nun anklicken, meldet die zweite **MsgBox** den passenden Wert. Dieser lautet entweder **6** oder **7**, je nachdem, ob Sie die Schaltfläche **Ja** oder **Nein** anklicken (s. Bild 2).

Nanu, Zahlen? Sie hatten doch Schaltflächen mit den Texten **Ja** und **Nein** angeklickt? Ja, aber es sind absichtlich keine Texte und das ist auch gut so.

Stellen Sie sich vor, Sie schreiben eine solche Prozedur und prüfen, ob der Benutzer **Ja** eingegeben hat. Anschließend geben Sie das der Kollegin Bergmann zum Testen. Leider wussten Sie nicht, dass diese eine geborene Komarowska ist und ihren Rechner mit einem polnischen Windows betreibt. Dort antwortet die Meldung also mit **Tak**. Und bei Herrn Schneider läuft ein englischsprachiges Windows, weil er das cooler findet. Dort antwortet Ihre Meldung mit **Yes**. Wäre das praktisch, wenn Sie für jede Sprache ein eigenes **Ja/Yes/Tak/Oui...** auswerten müssten?

Nein, natürlich wäre das völlig unbrauchbar. Daher gibt es sprachneutrale Zahlen, die in jedem Land gleich lauten. Der Benutzer merkt davon ja nichts. Ihr Code hingegen wird dadurch viel einfacher, denn er braucht sich nur um Zahlenwerte zu kümmern:

```
Sub BenutzerFragen()
    Dim intAntwort As Integer
    intAntwort = MsgBox(„Gefällt Ihnen das?“, 7
                                          vbYesNo)
    If intAntwort = 6 Then
        Msgbox „Ja angeklickt!“
    End If
End Sub
```

Die zweite Meldung ist nun entfallen und dafür erscheint nur dann eine Folgemeldung, wenn **Ja** angeklickt wurde (s. Bild 3). Das Programm entscheidet sich also je nach dem Wert von **intAntwort** dafür, eine Meldung auszugeben oder auch nicht.

Damit sehen Sie auch die Grundkonstruktion von **If... Then**, die zwischen diesen beiden Schlüsselwörtern immer (mindestens) eine Bedingung stehen hat. Dann folgen beliebig viele Code-Zeilen (hier allerdings nur eine) und das Ende des Ausführungsblocks wird durch ein **End If** gekennzeichnet. Tatsächlich gibt es auch ein eher selten benutztes einzeiliges **If...Then** ohne **End If**. Die gleiche Bedingung sieht dann so aus:

```
If intAntwort = 6 Then Msgbox „Ja angeklickt!“
```

Wichtig dabei ist, dass die einzige ausführbare Code-Zeile direkt auf das **Then** folgt und nirgends ein **End If** steht. Das spart zwar ein paar Zeilen im VBA-Editor ein, macht Ihren Code aber nicht unbedingt übersichtlicher.

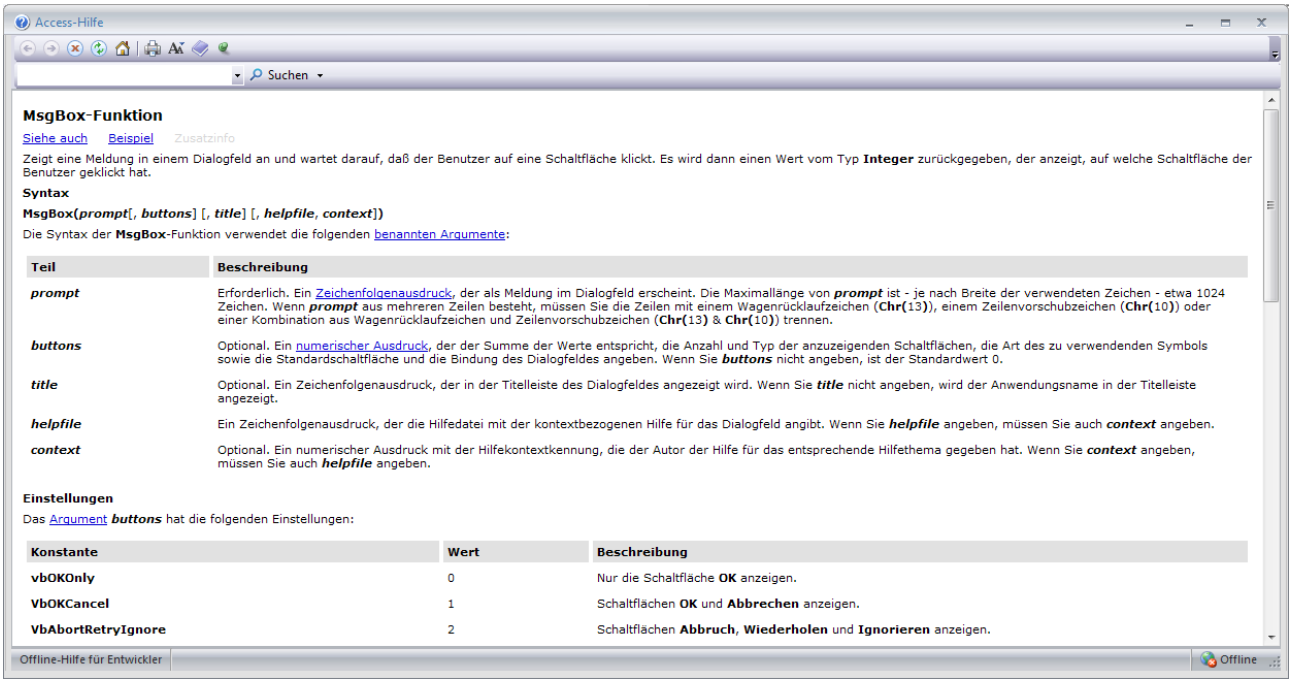


Bild 4: Die Hilfe zur MsgBox nennt die Konstanten

Für die Variante mit **End If** spricht auch, dass Sie damit auf die Fälle reagieren können, bei denen die in der Zeile **If...Then** angegebene Bedingung nicht erfüllt wird. Dazu bauen wir einen **Else**-Teil in die Prozedur ein:

```
Sub BenutzerFragen()
    Dim intAntwort As Integer
    intAntwort = MsgBox(„Gefällt Ihnen das?“, 7
                                        vbYesNo)
    If intAntwort = 6 Then
        Msgbox „Ja angeklickt!“
    Else
        MsgBox „Nein angeklickt!“
    End If
End Sub
```

Eine weitere Bedingung ist nicht nötig, denn bei zwei Schaltflächen werden hiermit alle Möglichkeiten abgedeckt.Trotzdem gibt es natürlich viele Fälle, wo mehr als zwei Entscheidungen zu treffen sind. Bevor wir uns denen widmen, möchten wir allerdings diesen Code noch verbessern. Nicht in seiner Funktionsfähigkeit, sondern in seiner Lesefreundlichkeit. Für die Zahlen, die durch das Anklicken der Schaltflächen entstehen, gibt es nämlich Konstanten:

```
Sub BenutzerFragen()
    Dim intAntwort As Integer
    intAntwort = MsgBox(„Gefällt Ihnen das?“, 7
                                        vbYesNo)
    If intAntwort = vbYes Then
        Msgbox „Ja angeklickt!“
    Else
        MsgBox „Nein angeklickt!“
    End If
End Sub
```

Der der VBA-Editor Ihnen diese Konstanten beim Programmieren nicht anbietet, müssen Sie diese nachlesen. Klicken Sie dazu einfach auf das Schlüsselwort **MsgBox** und drücken dann die **F1**-Taste, um die VBA-Hilfe zu öffnen. Dort finden Sie oben eine Tabelle für die möglichen Schaltflächen-Einstellungen und darunter für die Rückgabewerte (s. Bild 4). Damit müssen Sie nicht mehr raten, was denn nun eine **6** als Antwort bedeuten mag.

Mehr als zwei Entscheidungen

Damit es jetzt auch für mehrere Entscheidungen einfach bleibt, wechseln wir von einer banalen Meldung zu einem minimalen Eingabedialog. Statt **MsgBox** steht dort nun **InputBox**, womit der Benutzer einen beliebigen Text eingeben darf:

```
Sub BenutzerMehrFragen()
    Dim intAntwort As Integer
    intAntwort = InputBox(„Welche Zahl?“)
    If intAntwort = 1 Then
        MsgBox „Eins“
    ElseIf intAntwort = 2 Then
```

Access [basics]
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

Abfragen für die Datenauswahl

Teil III: Filtern von Textfeldern

Im vorliegenden und den folgenden Teilen dieser Artikelserie zum Thema Abfragen beschäftigen wir uns mit den verschiedenen Möglichkeiten zum Auswählen von Daten über verschiedene Kriterien. Den Start machen einfache Textkriterien.

Sie erinnern sich noch an den Chef unserer kleinen Beispielfirma Schwuppdiwupp? Der saß doch am Wochenende immer am Computer und hat die Daten der verschiedenen Mitarbeiter zusammengeführt. Inzwischen hat ihm sein neuer Mitarbeiter die Daten ja in Access-Tabellen zusammengefasst, aber so richtig glücklich ist er noch nicht. Jetzt will er doch wieder einzelne Daten sehen...

Beispielsweise konnte er früher einfach in der jeweiligen Excel-Tabelle nachgucken, welche Projekte Tabea Neukirch bearbeitet hat. Jetzt stehen alle Mitarbeiter zusammen in der gleichen Tabelle und er muss die erst wieder herausfiltern. Dabei ist das gar nicht so schwierig: in 03/2010 haben wir schon gezeigt, wie so eine Auswahlabfrage aussieht. Es geht aber noch besser.

Basisabfrage erstellen

Erstellen Sie also zuerst einmal eine neue Abfrage mit den drei Tabellen **tblProjekte**, **tblMitarbeiterProjekte** und **tblMitarbeiter**. Aus diesen fügen Sie per Doppelklick die Felder **Projektbezeichnung**, **Vorname** und **Nachname** zum Entwurfsraster der Abfrage hinzu.

Auch wenn Sie aus der Tabelle **tblMitarbeiterProjekte** selber gar kein Feld brauchen, erinnern Sie sich sicher, dass diese als Verbindung zwischen den beiden anderen Tabellen notwendig ist (siehe auch **Tabellen entwerfen - Teil III: Mitarbeiter per m:n-Beziehung zu Projektteams zusammenstellen**).

Was Sie in diesem Artikel lernen

» Textfelder nach einfachen Zeichenketten filtern
» Platzhalter für das Filtern einsetzen
» Und- und Oder-Verknüpfungen verwenden

Abfragen im Griff

Diese Ausgabe von Access [basics] stellt einerseits das Thema Abfrage und andererseits eine Lösung zum Verwalten von m:n-Beziehungen mit Listenfeldern in den Mittelpunkt. Nebenher lernen Sie noch den Umgang mit VBA-Variablen kennen.

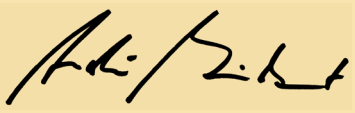
Doch zunächst zu den Abfragen: Zu diesem Thema finden Sie gleich drei Artikel. Im ersten zeigen wir Ihnen, wie Sie die Datensätze der Abfrage nach dem Inhalt von Textfeldern filten können. Es geht allerdings nicht nur um den einfachern Vergleich mit Zeichenketten, denn wir nehmen auch Spezialitäten wie die Platzhalter unter die Lupe. Außerdem erfahren Sie, wie Sie verschiedene Kriterien mit Operatoren wie OR oder AND verknüpfen können.

Der zweite Artikel zum Thema Abfragen beschäftigt sich mit Abfrageparametern: Sie können eine Abfrage nämlich so anlegen, dass der Benutzer beim Ausführen selbst die Kriterien festlegen kann – ohne selbst den Entwurf zu ändern! Und schließlich schauen wir uns an, wie Sie Daten nach Zahlenfeldern filtern können – einen kleinen Abstecher in die Welt der Unterabfragen inklusive.

Wer mit VBA programmiert, kommt selten ohne den Einsatz von Variablen aus. Dabei handelt es sich um Platzhalter für Werte, die während des Ablaufs einer VBA-Routine gefüllt und auch wieder ausgelesen werden können. Alles zu diesem Thema finden Sie hier.

Und schließlich wollen wir am Beispiel der Projektzeiterfassung zeigen, wie Sie die Werte aus per m:n-Beziehung verknüpften Tabellen mit Hilfe eines einfachen Formulars und zweier Listenfelder verwalten können.

Viel Spaß beim Lesen!



[André Minhorst]

In dieser Ausgabe:

Abfragen für die Datenauswahl - Teil III: Filtern von Textfeldern

Abfragen für die Datenauswahl – Teil IV: Einsatz von Parametern

Abfragen für die Datenauswahl – Teil V: Filtern nach Zahlen

VBA-Programmierung – Teil II: Variablen in VBA

Formulare für die Datenbearbeitung – Teil V: m:n-Beziehungen in Listenfeldern darstellen

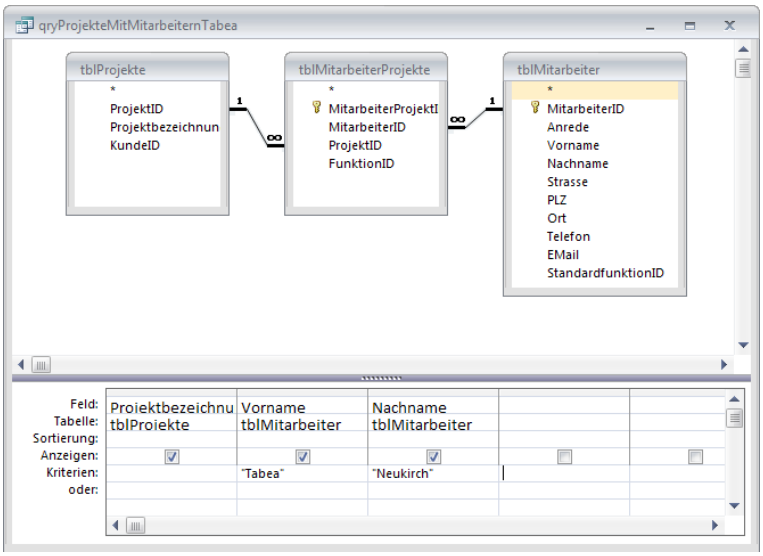


Bild 1: Der Entwurf der Abfrage, um die Projekte bestimmter Mitarbeiter zu finden

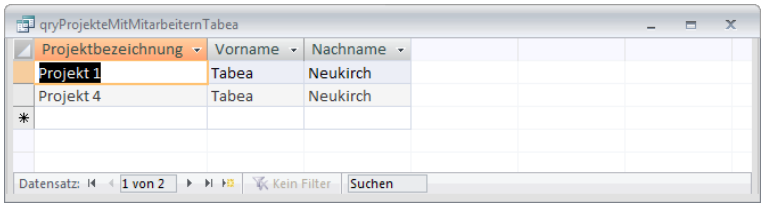


Bild 2: Das Abfrage-Ergebnis für Tabea Neukirch

Kriterien definieren

Jetzt tragen Sie nur noch den Wert **Tabea** als Kriterium in die Spalte **Vorname** und **Neukirch** in die Spalte **Nachname** ein und die Abfrage ist fertig. Sie können diese direkt als **qryProjekteMitMitarbeiternTabea** speichern (s. Bild 1).

Access erkennt automatisch, dass es sich bei beiden Feldern um Textfelder handelt. Daher werden die Kriterien beim Verlassen der Zelle oder beim Speichern der Abfrage automatisch in Gänsefüßchen gesetzt, damit es syntaktisch korrekt ist.

Wenn der Chef nun diese Abfrage ausführt, sieht er tatsächlich die beiden Projekte, an denen die Mitarbeiterin arbeitet (s. Bild 2).

Es ist wichtig, dass Sie sowohl den Vor- als auch den Nachnamen in den Filterkriterien angeben. Lassen Sie das Kriterium **Tabea** für **Vorname** weg, sehen Sie insgesamt vier Ergebnisse (s. Bild 3).

Da auch Tabeas Bruder Erich in der Firma arbeitet, werden nun die Projekte der beiden angezeigt. Da der Nachname eben nicht eindeutig ist, müssen Sie mit dem Vornamen ein weiteres Kriterium hinzufügen.

Es ist übrigens wichtig, dass der Vorname als zweites Kriterium in der gleichen Zeile steht. Nur dann liefert die Abfrage alle Datensätze, die beide Kriterien gleichzeitig erfüllen.

Und-Verknüpfung

Das nennt sich in mathematischer Logik eine **UND**-Verknüpfung: Beide Kriterien müssen gleichzeitig erfüllt werden. Leider spielt uns da unser täglicher Umgang mit der Sprache einen Streich. Wenn Frau Köhler aus unserer Beispielfirma morgens in der Küche erzählt, dass Sie am Wochenende in einem Konzert mit Musik von Bach und Bartók gewesen war, dann mag sie das für einen unvergesslichen Abend halten.

Ein Logiker hingegen wundert sich, denn Béla Bartók wurde erst rund 200 Jahre nach Johann Sebastian Bach geboren. Wie können die beiden zusammen Musik komponiert haben? Oder wurden im Konzert die Stücke der beiden Komponisten gleichzeitig gespielt? Das muss ja furchtbar geklungen haben!

Oder-Verknüpfung

Selbstverständlich meinte Frau Köhler nicht, dass die Musik von Bach und Bartók war, sondern dass es entweder ein Stück von Bach oder von Bartók war. Das sprachliche „und" ist im logischen Sinne in den meisten Fällen ein ODER. Warum das hier so wichtig ist? Na, dann suchen Sie doch mal nach den Projekten der Kollegen **Fischer** und **Schneider**!

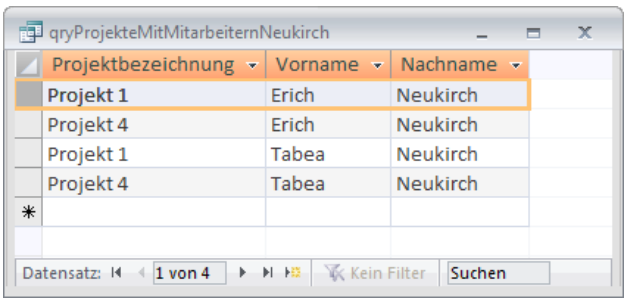


Bild 3: Das Abfrage-Ergebnis nur für Neukirch

Abfragen für die Datenauswahl

Teil IV: Einsatz von Parametern

Der Aufbau von Abfragen ist gar nicht so schwer – zumindest wenn Sie sich auf Abfragen mit einer überschaubaren Anzahl verknüpfter Abfragen beschränken. Allein eines nervt: Wenn Sie mal die Kriterien für die Auswahl der gesuchten Datensätze ändern möchten, müssen Sie jedesmal in den Abfrageentwurf wechseln, dort die gewünschten Einstellungen vornehmen und wieder zur Datenblattansicht zurückkehren. Damit dies etwas einfacher wird, stellt Ihnen dieser Artikel Abfrage-Parameter vor.

Finden Sie es praktisch, durch ständige Anpassung der Kriterien nach den Mitarbeitern zu suchen? Wenn das so bliebe, müsste der Chef jedes Mal in die Entwurfsansicht wechseln, dort den Namen eintragen und erst dann diese Abfrage ausführen.

Alternativ kommen viele Access-Nutzer jetzt auf die Idee, für jeden Mitarbeiter eine Abfrage zu speichern, also **qryXYZ_Fischer**, **qryXYZ_TabeaNeukirch**, **qryXYZ_ErichNeukirch** und so weiter.

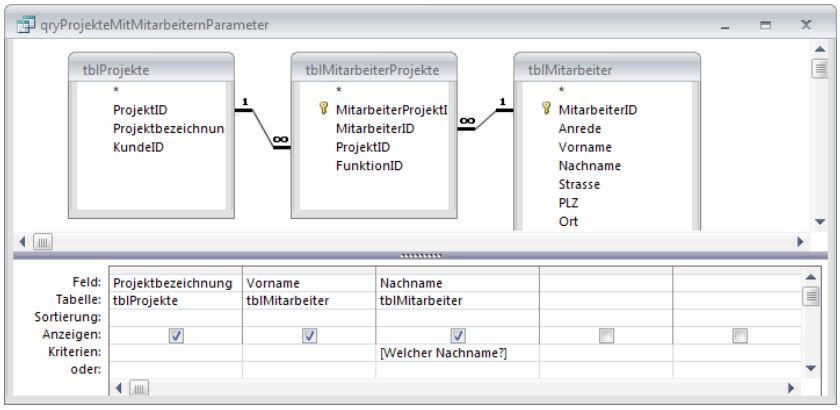


Bild 1: So geben Sie einen Parameter an.

Das heißt, dass Sie für jeden Mitarbeiter eine eigene Abfrage anlegen müssten – und das für jede gewünschte Information.

Vom Platzbedarf in der Access-Datenbank her ist das relativ unproblematisch. Abfragen brauchen nur wenig Speicher, denn sie speichern nur die Frage (Sie werden beim Stichwort **SQL** sehen, das es sich nur um einen schlichten Text handelt) und nicht die Antwort (die aus Millionen Datensätzen bestehen könnte).

Manch ein Entwickler nutzt dies ohne Rücksicht aus und erstellt Abfragen nach diesem Muster: **qryUmsatzJanuar**, **qryUmsatzFebruar** bis **qryUmsatzDezember**.

Aber wo bleibt da die Übersicht für Sie als Nutzer der Datenbank? Sie müssten mit zehn Mal so vielen Abfragen zurechtkommen und für jeden neuen Mitarbeiter außerdem eine neue Abfrage anlegen.

Was Sie in diesem Artikel lernen

» Wozu brauche ich Parameter-Abfragen?
» Wie formulare ich eine Parameter-Abfrage?
» Wie kann ich mit einer Parameter-Abfrage viele gleichartige Abfragen ersetzen?

Parameter-Abfrage

Es gibt in Access einen besonderen Typ Auswahl-Abfragen, dessen Kriterien flexibel sind: die Parameter-Abfragen. Sie fragen bei jeder Ausführung immer nach, welches Kriterium dieses Mal berücksichtigt werden soll. Das ist genau für den Fall gedacht, dass Sie bei jedem neuen Aufruf der Abfrage die Daten zu einem anderen Mitarbeiter in Erfahrung bringen möchten.

Bauen wir doch gleich eine erste Parameter-Abfrage. Dazu kopieren Sie die bereits erstellte Abfrage **qryProjekteMitMitarbeiternFischerUndSchneider** und geben der Kopie den Namen **qryProjekteMitMitarbeiternParameter**.

Dies erledigen Sie am schnellsten, indem Sie **qryProjekteMitMitarbeiternFischerUndSchneider** im Datenbankfenster (Access 2003 und älter) beziehungsweise im Navigationsbereich (Access 2007 und jünger) markieren und nacheinander die Tastenkombinationen **Strg + C** und **Strg + V** betätigen. Access fragt dann nach dem Namen für die Kopie des Objekts.

Öffnen Sie die neue Abfrage und löschen Sie alle bisherigen Kriterien. Dort, wo bisher der Text **„Fi-**

Abfragen für die Datenauswahl

Teil V: Filtern nach Zahlen

Viele Auswertungen basieren auf Zahlenkriterien. Welche Artikel kosten mehr als 100,- Euro? Welcher Kunde hat den größten Umsatz generiert? Welcher Artikelbestand erfordert eine Nachbestellung? Aber auch: Welche Bestellungen gehören zum Kunden mit der Kundennummer 1234? Dieser Artikel zeigt, wie Sie Daten mit Zahlenkriterien filtern.

Da die Projektzeiterfassung als eigentliche Beispieldatenbank von Access [basics] zum aktuellen Zeitpunkt keine umfassenden Zahlenfelder bietet, die wir nach Herzenslust für das Durchführen von Beispielen heranziehen können, weichen wir im Rahmen dieses Artikels auf die Südsturm-Datenbank aus. Südsturm-Datenbank? War da nicht was? Gibt es da nicht eine Beispieldatenbank von Microsoft namens Nordwind? Genau: Und eben dieser haben wir die Tabellen entnommen.

Und nicht nur das: Wir haben auch noch einige Anpassungen durchgeführt, damit die Tabellen- und Feldnamen den unter →*[basics]* **Konventionen** beschriebenen Regeln entsprechen.

Doch zur Sache: Wir beginnen mit der Tabelle **tblArtikel**, die einige interessante Zahlenfelder enthält. Da wäre zum Beispiel der Einzelpreis, der Bestand, die bestellten Einheiten oder der Lagerbestand.

Feld mit bestimmtem Zahlenwert

Die einfachsten Abfragen mit Zahlenkriterien enthalten nur den Vergleichswert und liefern so alle Datensätze, bei denen etwa das Feld **Lagerbestand** den Wert **0** aufweist.

Dazu legen Sie eine neue Abfrage mit der Tabelle **tblArtikel** an und fügen die Felder **ArtikelID**, **Artikelname** und **Lagerbestand** zum Entwurfsraster hinzu (siehe →*[basics]* **Abfrage erstellen**) und speichern diese unter dem Namen **qryLagerbestandIstNull**. Tragen Sie in der Zeile **Kriterien** den Wert **0** für das Feld **Lagerbestand** ein (siehe Bild 1). Das Ergebnis liefert genau die gewünschten Datensätze (siehe Bild 2).

Was Sie in diesem Artikel lernen

» Einsatz von Kriterien für Zahlenfelder
» Verwendung von Operatoren wie AND, OR, BETWEEN oder IN
» Kombination mehrerer Kriterien in Abfragen
» Einsatz des Schlüsselworts NULL

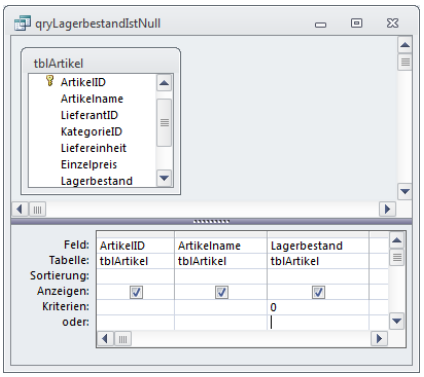


Bild 1: Abfrage aller Datensätze, deren Lagerbestand den Wert 0 hat

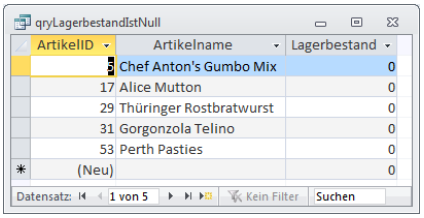


Bild 2: Ergebnis der Abfrage aller ausverkauften Datensätze

Nur Zahlen!

Wenn Sie Vergleichswerte für Zahlenfelder angeben, dürfen Sie ebenfalls nur Zahlen verwenden (abgesehen von den nachfolgend vorgestellten Operatoren). Sie dürfen keine Texte wie beispielsweise **Euro** oder **%** zusätzlich zum Zahlenwert eingeben – Access meldet dann einen Fehler.

Größer, kleiner, gleich

Dieses Kriterium entspricht prinzipiell dem Ausdruck **=0**. Diesen können Sie alternativ eingeben. Von dort aus kommen wir schnell auf die übrigen möglichen Operatoren für das Filtern von Zahlenwerten:

- Größer (**>**)
- Größer gleich (**>=**)
- Kleiner (**<**)
- Kleiner gleich (**<=**)

Statt =0 können Sie also beispielsweise Werte wie >0, >=10 oder <100 eingeben.

Ungleich

Für das Finden von Datensätzen, deren Feldwert ungleich dem betreffenden Wert ist, haben Sie zwei Möglichkeiten: Sie können erstens die Kombination aus größer (**>**) und kleiner (**<**) verwenden, also **<>**, aber auch den Operator **NOT** einsetzen. Dieser ist übrigens mit beliebigen Ausdrücken kombinierbar.

Wenn Sie alle Artikel finden möchten, deren Lagerbestand ungleich **10** ist, verwenden Sie also entweder das Kriterium **<>10** oder **NOT 10**. Lassen Sie sich nicht täuschen, wenn Sie **NOT 10** als Kriterium eintragen und Access dies gleich nach dem Verlassen des Feldes in **Nicht 10** umwandelt (siehe Bild 4): Dies ist eine Eigenschaft der deutschsprachigen Access-Version.

Im Hintergrund speichert Access ohnehin den SQL-Ausdruck, und zwar mit englischen Schlüsselwörtern. Zum Thema SQL kommen wir zu einem späteren Zeitpunkt.

Zwischen zwei Werten

Vielleicht möchten Sie auch einmal alle Artikel ermitteln, deren Preis zwischen 10,- Euro und 20,- Euro liegt. Dann haben Sie ebenfalls zwei Möglichkeiten:

- Sie verwenden den **AND**-Operator, um die Kriterien **>=10** und **<=20** zu verknüpfen. Der Abfrage-Editor macht dann **>=10 Und <=20** daraus (s. Bild 5).

- Oder Sie verwenden direkt den für solche Fälle vorgesehen **BETWEEN ... AND ...**-Operator. Dazu geben Sie den Ausdruck **BETWEEN 10 AND 20** als Kriterium, was umgehend in **Zwischen 10 Und 20** umgewandelt wird (s. Bild 6).

Englische und deutsche Operatoren

Warum führen wir immer die englischen Schlüsselwörter an, wenn der Abfrage-Editor diese doch in die deutschen Pendants umwandelt – zumindest in der deutschen Version von Access? Ganz einfach: Access speichert die Ausdrücke intern in der englischen Version. Und das ist auch die Version, die Sie später kennenlernen, wenn Sie einmal manuell Abfrageausdrücke zusammenstellen. Das kommt zwar kaum vor, weil Sie immer den Abfrage-Designer zu Hilfe nehmen können, aber es gibt auch Ausnahmen.

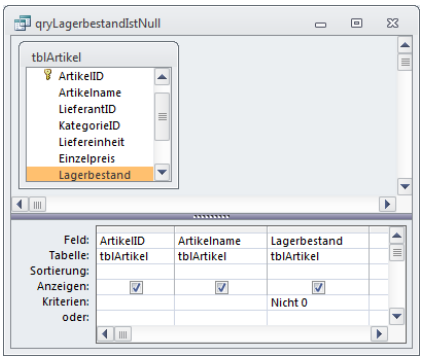


Bild 3: Schlüsselwörter wie **NOT** werden im Abfrageentwurf der deutschen Access-Version eingedeutscht.

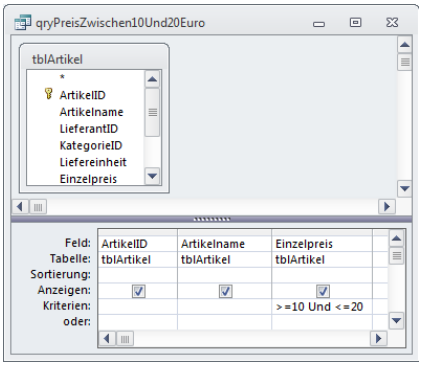


Bild 4: Abfragen eines Bereiches von Werten

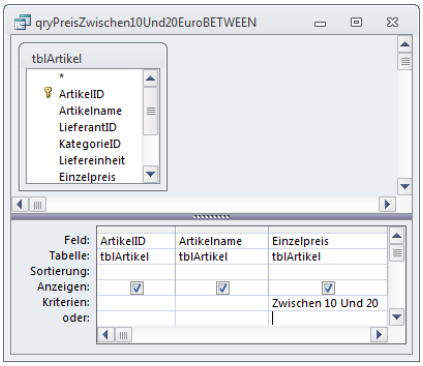


Bild 5: Bereichsabfrage, diesmal mit BETWEEN

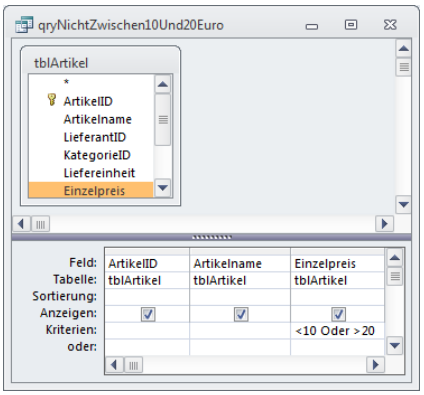


Bild 6: Artikel, die weniger als 10,- Euro oder mehr als 20,- Euro kosten

Werte außerhalb eines Bereichs

Vielleicht möchten Sie auch einmal alle Artikel auflisten, die gerade nicht im Preissegment zwischen 10,- Euro und 20,- Euro liegen. Dann hilft der **AND**-Operator nicht weiter: Er liefert nur Datensätze, für die alle mit **Und** verknüpften Kriterien gelten. Hier kommt der **OR**-Operator ins Spiel. Der Ausdruck **<10**

VBA-Programmierung

Teil II: Variablen in VBA

Eine der wichtigsten Techniken in Programmiersprachen besteht darin, dass Sie sich veränderliche Werte zur Laufzeit merken können. Dazu dienen Variablen, in denen der Inhalt vorübergehend gespeichert wird.

VBA bietet die Möglichkeit zum Speichern von Werten wie beispielsweise von Berechnungsergebnissen in sogenannten Variablen. Eine Variable ist im Prinzip ein Aufbewahrungsort für einen Wert. Bevor Sie eine Variable in einer VBA-Routine verwenden, sollten Sie diese einen aussagekräftigen Namen geben. Im gleichen Zuge legen Sie die Art der darin gespeicherten Daten fest. Es gibt beispielsweise Variablentypen zum Speichern von Text, Zahlen oder Datumsangaben. Das Benennen und das Zuweisen des Datentyps nennt sich Deklarieren und sieht beispielsweise so aus:

```
Dim intZahl As Integer
```

Diese Anweisung deklariert eine Variable namens **intZahl** und weist ihr den Datentyp Integer zu. Der Variablenname setzt sich aus einem Präfix (**int**) und der Beschreibung des Inhalts (**Zahl**) zusammen.

Durch den Einsatz des Präfix können Sie beim Programmieren gleich erkennen, welchen Datentyp eine Variable besitzt und brauchen nicht erst die Deklarationszeile zu konsultieren.

Hinter dem Präfix steht eine kurze Beschreibung des Werts, den die Variable enthalten soll (hier schlicht **Zahl**). Die Regeln zur Benennung von Variablen haben wir unter →*[basics]* **Konventionen** zusammengefasst.

Wenn Sie eine Variable deklariert haben, können Sie diese Werte zuweisen und über die Variable auf den gespeicherten Wert zugreifen:

```
Dim intZahl As Integer
intZahl = 100
Debug.Print intZahl
```

Was Sie in diesem Artikel lernen

» Welche Basisdatentypen stellt VBA bereit?
» Wie speichere ich einen Wert in einer Variablen und rufen ihn wieder ab?
» Welchen Wertebereich haben die verschiedenen Datentypen?
» Was ist der Unterschied zwischen Gleit- und Festkommazahlen?

Wenn Sie die drei Zeilen in eine Sub-Prozedur einfügen und diese ausführen (→*[basics]* **VBA-Prozedur ausführen**), erscheint im Direktfenster der Wert **100**.

Variablendeklaration erzwingen

Wenn Sie Variablen auf die oben beschriebene Art deklarieren, könnten einfache Tippfehler dazu führen, dass die Routine nicht mehr wie erwartet funktioniert. Wenn die dritte Zeile beispielsweise wie folgt lautet, gibt die Prozedur gar keinen Wert mehr im Direktfenste raus:

```
Debug.Print ingZahl
```

Diese Zeile greift schlicht auf eine zuvor nicht gefüllte Variable zu. Solche Fehler vermeiden Sie auf einfache Weise. Schreiben Sie einfach die folgende Zeile in den Modulkopf:

```
Option Explicit
```

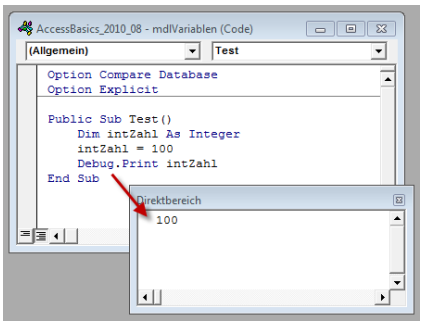


Bild 1: Ausgabe des in einer Variablen gespeicherten Wertes im Direktfenster

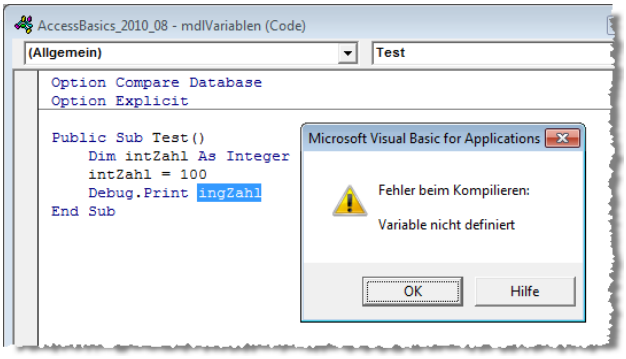


Bild 2: Fehlermeldung beim Verwenden einer nicht deklarierten Variablen

Der VBA-Editor prüft nun vor dem Ausführen der Routine, ob alle verwendeten Variablen auch deklariert wurden.

Dies ist bei **ingZahl** nicht der Fall (deklariert wurde ja nur **intZahl**). Der VBA-Editor liefert daraufhin eine entsprechende Fehlermeldung und markiert die betroffene Zeile (s. Bild 2).

Diese Option gilt nicht anwendungsweit, sondern nur in den Modulen, in denen diese enthalten ist. Wenn die Option nicht in allen Modulendes VBA-Projekts einer Access-Anwendung aktiv ist, sollten Sie dies manuell nachholen.

Für neue Module können Sie dies automatisieren: Dazu stellen Sie einfach eine Option des VBA-Editors ein. Klicken Sie im VBA-Editor auf den Menüeintrag **Extras|Optionen** und aktivieren Sie im nun erscheinenden Dialog die Option **Variablendeklaration erforderlich** (s. Bild 3). Fortan wird die Zeile **Option Explicit** automatisch in alle neuen Module integriert.

Deklaration mehrere Variablen in einer Zeile

Aus Gründen der Übersicht sollten Sie jede Variable in einer eigenen Zeile deklarieren, also zum Beispiel so:

```
Dim i As Integer
Dim strVorname As String
Dim datStart as Date
```

Sie können allerdings auch mehrere Deklarationen in eine Zeile schreiben:

```
Dim i As Integer, j As Integer
```

Beachten Sie, dass Sie zu jeder Variablen den Datentyp angeben müssen – anderenfalls nimmt VBA automatisch den Variablentyp **Variant** an.

Dies ist zwar nicht unbedingt fehlerhaft, aber Variablen dieses Datentyps nimmt den meisten Speicherplatz ein (mehr dazu weiter unten).

Basisdatentypen

Die Programmiersprache VBA kennt einige Basisdatentypen. Die meisten davon speichern Zahlen, aber Sie können auch Zeichenketten in Variablen zwischenlagern.

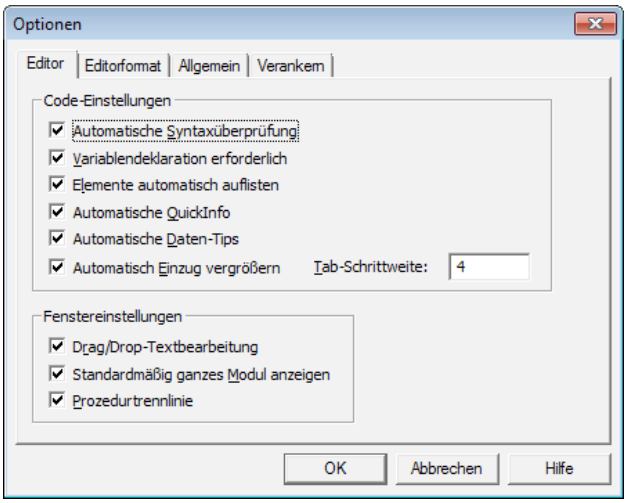


Bild 3: Aktivieren des Kontrollkästchens **Variablendeklaration erforderlich**

Warum gibt es mehrere Datenzahlentypen? Nun: Es gibt unterschiedlich große Zahlen mit und ohne Nachkommastellen. VBA stellt für die verschiedenen Fälle unterschiedliche Datentypen bereit.

Ganzzahlen

Bei den Zahlen gibt es zunächst die folgenden Ganzzahlentypen:

- **Byte**: Ganzzahlen von 0 bis 255 (ein Byte Speicherbedarf)
- **Integer**: Ganzzahlen von -32.768 bis 32.767 (zwei Bytes Speicherbedarf)
- **Long**: Ganzzahlen von -2.147.483.648 bis 2.147.483.647 (vier Bytes Speicherbedarf)

Achtung: Wenn Sie einer Ganzzahl-Variablen eine Zahl mit Nachkommastellen zuweisen, werden diese kommentarlos abgeschnitten. Wenn Sie einer Zahl-Variablen einen Wert zuweisen, der außerhalt des angegebenen Wertebereichs liegt, löst dies einen Fehler aus. Das wäre zum Beispiel hier der Fall – der Wertebereich für den Datentyp **Integer** reicht nur bis 32.767:

```
Dim intZuGross As Integer
intZuGross = 32768
```

Gleitkommazahlen

Daneben gibt es zwei Gleitkommazahlentypen:

- **Single**: Gleitkommazahlen von -3,402823E38 bis -1,401298E-45 für negative Werte und 1,401298E-

Formulare für die Datenbearbeitung

Teil V: m:n-Beziehungen in Listenfeldern darstellen

m:n-Beziehungen kann man auf verschiedene Arten abbilden. Die Variante mit Haupt- und Unterformular haben Sie bereits kennengelernt. In der Variante dieses Artikels kommen zwei Listenfelder zum Einsatz. Das erste enthält alle Datensätze, die mit dem im Formular angezeigten Datensatz verknüpft sind und das zweite alle übrigen. Das Hinzufügen und Entfernen verknüpfter Datensätze erledigen wir elegant per Schaltfläche oder per Doppelklick.

Das Beispiel zu diesem Artikel soll mal wieder unsere Projektzeiterfassung in den Mittelpunkt stellen. Wenn mal ein Kunde anruft und eine Frage zu einem speziellen Thema hat, soll er bei der Firma Schwuppdiwupp gleich mit einem kompetenten Kollegen verbunden werden. Dummerweise weiß aber nicht jeder, welcher Mitarbeiter sich mit welchen Themen beschäftigt. Was liegt also näher, als in der Datenbank Informationen über die Fähigkeiten (neudeutsch Skills) der einzelnen Mitarbeiter zusammeln?

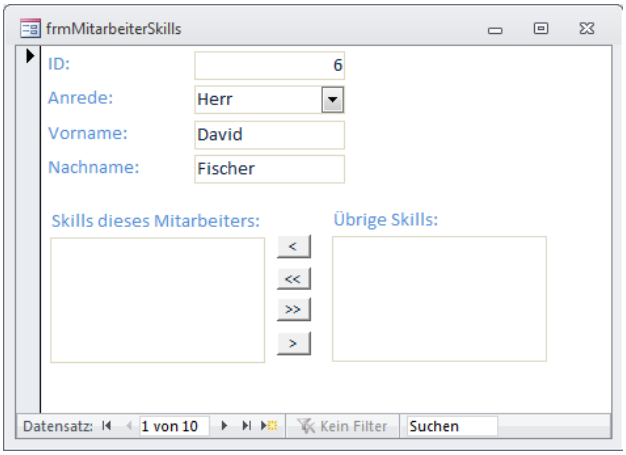


Bild 1: Hinzufügen und Entfernen der Skills eines Mitarbeiters

Im Endergebnis soll dies wie in Bild 1 aussehen. Ein Doppelklick auf einen Eintrag im linken Listenfeld soll diesen ins rechte Listenfeld verschieben und umgekehrt. Mit den Schaltflächen mit dem Größer- und dem Kleiner-Zeichen lässt sich der jeweils markierte Eintrag verschieben. Die doppelten Größer/Kleiner-Zeichen verschieben gleich alle Einträge zum jeweils anderen Listenfeld.

Datenmodell

Für dieses Beispiel benötigen Sie zwei neue Tabellen. Die erste heißt **tblSkills** und speichert die einzelnen Skills. Sie enthält ein Primärschlüsselfeld namens **SkillID** und ein Textfeld namens **Skill** zum Eintragen der Skills.

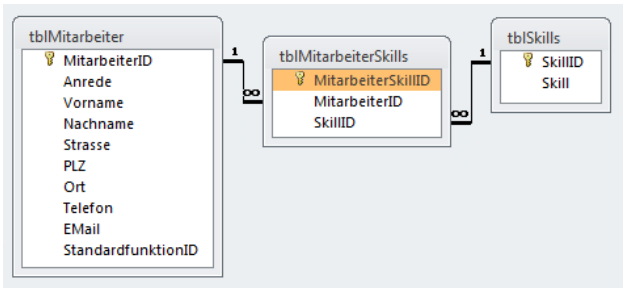


Bild 2: Mitarbeiter und ihre Skills im Datenmodell

Die Zuordnung eines Skills zu einem Mitarbeiter erfolgt über die m:n-Verknüpfungstabelle **tblMitarbeiterSkills**. Die Tabelle enthält das Primärschlüsselfeld **MitarbeiterSkillID** sowie die beiden Felder **MitarbeiterID** und **SkillID**. Die beiden Felder sind mit den jeweili-

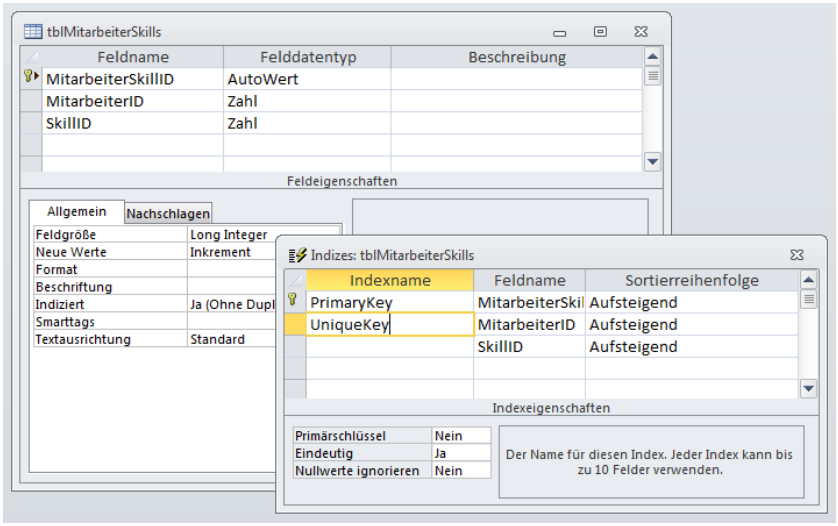


Bild 3: Festlegen eines zusammengesetzten eindeutigen Schlüssels für die Tabelle tblMitarbeiterSkills

Was Sie in diesem Artikel lernen

» m:n-Beziehungen mit Listenfeldern verwalten
» Daten zu einer m:n-Beziehung hinzufügen
» Daten einer m:n-Beziehung löschen
» Listenfelder mit Inhalten füllen

gen Primärschlüsselfeldern der Tabellen **tblMitarbeiter** und **tblSkills** verknüpft (s. Bild 2).

Damit die Benutzer der Datenbank keinem Mitarbeiter den gleichen Skill zweimal zuweisen können, legen Sie einen zusammengesetzten eindeutigen Index für die beiden Felder **MitarbeiterID** und **SkillID** der Tabelle **tblMitarbeiterSkills** fest (s. Bild 3).

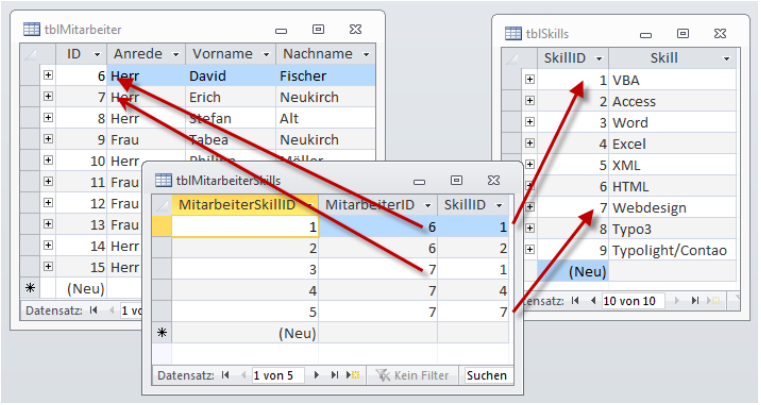


Bild 4: Zuordnen von Skills zu Mitarbeitern per Verknüpfungstabelle

Wenn Sie die Tabellen angelegt haben, geben Sie wie in Bild 4 ein paar Datensätze in die Tabelle **tblSkills** und **tblMitarbeiterSkills** ein – dies erleichtert die Entwicklung auf Basis des Formulars mit den beiden Listenfeldern.

Basisformular

Das Formular bauen Sie wie in →*[basics]* **Gebundenes Formular anlegen** beschrieben auf. Dabei verwenden Sie die Tabelle **tblMitarbeiter** als Datenherkunft und ziehen die wichtigsten Felder, hier beispielsweise die **MitarbeiterID**, **AnredeID**, **Vorname** und **Nachname** in den Detailbereich. Speichern Sie das Formular unter dem Namen **frmMitarbeiterSkills**.

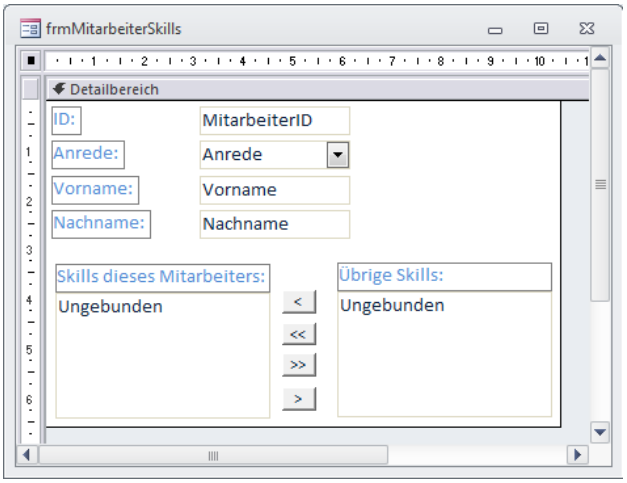


Bild 5: Entwurfsansicht des Formulars mit allen Steuerelementen

Damit ist die m-Seite der Beziehung bereits erledigt, aber das ist bei weitem nicht der aufwendigste Teil der Aufgabe. Nun folgen noch die beiden Listenfelder, die Schaltflächen zum Hin- und Herschieben der Skills zwischen den Listenfeldern und die notwendigen VBA-Routinen.

Fügen Sie also zunächst zwei Listenfelder zum Formular hinzu und nennen Sie diese **lstZugeordnet** und **lstNichtZugeordnet**. Dazwischen legen Sie gleich vier Schaltflächen an, die folgende Namen erhalten: **cmdEinzelnHinzufuegen**, **cmdAlleHinzufuegen**, **cmdAlleEntfernen** und **cmdEinzelnEntfernen**. Als Beschriftung verwenden Sie der Einfachheit halber entsprechende Größer/Kleiner-Zeichen (**<**, **<<**, **>>** und **>**). Wie Bild 5 zeigt, ist das Formular nun bereits mit allen notwendigen Steuerelementen ausgestattet.

Listenfeld mit zugeordneten Skills

Nun weisen Sie dem Listenfeld der für den aktuellen Mitarbeiter ausgewählten Skills eine entsprechende Datensatzherkunft zu. Die Besonderheit dieser Abfrage ist, dass sie einen Wert des aktuell im Formular angezeigten Datensatzes als Kriterium verwenden soll. Das Listenfeld soll schließlich nur die Skills für den aktuell im Formular sichtbaren Mitarbeiter anzeigen. Wie muss die Datenherkunft nun aussehen? Sie soll erstens die Skills anzeigen, also brauchen wir zumindest die Tabelle **tblSkills**. Um aber nur die Skills des aktuellen Mitarbeiters zu ermitteln, brauchen wir auch noch die Tabelle **tblMitarbeiterSkills**, welche die Skills über das Fremdschlüsselfeld **MitarbeiterID** mit dem jeweiligen Mitarbeiter verbindet.

Die Tabelle **tblMitarbeiter** selbst brauchen wir nicht: Wir wollen im Listenfeld ja keine der eigentlichen Mitarbeiterdaten anzeigen, sondern nur die über die Tabelle **tblMitarbeiterSkills** verknüpften Einträge der Tabelle **tblSkills**.

Erstellen Sie also eine neue Abfrage und fügen Sie die beiden Tabellen **tblSkills** und **tblMitarbeiterSkills** zur Abfrage hinzu (siehe →*[basics]* **Abfrage erstellen**). Ziehen Sie die beiden Felder **SkillID** und **Skill** der Tabelle **tblSkills** und das Feld **MitarbeiterID** der

Tabelle **tblMitarbeiterSkills** in das Entwurfsraster. Speichern Sie die Abfrage dann zunächst unter dem Namen **qrySkillsEinesMitarbeiters**. Dieser Name ist prinzipiell noch nicht korrekt: Die Abfrage zeigt nun nämlich alle Skills aller Mitarbeiter an. Damit nur die Skills des aktuell im Formular **frmMitarbeiterSkills** angezeigten Mitarbeiters anzeigt, müssen wir der Spalte **MitarbeiterID** der Abfrage ein Kriterium hinzufügen, das genau dem Wert von **MitarbeiterID** im Formular entspricht. Zeigt also das Formular den Mitarbeiter mit der **MitarbeiterID 4711** an, soll die Abfrage für das Feld **MitarbeiterID** als Parameter den Wert **4711** erhalten. Die Abfrage sähe im Entwurf dann wie in Bild 6 aus.

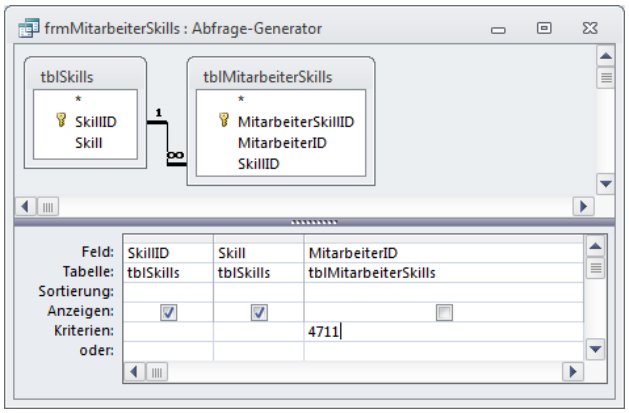


Bild 6: Datenherkunft des linken Listenfeldes mit vorläufigem Kriterium

Nun können Sie diesen Wert natürlich nicht fest dort eintragen. Die Abfrage muss diesen Wert in Abhängigkeit des Wertes im Formular dort einsetzen. Das erledigen Sie durch einen Verweis auf das entsprechende Formularfeld. In diesem Fall lautet dieser so:

```
Forms!frmMitarbeiterSkills!MitarbeiterID
```

Die einzelnen Bestandteile sind schnell erläutert: **Forms** verweist auf die Auflistung aller geöffneten Formulare. Wenn Sie auf ein Element einer Auflistung selbstgebauter Objekte, also beispielsweise das Formular **frmMitarbeiterSkills**, zugreifen möchten, hängen Sie dieses Element durch ein Ausrufezeichen getrennt an die Auflistung an. **frmMitarbeiterSkills** enthält wiederum benutzerdefinierte Objekte, in diesem Fall das Steuerelement **MitarbeiterID** – auch diese hängen Sie mit führendem Ausrufezeichen an den bestehenden Ausdruck an (weitere Informationen siehe →*[basics]* **Formularelemente referenzieren**).

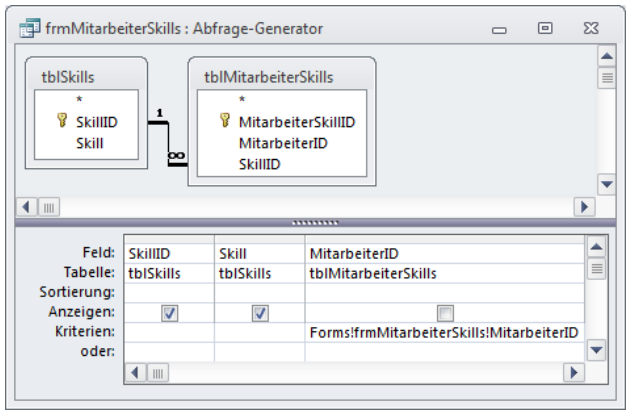


Bild 7: Endgültige Datenherkunft des Listenfeldes **lstAusgewaehlt**

Beachten Sie, dass wir den Haken in der Zeile **Anzeigen** für die Spalte **MitarbeiterID** entfernt haben – das Feld dient nur als Kriterium und soll nicht im Listenfeld angezeigt werden.

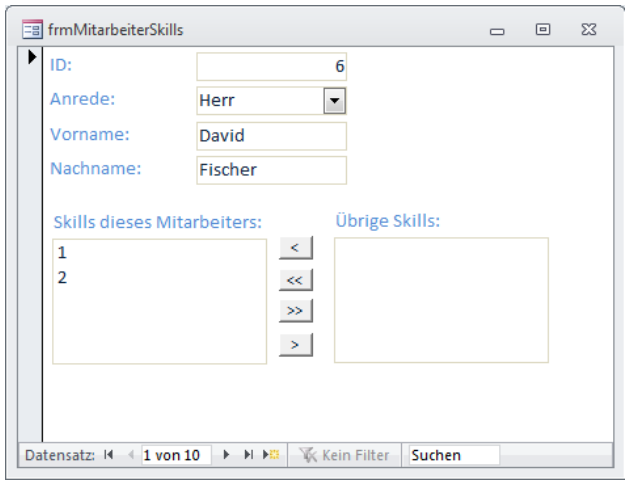


Bild 8: Das Listenfeld zeigt vorerst nur die Primärschlüsselwerte der verknüpften Skills an.

Auch das Feld **SkillID** soll dort eigentlich nicht erscheinen. Dennoch benötigen wir es: Wenn der Benutzer einen Eintrag eines Listenfeldes markiert und diesen zum anderen Listenfeld verschieben möchte, müssen wir diesen Datensatz schließlich eindeutig identifizieren können. Und das gelingt am besten mit dem Primärschlüsselfeld.

Listenfeld füllen

Damit das Listenfeld **lstAusgewaehlt** gefüllt wird, stellen Sie die Eigenschaft **Datensatzherkunft** des Listenfeldes auf den Namen der soeben erstellten Abfrage ein, also auf **qrySkillsEinesMitarbeiters**.

Ein Wechsel in die Formularansicht zeigt, dass wir das Listenfeld noch optimieren müssen (s. Bild 7). Da wir bereits einige Daten in die Tabellen **tblSkills**

und **tblMitarbeiterSkills** eingetragen haben, zeigt es zwar bereits einige Daten an, aber leider nur das erste Feld der Datensatzherkunft. Es soll aber das zweite Feld anzeigen. Dies ändern wir durch Ändern der beiden Eigenschaften **Spaltenanzahl** und **Spaltenbreiten** auf die Werte **2** und **0cm**. Ersteres sorgt dafür, dass theoretisch beide Felder im Listenfeld erscheinen, und Zweiteres blendet das erste der beiden Felder durch Einstellen der Breite auf **0cm** gleich wieder aus. Durch das Auslassen einer Breite für das zweite, letzte Feld nimmt dieses den kompletten verbleibenden Platz ein. Das Ergebnis sieht nun schon viel besser aus, wie Bild 9 zeigt.

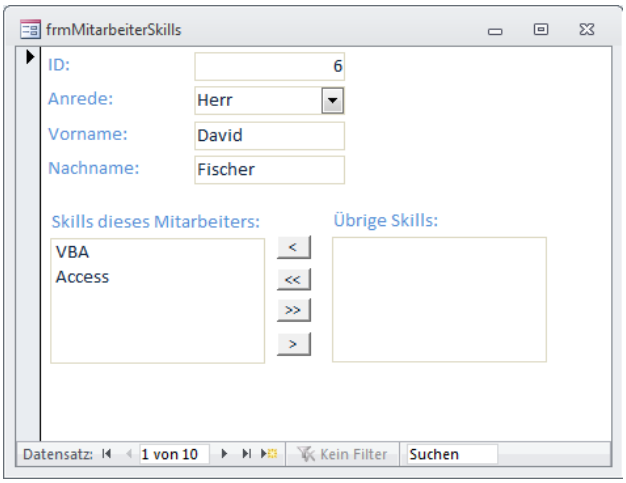


Bild 9: Listenfeld mit korrekter Anzeige der Skill-Einträge

Listenfeld mit den übrigen Skills

Die Skills eines Mitarbeiters zu ermitteln war einfach: Wir brauchten einfach nur die aktuelle **MitarbeiterID** als Kriterium für das entsprechende Feld der Datensatzherkunft des Listenfeldes einzutragen. Das Auffinden der Einträge der Tabelle **tblSkills**, die dem aktuell angezeigten Mitarbeiter nicht zugeordnet sind, ist etwas schwieriger. Wir beginnen mit einer Abfrage namens **qrySkillsNichtZugeordnet**, die schlicht die Tabelle **tblSkills** enthält und beide Felder dieser Tabelle anzeigt. Bild 10 zeigt die Entwurfsansicht dieser Abfrage.

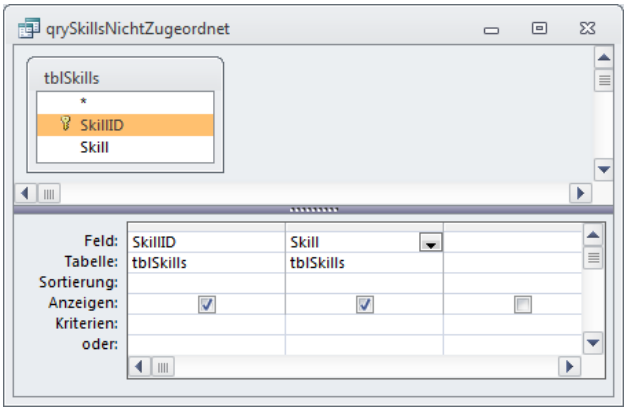


Bild 10: Diese Abfrage gibt zunächst alle Datensätze der Tabelle **tblSkills** zurück.

Nun liefert diese alle Skills. Die Skills, die dem aktuellen Mitarbeiter zugeordnet sind, soll sie jedoch nicht anzeigen. Wie aber schließen wir genau diese aus? Dazu verwenden wir eine Kriterium, das genau die **SkillID**s auflistet, die dem Mitarbeiter bereits zugeteilt wurden. Dieses Kriterium verwendet das **IN**-Schlüsselwort mit einer Abfrage als Parameter, die genau die gewünschten Daten liefert (grundlegende Informationen hierzu finden Sie in **Abfragen für die Datenauswahl – Teil V: Filtern nach Zahlen**). Diese Abfrage sieht als SQL-Ausdruck wie folgt aus:

```
SELECT SkillID FROM qrySkillsEinesMitarbeiters
```

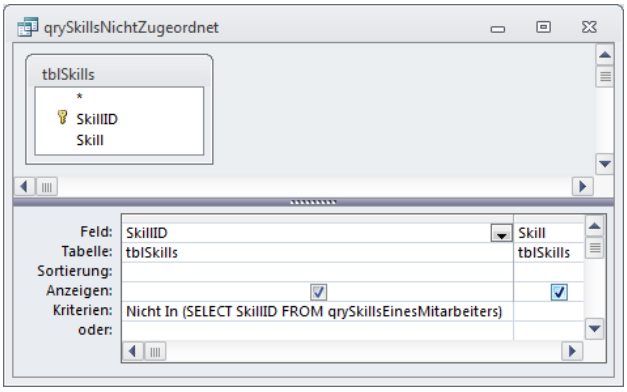


Bild 11: Diese Abfrage liefert alle Skills, die dem aktuellen Mitarbeiter nicht zugeordnet wurden.

Die Abfrage liefert genau die gleichen Datensätze wie die Abfrage, die als Datensatzherkunft des Listenfeldes **lstAusgewaehlt** dient – allerdings beschränkt sie sich auf das Primärschlüsselfeld **SkillID**. Das ist auch notwendig, denn die **IN**-Klausel, welche das Ergebnis der Abfrage weiterverwenden soll, kann nur ein einziges Feld verarbeiten.

Durch die Verwendung des Ausdrucks **NOT IN (...)** als Kriterium liefert die Abfrage **qrySkillsNichtZugeordnet** aus Bild 11 alle Datensätze der Tabelle **tblSkills**, deren **SkillID** nicht durch die in Klammern angegebene Abfrage geliefert werden.

Stellen Sie nun die Datensatzherkunft des Listenfeldes **lstNichtZugeordnet** auf den Namen der neuen Abfrage **qrySkillsNichtZugeordnet** ein. Passen Sie außerdem wie schon beim ersten Listenfeld die Ei-

Tabellen entwerfen

Teil IV: Felder und ihre Eigenschaften

In den bisherigen Artikeln haben wir bereits einige Tabellen erstellt. Dabei haben wir uns jedoch auf die benötigten Felder und deren Datentypen konzentriert. Tabellen und Felder bieten jedoch noch eine ganze Reihe weiterer Einstellmöglichkeiten. In diesem Artikel stellen wir die Wichtigsten vor.

Felder anlegen, löschen und verschieben

Wenn Sie eine neue Tabelle in der Entwurfsansicht vor sich sehen, können Sie in jeder Zeile des Entwurfsrasters ein Feld anlegen und Feldname, Felddatentyp und Beschreibung festlegen. Im unteren Bereich des Entwurfsbereichs finden Sie weitere Eigenschaften, die aber nur für das jeweils markierte Feld angezeigt werden.

In der Regel fügen Sie neue Felder nacheinander an, indem Sie Zeile für Zeile des Datenentwurfs füllen. Sie werden jedoch von Zeit zu Zeit auch Felder löschen, umsortieren oder neue Felder zwischen zwei bestehenden Feldern einsortieren wollen.

Zum Löschen eines Feldes gibt es beispielsweise die folgenden beiden Möglichkeiten:

- Markieren Sie die komplette Zeile des zu löschenden Feldes durch einen Mausklick auf den grauen Bereich links neben dem Feld. Betätigen Sie dann entweder die **Entfernen**-Taste, den Menüeintrag **Bearbeiten|Zeilen löschen** beziehungsweise den entsprechenden Symbolleisteneintrag (Access 2003 und jünger) oder den Ribbon-Eintrag **Entwurf|Tools|Zeilen löschen** (Access 2007 und jünger).

- Oder Sie klicken mit der rechten Maustaste auf die zu löschende Zeile und wählen den Eintrag **Zeilen löschen** des Kontextmenüs aus (s. Bild 1).

Sie können auch gleich mehrere zusammenhängende Felder löschen. Drücken Sie die linke Maustaste auf dem grauen Bereich links vom ersten zu löschenden Feld herunter, bewegen Sie die Maus zum letzten zu löschenden Feld und lassen Sie die Maustaste wieder los (s. Bild 2). Betätigen Sie dann auf die oben beschriebene Weise den Befehl **Zeilen löschen** oder betätigen Sie einfach die **Entfernen**-Taste.

Für die in der Tabelle zu speichernden Daten spielt es keine Rolle, in welcher Reihenfolge Sie die Felder anlegen. Wenn Sie später jedoch Formulare und Berichte auf Basis der Tabelle anlegen möchten, erscheinen die Felder in genau dieser Reihenfolge in

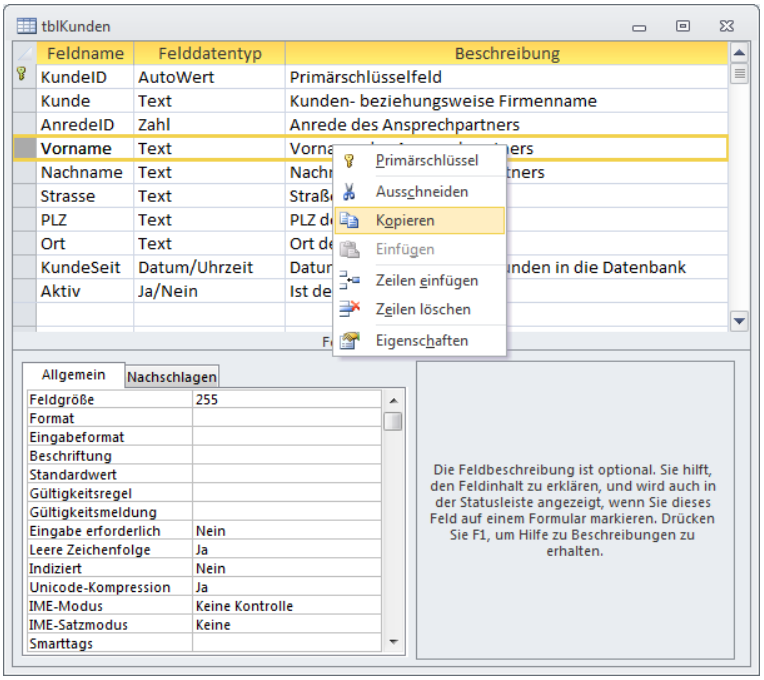


Bild 1: Bearbeiten einer Tabelle in der Entwurfsansicht

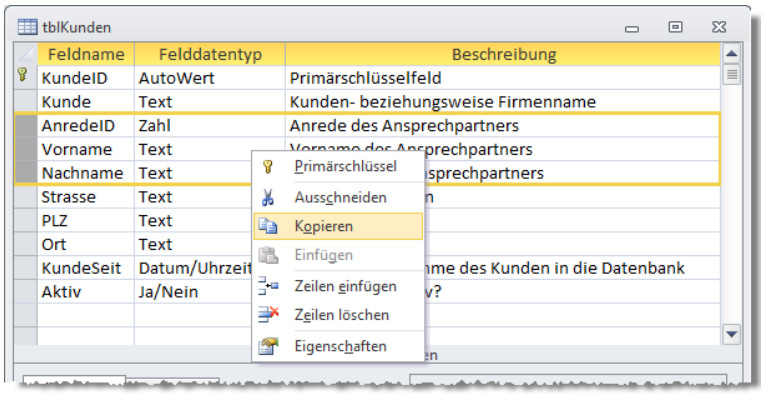


Bild 2: Markieren mehrerer zusammenhängender Zeilen

der Feldliste. Oft zieht man gleich alle Felder aus der Feldliste in den Formularentwurf. In diesem Fall ist es natürlich sinnvoll, wenn diese gleich in der richtigen Reihenfolge vorliegen – anderenfalls müssten Sie die Felder einzeln an die richtige Position im Formular ziehen.

Sollten Sie also einmal nicht gleich beim ersten Anlauf die richtige Reihenfolge der Felder im Tabellenentwurf festgelegt haben, können Sie dies nachholen.

Um ein Feld zu verschieben, markieren Sie es durch einen Mausklick auf den grauen Bereich links vom Feld. Ziehen Sie es dann bei gedrückter Maustaste an die gewünschte Position. Eine schwarze Linie deutet an, zwischen welchen beiden Feldern das zu verschiebende Feld eingefügt wird. Haben Sie die richtige Position erreicht, lassen Sie die Maustaste einfach los.

Sie können auch neue Zeilen zwischen zwei bestehende Zeilen einfügen. Dazu markieren Sie die Zeile, vor der die neue Zeile eingefügt werden soll und rufen den Befehl **Zeile einfügen** auf. Diesen finden Sie an den gleichen Stellen wie den bereits oben beschriebenen Befehl zum Löschen eines Feldes.

Felder umbenennen

Solange Sie noch keine Abfragen, Formulare und Berichte erstellt haben, die auf die Felder einer Tabelle verweisen, können Sie die Feldnamen ohne Probleme ändern.

Später sollten sorgfältig abwägen, ob das Anpassen in allen abhängigen Objekten einschließlich eventuell darauf zugreifenden VBA-Routinen nicht zu aufwendig wird.

Immerhin treten Feldnamen in der Benutzeroberfläche nur als Standardwerte für die Bezeichnung von Steuerelementen auf Basis dieser Felder in Erscheinung – und die kann man immer noch unabhängig vom Feldnamen selbst anpassen.

Felder ausschneiden, kopieren und einfügen

Brauchen Sie ein bereits bestehendes Feld erneut, können Sie es zum Beispiel über den entsprechenden Befehl des Kontextmenüs oder über die Tastenkombination **Strg + C** dieses Feldes kopieren. Vorher müssen Sie die komplette Zeile dieses Feldes durch einen Klick auf den grauen Bereich links markieren. Anschließend markieren Sie den Zielort für dieses Feld. Hier gibt es verschiedene Möglichkeiten:

- Markieren einer oder mehrerer kompletter Zeilen: Die Zeilen werden überschrieben.
- Setzen der Einfügemarke in eine Zeile: Einfügen der zu kopierenden Zeile vor die markierte Zeile.

Anschließend fügen Sie das zu kopierende Feld entweder über den Menü-, Symbolleisten- oder Ribbon-Eintrag **Einfügen** oder die Tastenkombination **Strg + V** ein.

Auf ähnliche Weise schneiden Sie ein Feld aus und fügen es an anderer Stelle wieder ein. Achtung: Das Ausschneiden und Einfügen löscht alle im Feld enthaltenen Daten. Zum Verschieben eines Feldes mit Inhalten sollten Sie also den entsprechenden Befehl verwenden.

Kopieren, Ausschneiden und Einfügen funktioniert natürlich auch tabellenübergreifend – und Sie können sogar Felder aus einer Datenbank in eine Tabelle einer anderen Datenbank einfügen.

Datentypen

Als erstes werfen wir einen genaueren Blick auf die einzelnen Datentypen. Es gibt die folgenden Datentypen, die Sie wie in Bild 3 festlegen:

- **Autowert**: siehe →*[basics]* **Autowertfeld anlegen**
- **Text**: siehe →*[basics]* **Textfeld anlegen**

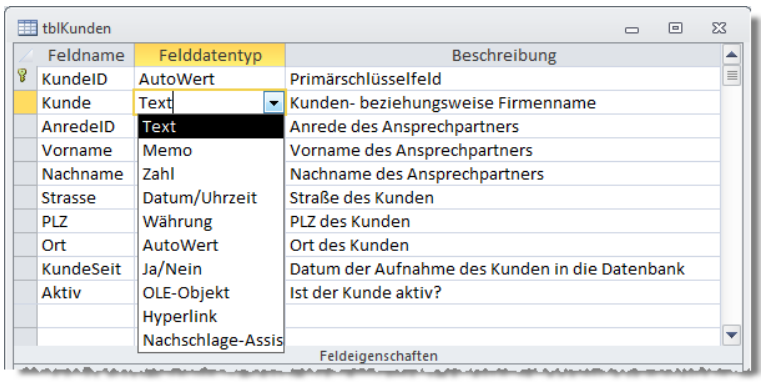


Bild 3: Auswahl des Datentyps

- **Memo**: speichert lange Texte von bis zu 64.000 Zeichen
- **Zahl**: speichert numerische Daten
- **Datum/Zeit**: siehe →[basics] **Datum/Zeit-Feld anlegen**
- **Währung**: speichert numerische Daten als Festkommazahl mit vier Nachkommastellen
- **Ja/Nein**: siehe →[basics] **Ja/Nein-Feld anlegen**
- **OLE-Objekt**: speichert beispielsweise komplette Windows-Dokumente, die über ein entsprechendes Steuerelement etwa innerhalb eines Formulars angezeigt werden können
- **Hyperlink**: speichert E-Mail- und Internet-Adressen oder Position in der Datenbank, die per Mausklick auf den Text angesprungen werden können
- **Nachschlage-Assistent**: siehe →[basics] **Nachschlagefeld für eine 1:n-Beziehung einrichten** und →[basics] **Nachschlagefeld für Wertliste einrichten**
- **Anlage** (ab Access 2007): speichert Dateien
- **Berechnet** (ab Access 2010): erweitert eine Tabelle um ein berechnetes Feld, etwa zum Zusammensetzen von Vor- und Nachname oder zum Berechnen der Mehrwertsteuer bei gegebenem Preis und Mehrwertsteuersatz

Eine Übersicht der Datentypen und wichtige Informationen finden Sie unter **Datentypen**.

Feldgrößen

Die Eigenschaft **Feldgröße** gibt es nur für bestimmte Datentypen. Nicht jeder Datentyp bietet die gleichen Werte für die Feldgröße an.

Beim Datentyp **Text** geben Sie beispielsweise die maximale Anzahl Zeichen an, wobei 255 der größtmögliche Wert ist.

Der Datentyp **Zahl** bietet gleich eine ganze Reihe von Einstellungen, die sich auf die Größe und Art der möglichen Werte auswirken (siehe Bild 4). Hier sind die wichtigsten:

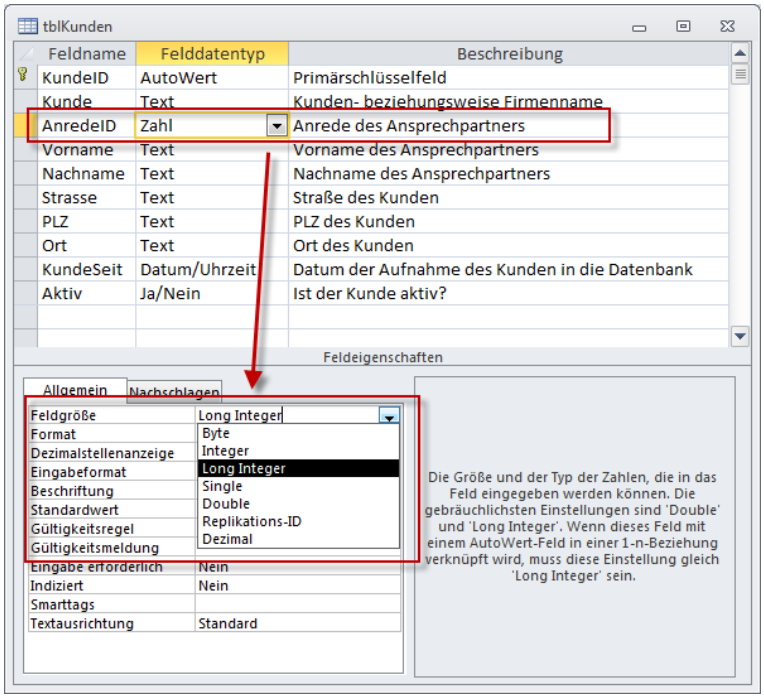


Bild 4: Festlegen der Feldgröße

- **Byte**: Ganzzahlen von 0 bis 255 (ein Byte Speicherbedarf)
- **Integer**: Ganzzahlen von -32.768 bis 32.767 (zwei Bytes Speicherbedarf)
- **Long**: Ganzzahlen von -2.147.483.648 bis 2.147.483.647 (vier Bytes Speicherbedarf)
- **Single**: Gleitkommazahlen von -3,402823E38 bis -1,401298E-45 für negative Werte und 1,401298E-45 bis 3,402823E38 für positive Werte (vier Bytes Speicherbedarf)
- **Double**: Gleitkommazahlen von -1,79769313486232E308 bis 4,94065645841247E-324 für negative Werte und 4,94065645841247E-324 bis 1,79769313486232E308 für positive Werte (acht Bytes Speicherbedarf)

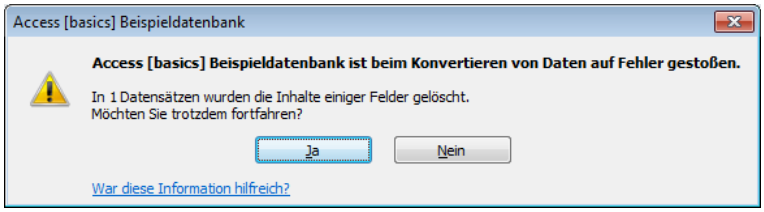


Bild 5: Meldung beim Ändern einer Feldgröße, die zum Ändern von Daten führen würde.

- **Dezimal**: Festkommazahlen im Bereich von -10E28 bis 20E28 (zwölf Bytes Speicherbedarf)

Informationen zum Unterschied zwischen Gleitkommazahlen und Festkommazahlen finden Sie im Artikel **VBA-Programmierung - Teil II: Variablen in VBA** unter **Gleitkommazahlen** und **Festkommazahlen**.

Achtung: Wenn Sie die Feldgröße für ein Feld ändern, das bereits Daten enthält, löschen Sie unter Umständen Daten. Dies kann zum Beispiel passieren, wenn Sie die **Feldgröße** eines Textfeldes von **255** auf **50** reduzieren.

Alle Inhalte dieses Feldes werden hinter dem fünfzigsten Zeichen abgeschnitten. Auch bei Zahlendatentypen kann dies geschehen. So schneidet etwa der Wechsel von **Double** auf **Long** alle Nachkommastellen ab.

Und wenn Sie beispielsweise die Feldgröße eines Zahlenfeldes von **Integer** auf **Byte** ändern, werden alle Werte größer **255** einfach gelöscht. Access weist jedoch per Meldung auf diesen Umstand hin (s. Bild 5).

Zahlenfelder beanspruchen immer den ihrer Feldgröße entsprechenden Speicherplatz.

Die übrigen Felder enthalten keine Eigenschaft zur Einstellung der Feldgröße. **Datum/Zeit**-Felder werden immer als **Double**-Zahl gespeichert.

Memofelder belegen immer den dem enthaltenen Text entsprechenden Platz (bis maximal 64.000 Zeichen) und auch OLE-Felder werden nur durch die Obergrenze von einem Gigabyte begrenzt.

Format festlegen

Für fast alle Datentypen können Sie ein individuelles Anzeigeformat definieren. Dieses Format legt fest, wie die enthaltenen Daten erscheinen sollen.

Access zeigt den Feldinhalt normalerweise so an, wie er eingegeben wird. Sie können jedoch beispielsweise für ein Datum angeben, dass es in Kurzform (1.1.2011) oder ausgeschrieben (Samstag, 1. Januar 2011) erscheinen soll.

Ein Datumsfeld bietet einige vorgefertigte Format (siehe Bild 6), aus denen Sie eines auswählen können.

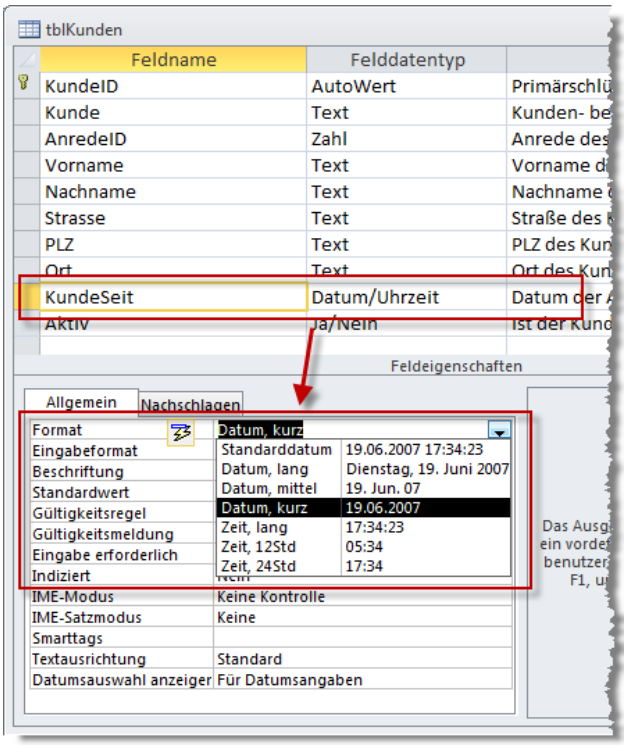


Bild 6: Auswahl eines der Formate für ein Datumsfeld

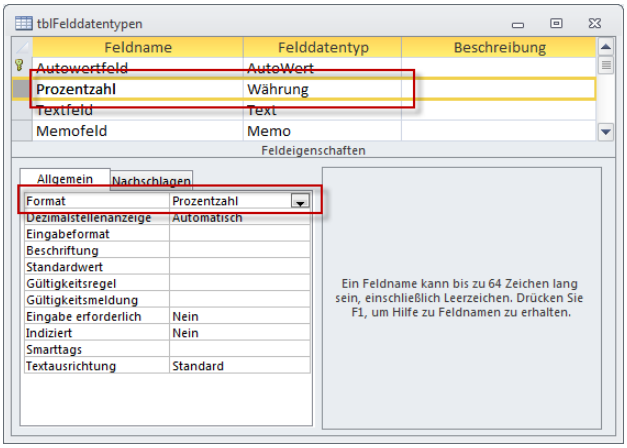


Bild 7: Festlegen des Formats für Prozentzahlen

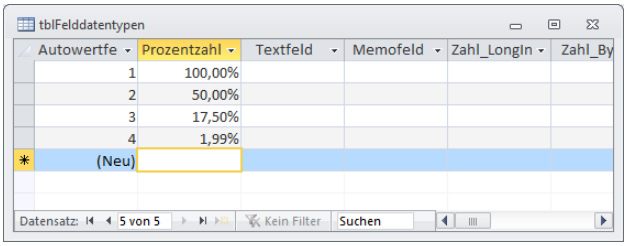


Bild 8: Prozentzahlen in der Datenblattansicht

Sie haben jedoch auch die Möglichkeit, ein benutzerdefiniertes Format anzulegen. Dazu müssen Sie die Buchstaben kennen, die als Platzhalter für die einzelnen Bestandteile des Datums dienen. Der Ausdruck **tt.mm.jjjj** entspricht beispielsweise **01.01.2011**. Sol-

Abfragen für die Datenauswahl

Teil VI: Daten zusammenführen mit UNION-Abfragen

Es kommt immer mal wieder vor, dass Sie gleichartige Daten aus mehreren Tabellen zusammenführen möchten. Ein einfaches Beispiel: Sie haben eine Kunden-, eine Mitarbeiter- und eine Lieferanten-Tabelle und möchten diese in einer einzigen Abfrage anzeigen, um den entsprechenden Personen eine Weihnachtskarte zukommen zu lassen. In diesem Fall gibt es im Wesentlichen zwei Möglichkeiten – und eine davon ist die sogenannte UNION-Abfrage.

Es gibt verschiedene Anlässe, Daten aus mehreren Tabellen oder Abfragen in einem Abfrageergebnis zusammenzuführen. Diese unterschieden sich wie folgt:

- Sie führen tatsächlich Daten aus zwei oder mehreren verschiedenen Tabellen oder Abfragen zusammen.

- Sie fügen den Daten einer Tabelle einen oder mehrere künstliche Datensätze hinzu.

Ein Beispiel für die erste Gruppe haben wir bereits in der Einleitung angerissen. Sie können damit beispielsweise die Adressen aus verschiedenen Tabellen wie Kunden-, Mitarbeiter- oder Lieferantentabellen zusammenführen, um beispielsweise alle Empfänger einer Weihnachtskarte zu ermitteln.

Die zweite Gruppe kommt zum Tragen, wenn eine Liste Einträge enthalten soll, welche die Datenherkunft normalerweise nicht hergibt. Ein gutes Beispiel finden Sie im Artikel **Datensatzauswahl per Kombinationsfeld - Kunden per Kombinationsfeld auswählen und im Formular anzeigen**.

Dort soll ein Kombinationsfeld alle Kundenbezeichnungen anzeigen und außerdem einen Eintrag namens **<Kunde auswählen>** bereithalten, der gleich nach dem Öffnen des Formulars angezeigt wird und dem Benutzer verdeutlicht, wozu das Kombinationfeld dient.

Auf ähnliche Weise könnte man einem Listenfeld oder einer Datenblattansicht Trennlinien oder ähnliches unterjubeln.

Kein UNION ohne SQL

Der kleine Haken beim Zusammenführen von Daten aus zwei oder mehr Tabellen oder Abfragen ist, dass UNION-Abfragen nicht wie die meisten anderen Abfragen einfach im Abfrageentwurf von Access zusammengestellt werden können, sondern den Einsatz von SQL erfordern. Und das, obwohl Sie als Leser dieses Magazins bisher noch gar nicht mit reinem SQL gearbeitet haben!

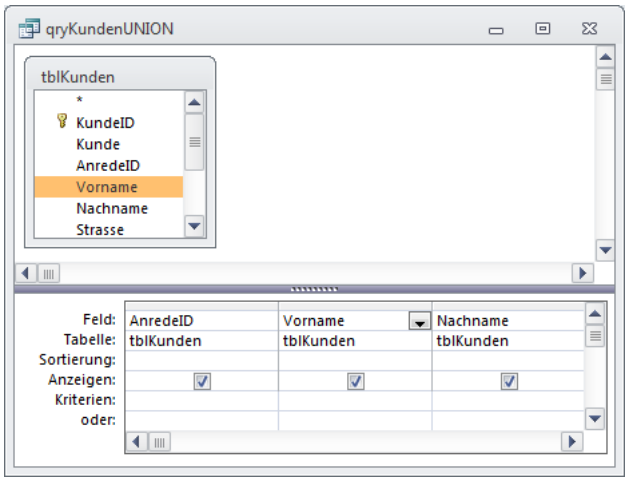


Bild 1: Der Kunden-Teil der UNION-Abfrage

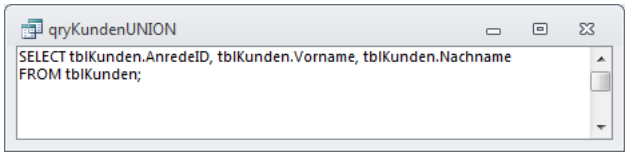


Bild 2: Die Abfrage in der SQL-Ansicht

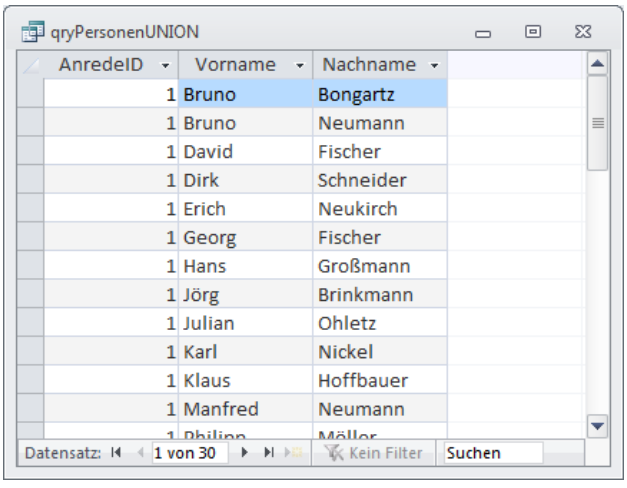


Bild 3: Das Ergebnis der UNION-Abfrage mit allen Mitarbeitern und Kunden

Genau genommen ist das auch der Grund, warum UNION-Abfragen gerade bei Einsteigern ein Schattendasein führen. Und das, obwohl sie in manchen Situationen unschätzbar wertvoll sein können!

Wir versuchen, für diesen Grundlagenartikel soweit wie möglich mit dem Abfrageentwurf zu arbeiten und das Thema SQL nur dort anzureißen, wo es unbedingt notwendig ist.

Gleiches zu gleichem

In einer **UNION**-Abfrage fügen Sie die Daten zweier oder mehrerer Abfragen zusammen, indem Sie die **SELECT**-Anweisungen zum Ermitteln dieser Daten durch das **UNION**-Schlüsselwort miteinander verknüpfen.

Diese **SELECT**-Anweisungen entsprechen schlicht und einfach einer anderen Schreibweise dessen, was Sie im Abfrageentwurf von Access als Abfrage festlegen.

Die **SELECT**-Anweisung enthält den Namen der Tabelle, aus der die Daten stammen sowie die Felder, die im Abfrageergebnis vorkommen sollen. Außerdem steht in der **SELECT**-Anweisung noch, wie die Daten gegebenenfalls gefiltert und sortiert werden sollen.

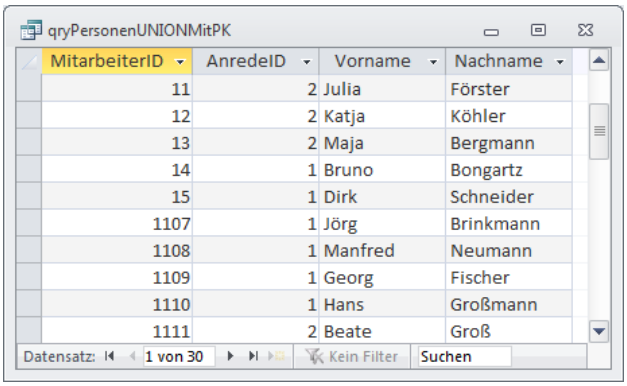
Die wichtigste Grundregel beim Erstellen von **UNION**-Abfragen ist die folgende: Alle **SELECT**-Anweisungen, die durch das **UNION**-Schlüsselwort verkettet werden, müssen genau die gleiche Anzahl Felder enthalten und die gleichen Namen enthalten.

Die Datentypen spielen hier allerdings keine Rolle: Sie können durchaus Text- und Zahlenfelder zusammenführen.

Daten aus mehreren Tabellen zusammenführen

Um dies zu verdeutlichen, beginnen wir einfach mit der ersten **UNION**-Abfrage. Die Beispieldatenbank liefert mit **tblKunden** und **tblMitarbeiter** zwei Tabellen, die sich als Grundlage für ein Beispiel einer **UNION**-Abfrage eignen. Wir wollen zunächst die drei Felder **AnredeID**, **Vorname** und **Nachname** betrachten (siehe Bild 11).

Um die Daten der beiden Tabellen zusammenzuführen, brauchen wir zunächst zwei SQL-Ausdrücke,


qryPersonenUNIONMitPK

MitarbeiterID	AnredeID	Vorname	Nachname
11	2	Julia	Förster
12	2	Katja	Köhler
13	2	Maja	Bergmann
14	1	Bruno	Bongartz
15	1	Dirk	Schneider
1107	1	Jörg	Brinkmann
1108	1	Manfred	Neumann
1109	1	Georg	Fischer
1110	1	Hans	Großmann
1111	2	Beate	Groß

Datensatz: 1 von 30 | Kein Filter | Suchen

Bild 4: Personen aus **tblMitarbeiter** und **tblKunden** mit Primärschlüsselwert

welche die gewünschten Daten liefern. Das ist kein Problem – beginnen wir einfach mit der Tabelle **tblKunden**.

Erstellen Sie eine neue Abfrage und fügen Sie im Dialog **Tabelle anzeigen** die Tabelle **tblKunden** zum Abfrageentwurf hinzu. Ziehen Sie dann die drei Felder **AnredeID**, **Vorname** und **Nachname** in das Entwurfsraster, sodass die Abfrage wie in Bild 1 aussieht.

Wechseln Sie nun zur SQL-Ansicht der Abfrage, indem Sie den Menübefehl **Ansicht|SQL-Ansicht** (Access 2003 und älter) oder den Ribbon-Eintrag **Entwurf|Ergebnisse|Ansicht|SQL-Ansicht** (Access 2007 und jünger) auswählen.

In der SQL-Ansicht sieht die Abfrage wie in Bild 2 aus.

Auf die gleiche Weise verfahren Sie mit der Tabelle **tblMitarbeiter**: Erstellen Sie eine neue, leere Abfrage, fügen Sie die drei Felder **AnredeID**, **Vorname** und **Nachname** der Tabelle **tblMitarbeiter** hinzu und wechseln Sie dann in die SQL-Ansicht.

Das Ergebnis sieht fast genauso aus wie das der Abfrage aller Kundennamen:

```
SELECT tblMitarbeiter.AnredeID,
  tblMitarbeiter.Vorname,
  tblMitarbeiter.Nachname
FROM tblMitarbeiter;
```

Mit diesen beiden **SELECT**-Anweisungen im Gepäck erstellen Sie nun die **UNION**-Abfrage. Dazu legen Sie eine weitere neue Abfrage an. Diesmal schließen Sie den Dialog **Tabellen anzeigen** jedoch direkt und finden einen völlig leeren Abfrage-Entwurf vor. Den können wir diesmal auch nicht gebrauchen und wechseln daher direkt in die SQL-Ansicht.

Steuerelemente

Teil III: Das Textfeld

Textfelder sind neben Beschriftungsfeldern wohl die meistgenutzten Steuerelemente in Access-Anwendungen. Sie dienen zur Anzeige und Eingabe einfacher Texte und können durch verschiedene Eigenschaften in Funktion und Aussehen beeinflusst werden. Sie zeigen Daten aus der Datenherkunft eines Formulars an oder erlauben die Eingabe von Texten, die einfach nur weiterverarbeitet werden, ohne selbst in einer Tabelle zu landen. Und natürlich lösen Textfelder auch Ereignisse aus, für die Sie wiederum entsprechende Prozeduren hinterlegen können – beispielsweise zum Validieren des eingegebenen Inhalts.

Die wichtigste Limitierung eines Textfeldes ist, das es schlicht und einfach nur Text anzeigen kann. Davon abgesehen können Sie mit einem Textfeld aber sehr flexibel arbeiten, wie der vorliegende Artikel zeigen wird.

Textfelder in Formularen und Berichten

Textfelder können Sie wie die meisten Steuerelemente sowohl in Formularen als auch in Berichten abbilden.

Im Gegensatz zu Formularen, wo Textfelder nicht nur der Anzeige, sondern auch der Bearbeitung von Texten dienen, zeigen diese in Berichten lediglich Inhalte an.

Während in Formulare noch eine ganze weiterer Steuerelemente wie Schaltflächen, Kombinationsfelder, Listenfelder und viele mehr eine Daseinsberechtigung haben, sind Textfelder in Berichten quasi die einzige Möglichkeit, Texte sinnvoll darzustellen.

Nebenher können Sie noch mit Linien arbeiten, um den Inhalt zu strukturieren, aber Steuerelemente wie Kombinationsfelder, Listenfelder, Schaltflächen et cetera haben in einem Bericht schlicht nichts verloren. Zum Einsatz von Textfeldern in Berichten kommen wir jedoch in einem späteren Artikel.

Gebundene und ungebundene Textfelder

Manchmal sollen Textfelder einfach nur für den Zeitraum, in dem ein Formular angezeigt wird, zur Eingabe bestimmter Informationen dienen. Danach wird der Inhalt einfach verworfen.

Ein klassisches Beispiel dafür ist die Eingabe eines Dateinamens, der nur zum Öffnen oder Anlegen einer Datei verwendet wird. Dies ist ein ungebundenes Textfeld.

Andere Textfelder sollen das Bearbeiten der Daten einer Tabelle ermöglichen. Dabei wird zunächst das Formular an die Tabelle oder Abfrage gebunden, indem Sie der Eigenschaft **Datenherkunft** des Formulars deren Namen zuweisen, also beispielsweise **tblKunden** (s. Bild 1).

Wenn ein Textfeld dann den Inhalt eines der Felder dieser Tabelle anzeigen soll, also etwa **Vorname** oder **Nachname**, benötigen Sie ein Textfeld, dessen Eigenschaft **Steuerelementinhalt** den Namen des entsprechenden Feldes der zugrunde liegenden Ta-

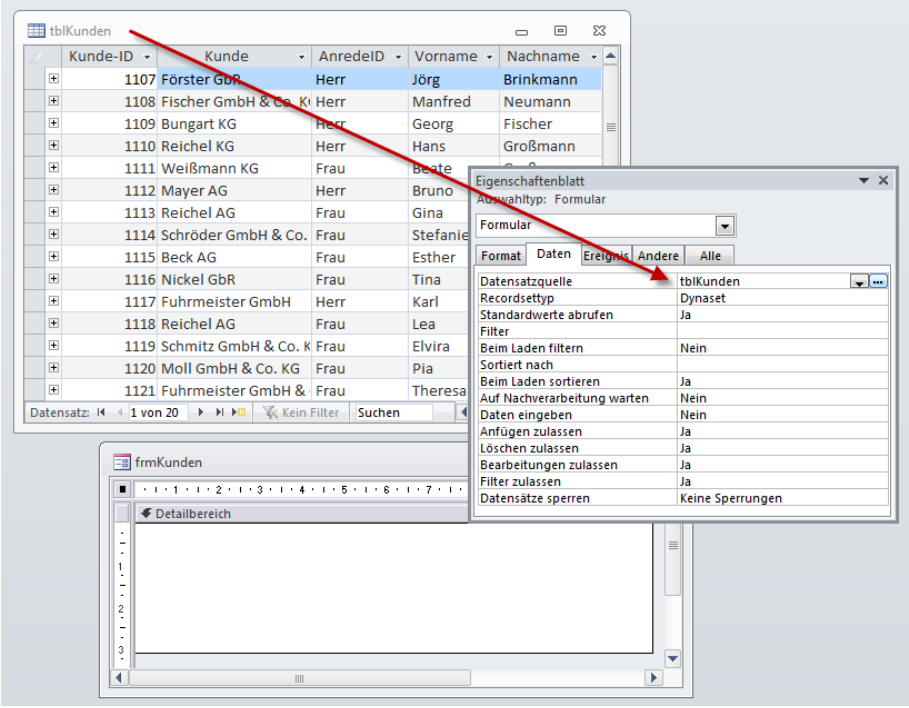


Bild 1: Zuweisen der Datenherkunft eines Formulars

belle enthält. Das erreichen Sie auf zwei Arten:

- Sie aktivieren die Feldliste, also die Liste der in der Datenherkunft des Formulars enthaltenen Felder, und ziehen mit der Maus die gewünschten Felder in das Formular. Diese werden dann in der Regel als Textfelder angelegt und erhalten automatisch den Feldnamen als Steuerelementname und auch als Steuerelementinhalt (siehe →*[basics]* **Gebundene Steuerelemente zu Formularen und Berichten hinzufügen**). Außerdem erhält das Bezeichnungsfeld dieses Textfelds den Namen des Feldes oder, falls vorhanden, den Wert der Eigenschaft **Beschriftung** des Feldes.

- Sie erstellen manuell ein neues Textfeld und stellen dann seine Eigenschaft **Steuerelementinhalt** auf den Namen des entsprechenden Felds der Datenherkunft ein, also beispielsweise **Vorname**. Im Gegensatz zur ersten Methode müssen Sie außerdem noch den Namen des neuen Textfelds anpassen (etwa auf **txtVorname**) und die Beschriftung des zugehörigen Bezeichnungsfeldes einstellen.

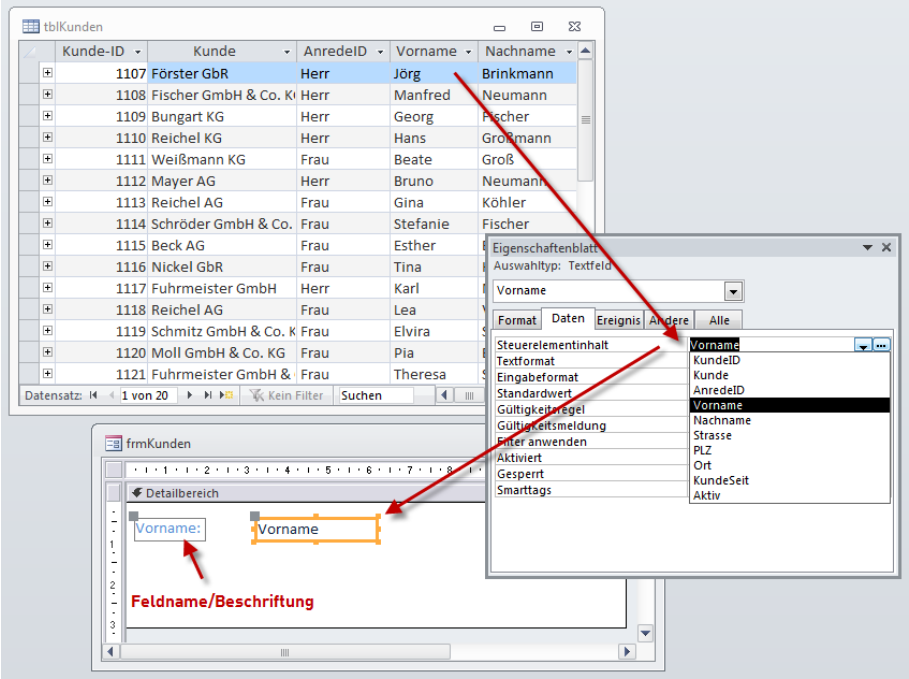


Bild 2: Binden eines Textfelds an ein Feld der Datenherkunft

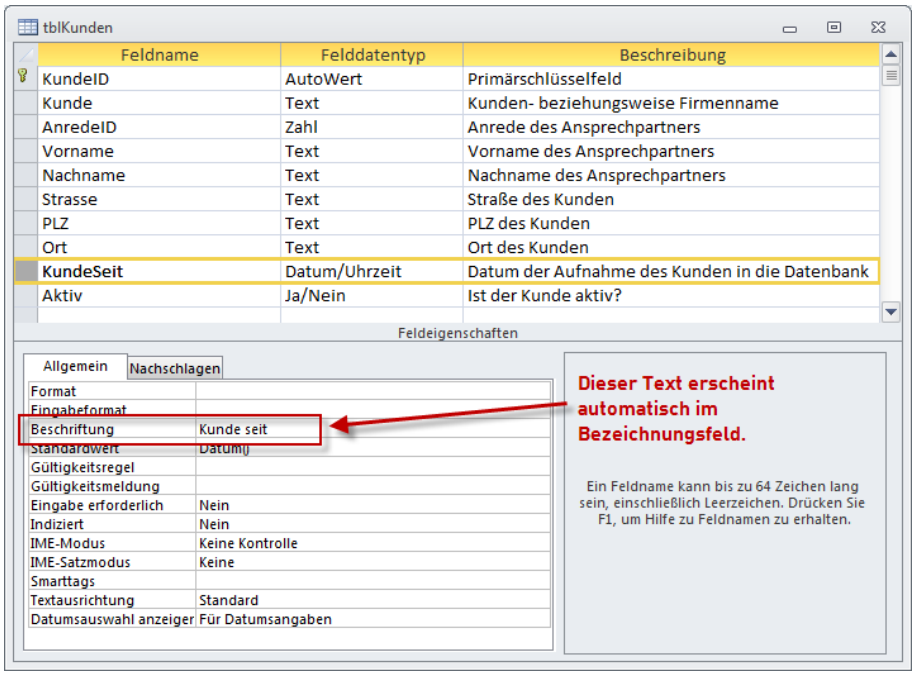


Bild 3: Diese Beschriftung wird in das Bezeichnungsfeld eines Textfelds auf Basis dieses Feldes übernommen.

Bild 2 zeigt einige Schlüsselelemente beim manuellen Anlegen eines gebundenen Textfelds.

Die zweite Methode haben wir nur vorgestellt, damit Sie erstens erkennen, welche Schritte zum Hinzufügen eines gebundenen Textfeldes nötig sind und an welchen Eigenschaften Sie Änderungen vornehmen müssen, um beispielsweise den Namen des Textfelds oder die Bezeichnung zu ändern.

Arbeit sparen

Sie haben bereits gesehen, dass einige Schritte beim Ziehen eines Feldes aus der Feldliste automatisch geschehen. Um hier besonders effizient zu arbeiten,

Steuerelemente

Teil IV: Das Kombinationsfeld

Kombinationsfelder sind ein sehr wichtiges Steuerelement. Sie können damit Daten aus einer Liste auswählen und diese auf verschiedene Arten weiterverwenden. Doch nicht nur das: Kombinationsfelder dienen nicht nur zur Datenauswahl, sondern gegebenenfalls auch zur Eingabe von Daten. In diesem Artikel lernen Sie zunächst die Grundlagen zu Kombinationsfeldern kennen.

Ordnen wir zunächst die Einsatzfelder von Kombinationsfelder ein. Diese hängen maßgeblich von zwei Faktoren ab. Der erste ist die eventuelle Bindung des Kombinationsfeldes an ein Feld einer zugrunde liegende Tabelle:

- Das Formular, welches das Kombinationsfeld enthält, besitzt eine Tabelle oder Abfrage als Datenherkunft und das Kombinationsfeld zeigt die Daten eines der Felder dieser Tabelle an.
- Das Kombinationsfeld ist nicht an ein Feld gebunden, sondern bietet einfach nur Listeneinträge zur Auswahl an.

Der zweite wichtige Faktor ist die Herkunft der Daten des Kombinationsfeldes:

- Das Kombinationsfeld bezieht seine Daten aus einer Tabelle oder Abfrage.
- Das Kombinationsfeld zeigt manuell hinzugefügte Werte an.
- Dritte, hier nur der Vollständigkeit halber aufgeführte Variante: Das Kombinationsfeld wird über eine spezielle Funktion mit Daten gefüllt.

Alle Varianten der beiden Faktoren lassen sich miteinander kombinieren (daher bezieht dieses Steuelement aber nicht seinen Namen ...). Die folgenden Abschnitte zeigen die grundlegenden Techniken zum Füllen eines Kombinationsfeldes.

Anreden, manuell zusammengestellt

Nehmen wir an, Sie verwenden eine Tabelle zum Speichern von Kundendaten (siehe Beispieldatenbank, **tblKunden_AnredeText** – Entwurfsansicht wie in Bild 1). Wenn Sie ein Formular auf Basis dieser Tabelle anlegen und alle Felder der Tabelle zum Detailbereich hinzufügen (siehe →*[basics]* **Gebundenes Formular anlegen**), landen die Anreden dieser Kunden in einem Textfeld namens **Anrede** (siehe Bild 2). Der Benutzer muss nun beim Anlegen neuer Kunden jede Anrede von Hand eintragen. Einfacher wäre es, wenn er einen der Einträge **Herr** oder **Frau** mit einem Kombinationsfeld auswählen könnte.

Dazu sind nur wenige Schritte nötig:

- Öffnen Sie das Formular in der Entwurfsansicht.
- Klicken Sie mit der rechten Maustaste auf das Textfeld **Anrede**.
- Wählen Sie aus dem Kontextmenü den Eintrag **Ändern zu|Kombinationsfeld** aus (siehe Bild 3).

tblKunden_AnredeText

Feldname	Felddatentyp	Beschreibung
KundeID	AutoWert	Primärschlüsselfeld
Kunde	Text	Kunden- beziehungsweise Firmenname
Anrede	Text	Anrede des Ansprechpartners
Vorname	Text	Vorname des Ansprechpartners
Nachname	Text	Nachname des Ansprechpartners
Strasse	Text	Straße des Kunden
PLZ	Text	PLZ des Kunden
Ort	Text	Ort des Kunden
KundeSeit	Datum/Uhrzeit	Datum der Aufnahme des Kunden in die Datenbank
Aktiv	Ja/Nein	Ist der Kunde aktiv?
EMail	Text	E-Mail-Adresse des Kunden

Bild 1: Das Feld **Anrede** dient als Steuerelementinhalt des Kombinationsfeldes für die Auswahl der Anrede.

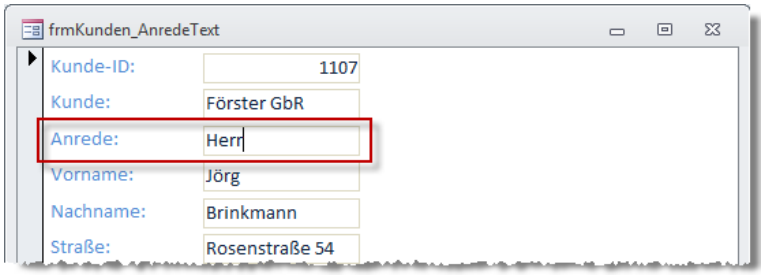


Bild 2: In das Formular übernommen erscheint das Feld Anrede zunächst als Textfeld.

Sie haben das Textfeld nun bereits in ein Kombinationsfeld umgewandelt. Nach einem Wechseln in die Formularansicht zeigt dieses den zuvor enthaltenen Eintrag an, aber wenn Sie auf die Schaltfläche am rechten Rand des Kombinationsfeldes klicken, erscheint lediglich eine leere Liste. Kein Wunder: Sie haben dem Kombinationsfeld ja auch noch nicht mitgeteilt, welche Werte es zur Auswahl anbieten soll.

Dies erledigen Sie durch einen weiteren Wechseln in die Entwurfsansicht. Klicken Sie auf das Kombinationsfeld **Anrede**, aktivieren Sie mit **F4** das Eigenschaftsfenster und nehmen Sie zwei Änderungen im Bereich **Daten** vor:

- Stellen Sie die Eigenschaft **Herkunftstyp** auf den Wert **Wertliste** ein.
- Legen Sie **Datensatzherkunft** auf den Ausdruck **"Herr";"Frau"** fest.

In der Formularansicht erscheint nun die gewünschte Liste mit den beiden Einträgen **Herr** und **Frau**.

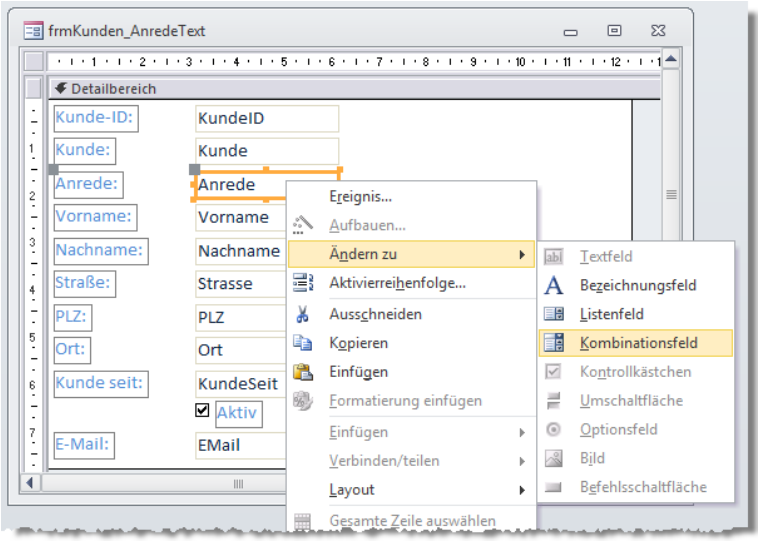


Bild 3: Umwandeln eines Textfelds in ein Kombinationsfeld

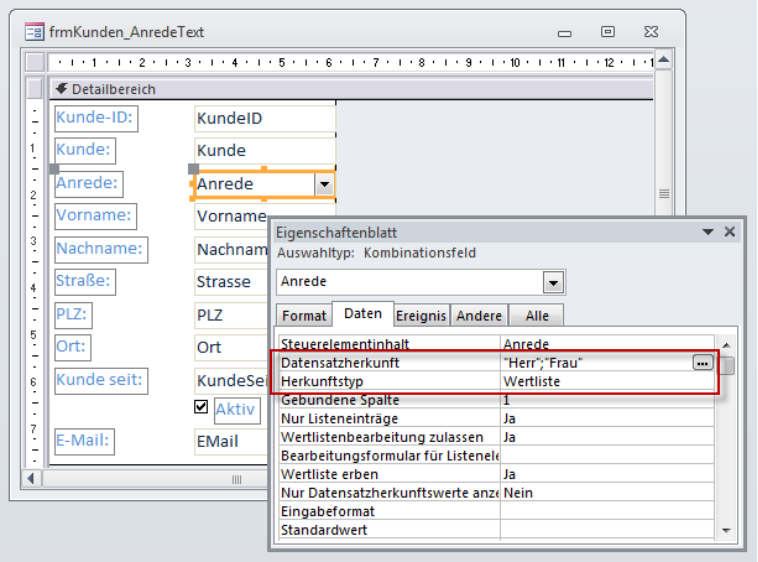


Bild 4: Einstellen der anzuzeigenden Einträge

Herkunftstypen

Die Eigenschaft **Herkunftstyp** erwartet einen der drei folgenden Werte:

- **Tabelle/Abfrage**: Die im Kombinationsfeld anzuzeigenden Daten stammen aus einer Tabelle.
- **Wertliste**: Die anzuzeigenden Daten stammen aus einer Semikola-getrennten Liste.
- **Feldliste**: Das Kombinationsfeld zeigt die Namen der Felder einer bestimmten Tabelle an.

Egal, welche der drei Einstellungen Sie auswählen: Mit der Eigenschaft **Datensatzherkunft** legen Sie weitere Details der anzuzeigenden Daten fest.

Datensatzherkunft

Wenn Sie **Tabelle/Abfrage** eingestellt haben, wählen Sie mit der Eigenschaft **Datensatzherkunft** die Tabelle oder Abfrage aus, aus der die anzuzeigenden Daten stammen sollen.

Bei der bereits im ersten Beispiel verwendeten Option **Wertliste** geben Sie manuell eine Liste der anzuzeigenden Ausdrücke an. Die Ausdrücke werden dabei durch Semikola getrennt, zum Beispiel **"Herr";"Frau"**.

Die dritte Option **Feldliste** erwartet ebenfalls die Auswahl einer Tabelle oder Abfrage für die Eigenschaft **Datensatzherkunft**. Hier zeigt das Kombinationsfeld jedoch die Namen aller Felder der ausgewählten Tabelle oder Abfrage an. Wofür das gut ist, werden Sie in späteren Beispielen erfahren.

Anreden aus der Tabelle

Mit den beiden Eigenschaften **Datensatzherkunft** und **Wertliste** ausgestattet machen wir uns gleich

Datensatzauswahl per Kombinationsfeld

Kunden per Kombinationsfeld auswählen und im Formular anzeigen

Wer Daten in einem Detailformular anzeigt, wird früher oder später schnell mal zu einem anderen Datensatz wechseln wollen – und das, ohne zwischendurch eine Liste aller Datensätze zur Auswahl aufzurufen. Eine einfache Lösung liegt im Einsatz eines Kombinationsfeldes im Formularkopf, das eine Schnellauswahl des gewünschten Datensatzes ermöglicht.

Die fertige Lösung dieses Artikels soll im Kopf ein Kombinationsfeld enthalten, dass alle Kunden alphabetisch sortiert anzeigt und nach der Auswahl eines Kunden seine Detaildaten im Formular anzeigt (s.Bild 1).

Wie die Schnellauswahl funktioniert, zeigen wir am Beispiel des Formulars **frmKundenDetail**, das wir zu diesem Zweck kopieren (im Datenbankfenster/Navigationsbereich markieren, **Strg + C**, **Strg + V**, als neuen Namen **frmKundenSchnellauswahl** angeben).

Um das Formular ein wenig zu entschlacken, stellen Sie die Eigenschaften **Datensatzmarkierer**, **Navigationsschaltflächen**, **Trennlinien** und **Bildlaufleisten** jeweils auf den Wert **Nein** ein. Das Formular sieht in der Formularansicht dann vorläufig wie in Bild 2 aus.

Da das Kombinationsfeld zur Auswahl der Kunden nicht zu den eigentlichen Daten gehört, bringen wir es der Ordnung halber im Formularkopf unter. Im Beispielformular ist dieser Bereich deaktiviert, daher müssen Sie ihn noch sichtbar machen. Dazu klicken Sie am einfachsten mit der rechten Maustaste auf den Balken mit der Beschriftung **Detailbereich** und wählen den Eintrag **Formularkopf/-fuß** aus (s. Bild 3).

In diesem Zuge verschieben Sie gleich noch die OK-Schaltfläche des Formulars in den Fußbereich. Dies erledigen Sie einfach per Drag and Drop. Die Höhe des Formularkopfes stellen Sie so ein, dass ein Kombinationsfeld leicht Platz darin findet.

Anschließend klicken Sie entweder auf die Schaltfläche **Kombinationsfeld** der Symbolleiste **Toolbox** (Access 2003 und älter) oder auf den Ribbon-Eintrag **Entwurf|Steuerelement|Kombinationsfeld** (Access 2007 und neuer) und fügen das Steuerelement durch einen Klick an die gewünschte Stelle im Formularkopf zu diesem hinzu.

Anschließend nehmen Sie noch ein paar Detaileinstellungen am Kombinationsfeld vor: Ändern Sie die

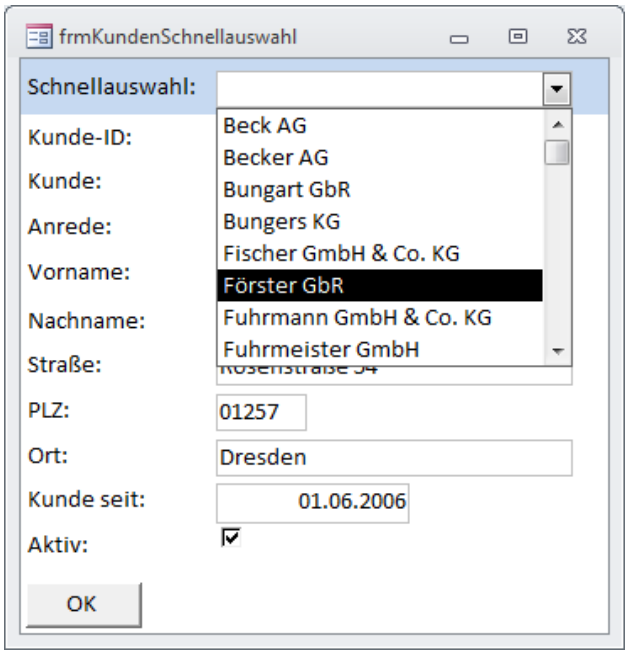


Bild 1: Das Suchkombinationsfeld erlaubt das schnelle Ansteuern von Datensätzen.

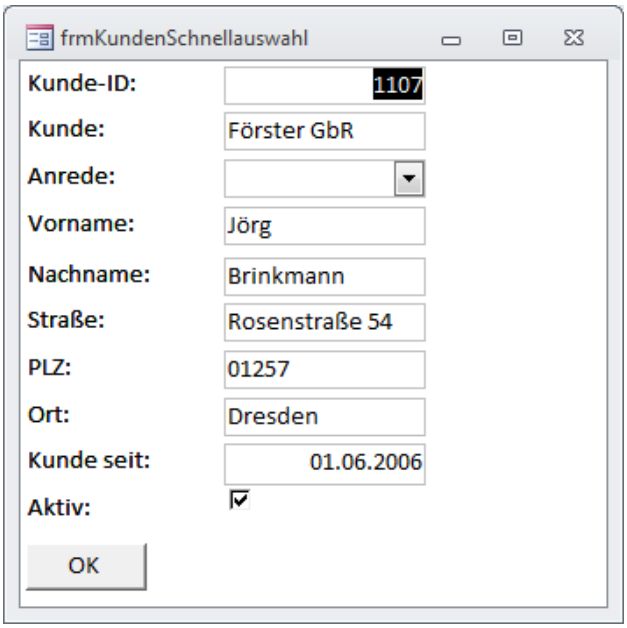


Bild 2: Der Ausgangspunkt: Eine Kopie des Formulars **frmKundenDetail** der Beispieldatenank

Beschriftung des Bezeichnungsfeldes auf **Schnellauswahl** und passen Sie außerdem den Namen des Kombinationsfeldes selbst auf **cboSchnellauswahl** an. Das Ergebnis sollte nun wie in Bild 4 aussehen.

Kombinationsfeld mit Kunden füllen

Anschließend sorgen wir dafür, dass das Kombinationsfeld die gewünschten Daten anzeigt. Die Grundlagen hierfür finden Sie übrigens im Artikel **Steuerelemente - Teil IV: Das Kombinationsfeld**, daher gehen wir hier nur auf die notwendigen Schritte ein.

Das Kombinationsfeld ist nicht an die Datenherkunft des Formulars gebunden, daher kann die Eigenschaft Steuerelementinhalt leer bleiben. Sie soll aber alle Kundenbezeichnungen anzeigen, die im Feld **Kunde** der Tabelle **tblKunden** zu finden sind.

Wir könnten nun faul sein und einfach die Tabelle **tblKunden** als Datensatzherkunft des Kombinationsfeldes festlegen, aber wir benötigen nicht alle Felder, sondern nur die beiden Felder **KundeID** und **Kunde**.

Wofür dient das Feld **KundeID**, wenn wir doch nur die Kundenbezeichnungen anzeigen möchten? Nun: Nach der Auswahl eines Kunden sollen seine Detailinformationen unten im Formular angezeigt werden.

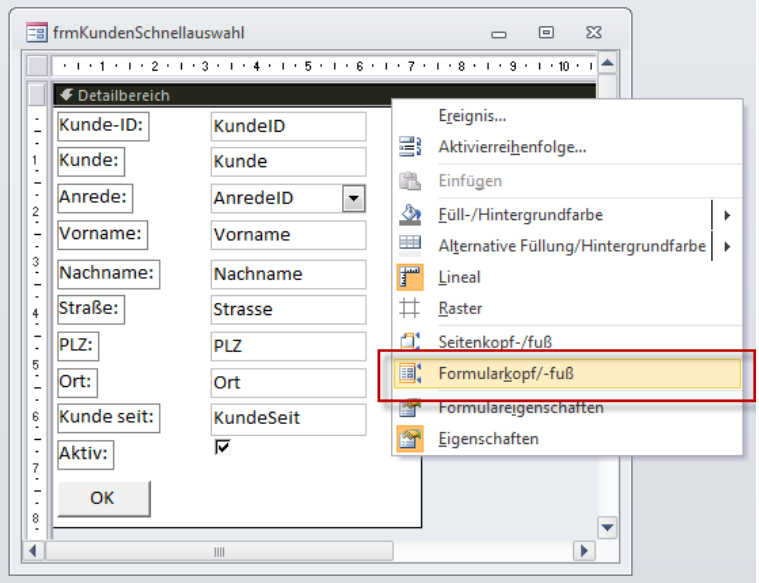


Bild 3: Aktivieren von Formularkopf und -fuß

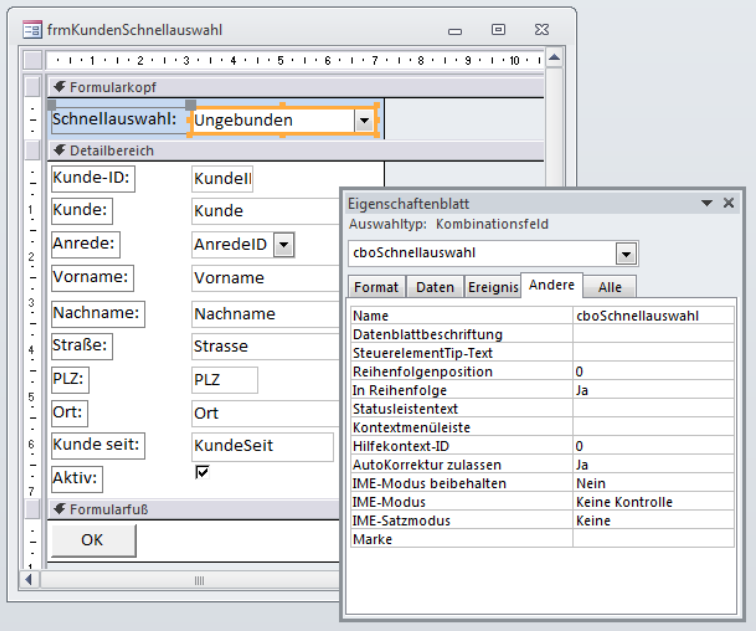


Bild 4: Hinzufügen eines Kombinationsfeldes zur Schnellauswahl

Dazu müssen wir zu dem gewünschten Datensatz springen. Ein Kundendatensatz wird eindeutig über das Primärschlüsselfeld **KundeID** identifiziert, also ziehen wir dieses hinzu.

Der Wert von **KundeID** soll Bestandteil des im Kombinationsfeld ausgewählten Eintrags sein, damit wir diesen Wert für das Filtern des Formulars nach diesem Kunden verwenden können.

Um nicht alle Felder der Tabelle als Datensatzherkunft des Kombinationsfeldes zu verwenden, sondern nur die beiden benötigten, definieren wir eine entsprechende Abfrage.

Die Abfrage wollen wir in diesem Fall nicht unter einem bestimmten Namen speichern und im Datenbankfenster/Navigationsbereich anzeigen, sondern einfach nur der Eigenschaft **Datensatzherkunft** des Kombinationsfeldes hinterlegen.

Dies funktioniert so:

- Klicken Sie auf die Eigenschaft **Datensatzherkunft** des Kombinationsfeldes.
- Klicken Sie dann auf die nun erscheinende Schaltfläche mit den drei Punkten (...).

- Wählen Sie im Dialog **Tabelle anzeigen** den Eintrag **tblKunden** aus und klicken Sie auf die Schaltfläche **Hinzufügen** und dann auf **Schließen**.

- Fügen Sie dem Entwurfsraster die beiden Felder **KundeID** und **Kunde** hinzu.

- Stellen Sie für das Feld **Kunde** in der Zeile **Sortierung** den Wert **Aufsteigend** ein (s. Bild 5).

Fertig! Sie können die Abfrage nun testweise in der Datenblattansicht betrachten und diese dann schließen.

Die Eigenschaft **Datensatzherkunft** des Kombinationsfeldes sollte nun einen SQL-Ausdruck enthalten, welcher die soeben im Abfrageentwurf eingegebene Abfrage repräsentiert (s. Bild 6).

Keine Sorge: Sie können die Abfrage problemlos wieder im Entwurf öffnen und erneut bearbeiten.

Damit das Kombinationsfeld nur den Inhalte des Feldes **Kunde** anzeigt, stellen Sie die beiden Eigenschaften **Spaltenanzahl** und **Spaltenbreiten** auf die Werte **2** und **0cm** ein. Dadurch wird die erste Spalte **KundeID** ausgeblendet, kann aber dennoch ausgewertet werden.

Außerdem stellen wir die Eigenschaft **Zeilenanzahl** auf **8** ein, damit das Kombinationsfeld im aufgeklappten Zustand nicht das komplette Formular überdeckt.

Kombinationsfeld mit Leben füllen

Fehlt nur noch ein kleiner Schritt: Das Formular soll nach der Auswahl eines Eintrags im Kombinationsfeld **cboKunden** den entsprechenden Kunden anzeigen.

Dazu gibt es technisch verschiedene Möglichkeiten. Sie könnten erstens nach der Auswahl zu dem Datensatz mit der ausgewählten **KundeID** springen oder zweitens das Formular nach dem ausgewählten Wert von **KundeID** filtern.

Welche Variante Sie wählen, hängt beispielsweise davon ab, ob der Benutzer außer über das Kombinationsfeld noch über andere Elemente wie etwa die Navigationsleiste durch die Datensätze navigieren kann.

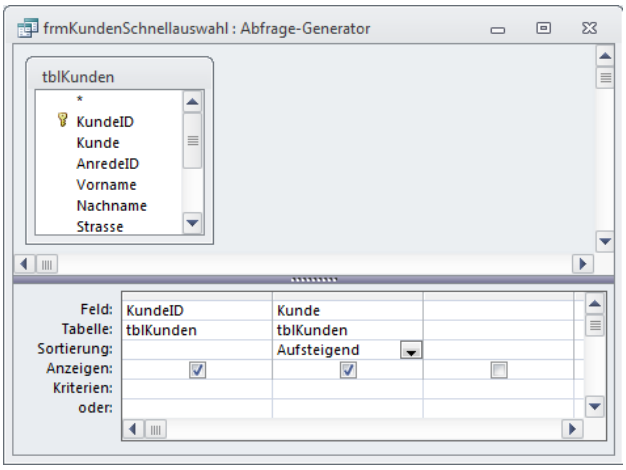


Bild 5: Abfrage als Datensatzherkunft des Kombinationsfeldes

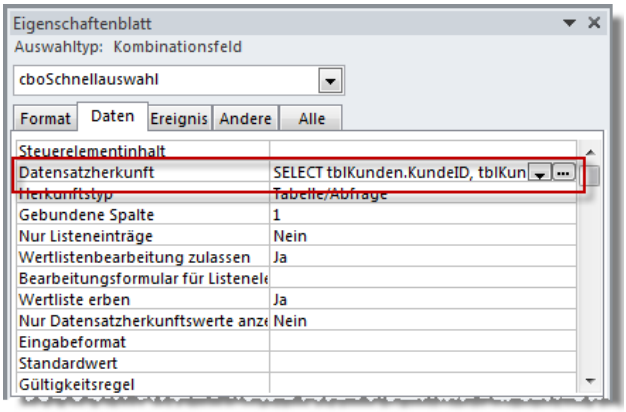


Bild 6: SQL-Ausdruck als Wert der Eigenschaft **Datensatzherkunft**

Da wir das Formular bereits von den Navigationsschaltflächen befreit haben, wenden wir die zweite Möglichkeit an.

Was muss nun geschehen? Als erstes müssen wir erkennen, wann der Benutzer einen neuen Eintrag im Kombinationsfeld **cboSchnellsuche** ausgewählt hat.

Dann müssen wir den Wert des Feldes **KundeID** des gewählten Kunden ermitteln und schließlich die Datenherkunft des Formulars so filtern, dass dieses nur noch den ausgewählten Datensatz anzeigt.

Schritt 1 ist einfach: Wechseln Sie in die Entwurfsansicht des Formulars und aktivieren Sie das Kombinationsfeld **cboSchnellsuche**. Zeigen Sie dann im Eigenschaftsfenster die Registerseite **Ereignisse** an. Dort finden wir schnell einen Eintrag namens **Nach Aktualisierung**.

Tabellen entwerfen

Teil V: Eigenschaften von Tabellen

Im vorherigen Teil dieser Artikelreihe haben wir uns die Eigenschaften der Felder einer Tabelle angesehen. Aber auch Tabellen selbst haben einige Optionen, die das Verhalten und das Aussehen steuern. Dieser Artikel beschreibt, welche Eigenschaften Tabellen haben und wie Sie diese einsetzen können.

Die Eigenschaften einer Tabelle finden Sie in der Entwurfsansicht. Klicken Sie hier mit der rechten Maustaste auf eines der Felder und wählen Sie den Kontextmenüeintrag Eigenschaften aus, um das Eigenschaftsfenster wie in Bild 1 anzuzeigen (hier unter Access 2010). Die folgenden Abschnitte stellen die Eigenschaften und ihre Bedeutung vor.

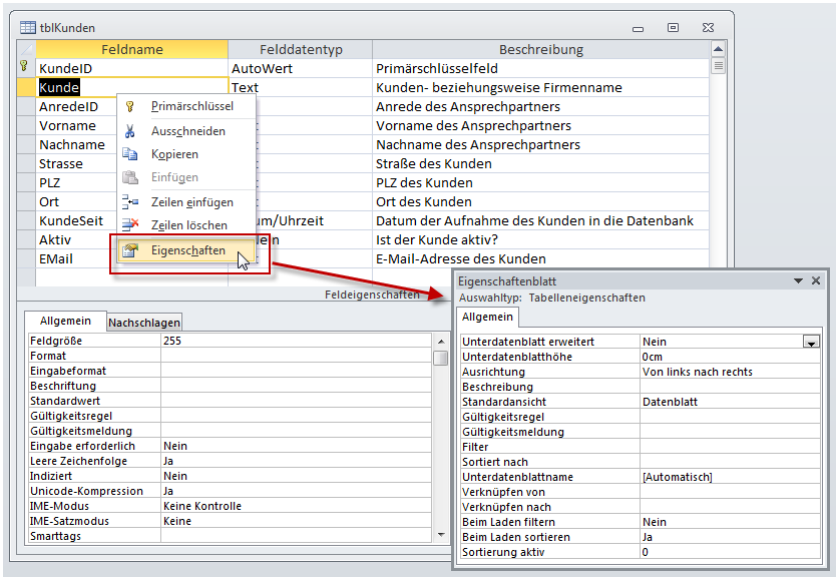


Bild 1: Anzeigen des Eigenschaftsfensters einer Access-Tabelle

Beschreibung (ab Access 2000)

Die Eigenschaft **Beschreibung** nimmt einen Text auf, der die Tabelle beschreibt. Dieser Text wird beispielsweise wie in Bild 2 im Navigationsbereich angezeigt (Access 2007 und jünger). Dazu müssen Sie allerdings den Eintrag **Anzeigen nach|Details** aus dem Kontextmenü des Navigationsbereich-Kopfes auswählen. Access 2003 und älter zeigen diesen Text im Datenbankfenster an (s. Bild 3).

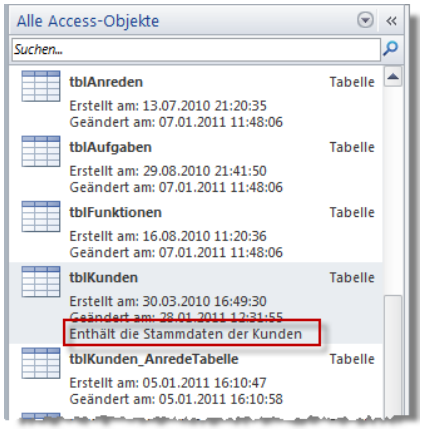


Bild 2: Beschreibung einer Tabelle im Navigationsbereich ...

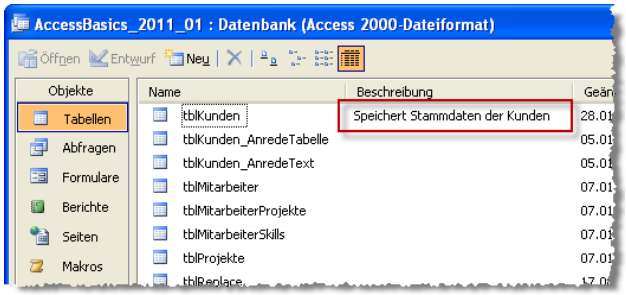


Bild 3: ... und im Datenbankfenster.

Standardansicht (2003)

Die Standardansicht kann die drei Werte **Datenblatt**, **PivotTable** und **PivotChart** annehmen. Die Datenblattansicht kennen Sie bereits, die beiden übrigen Ansichten nicht. Das werden wir an dieser Stelle auch nicht ändern – es würde schlicht den Rahmen sprengen. Stattdessen widmen wir diesen beiden Ansichten, die übrigens auch in Formularen und Berichten eine Rolle spielen, in einem eigenen Artikel in einer späteren Ausgabe von **Access [basics]**.

Gültigkeitsregel und Gültigkeitsmeldung (2000)

Diese beiden Eigenschaften kennen Sie eigentlich schon, da der oben genannte Artikel zum Thema Feldeigenschaften sie bereits vorgestellt hat. Allerdings bieten die Eigenschaften in Zusammenhang mit Tabellen andere Möglichkeiten. So können Sie diese beiden Eigenschaften nicht nur zum Prüfen

Der Ausdrucks-Generator

Ausdrücke schnell zusammenstellen mit einem praktischen Tool

Der Ausdrucks-Generator ist eines von vielen praktischen Helferlein, die gerade dem Einsteiger gute Dienste erweisen können. Mit dem Ausdrucks-Generator stellen Sie Ausdrücke für verschiedene Eigenschaften zusammen.

Der Ausdrucks-Generator ist an vielen Stellen verfügbar – in Tabellen, Abfragen, Formularen, Berichten und Makros. Ob Sie den Ausdrucks-Generator zum Formulieren eines Ausdrucks heranziehen können, erkennen Sie an der Schaltfläche mit den drei Punkten (...), die beim Aktivieren einer Eigenschaft rechts neben dieser erscheint. Aber Achtung: Bei manchen Eigenschaften aktiviert ein Klick auf diese Schaltfläche auch andere Assistenten, die beim Zuweisen eines Wertes helfen sollen. Dies ist etwa bei der Eigenschaft Eingabeformat eines Tabellenfeldes in der Entwurfsansicht der Fall: Dort wird etwa der Eingabeformat-Assistent gestartet.

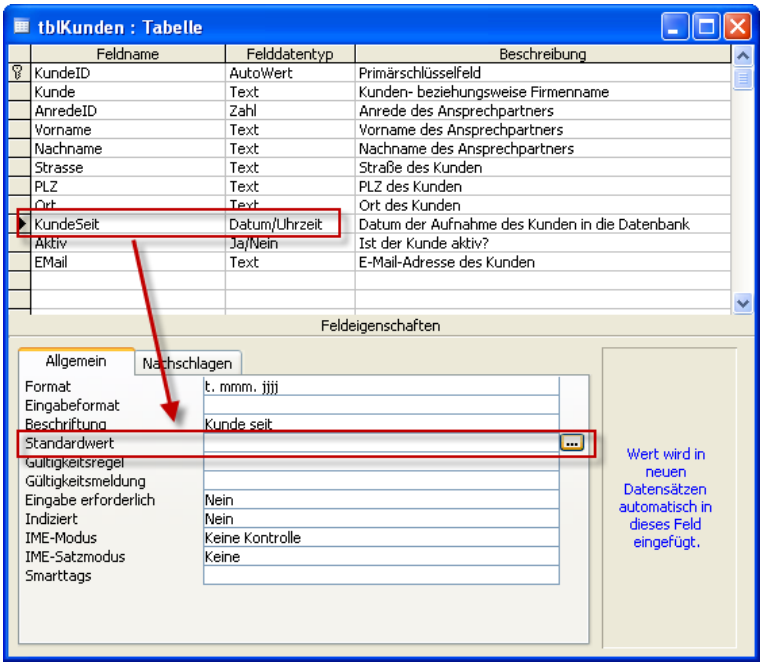


Bild 1: Eigenschaften eines Datumsfeldes

Statt durch einen Klick auf die Schaltfläche mit den drei Punkten können Sie den Ausdrucks-Generator auch mit der Tastenkombination **Strg + F2** öffnen (**Umschalt + F2** öffnet das Zoom-Fenster).

Aktuelles Datum als Standardwert

Die Funktionsweise lässt sich am einfachsten an einem Beispiel erläutern. Hier möchten wir als Standardwert für ein Datumsfeld das aktuelle Datum festlegen. Das bedeutet, dass beim Anlegen eines neuen Datensatzes automatisch das aktuelle Datum für dieses Feld eingetragen werden soll.

Wie Bild 1 zeigt, ist die erste Voraussetzung für dieses Feld gegeben: Beim Aktivieren der Eigenschaft **Datum** erscheint schon einmal die Schaltfläche mit den drei Punkten. Ein Klick öffnet dann auch prompt den Ausdrucks-Generator (s. Bild 2). Unter Access 2010 sieht der Ausdrucks-Generator übrigens etwas anders aus, auf die Änderungen gehen wir später ein.

Theoretisch könnten Sie den gewünschten Ausdruck nun einfach in das Feld oben eintragen. Wenn Sie genau wissen, welchen Inhalt Sie der Eigenschaft hinzufügen möchten, brauchen Sie den Ausdrucks-Generator jedoch gar nicht erst zu öffnen und können gleich im Textfeld der jeweiligen Eigenschaft zu Werke gehen (nach einiger Zeit kennen Sie sich mit den Ausdrücken wahrscheinlich auch so gut aus, dass Sie dies auf diese Weise erledigen werden).

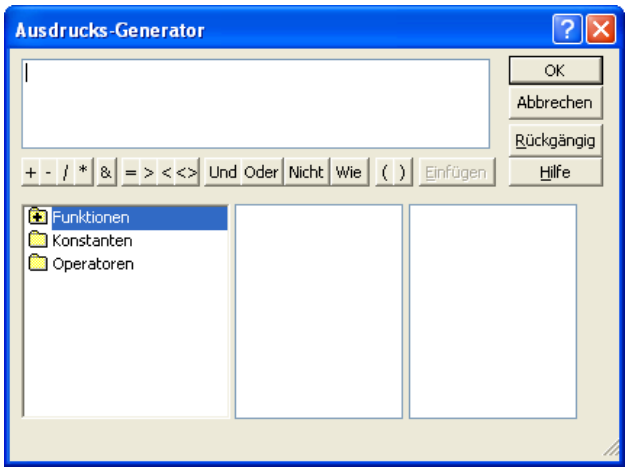


Bild 2: Der Ausdrucks-Generator unter Access 2007 und jünger

Also schauen wir uns an, wie der Ausdrucks-Generator beim Auffinden der benötigten Funktion helfen kann. Die drei Ordner im linken Listenfeld bieten die Grundlage für die Auswahl eines Ausdrucks,

der grundsätzlich aus einem oder mehreren Teilausdrücken besteht, die durch spezielle Operatoren verknüpft werden. Diese Operatoren fügen Sie ganz einfach über die Schaltflächen zwischen dem oberen Textfeld und den drei Listenfeldern ein.

Im vorliegenden Fall soll der Ausdruck jedoch nur das aktuelle Datum liefern. Im linken Listenfeld landen wir damit schnell im Ordner **Funktionen**, denn im Gegensatz zu **Konstanten** und **Operatoren** liefern nur Funktionen auch dynamische Werte wie etwa das Datum. Ein Doppelklick auf den Eintrag **Funktionen** öffnet den Unterordner **Eingebaute Funktionen**, der wiederum eine Reihe Einträge im mittleren und linken Listenfeld erscheinen lässt (wenn Sie öffentliche VBA-Funktionen programmiert haben, finden Sie hier für manche Eigenschaften einen weiteren Ordner – dazu später mehr).

Klicken Sie im mittleren Ordner auf **Datum/Uhrzeit** und wählen Sie rechts den Eintrag **Datum** aus. Diesen übertragen Sie entweder durch einen Klick auf die Schaltfläche **Einfügen** oder durch einen Doppelklick auf diesen Eintrag ins obere Textfeld. Das Ergebnis sieht schließlich wie in Bild 3 aus.

Nach dem Schließen des Ausdrucks-Generators mit der Schaltfläche **OK** finden Sie den Ausdruck in der Eigenschaft wieder. Wenn Sie in die Datenblattansicht der Tabelle wechseln und einen neuen Datensatz anlegen, trägt Access dort automatisch das aktuelle Datum ein (siehe Bild 4).

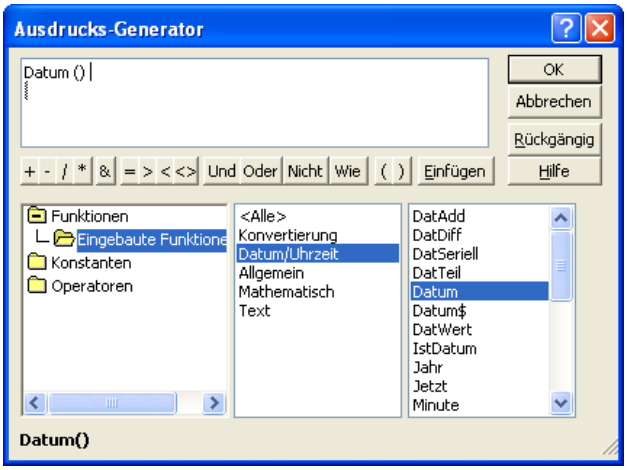


Bild 3: Die Funktion zum Einfügen des aktuellen Datums ist schnell gefunden.

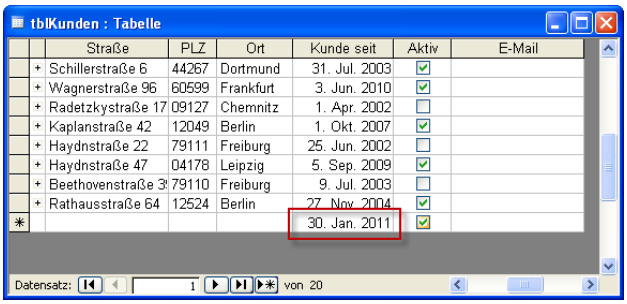


Bild 4: Automatisches Datum für neue Datensätze

Gültigkeitsregel mit dem Ausdrucks-Generator

Während Sie den Ausdruck **Datum()** schnell verinnerlichen werden, ist der Ausdrucks-Generator beim Anlegen eines Ausdrucks für die Gültigkeitsregel einer Tabelle schon hilfreicher. Hier kommt es schnell vor, dass man eines der Tabellenfelder referenzieren möchte, aber vielleicht den Namen nicht mehr kennt. Legen wir also eine Gültigkeitsregel für die Tabelle **tblProjekte** an, die festlegt, dass das Startdatum vor dem Enddatum liegen muss. Hierbei handelt es sich um eine Gültigkeitsregel auf Tabellenebene, nicht auf Feldebene.

Deshalb aktivieren Sie in der Entwurfsansicht der Tabelle das Eigenschaftsfenster, klicken in die Eigenschaft **Gültigkeitsregel** und dann auf die Schaltfläche mit den drei Punkten (siehe Bild 5).

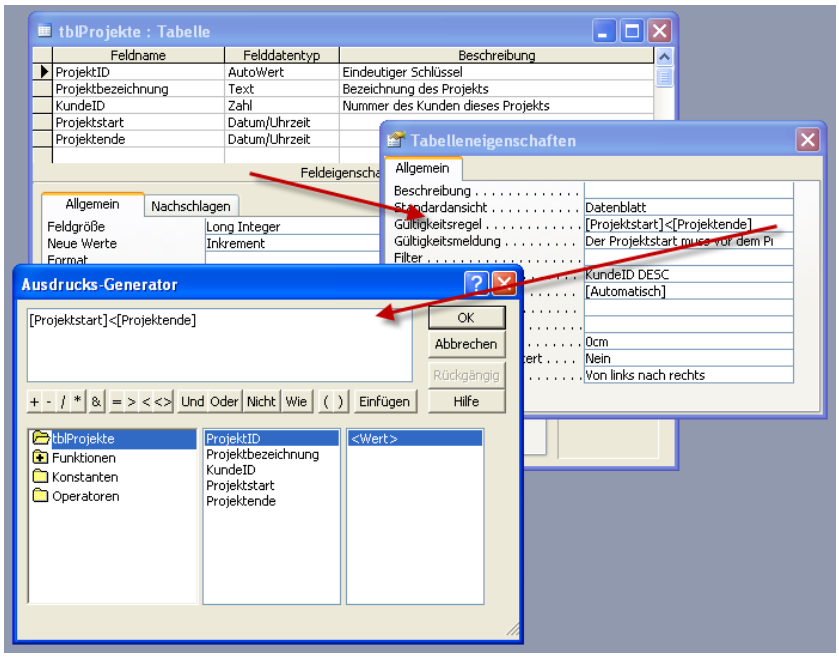


Bild 5: Erstellen einer Gültigkeitsregel mit dem Ausdrucks-Generator

Abfragen für die Datenauswahl

Teil VI: Berechnete Felder in Abfragen und Eigenschaften

Bislang haben wir in Abfragen nur die Inhalte einzelner Felder angezeigt, also beispielsweise Texte, Zahlen oder Datumsangaben. Sie können in einem Feld einer Abfrage aber auch gleich mathematische Berechnungen durchführen, Texte zusammenführen oder beliebige Feldinhalte mit eingebauten oder benutzerdefinierten Funktionen behandeln und das Resultat im Abfrageergebnis ausgeben. Wie das funktioniert, zeigen wir Ihnen in diesem Artikel.

Der Oberbegriff für alle Felder in Abfrageergebnissen, die nicht den Wert genau eines einzigen Feldes ausgeben, lautet „Berechnete Felder". Genaugenommen handelt es sich um Ausdrücke, die auf mehr als einem Feld basieren oder aber den Inhalt eines oder mehrere Felder durch eine Funktion verändert – natürlich sind auch Kombinationen möglich. Solche berechneten Felder gibt es unter anderem Namen auch in verschiedenen Eigenschaften von Formularen, Berichten und Steuerelementen oder auch als Kriterien in Abfragen. Wir konzentrieren uns in diesem Artikel jedoch auf die Verwendung berechneter Felder zur Ermittlung der in einem Abfrageergebnis anzuzeigenden Daten.

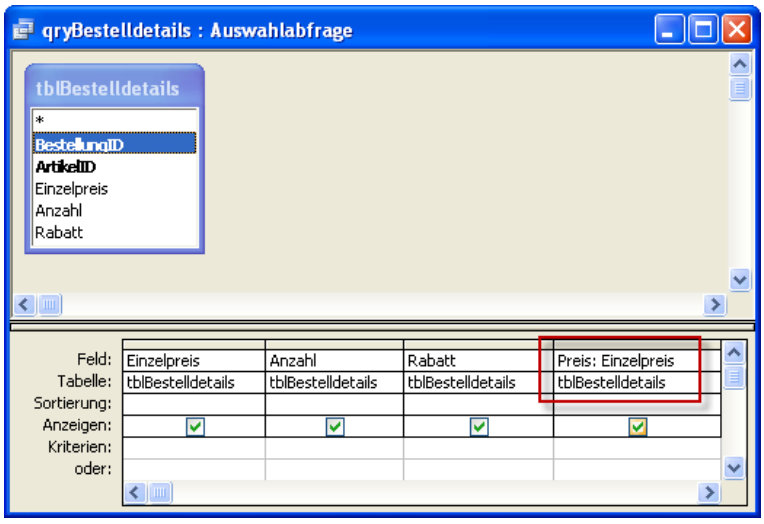


Bild 1: Ein berechnetes Feld, das einfach nur umbenannt wurde und den Wert eines anderen Feldes anzeigen soll

Berechnete Felder

Der Schritt von einem einfachen Abfragefeld zu einem berechneten Feld ist denkbar einfach: Sie brauchen dem Feld einfach nur einen neuen Namen zu geben und den Feldinhalt beizubehalten. Wir greifen für diesen Artikel auf die Beispieldatenbank **Südsturm.mdb** zurück, die Sie im Downloadbereich finden – diese Datenbank liefert besseres Futter für die Beispiele dieses Artikels.

Erstellen Sie zunächst eine neue Abfrage (siehe **Abfrage erstellen**) und fügen Sie die Tabelle **tblBestelldetails** hinzu. Ziehen Sie alle Felder dieser Tabelle in das Entwurfsraster. Nun legen wir ein neues berechnetes Feld in der ersten leeren Spalte rechts an. Es soll Preis heißen und den gleichen Inhalt wie das Feld **Einzelpreis** anzeigen. Dazu tragen Sie den folgenden Ausdruck dort ein, wo sonst die Feldnamen stehen (siehe Bild 1): **Preis: Einzelpreis**

Ein Wechseln in die Datenblattansicht zeigt schnell, dass dieses Feld tatsächlich genau den gleichen Inhalt wie das Feld **Einzelpreis** anzeigt – es hat sogar den gleichen Spaltennamen (siehe Bild 2).

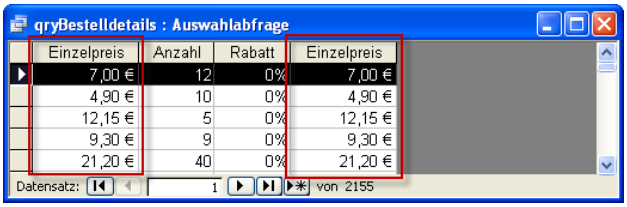


Bild 2: Das berechnete Feld und das Originalfeld sehen genau gleich aus.

Um aus diesem Feld ein richtiges berechnetes Feld zu machen, ersetzen wir den Ausdruck durch den folgenden:

```
Preis: [Einzelpreis]*[Anzahl]*(1-[Rabatt])
```

Wir möchten in dieser Spalte den Gesamtpreis für diese Position ermitteln, indem wir den Einzelpreis mit der Anzahl multiplizieren und außerdem den Rabatt hinzuziehen. Die erste Auswirkung dieser Maßnahme ist, dass in der Entwurfsansicht der Tabellenname aus der Zeile **Tabelle** verschwindet (siehe Bild 3).

Der Wechsel in die Datenblattansicht offenbart den zweiten Unterschied: Nun wird auch die vor dem Doppelpunkt angegebene Bezeichnung als Spalten-

kopf ausgegeben (siehe Bild 4). Außerdem liefert Access die richtige Berechnung für den Gesamtpreis dieser Position. Für die mathematische Berechnung brauchen Sie also nur die betroffenen Felder durch die entsprechenden Operatoren zu verknüpfen. Dabei können Sie auch Klammern verwenden, um die Reihenfolge der Berechnung anzupassen.

Halten wir also fest: Ein berechnetes Feld besteht aus dem neuen Feldnamen, einem Doppelpunkt sowie dem eigentlich anzuzeigenden Ausdruck.

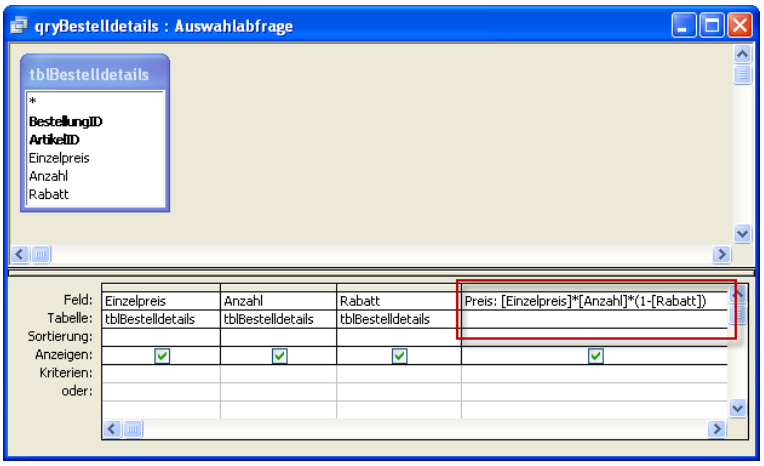


Bild 3: Besteht das berechnete Feld aus mehr als nur einem Feld der Datenherkunft der Abfrage, verschwindet die Anzeige der Herkunftstabelle.

Immer aktuell

Berechnete Felder sind immer aktuell. Wenn Sie also wie oben eine Berechnung auf zwei oder mehr Feldern durchführen lassen und während der Anzeige des Ergebnisses Änderungen an den zugrunde liegenden Feldern vornehmen, werden diese Änderungen gleich in das Berechnungsergebnis übernommen.

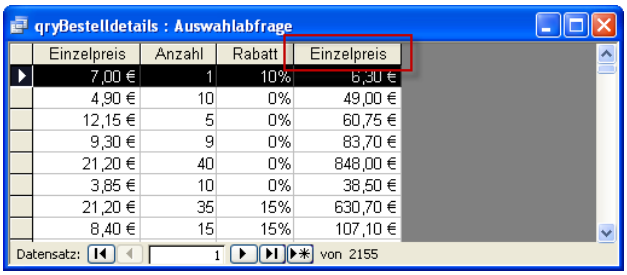


Bild 4: Außerdem zeigt der Spaltenkopf nun die Bezeichnung des berechneten Feldes an.

Einfachstes berechnetes Feld

Wenn Sie gedacht haben, die Angabe eines einfachen Feldes in einem berechneten Feld sei die einfachste Variante, haben Sie sich getäuscht: Sie können hier auch schlicht und einfach einen konstanten Ausdruck hinterlegen, beispielsweise wie folgt:

```
KonstanterAusdruck: 100
```

Die Abfrage liefert für das Feld **Konstanter Ausdruck** dann in jedem Datensatz den Wert **100**. Wie Sie hier erkennen können, kann der Feldname eines berechneten Feldes durchaus auch Leerzeichen und weiter Zeichen wie Umlaute, Bindestriche et cetera enthalten. Wir empfehlen jedoch, dies nicht zu tun und einen Ausdruck wie **KonstanterAusdruck** zu verwenden.

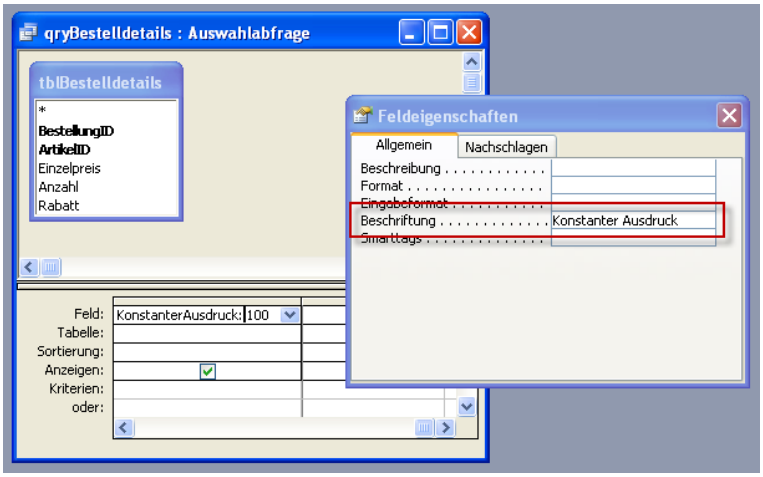


Bild 5: Anpassen des Spaltenkopfes eines berechneten Feldes

Soll die Abfrage in der Datenblattansicht eine abweichende Spaltenüberschrift anzeigen, legen Sie diese einfach in der Eigenschaft **Beschriftung** der Feldeigenschaften fest (siehe Bild 5).

Mathematische Funktionen

Access bietet eine Reihe mathematischer Funktionen an, die wir an anderer Stelle im Detail vorstellen.

Am einfachsten finden Sie diese Funktionen über den Ausdrucks-Generator, den wir im Artikel **Der Ausdrucks-Generator** genau beschreiben. Sie können sämtliche dieser Funktionen auch zur Definition berechneter Felder heranziehen.

Abfragen für die Datenauswahl

Teil VII: Benutzerdefinierte Funktionen in Abfragen

Wie Sie eingebaute VBA-Funktionen in berechneten Feldern von Abfragen einsetzen, haben Sie ja bereits unter Abfragen für die Datenauswahl - Teil VI: Berechnete Felder in Abfragen und Eigenschaften erfahren. In diesem Artikel zeigen wir Ihnen nun, wie Sie selbst erstellte VBA-Funktionen in Access-Abfragen einsetzen.

Manchmal reichen die eingebauten VBA-Funktionen einfach nicht aus, um in einer Abfrage ein Feld mit den gewünschten Informationen zu füllen.

Im oben genannten Artikel gibt es ein Beispiel, in dem aus dem Einzelpreis, der Anzahl und dem Rabatt für eine Bestellposition der Gesamtpreis berechnet wird. Dort verwenden wir gleich in einem berechneten Abfragefeld eine entsprechende Berechnungsvorschrift.

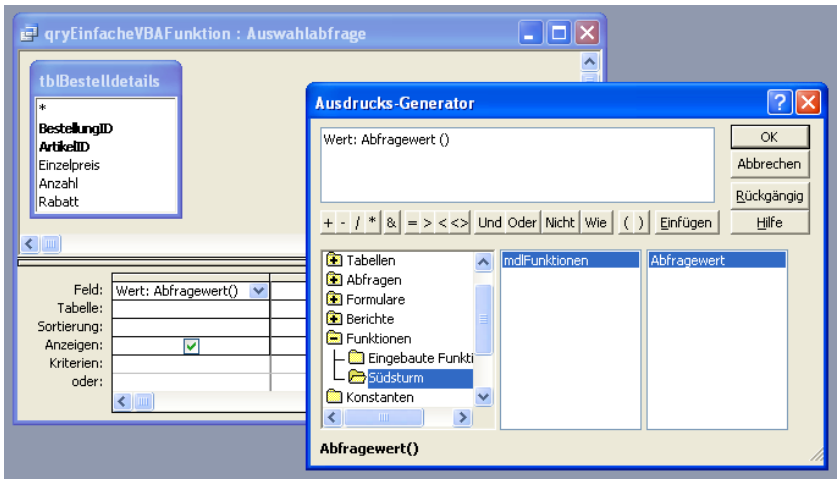


Bild 1: Hinzufügen eines berechneten Felds auf Basis einer benutzerdefinierten VBA-Funktion

Da wir uns an dieser Stelle noch nicht um komplizierte VBA-Funktionen kümmen möchten, wollen wir diese einfache Berechnung in eine VBA-Funktion auslagern.

Ergebnis einer VBA-Funktion in einer Abfrage

Wir starten jedoch zunächst mit dem einfachsten Fall: Die erste Beispielfunktion soll schlicht einen statischen Wert an die Abfrage zurückgeben. Die Funktion sieht wie folgt aus und wird in einem Standardmodul namens **mdlFunktionen** der Südwind-Beispieldatenbank angelegt (siehe **VBA-Modul anlegen** und **VBA-Routine schreiben**):

```
Public Function Abfragewert()
    Abfragewert = 100
End Function
```

Auf diese Funktion greifen Sie in der Abfrage am einfachsten über den Ausdrucks-Generator zu (siehe auch **Der Ausdrucks-Generator**).

Wenn Sie den Namen der Funktion gerade im Kopf haben, brauchen Sie diesen allerdings gar nicht – tragen Sie einfach den Namen des Feldes sowie den Funktionsaufruf mit einem Klammernpaar in die Zeile **Feld** der Abfrage ein:

```
Wert: Abfragewert()
```

Beachten Sie, dass die Abfrage mindestens eine Tabelle als Datenherkunft haben muss, da diese sonst gar nicht erst in der Datenblattansicht angezeigt werden kann. Im Beispiel aus Bild 1 haben wir bereits die Tabelle **tblBestelldetails** hinzugefügt. Allerdings soll diese erste Beispielabfrage keine Felder der zugrunde liegenden Tabelle anzeigen, sondern nur das Ergebnis der Funktion **Abfragewert**.

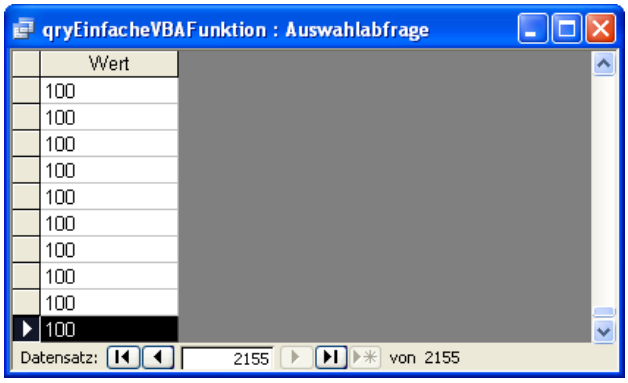


Bild 2: Das Ergebnis der Abfrage mit dem berechneten Feld

Access-Funktionen

Teil I: Domänenfunktionen

In der Reihe Access-Funktionen stellen wir Ihnen in Access eingebaute Funktionen vor, die Sie sowohl in Eigenschaften von Tabellen, Abfragen, Formularen und Berichten als auch in VBA-Routinen einsetzen können. Im ersten Teil dieser Artikelreihe beschäftigen wir uns mit den sogenannten Domänenfunktionen. Auch wenn die Bezeichnung nicht bei jedem gleich die richtige Assoziation auslöst: Hier geht es um Funktionen, mit denen Sie auf die in Tabellen gespeicherten Daten zugreifen können.

Domänenfunktionen ermöglichen den Zugriff auf genau einen Wert einer Datensatzgruppe, also eines oder mehrere Datensätze einer Tabelle oder Abfrage. Sie können damit beispielsweise folgende Informationen abfragen:

- Vorname des Kunden mit der Kunden-ID **12345** aus der Tabelle tblKunden
- Anzahl der Artikel aus einer Tabelle **tblArtikel**, die teurer als 100 Euro sind
- Preis des teuersten Artikels der Tabelle **tblArtikel**
- Durchschnittspreis aller Artikel der Tabelle **tblArtikel**

Einsatzorte

Domänenfunktionen und Access-Funktionen im allgemeinen setzen Sie beispielsweise an folgenden Stellen ein:

- Feldinhalt von Abfragen
- Kriterium von Abfragen
- Steuerelementinhalt von Steuerelementen in Formularen oder Berichten

Grundlagen

Grundsätzlich sind Domänenfunktionen mit Abfragen vergleichbar, die jedoch eine entscheidende Einschränkung aufweisen müssen: Sie dürfen das Ergebnis in nur einen einzigen Datensatz mit einem einzigen Feld zurückliefern.

Der Begriff **Domänenfunktion** wird vom englischen Domain abgeleitet. Domain bezieht sich im Datenbankbereich auf alle Werte eines Feldes einer Tabelle oder Abfrage. Die Domänenfunktionen erlauben es, die Werte dieser „Domäne" auf verschiedene Arten abzufragen.

Für alle Domänenfunktionen gibt es eine englische Version, die grundsätzlich unter VBA zum Einsatz kommt, sowie eine Version in der jeweiligen Anwendungssprache. Diese Variante verwenden Sie in jeglichen Eigenschaften, die Sie über die Benutzeroberfläche von Access in den verschiedenen Entwurfsansichten einstellen.

Sie können den Namen einer Domänenfunktion als Feldwert, Kriterium oder Eigenschaftswert zwar auch in Englisch angeben – Access benennt den Namen allerdings direkt nach dem Eingeben in die Anwendungssprache um. Es gibt noch einen weiteren wesentlichen Unterschied bei der Eingabe von Access-Funktionen in VBA-Routinen auf der einen und Feldwerten, Kriterien und Eigenschaften auf der anderen: Unter VBA trennen Sie die Parameter durch Kommata voneinander, sonst durch Semikola.

Domänenfunktionen im Überblick

Access stellt insgesamt zwölf Funktionen zur Verfügung, die allesamt mit „D" beginnen. Die folgende Auslistung liefert die deutsche Bezeichnung inklusive Syntax, die englische Bezeichnung mit Syntax und eine Erläuterung. Für alle Domänenfunktionen gilt: Sie berücksichten nur solche Datensätze, in denen der unter Ausdruck angegebene Wert nicht NULL ist, das Feld beziehungsweise die Felder müssen also einen Wert enthalten. Außerdem werden natürlich nur solche Datensätze berücksichtigt, die dem unter **Kriterium** angegebenen Ausdruck entsprechen:

- **DomAnzahl(Ausdruck;Domäne[;Kriterium]**
  DCount(Ausdruck,Domäne[,Kriterium])
  Liefert die Anzahl der Datensätze.
- **DomErsterWert(Ausdruck;Domäne[;Kriterium]**
  DFirst(Ausdruck,Domäne[,Kriterium])
  Liefert den Feldinhalt des ersten Datensatzes.

- **DomLetzterWert(Ausdruck;Domäne[;Kriterium])**
  DLast(Ausdruck,Domäne[,Kriterium])
  Liefert den Feldinhalt des letzten Datensatzes.

- **DomMax(Ausdruck;Domäne[;Kriterium])**
  DMax(Ausdruck,Domäne[,Kriterium])
  Liefert den maximalen Wert eines Feldes.

- **DomMin(Ausdruck;Domäne[;Kriterium])**
  DMin(Ausdruck,Domäne[,Kriterium])
  Liefert den minimalen Wert eines Feldes.

- **DomMittelwert(Ausdruck;Domäne[;Kriterium])**
  DAvg(Ausdruck,Domäne[,Kriterium])
  Liefert den Durchschnittswert des Feldinhaltes. Nullwerte werden ignoriert.

- **DomStAbw(Ausdruck;Domäne[;Kriterium])**
  DStDev(Ausdruck,Domäne[,Kriterium])
  Liefert die Standardabweichung eines Feldes, bezogen auf eine Stichprobe.

- **DomStAbwn(Ausdruck;Domäne[;Kriterium])**
  DStDevP(Ausdruck,Domäne[,Kriterium])
  Liefert die Standardabweichung eines Feldes, bezogen auf die Gesamtmenge.

- **DomSumme(Ausdruck;Domäne[;Kriterium])**
  DSum(Ausdruck,Domäne[,Kriterium])
  Liefert die Summe der Feldinhalte.

- **DomVarianz(Ausdruck;Domäne[;Kriterium])**
  DVar(Ausdruck,Domäne[,Kriterium])
  Liefert die Varianz eines Feldes, bezogen auf eine Stichprobe.

- **DomVarianzen(Ausdruck;Domäne[;Kriterium])**
  DVarP(Ausdruck,Domäne[,Kriterium])
  Liefert die Varianz eines Feldes, bezogen auf die Gesamtmenge.

- **DomWert(Ausdruck;Domäne[;Kriterium])**
  DLookup(Ausdruck,Domäne[,Kriterium])
  Liefert den Feldinhalt des ersten Datensatzes.

Beispieldatenbank

Die Beispiele zu diesem Artikel finden Sie in der Beispieldatenbank **Südsturm.mdb**.

Domänenfunktionen einsetzen

Zum einfachen Ausprobieren der Domänenfunktionen erstellen Sie zunächst ein einfaches Formular und fügen diesem ein Textfeld hinzu. Nennen Sie dieses **txtDCount** und weisen Sie seiner Eigenschaft **Steuerelementinhalt** den folgenden Ausdruck zu (siehe Bild 1):

```
=DomAnzahl("ArtikelID";"tblArtikel")
```

Sie sehen hier bereits, dass die Bezeichnungen von Feldern und Tabellen/Abfragen normalerweise in Anführungszeichen eingefasst und somit als Zeichenkette übergeben werden. Das gilt übrigens auch für das Kriterium.

Es gibt jedoch auch Ausnahmen, zum Beispiel wenn ein Parameter in einem Steuerelement oder einer Variablen enthalten ist – dazu später mehr.

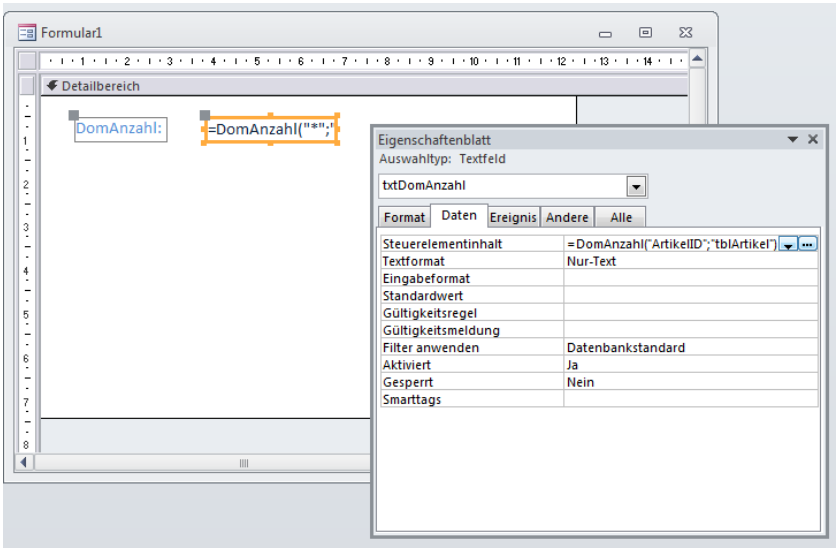


Bild 1: Textfeld mit einer Domänenfunktion als Steuerelementinhalt

Wenn Sie in die Formularansicht wechseln, zeigt das Textfeld die Anzahl der Datensätze der Tabelle **tblArtikel** an.

Dieser Funktionsaufruf liefert die Anzahl aller Datensätze der Tabelle **tblArtikel**, deren Feld **ArtikelID** einen Wert enthält. Da **ArtikelID** das Primärschlüsselfeld der Tabelle ist, enthält jeder Datensatz einen Wert für dieses Feld. Die **DomAnzahl**-Funktion liefert also die Gesamtzahl der enthaltenen Datensätze zurück.

Pflichtprogramm

Sie müssen für jede Domänenfunktion zumindest die ersten beiden Parameter übergeben. Der Parameter **Ausdruck** enthält eine der folgenden Möglichkeiten:

- Feldname: Gibt das auf dieses Feld bezogene Ergebnis der jeweiligen Domänenfunktion zurück.
- Ausdruck: Sie können auch einen beliebigen, aus mehreren Feldnamen und Operatoren zusammengesetzten Ausdruck angeben.
- Sternchen (*): Nur für die DomAnzahl-Funktion. Legt fest, das alle Datensätze gezählt werden und nicht nur diejenigen, die für das statt des Sternchens angegebene Feld einen Wert enthalten.

Für **Domäne** geben Sie den Namen einer beliebigen Tabelle oder Abfrage an.

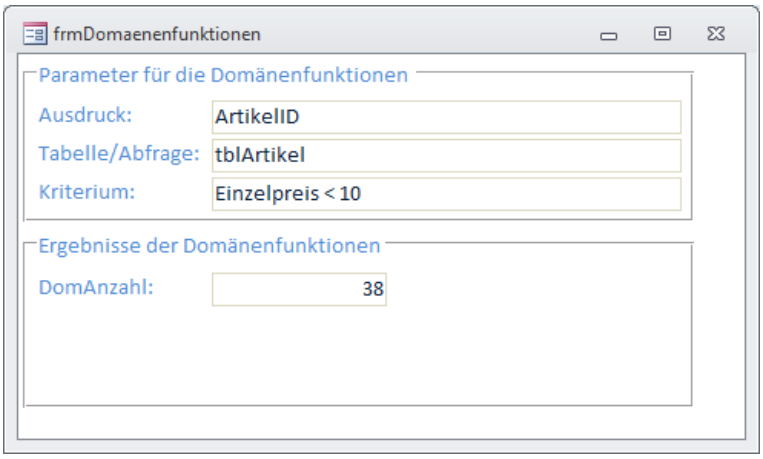


Bild 2: Parameter und Ergebnisse der Domänenfunktionen im Formular

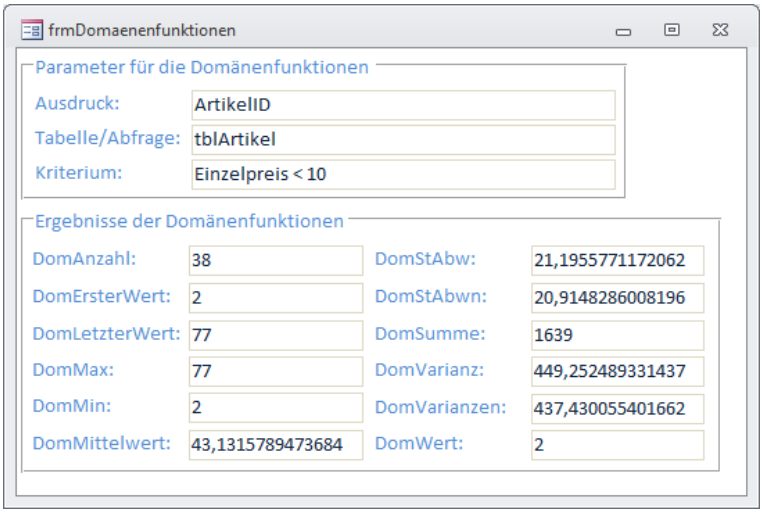


Bild 3: Ergebnisse aller Domänenfunktionen im Überblick

Kriterium verwenden

Im ersten Beispiel haben wir noch kein Kriterium angegeben. Ändern Sie den Wert der Eigenschaft Steuerelementinhalt des Textfeldes nun auf den folgenden Ausdruck:

```
=DomAnzahl("*";"tblArtikel";"Einzelpreis < 10")
```

Dies liefert die Anzahl aller Datensätze der Tabelle **tblArtikel**, deren Preis kleiner als zehn Euro ist.

Als Kriterium können Sie genau die gleichen Ausdrücke angeben, die Sie in Abfragen verwenden – also beispielsweise wie in **Abfragen für die Datenauswahl - Teil III: Filtern von Textfeldern** oder **Abfragen für die Datenauswahl - Teil V: Filtern nach Zahlen**.

Fasst man die drei Kriterien zusammen, könnte man diese auch als SQL-Abfrage formulieren, schematisch also etwa wie die Abfrage **SELECT Ausdruck FROM Domäne WHERE Kriterium**.

Dabei gilt die bereits erwähnte Einschränkung, dass das Ergebnis nur einen einzigen Wert zurückliefern darf.

Beispielformular erweitern

Als nächstes erweitern wir das Beispielformular so, dass Sie ein wenig mit den verschiedenen Domänenfunktionen experimentieren können. Dazu fügen Sie zunächst drei Textfelder mit folgenden Namen zum Formular hinzu:

- **txtAusdruck**
- **txtTabelleAbfrage**
- **txtKriterium**

Danach ändern Sie die Eigenschaft **Steuerelementinhalt** des Textfeldes **txtDomAnzahl** erneut ab, und zwar so:

```
=DomAnzahl(txtAusdruck;txtTabelleAbfrage;
txtKriterium)
```

Abfragen für die Datenauswahl

Teil VIII: Abfrageeigenschaften

Die Entwurfsansicht für Abfragen ist ja eigentlich nur eine vereinfachte Eingabemöglichkeit für SQL-Ausdrücke. Tabellen, Felder und Kriterien lassen sich prima im Entwurfsraster festlegen. Aber es gibt noch eine Reihe weiterer Abfrageeinstellungen, für die der Abfrageentwurf ein eigenes Eigenschaftsfenster vorsieht. In diesem Teil der Artikelreihe schauen wir uns an, welche Eigenschaften das sind und wie Sie diese einsetzen können.

Die Eigenschaften einer Abfrage bekommt man nur zufällig zu sehen, wenn man es nicht gerade darauf anlegt. Das liegt daran, dass es entweder gerade die Eigenschaften eines Feldes der zugrunde liegenden Tabelle oder die Eigenschaften eines der Abfragefelder anzeigt – je nachdem, was Sie zuletzt bearbeitet haben (vorausgesetzt, das Eigenschaftsfenster ist überhaupt aktiviert – ansonsten holen Sie dies schnell mit der Taste **F4** nach).

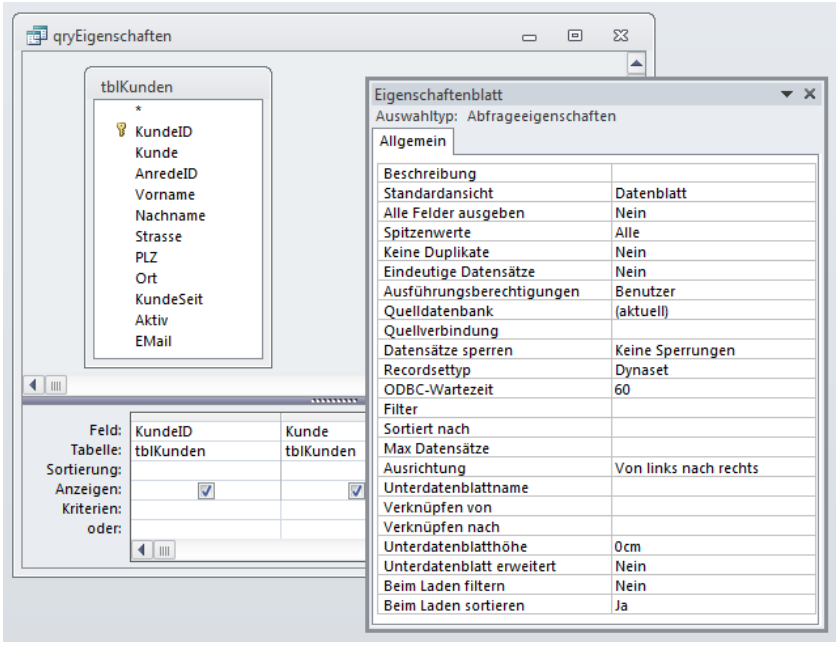


Bild 1: Das Eigenschaftsfenster einer Abfrage

Die Eigenschaften der Abfrage selbst bekommen Sie erst zu Gesicht, wenn Sie in den grauen, leeren Bereich oben in der Abfrage klicken (bei aktiviertem Eigenschaftsfenster) oder dort den Kontextmenüeintrag **Eigenschaften** auswählen.

Dann erscheint der Dialog aus Bild 1 (hier unter Access 2010 – jüngere Versionen stellen einige Eigenschaften weniger bereit, dazu später mehr).

Die Eigenschaften **Beschreibung**, **Standardansicht**, **Filter**, **Sortiert nach**, **Ausrichtung**, **Unterdatenblattname**, **Verknüpfen von**, **Verknüpfen nach**, **Unterdatenblatthöhe**, **Unterdatenblatt erweitert**, **Beim Laden filtern** und **Beim Laden sortieren** funktionieren wie die gleichnamigen Eigenschaften von Tabellen (siehe **Tabellen entwerfen – Teil V: Eigenschaften von Tabellen**).

Die übrigen Eigenschaften schauen wir uns in den folgenden Abschnitten an. Die Beispiele zu diesem Artikel finden Sie in der Datenbank **Südsturm.mdb**.

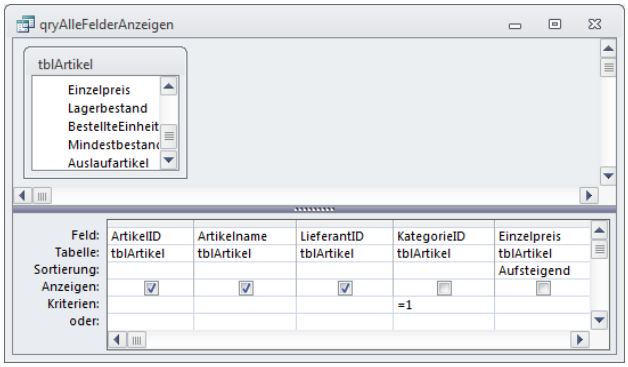


Bild 2: Eine Abfrage mit Feldern, die nicht im Ergebnis erscheinen sollen

Alle Felder ausgeben

Wenn Sie die Felder der Tabellen einer Abfrage zum Entwurfsraster hinzufügen, können Sie noch festlegen, ob diese auch angezeigt werden. Dies erledigen Sie durch Setzen oder Entfernen des Kontrollkästchens in der Zeile **Anzeigen**.

Warum sollte man Felder in das Entwurfsraster ziehen, wenn man diese gar nicht anzeigen möchte? Nun: Manchmal dienen Felder nur als Sortier- oder Filterkriterium, und dann müssen Sie auch in der Abfrage enthalten sein.

Adressblöcke in Berichten

Anschreiben mit Empfänger-Adressen ausstatten

Einer der wichtigsten Anwendungsfälle für Berichte ist die Ausgabe von Anschreiben und hier besonders von Rechnungen. Für das Anschreiben ist zunächst einmal die richtige Adressierung wichtig. Dieser Artikel zeigt, wie Sie Daten wie Anrede, Vorname, Nachname, Straße, PLZ, Ort, Land und weitere Informationen dort unterbringen.

Beispieldatenbank

Die in diesem Artikel vorgestellten Objekte finden Sie in der Beispieldatenbank **AccessBasics_2011_02.mdb**. Dort haben wir zunächst die Tabelle **tblKunden** um das Feld **Land** erweitert, das Einträge wie **Deutschland**, **Österreich**, **Schweiz** und so weiter aufnehmen soll.

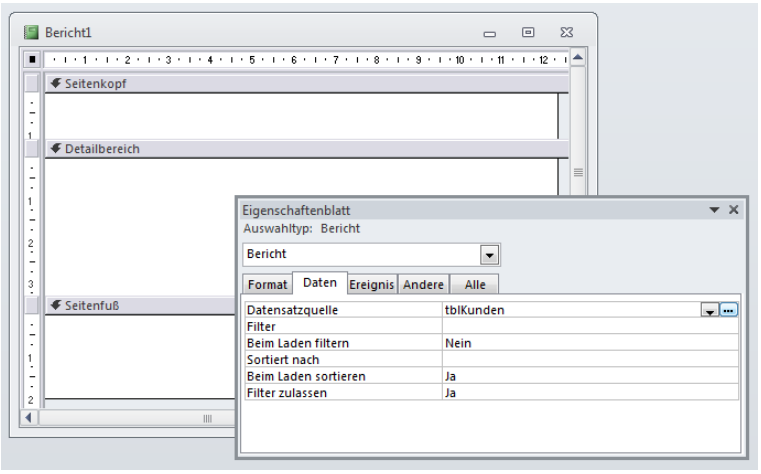


Bild 1: Zuweisen der Datenherkunft zum Bericht

Grobkonzept

Wenn Sie einen Bericht erstellen, der ein Adressfeld enthält, handelt es sich in der Regel um eine Rechnung oder ein ähnliches Dokument. Die Erstellung einer Rechnung als Access-Bericht kann sehr aufwendig werden, je nachdem, welche Elemente Sie dort integrieren möchten. Informationen wie Datum und Seitenzahlen sind noch einfach handzuhaben, aber wenn es an Zwischensummen, Überträge und ähnliche Informationen geht oder wenn Sie mehrere Rechnungen in einem Bericht drucken wollen, kann es kompliziert werden.

Dies bedeutet in erster Linie: Überlegen Sie sich vorab, welche Elemente Sie im Bericht anzeigen möchten und in welchen Bereichen des Berichts diese folglich landen. In folgenden Artikeln kommen wir auf dieses Thema zurück. Zuerst wollen wir uns jedoch auf die reinen Adressfelder konzentrieren, und auch diese lassen sich nicht so leicht zusammenstellen, wie man es sich vorstellen mag.

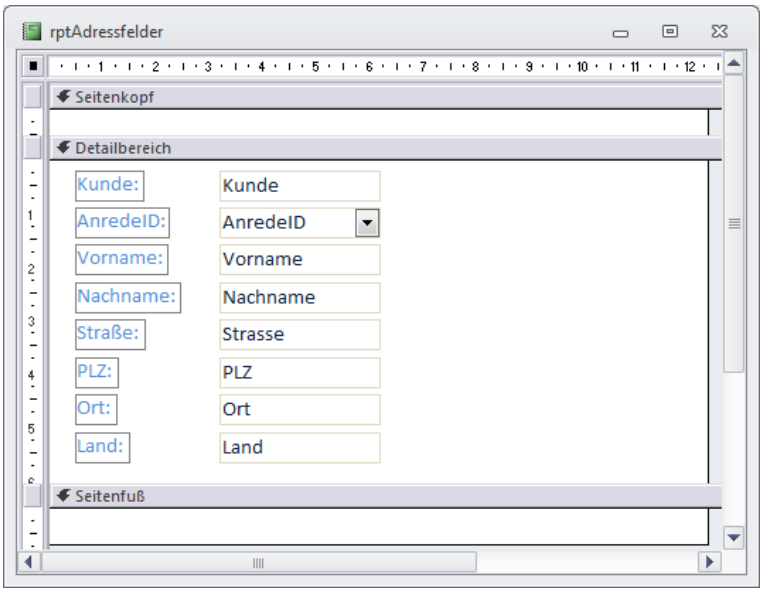


Bild 2: Adressfeld, ganz grob

Welcher Bereich?

Selbst für dieses Beispiel, dass nur die Erstellung von Adressfeldern beschreiben soll, brauchen wir ein kleines Konzept: Also legen wir fest, dass der Bericht je ein Adressfeld für jeden Datensatz der Tabelle **tblKunden** ausgeben soll. Für die Ausgabe von Daten für jeden Datensatz der Datenherkunft ist grundsätzlich der Detailbereich zuständig: Dieser wird für jeden Datensatz standardmäßig genau einmal angezeigt.

Da dieser Bericht sonst keine Daten anzeigen soll, reicht der Detailbereich aus – die übrigen Bereiche kommen nicht zum Einsatz.

Bericht erstellen

In einem ersten Ansatz erstellen Sie einen neuen, leeren Bericht und weisen seiner Eigenschaft **Datenherkunft** (seit Access 2007 **Datensatzquelle**) die Tabelle **tblKunden** zu (siehe Bild 1).

Danach können Sie gleich beginnen, Textfelder mit dem entsprechenden Steuerelementinhalt in den Detailbereich des Berichts zu ziehen. Für den Anfang ziehen Sie einfach die Felder **Kunde**, **AnredeID**, **Vorname**, **Nachname**, **Strasse**, **PLZ**, **Ort** und **Land** in den Detailbereich der Entwurfsansicht. Diesen verkleinern Sie gleich so, dass kein Platz in der Vertikalen verschenkt wird.

Bei dieser Gelegenheit speichern Sie den Bericht gleich unter dem Namen **rptAdressfelder**. Das Zwischenergebnis sieht wie in Bild 2 aus.

Textfelder anordnen

Die Bezeichnungsfelder brauchen Sie für diese Lösung nicht, also löschen Sie diese komplett. Außerdem orden Sie die einzelnen Felder zumindest schematisch schon einmal so an, wie diese auch später in der Berichtsausgabe erscheinen sollen (siehe Bild 3).

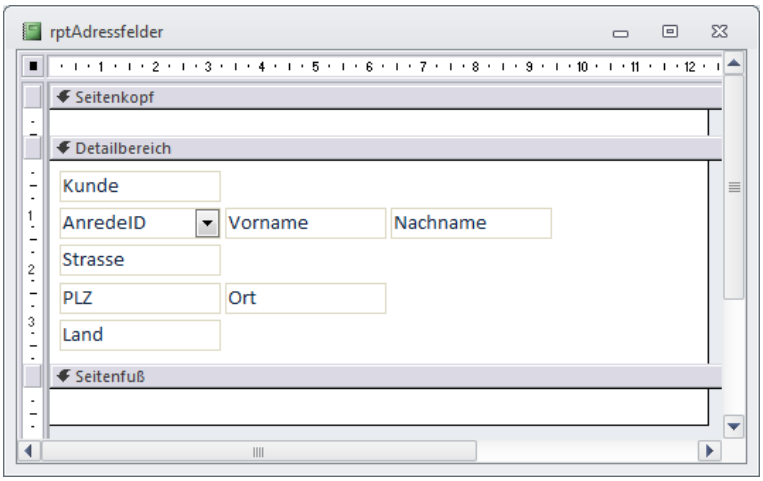


Bild 3: Adressfeld, ganz grob

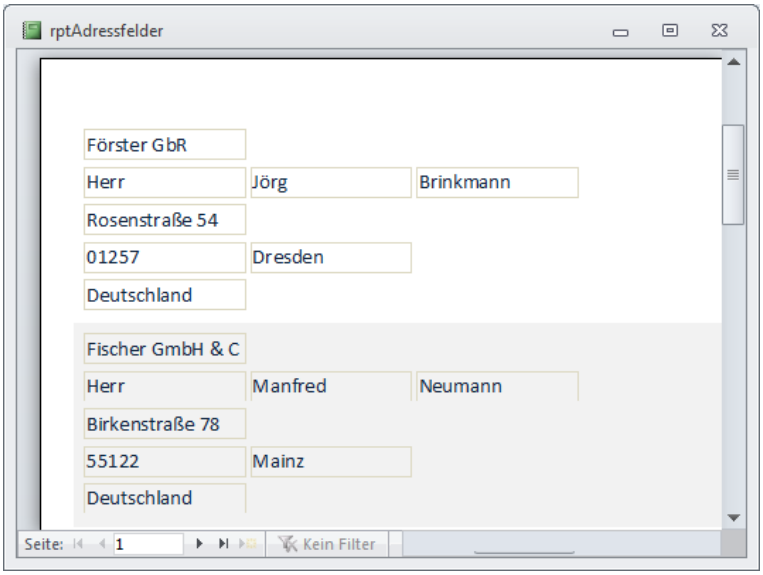


Bild 4: Ein erster Wechseln in die Seitenansicht zeigt dieses Bild.

An dieser Stelle wechseln Sie das erste Mal in die Seitenansicht des Berichts. Wie Bild 4 zeigt, liegen Sie zumindest schon einmal nicht ganz falsch. Access ist sogar so schlau, das Kombinationsfeld mit dem Steuerelementinhalt **AnredeID** nicht als Kombinationsfeld, sondern als Textfeld darzustellen.

Anrede, Vor- und Nachname zusammensetzen

Nun finden sich allerdings einige hässliche Zwischenräume etwa zwischen der Anrede, dem Vornamen und dem Nachnamen. Bei der Anrede, die ja nur die Werte **Herr** und **Frau** annehmen kann, ist dies kein Problem – dieses Textfeld könnten wir einfach ein wenig schmaler machen. Aber wie sollen wir die optimale Breite für das Textfeld zur Anzeige des Vornamens ermitteln?

Die Antwort ist einfach: Es geht schlicht und einfach nicht. Zum Glück können Sie aber in Berichten genau wie in Abfragen berechnete Felder verwenden (siehe auch **Abfragen für die Datenauswahl - Teil VI: Berechnete Felder in Abfragen und Eigenschaften**).

Wie Sie Vor- und Nachname zusammenfassen, haben Sie dort schon erfahren. In unserem Fall brauchen wir die beiden Informationen in der folgenden Reihenfolge und durch ein Leerzeichen voneinander getrennt:

```
=[Vorname] & " " & [Nachname]
```

Und wo wir schon einmal dabei sind, nehmen wir auch noch die Anrede mit hinzu:

```
=[AnredeID] & " " & [Vorname] & " " & [Nachname]
```

Access-Funktionen

Teil II: Bedingungen: Wenn(), Schalter() und Wahl()

Das es in VBA-Routinen Bedingungen wie If...Then, Select Case et cetera gibt, ist wohl bekannt. Weniger bekannt sind die Funktionen, mit denen Sie auch in Access-Ausdrücken logische Entscheidungen treffen können. Die Möglichkeiten halten sich zwar gegenüber den entsprechenden VBA-Strukturen in Grenzen, aber dennoch ersparen diese hin und wieder den Einsatz einer VBA-Funktion.

Es gibt insgesamt drei solcher Funktionen, die ein Äquivalent unter VBA besitzen. Diese funktionen kurz gefasst wie folgt:

- **Wenn** (**IIf**): Prüft den als ersten Paramter angegeben Ausdruck auf True oder False und gibt dann abhängig vom Ergebnis den als zweiten beziehungsweise dritten Parameter angegebenen Ausdruck aus.

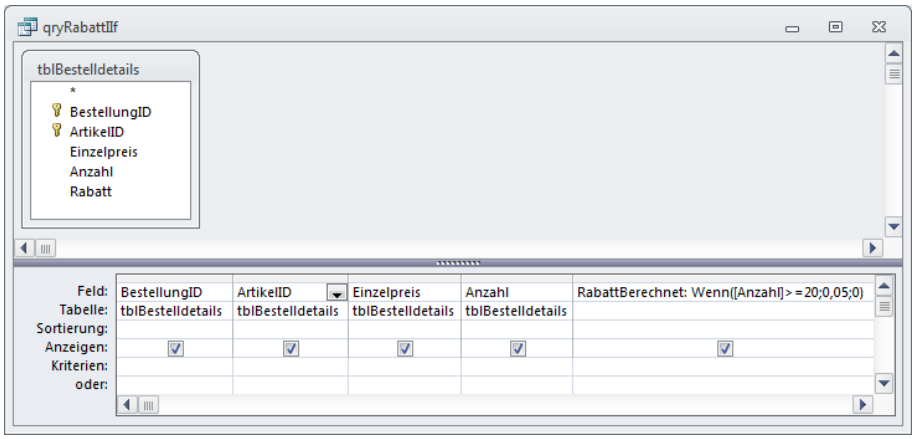


Bild 1: Eine Wenn-Funktion in einer Abfrage

- **Wahl** (**Choose**): Wertet den als ersten Parameter übergebenen Index von 1 bis n aus und liefert den ersten bis n-ten der danach folgenden Ausdrücke zurück.

- **Schalter** (**Switch**):Wertet den als ersten Parameter übergebenen Wert aus und liefert den jeweils zweiten Wert der danach aufgeführten Wertpaare zurück, dessen erster Wert dem ersten Parameter entspricht.

Bild 2: Das Abfrageergebnis mit dem berechneten Rabatt

Die Wenn-Funktion

Die Wenn-Funktion (unter VBA **IIf**) hat drei Parameter. Der erste entspricht in der Regel einem variablen Ausdruck, der etwa den Inhalt eines Feldes oder eine darauf basierende Funktion auswertet. Dieser Ausdruck wird von der **Wenn**-Funktion ausgewertet. Wenn der Ausdruck den Wert **True** liefert, dann entspricht der Rückgabewert dem im zweiten Parameter angegebenen Ausdruck, bei **False** dem dritten Parameter:

```
Wenn(<Ausdruck>;<Ergebnis bei True>;<Ergebnis bei False>)
```

Sie können die **Wenn**-Funktion beispielsweise in einer Abfrage einsetzen, die den Rabatt für eine Bestellposition entsprechend einer Staffelung zurückgeben soll.

Angenommen, bei einer Anzahl bestellter Artikel von mehr als 20 sollen 5% Rabatt gewährt werden, dann würden Sie in einer Abfrage über die Bestellpositionen den folgenden Ausdruck verwenden:

```
Wenn([Anzahl]>=20;0,05;0)
```

Im Beispiel aus Bild 1 haben wir in einer Abfrage ein neues Feld namens **RabattBerechnet** angelegt, das genau den obigen Ausdruck enthält. Das Ergebnis finden Sie in Bild 2. Was aber, wenn Sie weitere Staffelungen einbringen möchten, also beispielsweise 2% für Mengen ab 10 Stück? Dann verwenden Sie einfach zwei oder mehrere verschachtelte **Wenn**-Funktionen wie im folgenden Beispiel:

```
RabattWennVerschachtelt: Wenn([Anzahl]>=20;0,05;Wenn([Anzahl]>=10;0,02;0))
```

Formulare für die Dateneingabe

Teil VI: Abhängige Kombinationsfelder

Keine Sorge: Abhängige Kombinationsfelder müssen Sie keinesfalls zum Entzug schicken. Ganz im Gegenteil: Füttern Sie ordentlich mit Daten! Genaugenommen meint man mit abhängigen Kombinationsfelder zwei Kombinationsfelder, bei denen die Datenherkunft des zweiten von der aktuellen Auswahl des ersten Kombinationsfelds abhängt. Das einfachste Beispiel sind die Kategorien und Artikel der Südsturm-Datenbank: Wählen Sie im Kategorien-Kombinationsfeld eine Kategorie aus, soll das Artikel-Kombinationsfeld nur noch die Artikel der ausgewählten Kategorie zur Auswahl anbieten.

Die Beispieldatenbank zu diesem Artikel ist also die Südsturm-Datenbank, die fertigen Beispiele finden Sie im Formular **frmAbhaengigeKombinationsfelder**. Der grundlegende Ablauf ist, dass der Benutzer einen Eintrag des ersten Kombinationsfeldes auswählt. Dieses Kombinationsfeld soll als Datensatzherkunft alle Kategorien der Tabelle **tblKategorien** enthalten. Durch die Auswahl soll ein Ereignis ausgelöst werden, dass die Datensatzherkunft des zweiten Kombinationsfeldes so einstellt, dass diese nach der im ersten Kombinationsfeld ausgewählten Kategorie gefiltert wird.

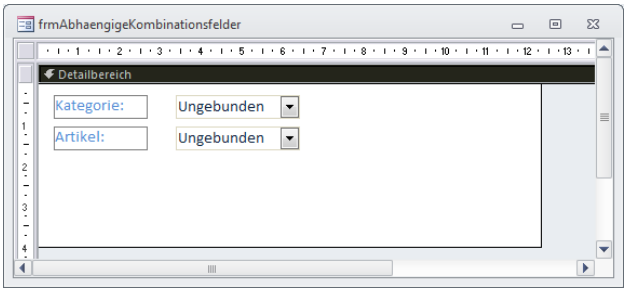


Bild 1: Formular mit den beiden Kombinationsfeldern im Rohbau

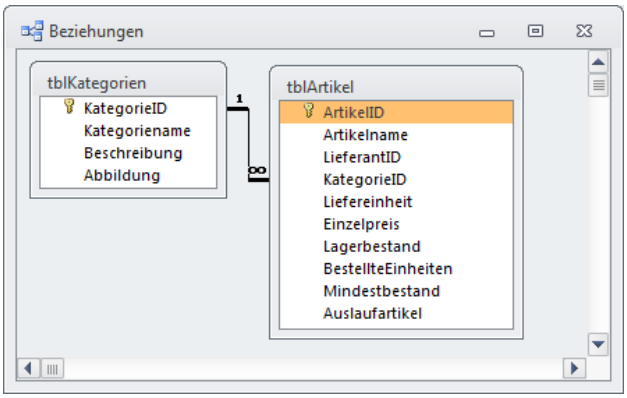


Bild 2: Beziehung zwischen Kategorien und Artikeln

Kombinationsfelder anlegen

Zum Nachvollziehen der hier vorgestellten Techniken benötigen Sie eine Datenbank mit den Tabellen **tblArtikel** und **tblKategorien** der Südsturm-Datenbank aus dem Download zu dieser Ausgabe.

Erstellen Sie zunächst ein neues Formular (siehe erste Abschnitte von →*[basics]* **Gebundenes Formular anlegen**) und fügen Sie zwei Kombinationsfelder hinzu (→*[basics]* **Steuerelement zu Formularen oder Berichten hinzufügen**). Stellen Sie die Beschriftungsfelder der beiden Formularen auf **Kategorie:** und **Artikel:** ein (siehe Bild 1). Legen Sie außerdem als Namen der beiden Kombinationsfelder die Werte **cboKategorien** und **cboArtikel** fest (→*[basics]* **Steuerelementeigenschaften festlegen**).

Kategorie-Auswahl

Das erste Kombinationsfeld soll alle Kategorien zur Auswahl anbieten. Dazu reicht es prinzipiell, eine Abfrage zu erstellen, die lediglich das Feld **Kategoriename** der Tabelle **tblKategorien** enthält. Nach der Auswahl soll aber die Datensatzherkunft des zweiten Kombinationsfeldes nach der gewählten Kategorie gefiltert werden. Die Datensätze der beiden Tabellen sind immerhin über das Fremdschlüsselfeld **KategorieID** der Tabelle **tblArtikel** miteinander verknüpft, dass jeweils den Wert des Primärschlüsselfeldes des betroffenen Kategorie-Datensatzes enthält (siehe Bild 2). Da ist es sinnvoll, diesen Primärschlüssel in die Datensatzherkunft des Kombinationsfeldes **cboKategorien** mit aufzunehmen, um später beim Definieren der Datensatzherkunft von **cboArtikel** auf die **KategorieID** der ausgewählten Kategorie zuzugreifen.

Also stellen Sie die Eigenschaften des Kombinationsfeldes **cboKategorien** wie folgt ein:

- **Datensatzherkunft**: **SELECT KategorieID, Kategoriename FROM tblKategorien;**
- **Spaltenanzahl**: **2**
- **Spaltenbreiten**: **0cm**

Das Ergebnis zeigt Bild 3. Durch die Einstellung für die Spalten zeigt das Kombinationsfeld zwar prinzipiell die beiden Felder **KategorieID** und **Kategoriename** an, die erste Spalte jedoch mit der Breite **0**. Da die erste Spalte jedoch den eigentlichen Wert des Kombinationsfeldes enthält (was durch die Eigenschaft **Gebundene Spalte** mit dem Wert **1** festgelegt ist), kann man diesen ganz einfach auslesen.

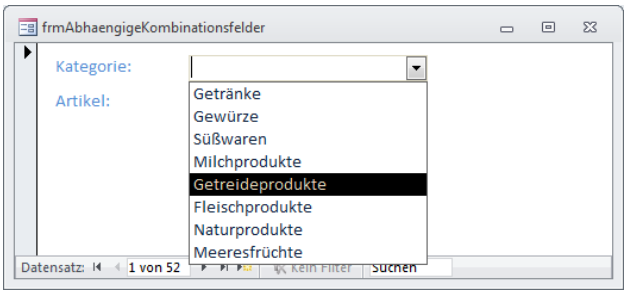


Bild 3: Auswahl der Kategorie per Kombinationsfeld

Um dies auszuprobieren, fügen Sie neben dem Kombinationsfeld ein weiteres Textfeld namens **txtKategorieID** hinzu und stellen dessen Eigenschaft **Steuerelementinhalt** auf **cboKategorieID** ein. Das Ergebnis sehen Sie in Bild 4: Nach dem Auswählen einer Kategorie im Kombinationsfeld **cboKategorieID** ändert sich auch der Wert des Textfeldes.

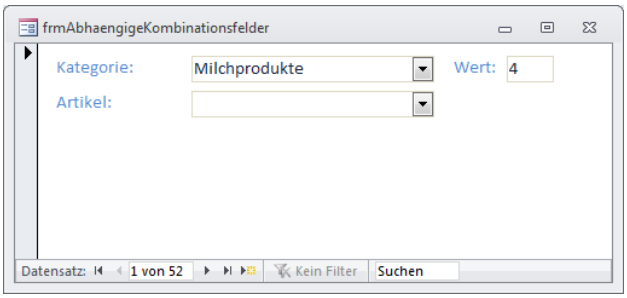


Bild 4: Anzeige des eigentlichen Wertes des Kombinationsfeldes

Dieses Textfeld brauchen wir im weiteren Verlauf nicht, es dient nur der Veranschaulichung des eigentlichen im Kombinationsfeld gespeicherten Wertes. Diesen nutzen wir nun, um die Datensatzherkunft des zweiten Kombinationsfeldes einzustellen. Wie Sie einer Abfrage ein Kriterium zuweisen, haben Sie bereits unter **Abfragen für die Datenauswahl - Teil V: Filtern nach Zahlen** erfahren. In diesem Fall verwenden wie eine Abfrage wie in Bild 5, deren SQL-Ausdruck in der Eigenschaft später so erscheint:

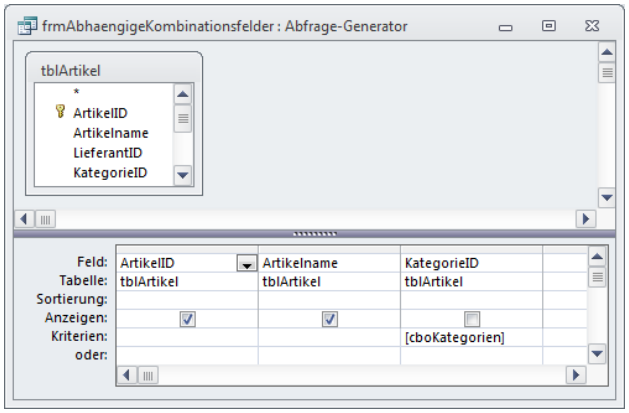


Bild 5: Datensatzherkunft des zweiten Kombinationsfeldes

```
SELECT ArtikelID, Artikelname FROM tblArtikel
WHERE KategorieID=[cboKategorien];
```

Dummerweise ändert sich die Auswahl im Kombinationsfeld **cboArtikel** jedoch nicht, wenn Sie eine andere Kategorie auswählen. Das hat aber doch beim Textfeld mit der **KategorieID** als **Steuerelementinhalt** geklappt? Nun: Genau das ist der Unterschied. Der **Steuerelementinhalt** wird bei Änderungen aktualisiert, andere Eigenschaften wie etwa die **Datensatzherkunft** jedoch nicht.

Sie müssen also anderweitig dafür sorgen, dass der Inhalt des Kombinationsfeldes **cboArtikel** beim Ändern der Auswahl in **cboKategorien** aktualisiert wird. Und dies gelingt beispielsweise mit einer entsprechenden Ereignisprozedur, die durch das Ereignis **Nach Aktualisierung** ausgelöst wird. Legen Sie also eine Ereignisprozedur für die Eigenschaft **Nach Aktualisierung** an (→*[basics]* **Ereignisprozedur hinzufügen**) und ergänzen Sie diese wie folgt:

```
Private Sub cboKategorien_AfterUpdate()
    Me!cboArtikel.Requery
End Sub
```

Nun filtert die Datensatzherkunft des Steuerelements **cboArtikel** seine Datensätze entsprechend dem angegebenen Kriterium.

Alle Datensätze anzeigen

Wenn Sie das Formular im aktuellen Zustand öffnen, enthält das erste Kombinationsfeld noch keinen Wert. Dementsprechend gestaltet sich auch die Auswahl im zweiten Kombinationsfeld: Das erste Kombinationsfeld enthält den Wert **Null**, und es gibt keinen Datensatz in der Tabelle **tblArtikel**, dessen Feld **KategorieID** ebenfalls den Wert **Null** liefert.

Das Kombinationsfeld **cboArtikel** soll aber alle Datensätze anzeigen, wenn **cboKategorien** keinen Wert

Formulare für die Dateneingabe

Teil VII: Standardwerte statisch und dynamisch einstellen

Den Standardwert eines Steuerelements stellen Sie im einfachsten Fall über die entsprechende Steuerelementeigenschaft ein. Es gibt jedoch auch anspruchsvollere Szenarien: So soll beispielsweise manchmal der Wert des zuvor eingegebenen Datensatzes übernommen werden, um schnell einige Datensätze mit gleich bleibenden Werten eingeben zu können.

Standardwerte können Sie in verschiedenen Ebenenfestlegen. Die unterste Ebene ist der Tabellenentwurf. Bereits da legen Sie etwa für das Feld Land einer Adresstabelle einen Standardwert wie Deutschland fest, wenn Sie davon ausgehen, dass der Benutzer größtenteils deutsche Adressen damit verwaltet. Neue Felder auf Basis dieses Feldes in Abfragen oder Formularen übernehmen diesen Standardwert, wobei Abfragen sich auf die ursprüngliche Eigenschaft des Feldes in der Tabelle beziehen. Legen Sie hingegen in einem Formular ein neues Steuerelement an, indem Sie es aus der Feldliste in den Entwurf ziehen, werden einige Eigenschaften des ursprünglichen Feldes auf das Steuerelement übertragen – darunter auch der Standardwert. Diesen können Sie nachträglich einstellen, wie in Bild 1 zu sehen.

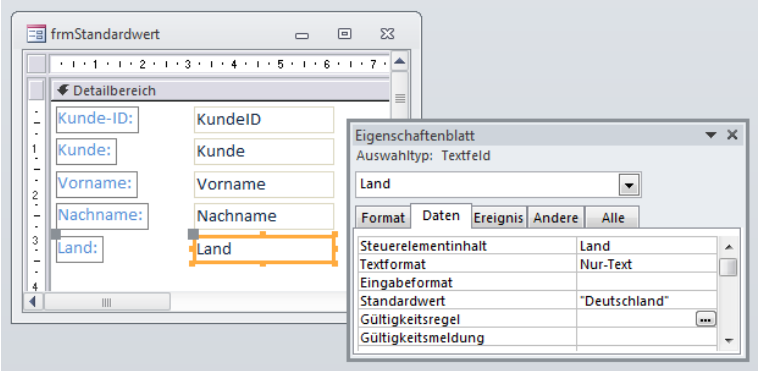


Bild 1: Einstellen des Standardwerts in einem Formular

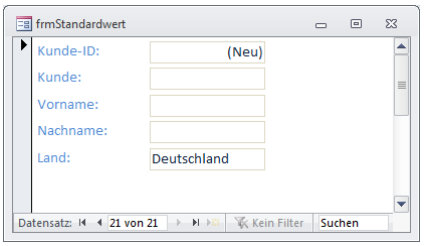


Bild 2: Felder mit Standardwert werden vorbelegt.

Dynamische Standardwerte

Wenn Sie beispielsweise bei der Eingabe von Adressen größtenteils deutsche Adressen eingeben, zwischendurch aber auch mal einen Block Adressen eines anderen Landes erhalten, ist ein dynamischer Standardwert sinnvoll. Das bedeutet nichts anderes, als dass der zuletzt eingegebene Wert als Standardwert für den neuen Datensatz verwendet wird. Ändern Sie diesen Wert in einen anderen Wert, wird dieser wiederum für die folgenden Datensätze vorgegeben.

Eine Änderung des Standardwertes wirkt sich immer erst beim Datensatzwechsel aus. Wenn Sie also nach dem Öffnen eines Datensatzes per VBA-Code den Wert der Eigenschaft **Standardwert** vornehmen, bekommen Sie den neuen Standardwert im aktuellen Datensatz nicht zu sehen.

Standardwert per VBA einstellen

Schauen wir uns an, wie der Standardwert im Beispielformular **frmStandardwert** auf Basis des zuletzt eingegebenen Wertes vorbelegt werden kann.

Das Formular verwendet die Tabelle **tblKunden** der Beispieldatenbank **AccessBasics_2010_02.mdb** als Datenherkunft.

Wenn Sie den Standardwert eines neuen Datensatzes einstellen möchten, müssen Sie dies nach dem Ändern des Wertes des vorherigen Datensatzes erledigen. Den richtigen Zeitpunkt dafür fängt das Ereignis **Nach Aktualisierung** des Formulars ab. Es findet noch vor dem Anzeigen des nächsten (gegebenenfalls neuen) Datensatzes statt, sodass die Änderung der Eigenschaft **Standardwert** dort bereits gültig ist.

Wenn Sie den Standardwert des Feldes **Land** also auf den zuvor eingestellten Wert ändern möchten, legen Sie eine Ereignisprozedur für das Ereignis **Nach Aktualisierung** an. Dort weisen wir der Eigenschaft **DefaultValue** (die unter VBA verwendete englische Bezeichnung für Standardwert) einfach den aktuellen Wert des Feldes **Land** zu (die hier explizit aufgeführte Eigenschaft **Value** brauchen Sie nicht unbedingt anzugeben – **Land** würde reichen, weil **Value** die Standardeigenschaft eines Textfeldes ist):

Projektzeiterfassung

Aufgaben und Tätigkeiten verwalten

In den vorherigen Ausgaben haben Sie bereits einige Formular für die Projektzeiterfassung erstellt, zum Beispiel zum Verwalten von Kunden, Mitarbeitern und Projekten. In diesem Artikel bauen Sie ein Formular zur Eingabe von Aufgaben und Tätigkeiten zu einem Projekt.

Die Projektzeiterfassung soll es ermöglichen, die Tätigkeiten zu einem Projekt zu erfassen. Dabei sollen die Zeit, der Mitarbeiter und das Thema der Tätigkeit aufgenommen werden. Um die Tätigkeiten zumindest grob kategorisieren zu können, gibt es die Zwischenebene der Aufgaben. Ein Projekt besteht aus einem oder mehreren Aufgaben und eine Aufgabe aus einer oder mehreren Tätigkeiten.

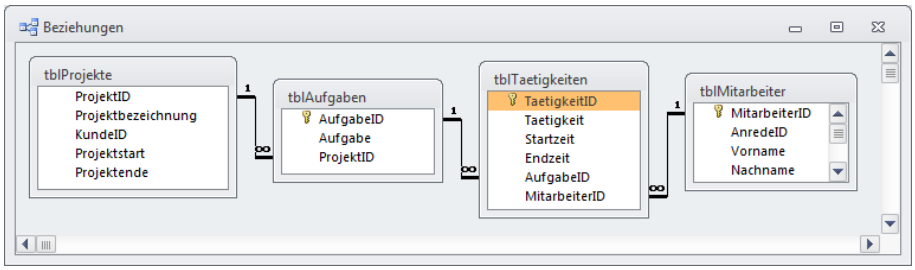


Bild 1: Hierarchie bestehend aus Projekten, Aufgaben, Tätigkeiten und Mitarbeitern

Die Erfassung dieser Daten ermöglicht später genaue Auswertungen der Zeit, die für bestimmte Teile eines Projekts aufgewendet wurde. Das ist ein fantastisches Hilfsmittel – egal, ob Sie Projekte zum Festpreis oder nach Aufwand berechnen. Bei Vereinbarung eines Festpreises können Sie später eine genaue Nachkalkulation durchführen und bei Projekten nach Aufwand ermitteln Sie schnell die benötigte Zeit.

Möglicherweise reicht für größere Projekte eine Zwischenebene nicht aus. Gegebenfalls beschäftigen wir uns später noch mit mehreren Aufgabenebenen – dies ermöglicht die Aufteilung einer Aufgabe in mehrere Teilaufgaben. Für den Moment ist diese Darstellung jedoch etwas zu kompliziert.

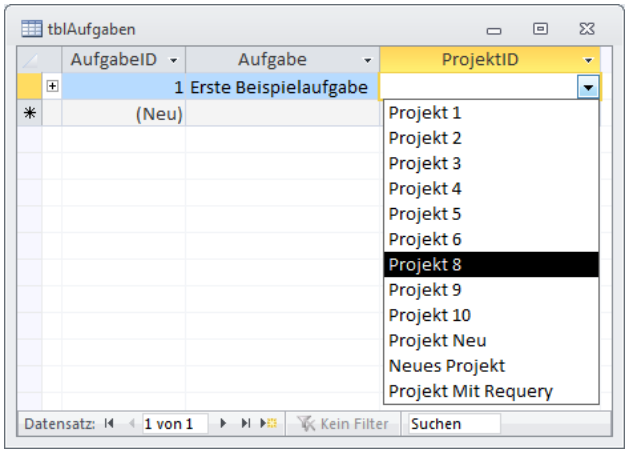


Bild 2: Hierarchie bestehend aus Projekten, Aufgaben, Tätigkeiten und Mitarbeitern

Datenmodell

Die für die in diesem Artikel beschriebenen Techniken benötigen wir vier Tabellen, von denen erst zwei vorhanden sind, nämlich **tblProjekte** und **tblMitarbeiter**. Das fertige Datenmodell aus der Beispieldatenbank **AccessBasics_2011_02.mdb** zeigt Bild 1.

Die Tabelle **tblAufgaben** enthält folgende Felder:

- **AufgabeID**: Primärschlüsselfeld (**Autowert**)
- **Aufgabe**: Bezeichnung der Aufgabe (**Text**)
- **ProjektID**: Fremdschlüsselfeld zur Tabelle **tblProjekte**, das die Zuordnung zum jeweiligen Projekt erlaubt (**Nachschlagefeld**)

Das Feld **ProjektID** wird als Nachschlagefeld ausgeführt, das die Einträge der Tabelle **tblProjekte** zur Auswahl anbietet (siehe Bild 2).

Für die Beziehung zwischen den Tabellen **tblAufgaben** und **tblProjekte** definieren Sie referentielle Integrität und Löschweitergabe. Das bedeutet, dass beim Löschen eines Projekts auch die damit verknüpften Aufgaben gelöscht werden (→*[basics]* **Referentielle Integrität festlegen**).

Die Tabelle **tblTaetigkeiten** auf die gleiche Weise mit **tblAufgaben** verknüpft wie **tblAufgaben** mit **tblProjekte**: Sie enthält ein Fremdschlüsselfeld namens

AufgabeID, das als Nachschlagefeld ausgeführt wird und die Auswahl der Aufgabe ermöglicht, zu deren Erledigung die Tätigkeit durchgeführt wird.

Die Felder im Überblick:

- **TaetigkeitID**: Primärschlüsselfeld der Tabelle (**Autowert**)
- **Taetigkeit**: Bezeichnung der Tätigkeit (**Text**)
- **Startzeit**: Zeitpunkt des Tätigkeitsstarts (**Datum**)
- **Endzeit**: Zeitpunkt der Beendigung der Tätigkeit (**Datum**)
- **AufgabeID**: Fremdschlüsselfeld zum Festlegen der Aufgabe aus **tblAufgaben**, zu deren Erledigung die Tätigkeit beiträgt (**Nachschlagefeld**)
- **MitarbeiterID**: Fremdschlüsselfeld zum Festlegen des Mitarbeiters aus **tblMitarbeiter**, der diese Tätigkeit durchgeführt hat (**Nachschlagefeld**)

Auch die beiden Beziehungen zur Tabelle **tblAufgaben** und **tblMitarbeiter** werden mit referentieller Integrität und Löschweitergabe definiert. In diesem Fall bedeutet dies: Wird ein Mitarbeiter gelöscht, verschwinden auch all seine Tätigkeiten und dem Löschen einer Aufgabe fallen ebenfalls die damit verknüpften Tätigkeiten zum Opfer.

Über die Konsequenzen beider Beziehungen müssen Sie sich klar sein: Eigentlich sollten weder Projekte noch Aufgaben gelöscht werden, wenn dazu bereits Tätigkeiten angelegt wurden. Diese müssen ja irgendwie abgerechnet werden – und wenn auch nur intern.Warum dann also die Löschweitergabe? Nun: Wenn Sie ein Projekt oder eine Aufgabe löschen, für die schon Tätigkeiten vorliegen, und keine Löschweitergabe definieren, dann enthält Ihr Datenbestand Aufgaben oder Tätigkeiten, deren Projekt Sie nicht mehr ermitteln können. Wenn als schon ein solches Projekt gelöscht wird, sollte man zumindest die entstehenden Inkonsistenzen mitlöschen.

Grundsätzlich sollten solche Vorgänge aber durch entsprechende Rückfragen abgesichert, wenn nicht sogar komplett verhindert werden. Immerhin müssen Sie ein Projekt ja nicht gleich löschen, wenn es etwa abgebrochen wurde – ganz im Gegenteil: Auch hier ist eine Nachkalkulation beziehungsweise eine Rechnungsstellung erforderlich.

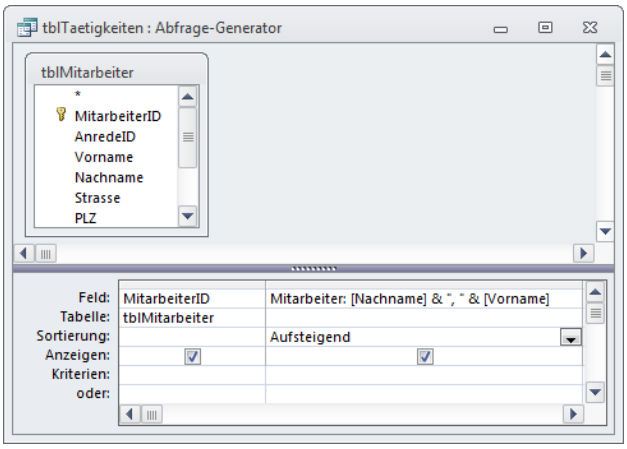


Bild 3: Datensatzherkunft des Nachschlagefeldes **MitarbeiterID** der Tabelle **tblTaetigkeiten**

Mitarbeiter-Auswahl

Für das Nachschlagefeld **MitarbeiterID** der Tabelle **tblTaetigkeiten** gibt es noch eine Besonderheit. Im Gegensatz zu den beiden Nachschlagefeldern **AufgabeID** (**tblTaetigkeiten**) und **ProjektID** (in **tblAufgaben**) werden hier nicht nur das Primärschlüsselfeld als gebundene, unsichtbare Spalte und eine weitere Spalte mit den auszuwählenden Daten benötigt.

Stattdessen soll der Mitarbeiter in der Form **<Nachname>, <Vorname>** erscheinen, und zwar in alphabetischer Reihenfolge.

Um dies anzupassen, öffnen Sie die Tabelle **tblTaetigkeiten** in der Entwurfsansicht und markieren dann das Feld **MitarbeiterID**. Wechseln Sie unten im Eigenschaftsbereich auf die Registerseite **Nachschlagen** und dann auf die Eigenschaft **Datensatzherkunft**.

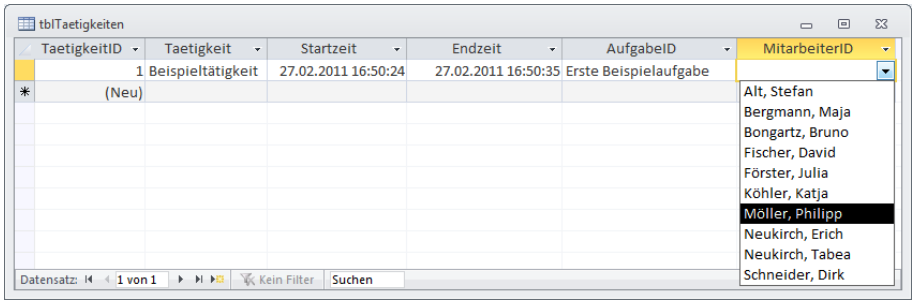


Bild 4: Komforable Auswahl des Mitarbeiters, der eine Tätigkeit durchführt

Es erscheint eine Schaltfläche mit drei Punkten, die Sie anklicken und somit den Entwurf der zugrunde liegenden Abfrage anzeigen. Hier tragen Sie in der zweiten Spalte den folgenden Ausdruck ein und legen außerdem in der Zeile **Sortierung** den Wert **Aufsteigend** fest (siehe Bild 4):

```
Mitarbeiter: [Nachname] & ", " & [Vorname]
```

Der Benutzer kann die Daten dann später wie in Bild 4 auswählen.

Eingabe von Tätigkeiten

Wie immer bei Softwareanwendungen geht es auch hier darum, eine möglichst effiziente Benutzeroberfläche zu gestalten. Wir gehen an dieser Stelle davon aus, dass der Benutzer die Projektzeiterfassung während seiner Arbeit immer geöffnet hat und beim Beginn der Bearbeitung einer Aufgabe zu einem Projekt wenig Zeit verlieren möchte, um den Startzeitpunkt in der Projektzeiterfassung zu verewigen.

Also legen wir fest, dass die Projektzeiterfassung gleich beim Starten der Anwendung das Formular zur Eingabe von Tätigkeiten anzeigt. Sollte der Benutzer anderes im Sinn haben, also beispielsweise das Verwalten von Kunden, Mitarbeitern, Projekten et cetera, kann er dieses Formular immer noch schließen. Wir gehen schlicht davon aus, dass sich über 80% der Vorgänge auf das Anlegen von Tätigkeiten konzentrieren und erlauben uns, das dafür geeignete Formular immer gleich beim Start zu öffnen.

Formular erstellen

Das Formular für die Eingabe der Tätigkeiten verwendet die Tabelle **tblTaetigkeiten** als Datenherkunft. Ziehen Sie die Felder **Taetigkeit**, **Startzeit**, **Endzeit**, **AufgabeID** und **MitarbeiterID** in den Detailbereich des Formulars. Die Steuerelemente benennen Sie um in **txtTaetigkeit**, **txtStartzeit**, **txtEndzeit**, **cboAufgabeID** und **cboMitarbeiterID**.

Außerdem fügen Sie ein weiteres Kombinationsfeld namens **cboProjekte** hinzu. Dieses Steuerelement soll helfen, die zur Verfügung stehenden Aufgaben einzuschränken. Das Steuerelement verwendet den folgenden Ausdruck als **Datensatzherkunft**:

```
SELECT ProjektID, Projektbezeichnung
FROM tblProjekte
ORDER BY Projektbezeichnung;
```

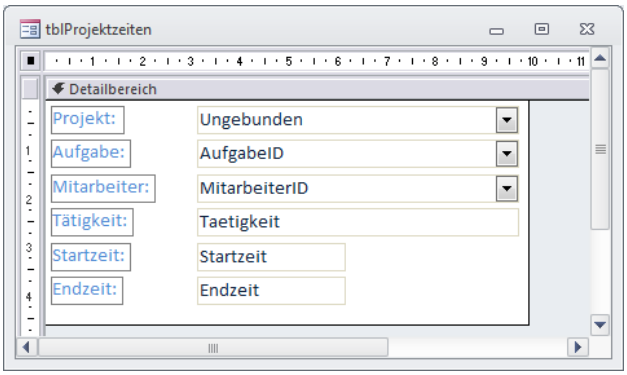


Bild 5: Steuerelemente zum Festlegen der Eigenschaften einer Tätigkeit

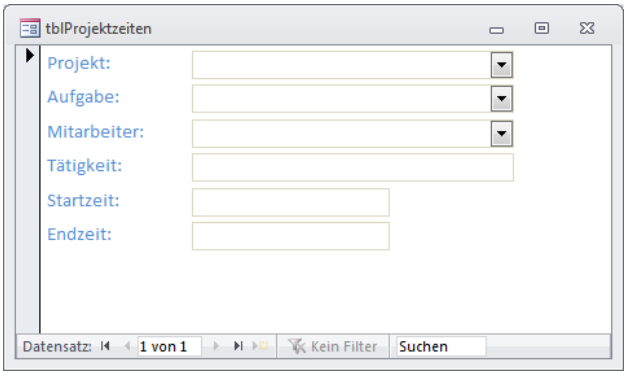


Bild 6: Das Formular mit einem leeren Datensatz in der Formularansicht

Es soll also alle Einträge der Tabelle **tblProjekte** in alphabetisch aufsteigender Reihenfolge anzeigen. Damit nicht der Wert des gebundenen Feldes, sondern der des Feldes **Projektbezeichnung** erscheint, stellen Sie die Eigenschaften **Spaltenanzahl** sowie **Spaltenbreiten** auf die Werte **2** und **0cm** ein.

Die Steuerelemente ordnen Sie nun etwa wie in Bild 5 an. Damit der Benutzer später in der richtigen Reihenfolge mit der Tabulator- oder der Eingabetaste durch die Steuerelemente navigieren kann, sollen Sie außerdem noch die Aktivierreihenfolge anpassen (siehe →*[basics]* **Aktivierreihenfolge von Steuerelementen**). Gegebenenfalls stellen Sie außerdem die Eigenschaft **Textausrichtung** der beiden Steuerelemente **txtStartzeit** und **txtEndzeit** auf **Links** ein.

Ergonomie verbessern

Wie Bild 6 zeigt, ist das Formular beim ersten Öffnen komplett leer. Klar, die Datenherkunft enthält ja auch noch keine Datensätze. Wenn wir nun allerdings etwa ein Projekt auswählen, tut sich nicht viel – es wäre aber schön, wenn danach die Einträge des Kombinationsfeldes **cboAufgabeID** nach den Aufgaben des ausgewählten Projekts gefiltert wäre. Das heißt also,

dass wir zwei wie in **Formulare für die Dateneingabe - Teil VI: Abhängige Kombinationsfelder** beschriebene abhängige Kombinationsfelder brauchen.

Dazu stellen Sie zunächst die Eigenschaft **Datensatzherkunft** von **cboProjekte** um:

```
SELECT 0 AS ProjektID,
"<Auswählen>" AS Projektbezeichnung
FROM tblProjekte
UNION
SELECT ProjektID, Projektbezeichnung
FROM tblProjekte
ORDER BY Projektbezeichnung;
```

Der **Standardwert** dieses Steuerelements soll den Wert **0** erhalten, damit das Kombinationsfeld beim Öffnen den Eintrag **<Auswählen>** anzeigt. Das Kombinationsfeld **cboAufgabeID** erhält nun diese **Datensatzherkunft** (weitere Informationen hierzu in oben genanntem Artikel):

```
SELECT 0 AS AufgabeID,
"<Auswählen>" AS Aufgabe
FROM tblAufgaben
UNION
SELECT AufgabeID, Aufgabe
FROM tblAufgaben
WHERE [ProjektID]=[cboProjektID]
OR [cboProjektID]=0
ORDER BY Aufgabe;
```

Auch den **Standardwert** dieses Steuerelements stellen Sie auf **0** ein.

Damit die drei Kombinationsfelder ein einheitliches Bild abgeben, stellen Sie auch noch die Eigenschaft **Datensatzherkunft** von **cboMitarbeiterID** um

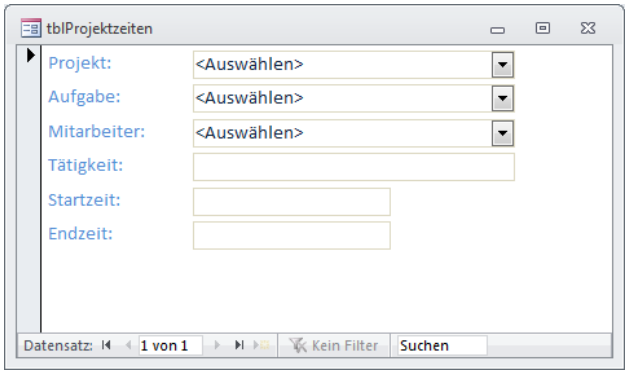


Bild 7: Die Kombinationsfelder enthalten nun voreingestellte Einträge, die den Benutzer auf die Auswahlmöglichkeit hinweisen.

(und auch die Eigenschaft **Standardwert** erhält den Wert **0**):

```
SELECT 0 AS MitarbeiterID,
"<Auswählen>" AS Mitarbeiter
FROM tblMitarbeiter
UNION
SELECT MitarbeiterID,
[Nachname] & ", " & [Vorname] AS Mitarbeiter
FROM tblMitarbeiter
ORDER BY Mitarbeiter;
```

Das vorläufige Ergebnis sieht wie in Bild 7 aus.

Abhängige Kombinationsfelder

Nun soll das Kombinationsfeld **cboAufgaben** nur noch die Einträge anzeigen, die mit dem jeweiligen Projekt verknüpft sind, also deren Wert **ProjektID** in der Tabelle **tblAufgaben** mit der **ProjekteID** im **cboProjektID** übereinstimmt.

Die Datensatzherkünfte sind bereits entsprechend eingestellt. Fehlt noch eine Ereignisprozedur, die durch das Ereignis **Nach Aktualisierung** des Steuerelements **cboProjektID** ausgelöst wird (siehe **Formulare für die Dateneingabe – Teil VI: Abhängige Kombinationsfelder**):

```
Dim lngProjektID As Long

Private Sub cboProjektID_AfterUpdate()
    Me!cboAufgabeID.Requery
    If Not (Me!cboProjektID = lngProjektID _
            Or Me!cboProjektID = 0) Then
        Me!cboAufgabeID = 0
    End If
    lngProjektID = Me!cboProjektID
End Sub
```

Tätigkeit eintragen

Die Tätigkeit muss der Benutzer im aktuellen Zustand jeweils beim Anlegen einer neuen Tätigkeit neu eintragen. Oft unterbricht man jedoch eine Tätigkeit und nimmt diese neu auf.

Im Laufe der Zeit gibt es sogar die meisten Tätigkeiten schon, sodass es praktisch wäre, wenn der Benutzer diese auswählen könnte.

Also wandeln wir das Textfeld **txtTaetigkeit** in ein Kombinationsfeld namens **cboTaetigkeit** um:

Tabellen entwerfen

Teil VI: Indizierung von Feldern

In den bisherigen Teilen dieser Artikelreihe haben Sie gelernt, wie Sie Tabellen und Felder anlegen und Tabellen miteinander verknüpfen. Das ist aber noch längst nicht alles: Damit Ihre Anwendung auch mit vielen Daten noch richtig schnell läuft, müssen Sie noch geeignete Indizes anlegen. Dieser Beitrag stellt die Funktion von Indizes vor und zeigt, wie Sie diese anlegen.

Als Beispiel schauen wir uns die Tabelle **tblKunden** an (siehe Bild 1). Die Tabelle enthält alle relevanten Kundendaten wie Vorname, Nachname, Adresse und so weiter.

Ein typischer Anwendungsfall wäre die Suche nach einem Kunden über seinen Nachnamen. Wenn Sie nun eine Abfrage erstellen, welche die Tabelle **tblKunden** nach einem bestimmten Wert für den Nachnamen filtert (etwa **LIKE 'B*'**, siehe Bild 2), durchläuft die Jet-Engine, die unter Access für alle Tabellenzugriffe verantwortlich ist, alle Datensätze der Tabelle und gleicht den Wert des Feldes **Nachname** mit dem Vergleichsausdruck ab. Nachdem alle Datensätze durchlaufen wurden, gibt Access das Abfrageergebnis mit allen gefundenen Daten aus (siehe Bild 3).

Diese Vorgehensweise funktioniert bei wenigen Datensätzen noch recht schnell, aber auch bei größeren Datenmengen müssten für jede Abfrage alle Datensätze durchlaufen werden – was natürlich entsprechend mehr Zeit kostet.

Wie lässt sich dieser Vorgang optimieren? Ausgehend davon, dass Sie nach einem konkreten Nachnamen suchen oder zumindest nach dem Beginn dieses Namens (Sie werden wohl kaum **Horst** als Suchbegriff eingeben, wenn Sie eigentlich nach **Minhorst** suchen), liegt die Antwort recht nah.

Stellen Sie sich vor, Sie suchen in einem Telefonbuch nach diesem Namen. Dann wissen Sie von vornherein, dass sich etwa der Name Achtermann recht weit vorn befindet und Zitzewitz eher hinten zu finden ist. Sie blättern also zur erwarteten Stelle, prüfen, ob der gesuchte Name entsprechend des Alphabets vor oder hinter dieser Seite liegt, blättern noch einige Seite vor oder zurück und finden schließlich die richtige Seite und somit auch den Namen.

Was ist der Unterschied zwischen einem Telefonbuch und den in der Tabelle eingetragenen Kunden? Richtig: Im Telefonbuch sind die Daten aufsteigend nach

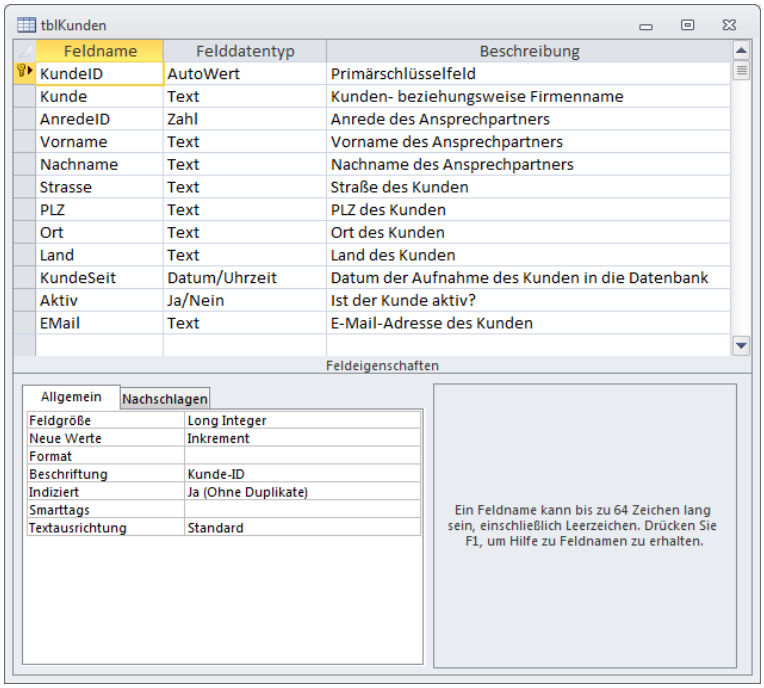


Bild 1: Die Tabelle **tblKunden** in der Entwurfsansicht

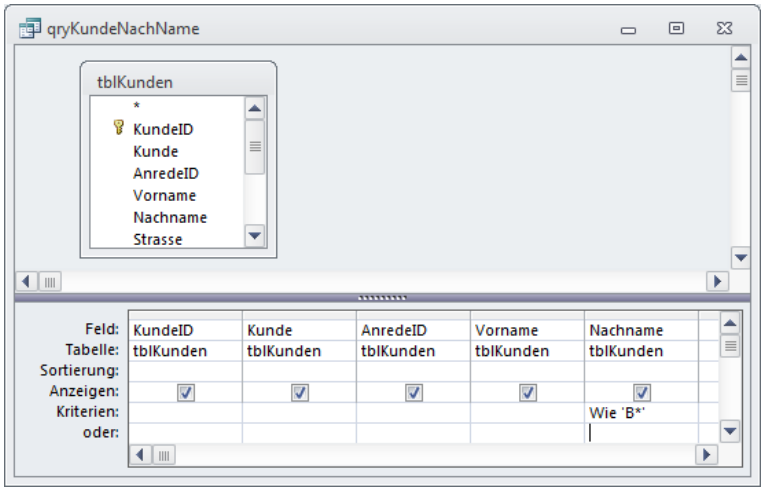


Bild 2: Abfrage nach dem Nachnamen in der Entwurfsansicht ...

dem Nachnamen sortiert, während dies in der Tabelle **tblKunden** nicht der Fall ist. Hier sind die Datensätze in der Anlegereihenfolge abgelegt – und die ist in der Regel nicht alphabetisch.

Indizierung

Wie also können Sie dies optimieren? Es gibt ein einfaches Mittel namens Indizierung. Oberflächlich erfolgt die Indizierung ganz schnell:

- Öffnen Sie die Tabelle **tblKunden** in der Entwurfsansicht.
- Markieren Sie das Feld **Nachname**.
- Wählen Sie für die Eigenschaft **Indiziert** den Wert **Ja (Duplikate möglich)** aus (siehe Bild 4).

Was passiert nun beim Speichern der Tabelle mit dem neuen Index? Access erstellt eine interne Index-Tabelle, welche nur die Primärschlüsselfelder der Ausgangstabelle enthält – allerdings in einer Reihenfolge, die der alphabetisch aufsteigenden Reihenfolge des Feldes **Nachname** entspricht (siehe Bild 5). Diese Indextabelle speichert Access intern, Sie bekommen diese also nicht zu sehen.

qryKundeNachName

Kunde-ID	Kunde	AnredeID	Vorname	Nachname
1107	Förster GbR	Herr	Jörg	Brinkmann
1115	Becker AG	Frau	Esther	Berger
1120	Moll GmbH & Co. KG	Frau	Pia	Beck
(Neu)				

Datensatz: 1 von 3 | Kein Filter | Suchen

Bild 3: ... und das Ergebnis der Abfrage.

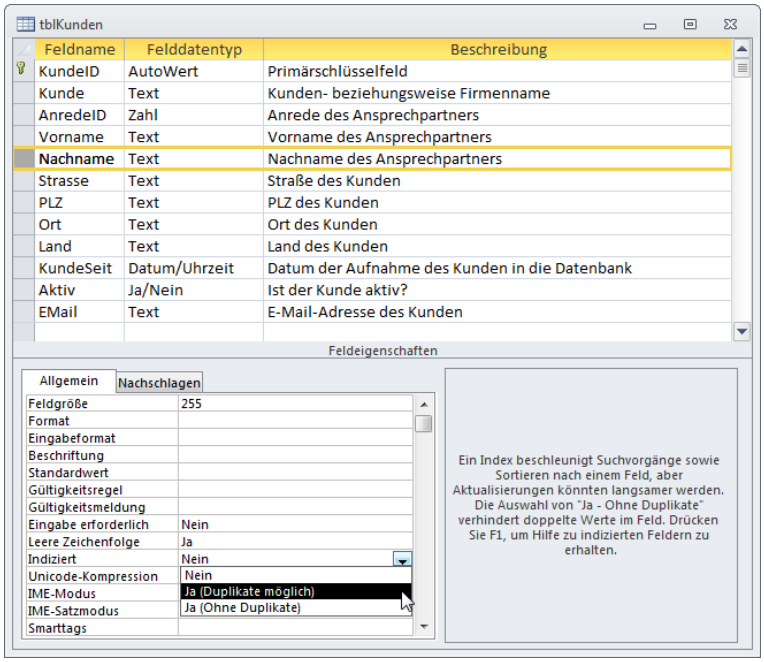


Bild 4: Anlegen eines Indexes für das Feld **Nachname**

Die Sortierung nach den Werten eines derart vorbereiteten Feldes erfolgt natürlich erheblich schneller als bei einem nicht indizierten Feld. Für die Suche werden noch etwas aufwendigere Techniken verwendet: Wenn Sie in einem Feld wie **Nachname** nach einem Eintrag suchen, der mit Z beginnt, immer noch fast alle Datensätze durchlaufen (außer, Sie beginnen unten). Daher kommen für die Suche beispielsweise Techniken wie binäre Suchbäume zum Tragen – an dieser Stelle gehen wir jedoch aus Platzgründen nicht weiter auf dieses Thema ein.

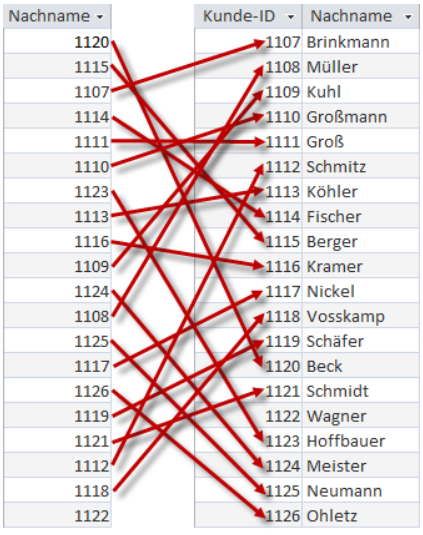


Bild 5: Schematisches Aussehen einer Indextabelle

Es geht vielmehr darum, eine wichtige Erkenntnis zu gewinnen: Das Sortieren von und das Suchen in indizierten Feldern erfolgt in der Regel wesentlich schneller als in nicht indizierten Feldern.

Die Medaille hat jedoch auch eine Kehrseite: Wenn Sie einen Datensatz zu einer Tabelle mit einem indizierten Feld hinzufügen, muss Access natürlich nicht nur die eigentlichen Informationen in der Tabelle speichern, sondern auch die Indextabellen mit entsprechenden Informationen füttern. Das heißt

Abfragen für die Datenauswahl

Duplikate per Abfrage finden

Es gibt verschiedene Anlässe, Duplikate in bestimmten Feldern einer Tabelle zu finden. Vielleicht möchten Sie einen eindeutigen Index für ein Feld festlegen, was aber am Vorhandensein doppelter Einträge für dieses Feld scheitert. Vielleicht wollen Sie aber auch einfach nur schauen, welche Kunden Sie doppelt in der Datenbank angelegt haben, was zu Inkonsistenzen führen kann. In beiden und auch in anderen Fällen hilft der Assistent zur Duplikatsuche, den dieser Artikel beschreibt.

Den Assistenten für die Duplikatsuche öffnen Sie unter den verschiedenen gängigen Access-Versionen wie folgt:

Access 2003 und älter: Menüeintrag **Einfügen|Abfrage,** dann im Dialog **Neue Abfrage** Eintrag **Abfrage-Assistent zur Duplikatsuche** auswählen

Access 2007: Ribbon-Eintrag **Erstellen|Andere|Abfrage-Assistent**, dann im Dialog **Neue Abfrage** den Eintrag **Abfrage-Assistent zur Duplikatsuche** auswählen

Access 2010: Ribbon-Eintrag **Erstellen|Abfragen|Abfrage-Assistent**, dann **Abfrage-Assistent zur Duplikatsuche** auswählen

Im Assistenten wählen Sie zunächst die betroffene Tabelle oder Abfrage aus, also beispielsweise **tblKunden** (siehe Bild 1).

Dann legen Sie fest, in welchem Feld beziehungsweise in welcher Feldkombination Sie nach Duplikaten suchen wollen. Verschieben Sie diese Felder (beispielsweise **Kunde**) in das rechte Listenfeld (siehe Bild 2).

Die doppelten Einträge allein helfen wahrscheinlich nicht bei der Entscheidung, welcher Datensatz zu löschen ist, wenn keine weiteren Informationen zu den jeweiligen Datensätzen vorliegen. Also legen Sie im dritten Schritt weitere Felder fest, die das Abfrageergebnis zurückgeben soll (siehe Bild 3).

Im letzten Schritt geben Sie schließlich noch den Namen ein, unter dem Access die Abfrage speichern soll (hier **qryDoppelteKunden**). Außerdem entscheiden Sie, ob Sie direkt die Ergebnisse anzeigen oder den Entwurf der automatisch erstellten Abfrage ändern möchten (siehe Bild 4).

Das Ergebnis zeigt genau die Datensätze, die mehr als einmal in der Tabelle vorkommen – und zwar mit allen Vorkommen (siehe Bild 5).

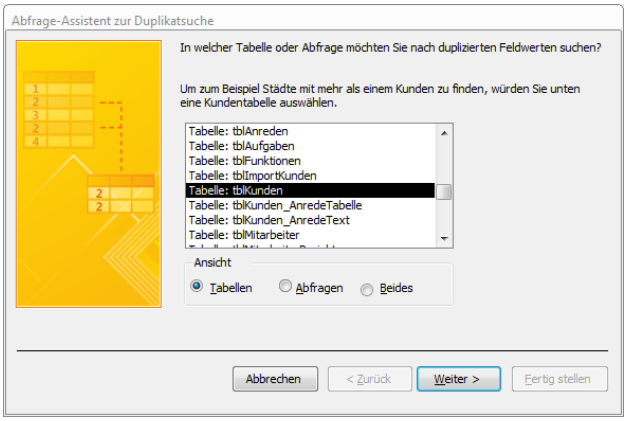


Bild 1: Zu untersuchende Tabelle oder Abfrage auswählen

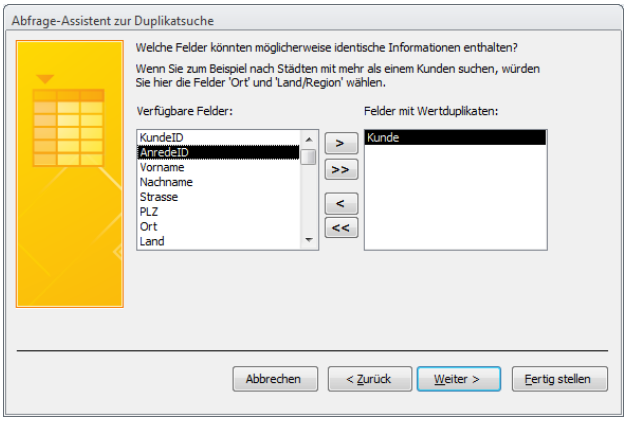


Bild 2: Auswählen der zu untersuchenden Felder

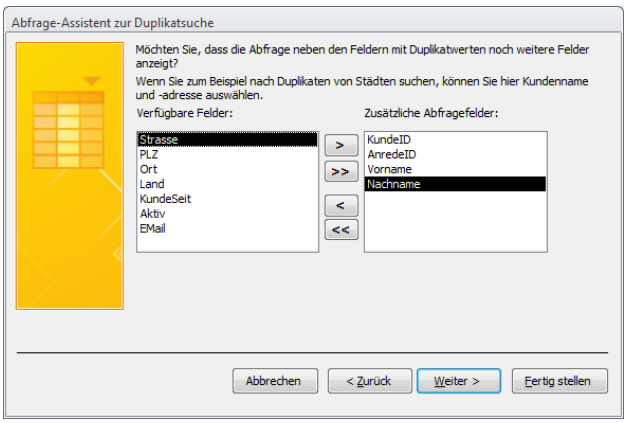


Bild 3: Weitere Felder für die Anzeige auswählen

Reporting mit Berichten

Einfache Tabelle in der tabellarischen Ansicht darstellen

Berichte liefern ihre Daten meist in tabellarischer Ansicht. Dabei gilt es, die Spaltenüberschriften und die eigentlichen Werte in den entsprechenden Bereichen des Berichts unterzubringen. Dabei gibt es einige Tricks und Kniffe, die Sie nach der Lektüre dieses Artikels aber komplett beherrschen sollten!

Der nachfolgend beschriebene Bericht soll die Datensätze der Tabelle **tblKunden** in tabellarischer Ansicht auflisten. Dazu erstellen Sie zunächst einen neuen Bericht, öffnen diesen in der Entwurfsansicht und stellen seine Eigenschaft **Datenherkunft** auf **tblKunden** ein (**Bericht erstellen**).

Ziehen Sie dann alle benötigten Felder aus der Feldliste in den Detailbereich der Entwurfsansicht, die danach wie in Bild 1 aussehen sollte.

Von tabellarischer Anordnung kann hier jedoch noch keine Rede sein. Dazu ist ein wenig Handarbeit gefragt: Ziehen Sie jeweils die Bezeichnungsfelder in den Bereich **Seitenkopf** und positionieren Sie die gebundenen Steuerelemente jeweils darunter im Detailbereich (siehe Bild 2).

Beachten Sie, dass Bezeichnungsfeld und gebundenes Steuerelement immer zusammenhängen und nur getrennt voneinander bewegt werden können, wenn Sie diese am grauen Kästchen links oben mit dem Mauszeiger anfassen.

Sie können die Bindung zwischen Bezeichnungsfeld und gebundenem Steuerelement auch aufheben, indem Sie das Bezeichnungsfeld markieren, es mit der Tastenkombination **Strg + X** ausschneiden und an anderer Stelle mit **Strg + C** wieder einfügen – das ist auf Dauer einfacher.

Ab Access 2007 haben Sie es noch einfacher: Sie brauchen dann einfach nur alle Elemente, die Sie in tabellarischer Ansicht darstellen möchten, zu markieren und dann aus dem Ribbon den Befehl **Anord-**

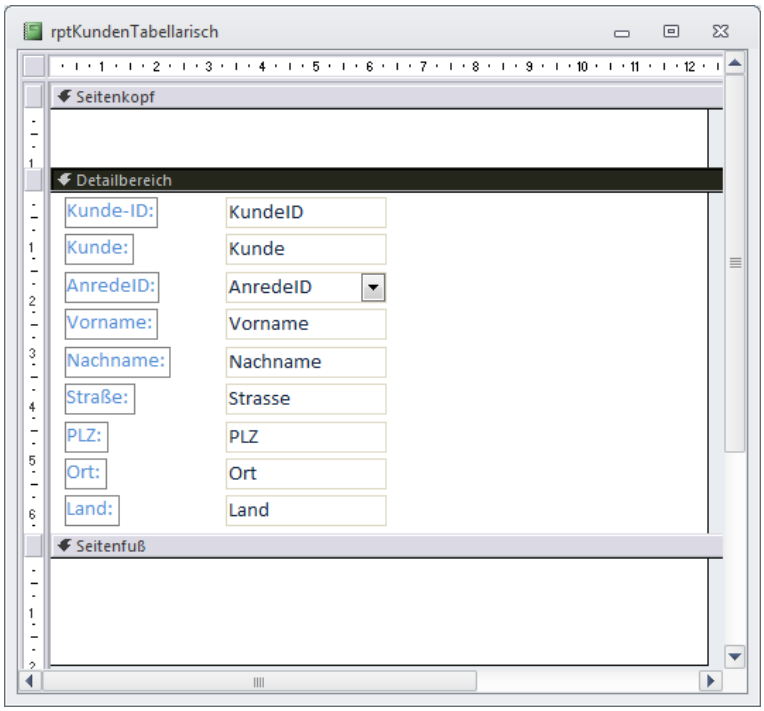


Bild 1: Die gewünschten Felder sind schonmal im Bericht gelandet.

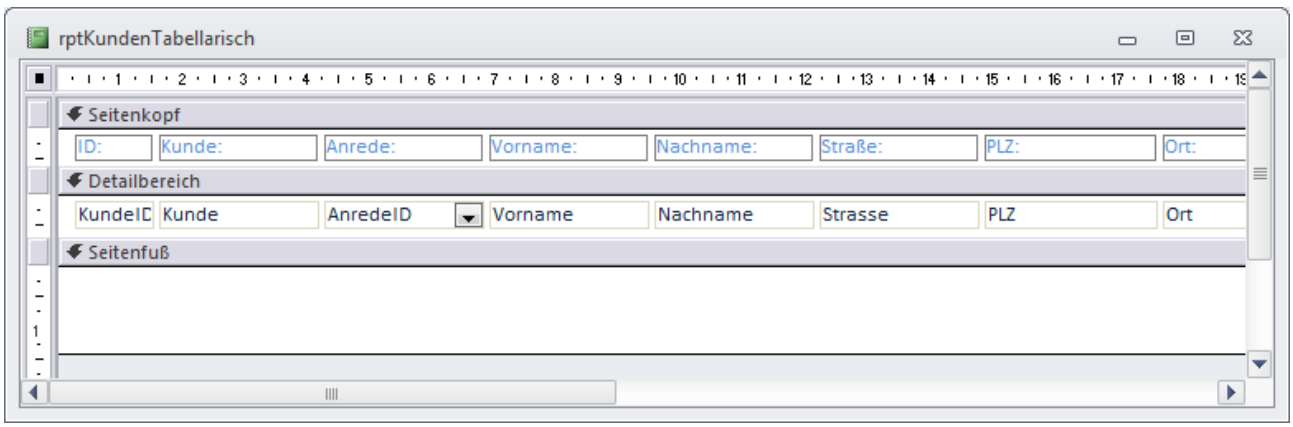


Bild 2: Die Bezeichnungsfelder landen im Seitenkopf, der Rest im Detailbereich.

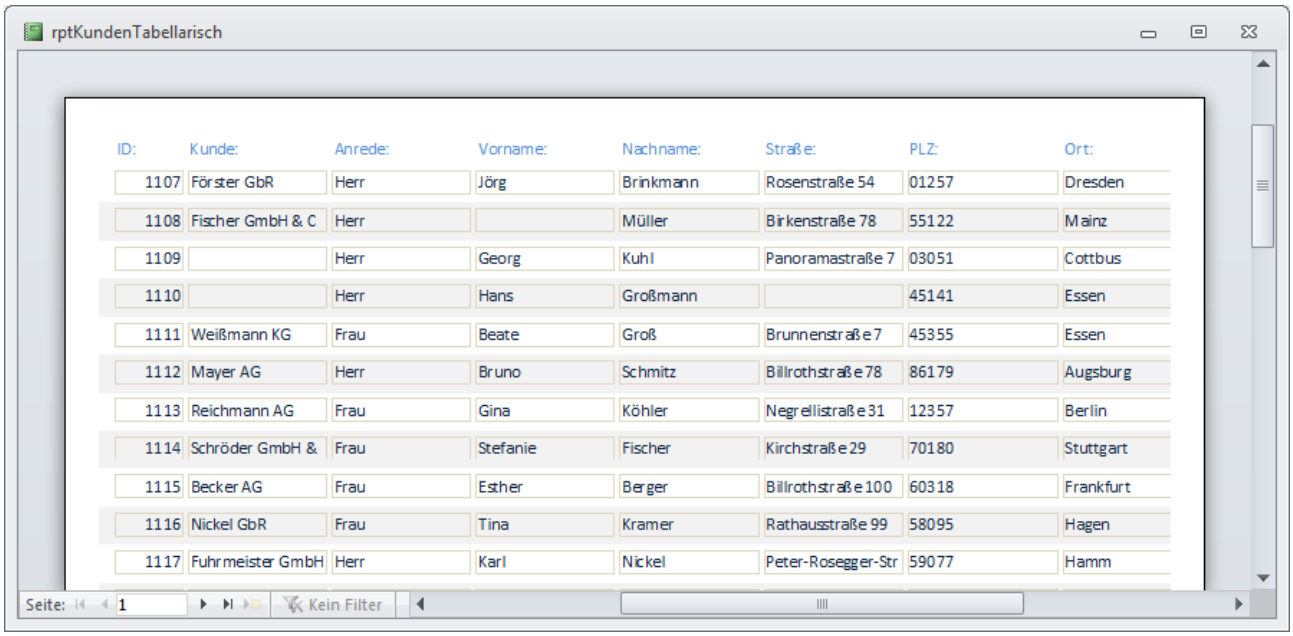


Bild 3: Der erste Entwurf in der Berichtsvorschau

nen|Tabelle|Tabelle auszuführen. Access legt dann ein sogenanntes Layout an, das die Ausrichtung wesentlich vereinfacht. Wir kommen in einer späteren Ausgabe auf dieses Feature zurück.

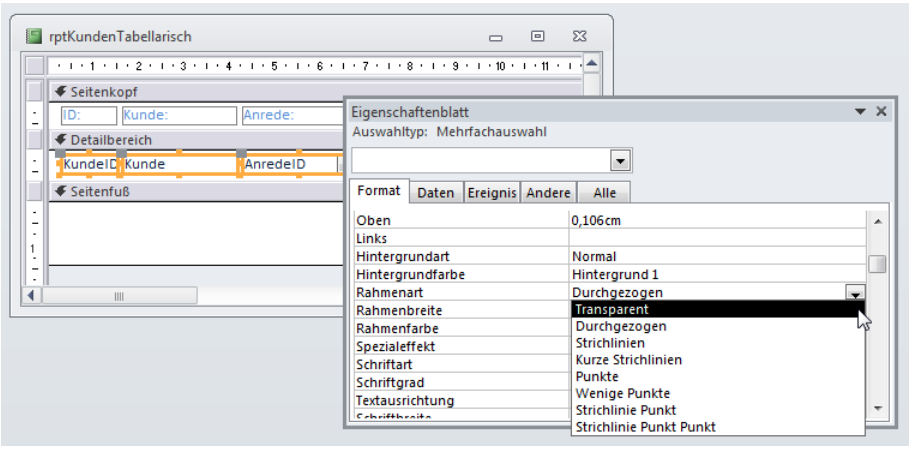


Bild 4: Entfernen der Textfeld-Rahmen

Warum teilen wir die Elemente auf den Seitenkopf- und den Detailbereich auf? Nun: Der Seitenkopf-Bereich wird nur einmal pro Seite angezeigt, war für die Überschriften genau richtig ist.

Die eigentlichen Daten hingegen landen im Detailbereich, weil dieser für jeden Datensatz einmal erzeugt wird.

Sollte die Datenherkunft soviele Datensätze enthalten, dass diese nicht auf einer einzigen Seite Platz finden, erstellt Access weitere Seiten. Diese enthalten wiederum die im Seitenkopf-Bereich angelegten Spaltenköpfe und darunter die entsprechenden Daten.

Nun schauen wir uns erst einmal in der Berichtsvorschau an, ob die tabellarische Ansicht unseren Vorstellungen entspricht (siehe Bild 3). Da Access 2010, in dem dieser Bericht erstellt wurde, einige Voreinstellungen wie etwa Steuerelement-Rahmen oder abwechselnd gefärbte Zeilenhintergründe verwendet, ist hier noch ein wenig Nacharbeit nötig (unter Access 2003 und früher sollten nur die reinen Inhalte zu sehen sein).

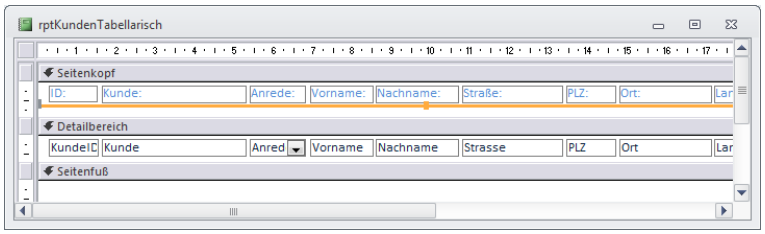


Bild 5: Positionierung der Trennlinie unter den Spaltenüberschriften

Die alternierenden Zeilenhintergründe können wir beibehalten, aber die Rah-

Reporting in Berichten

Darstellung von 1:n-Beziehungen per Gruppierung

Während man in Formularen meist ein aus Haupt- und Unterformular zur Darstellung von Daten aus 1:n-Beziehungen verwendet, ist dies in Berichten eher die Ausnahme. Hier setzen Sie meist Gruppierungen ein, um verknüpfte Daten anzuzeigen. Alternativ lassen sich solche Beziehungen auch mithilfe eines Unterberichts darstellen, was aber Thema eines anderen Artikels ist.

Im Beispiel dieses Artikels sollen Kunden und ihre Projekte in einem Bericht abgebildet werden. In Formularen würde man beispielsweise die Kunden im Hauptformular anzeigen und die Projekte dieses Kunden in einem Unterformular. Beim Wechseln des Kunden stellt das Unterformular dann die entsprechenden Projekte des nun angezeigten Kunden dar.

In Berichten ist dies grundsätzlich auch möglich. Sollten allerdings keine weiteren Verschachtelungen hinzu kommen, geht man einen anderen Weg: Dort teilt man die beiden Tabellen **tblKunden** und **tblProjekte** nicht auf zwei völlig verschiedene Bereiche auf, sondern bestückt bereits die Datenherkunft mit allen notwendigen Daten. Fügen Sie also zu einer neuen Abfrage die beiden Tabellen **tblKunden** und **tblProjekte** hinzu und ziehen Sie die Felder der beiden Tabellen außer dem Feld **KundeID** der Tabelle **tblKunden** in das Entwurfsraster (siehe Bild 1) – dieses Feld brauchen wir nur einmal und verwenden das der Tabelle **tblProjekte**. Speichern Sie die Abfrage unter **qryRptKundenProjekte**.

Erstellen Sie dann einen neuen, leeren Bericht und öffnen Sie diesen in der Entwurfsansicht. Fügen Sie der Eigenschaft **Datenherkunft** den Wert **qryRptKundenProjekte** ein. Wir bewegen uns nun Schritt für Schritt von einem naiven Ansatz zur richtigen Darstellung der Daten beider Tabellen.

Im ersten Ansatz ziehen Sie die Felder **KundeID**, **Kunde**, **AnredeID**, **Vorname** und **Nachname** sowie **ProjektID** und **Projektbezeichnung** in den Detailbereich (die übrigen Felder ziehen wir später nach – vorerst reichen ein paar Felder einer jeden Tabelle). Der Entwurf sieht nun wie in Bild 2 aus. Beim Wechsel in die Berichtsvorschau wird nun jede Kombination aus Kunden und Projekten einmal angezeigt. Wenn ein Kunde also zwei Projekte hat, werden dessen Kundendaten zwei mal mit den verschiedenen Projektdaten ausgegeben.

Das ist eigentlich unnötig: Warum sollen wir die Kundendaten zu jedem Projekt neu ausgeben? Das macht nur Sinn, wenn wir einzelne Projektdatenblätter benötigen. Außerdem möchten wir eine tabellarische Ansicht erreichen, bei der in der ersten Zeile jeweils der Kunde steht und in den folgenden Zeilen die Projekte des Kunden.

Projekte nach Kunden gruppieren

Das Zauberwort für die gewünschte Darstellung heißt Gruppierung. Sie können in Berichten Gruppierungen für beliebige Felder anlegen. Sie können beispielsweise die Datensätze der Tabelle **tblKunden** nach dem Ort gruppieren oder nach dem Nachnamen. Unsere Datenherkunft enthält aber Daten aus den Tabellen **tblKunden** und **tblProjekte** und wir wollen die Projekte nach den Kunden gruppieren.

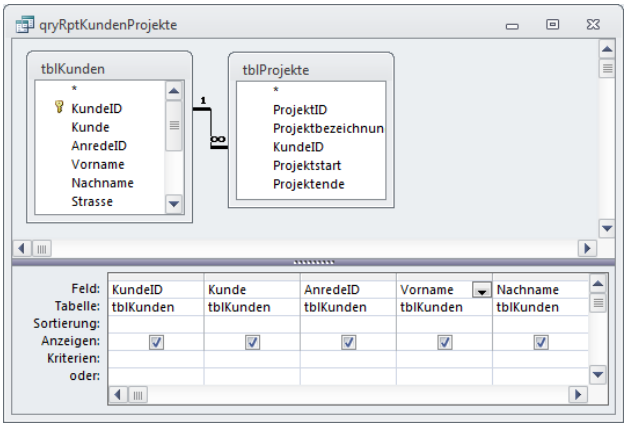


Bild 1: Datenherkunft des Berichts zur Anzeige von Kunden und Projekten

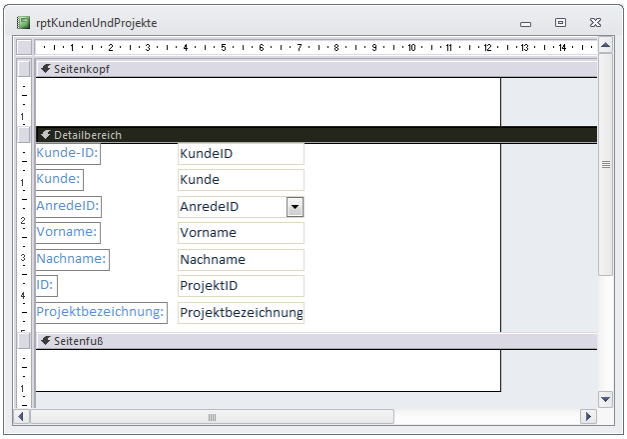


Bild 2: Erster Entwurf des Berichts

Nichts leichter als das: Die Tabelle **tblProjekte** enthält mit **KundeID** ja bereits ein eindeutiges Feld für die Zuordnung zur Tabelle **tblKunden**.

Das bedeutet, dass wir die Projekte nach dem Wert des Fremdschlüsselfeldes **KundeID** auf die verschiedenen Kunden aufteilen können.

Im Prinzip ist die Gruppierung eine Sortierung. Wir sortieren in diesem Fall alle Projekte nach dem Fremdschlüsseld **KundeID**. Es kommen also erst alle Projekte des Kunden **1**, dann die von Kunde **2** und so weiter.

Der kleine, aber feine Unterschied zwischen der Sortierung und der Gruppierung ist, dass eine Gruppierung im Bericht spezielle Bereiche enthält, die den Beginn und das Ende einer Gruppe markieren. Diese Bereiche heißen Gruppenkopf und Gruppenfuß.

Kommen wir zur Praxis und schauen uns an, wie wir dem Bericht entsprechende Gruppierungen hinzufügen. Unter Access 2003 und älter und unter Access 2007 und jünger gibt es verschiedene Dialog zum Anlegen der Gruppierungen eines Berichts, sodass wir uns nun beide Varianten ansehen.

Access 2003 und jünger: Hier klicken Sie am einfachsten mit der rechten Maustaste auf einen der Bereichsköpfe der Entwurfsansicht. Aus dem Kontextmenü wählen Sie den Eintrag **Sortieren und Gruppieren** aus. Es erscheint der Dialog **Sortieren und Gruppieren**, mit dem Sie in jeder Zeile eine Sortierung oder Gruppierung definieren können (siehe Bild 4).

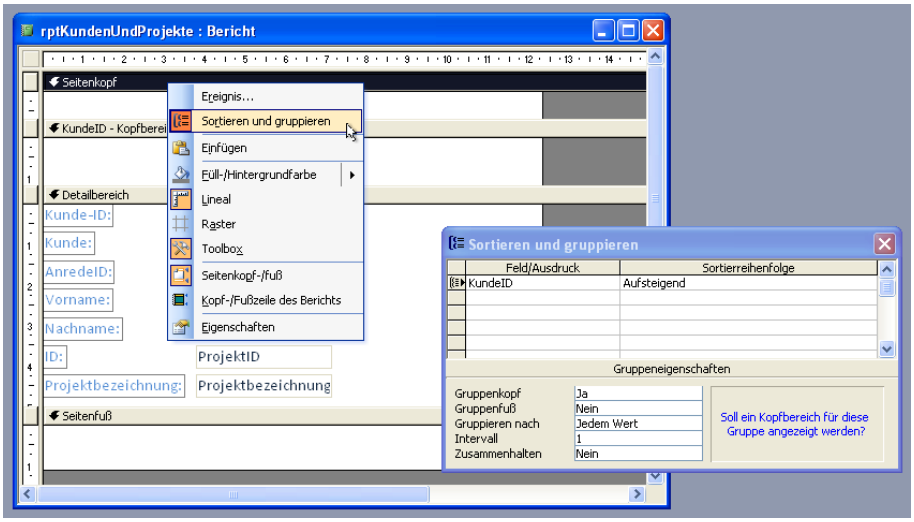


Bild 4: Einrichten einer Gruppierung unter Access 2003 und älter

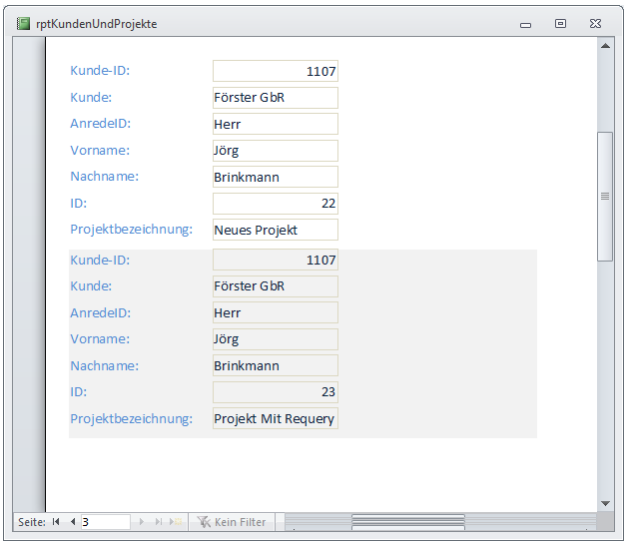


Bild 3: Im ersten Ansatz werden die Kundendaten für jedes Projekt des Kunden einmal ausgegeben.

In unserem Beispiel wählen Sie in der ersten Zeile einfach das Feld **KundeID** aus. Dies würde schlicht eine Sortierung der Datensätze nach diesem Feld bewirken.

Außerdem stellen Sie aber noch die Eigenschaft **Gruppenkopf** auf **Ja** ein. Sie können dann beobachten, wie Access dem Berichtsentwurf einen neuen Bereich namens **KundeID - Kopfbereich** hinzufügt.

Sie könnten auch noch einen Fußbereich hinzufügen, aber diesen benötigen wir aktuell nicht.

Access 2007 und jünger: Hier wählen Sie ebenfalls den Eintrag Sortieren und gruppieren aus dem Kontextmenü des Bereichskopfes im Berichtsentwurf aus. Die Optionen zum Einstellen von Sortierungen und Gruppierungen erscheinen allerdings nicht in einem eigenen Dialog, sondern in einem eigenen Bereich unten im Access-Fenster, der etwas unhandlicher als der Dialog aus älteren Access-Versionen scheint (siehe Bild 5).

Hier klicken Sie gleich auf die Schaltfläche **Gruppe hinzufügen**, um eine Gruppierung anzulegen.

Dies bewirkt das Erstellen einer Sortierung, wobei Sie

Datum und Zeit, Teil I

Umgang mit Datums- und Zeitangaben in Tabellen, Formularen, Berichten und VBA

Das Speichern von Datums- und Zeitangaben ist eine der wichtigsten Aufgaben in Datenbankanwendungen. Der Umgang damit ist allerdings nicht ganz trivial, weshalb wir diesem Thema einen größeren Artikel widmen. Sie erfahren, wie Access intern mit Zeit und Datum umgeht, wie Sie solche Daten in Tabellen speichern und diese in Formularen und Berichten ausgeben.

Access speichert Zeitangaben in Form von **Double**-Werten. **Double**-Variablen enthalten Vor- und Nachkommastellen. Die Vorkommastellen liefern eine Information über das Datum, während die Nachkommastellen die Zeit speichern. Bei den Vorkommastellen entspricht der Wert **1** dem **31.12.1899**. Für jeden weiteren Tag wird 1 hinzugezählt, sodass **40634** dem Datum **31.3.2011** entspricht. Bei früheren Daten werden entsprechende negative Zahlen verwendet, also beispielsweise **-1** für den **30.1.1899**. Der mögliche Bereich liegt zwischen -657.434 (1. Januar 100) und 2.958.465 (31. Dezember 9999).

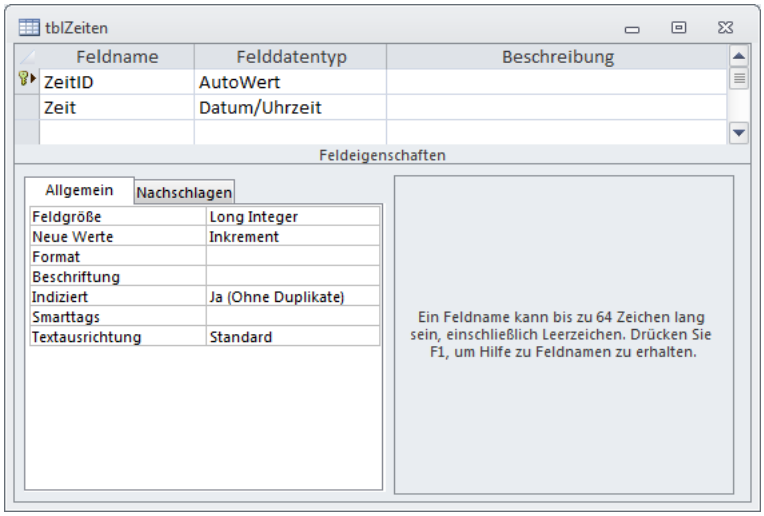


Bild 1: Die Tabelle **tblZeiten** in der Entwurfsansicht

Bei den Nachkommastellen entspricht **0** der Uhrzeit **0:00:00 Uhr**. **40634,0** entspricht also dem **31.3.2011**, **0:00:00 Uhr**. **40635,0** ist genau einen Tag später, also der **1.4.2011** um **0:00:00 Uhr**. Da eine ganze Zahl einem ganzen Tag entspricht, bilden die Nachkommastellen entsprechende Bruchteile des Tages ab. **0,5** ist etwa **12:00:00 Uhr**. Das Intervall für eine Stunde entspricht also **1/24** beziehungsweise ca. **0,016667**.

Zeiten in Tabellen speichern

Wenn Sie ein Datum samt Uhrzeit in einer Tabelle speichern wollen, brauchen Sie lediglich ein entsprechendes Feld des Datentyps **Datum/Uhrzeit**. Die Beispieltabelle **tblZeiten** aus Bild 1 enthält ein solches Feld, mit dem wir nun einige Experimente durchführen. Im Optimalfall geben Sie dort eine komplette Zeitangabe wie **31.3.2011 20:44:00** ein. Sie können aber auch Kurzformen eingeben oder auch nur ein Datum oder eine Uhrzeit eingeben. Folgendes ist beispielsweise möglich:

- **1/2011** (wird zu **1.1.2011**, der Tag wird also mit **1** ergänzt)
- **1.1.** (wird zu **1.1.2011**, das Jahr wird mit dem aktuellen Jahr ergänzt)
- **1:0** (wird zu **1:00:00** ergänzt)
- **1.0** (wird zu **1.1.2000** ergänzt)
- **1.13** (wird zu **13.1.2011** ergänzt)
- **1.32** (wird zu **1.1.1932** ergänzt)
- **1.1 1:0** (wird zu **1.1.2011 1:00:00**)

Sie sehen: Die Angabe nur von Bruchteilen eines Datums kann mitunter zu unerwarteten Datumsangaben führen. Woher weiß ich aber nun, dass Access tatsächlich intern einen **Double**-Wert speichert? Um dies herauszufinden, bemühen wir das Direktfenster

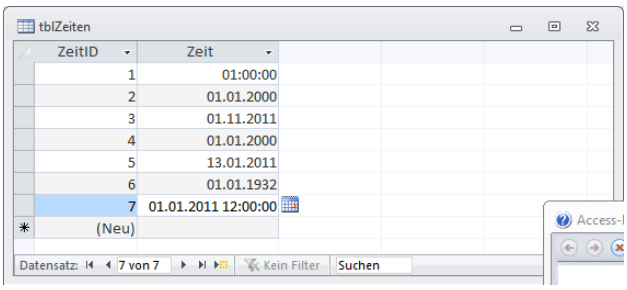


Bild 2: Die Zeit-Tabelle mit einigen Beispieldatensätzen

des VBA-Editors. Es gibt sogenannte Typumwandlungsfunktionen, die wir später im Detail besprechen – aber nun greifen wir dem ein wenig vor. Die VBA-Funktion **CDbl** beispielsweise wandelt den als Parameter angegebenen Wert in eine **Double**-Zahl um. Damit **CDbl** das Datum als solches erkennen kann, formatieren Sie es zuvor in der Form **#dd/mm/hhhh hh:nn:ss#**. Bild 3 zeigt am Beispiel des Datums **1.1.2011 12:00:00**, wie es funktioniert.

Alternativ geben Sie das Datum als Literal an und fassen es in Anführungszeichen ein. Dann ist zuvor noch ein Aufruf einer weiteren Typumwandlungsfunktion nötig, nämlich **CDate**:

```
? CDbl(CDate("1.1.2011 12:00:00"))
 40544,5
```

CDate wandelt einen Ausdruck, wenn möglich, in ein Datum um. Das funktioniert auch, wenn Sie mit der Domänenfunktion **DLookup** (**Access-Funktionen - Teil I: Domänenfunktionen**) auf einen Wert der Tabelle **tblZeiten** zugreifen – beispielsweise auf den Datensatz mit dem Wert **7** im Feld **ZeitID**:

```
? CDbl(DLookup("Zeit", "tblZeiten", "ZeitID = 7"))
 40544,5
```

Da Sie hier auf ein Feld mit dem Datentyp **Datum/Zeit** zugreifen, brauchen Sie keine spezielle Konvertierungsfunktion, um sicherzustellen, dass **DLookup** auch ein gültiges Datum ermittelt.

Eingabeformate

Access erlaubt für das Datum eine ganze Reihe verschiedener Eingabeformate, unter anderem die folgenden:

- dd.mm.yyyy
- dd-mm-yyyy
- dd/mm/yyyy
- mm/dd/yyyy

Wenn Access das eingegebene Datum nicht erkennt, erscheint eine entsprechende Fehlermeldung. Das gilt ebenfalls für ungültige Datumsangaben wie 29.02.2011. Sie können das Eingabeformat auch mit der gleichnamigen Eigenschaft in Tabellen, Abfragen oder Formularen vorgeben. Im Tabellenentwurf finden Sie diese Eigenschaft beispielsweise wie in Bild 4. Ein Klick auf die Schaltfläche mit den drei Punkten öffnet einen Dialog, der Sie beim Anlegen von Eingabeformaten unterstützt. Der folgende Ausdruck etwa erwartet die Eingabe von zwei, zwei und vier Zeichen, wobei der Unterstrich als Platzhalter verwendet wird:

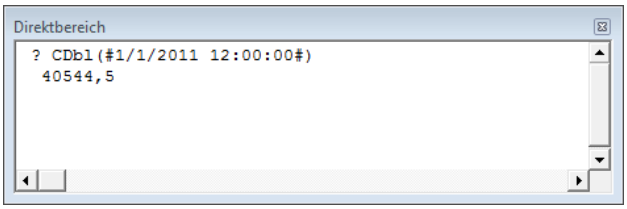


Bild 3: Umwandeln eines Datums mit der **CDbl**-Funktion

```
00.00.0000;0;_
```

Diese Eigenschaft hat drei Parameter:

- Das eigentliche Eingabeformat legt den Aufbau und den Wertebereich der zulässigen Daten fest. Das Zeichen **0** aus dem obigen Beispiel steht für Zahlen von 0 bis 9.
- Der zweite Parameter legt fest, ob die Nicht-Platzhalter-Zeichen, hier also etwa die Punkte zwischen den Platzhaltern, mit in der Tabelle gespeichert werden sollen. **0** bedeutet: Die Zeichen werden gespeichert, alle anderen Werte verhindern das Speichern. In diesem Fall würde bei Eingabe von **0** etwa der Ausdruck **01.01.2011** gespeichert, bei Eingabe eines anderen Wertes die Zahl **1012011**.

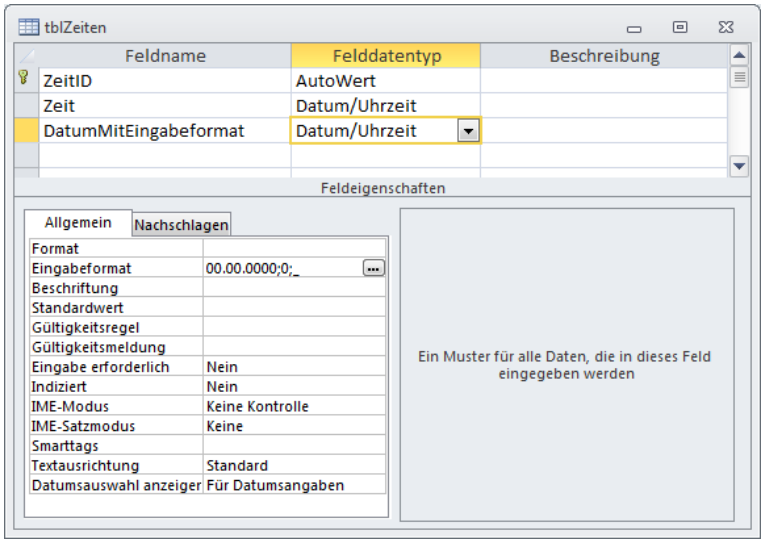


Bild 4: Eingabe eines Eingabeformats

- Der letzte Parameter legt das Platzhalterzeichen fest, hier den Unterstrich (_).

Ausgabeformate

Unabhängig davon, wie Sie das Datum eingeben, wird es immer in dem vom System oder von Ihnen festgelegten Format angezeigt. Das Standardformat können Sie ganz einfach durch die Ausgabe des aktuellen Datums mit der Funktion **Now()** im Direktfenster ermitteln:

```
? Now()
  01.04.2011 12:38:27
```

Das Standardformat ändert sich, wenn Sie in der Systemsteuerung unter **Region und Sprache** unter **Formate** ein anderes Land als das aktuelle auswählen (siehe Bild 5). Die unter **Datum (kurz)**, **Datum (lang)** et cetera angegebenen Datumsformate wirken sich nicht auf die Darstellung unter Access aus. Allein die Länderauswahl ist relevant. Wenn Sie dort etwa **Englisch (USA)** auswählen, liefert **Now** ein anderes Ergebnis:

```
? Now()
4/1/2011 1:13:04 PM
```

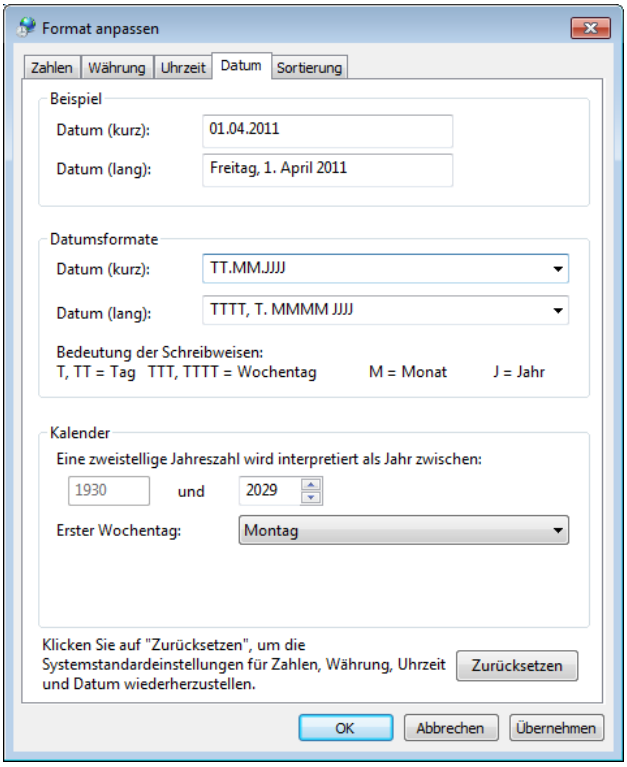


Bild 6: Weitere Optionen zum Datumsformat in der Systemsteuerung

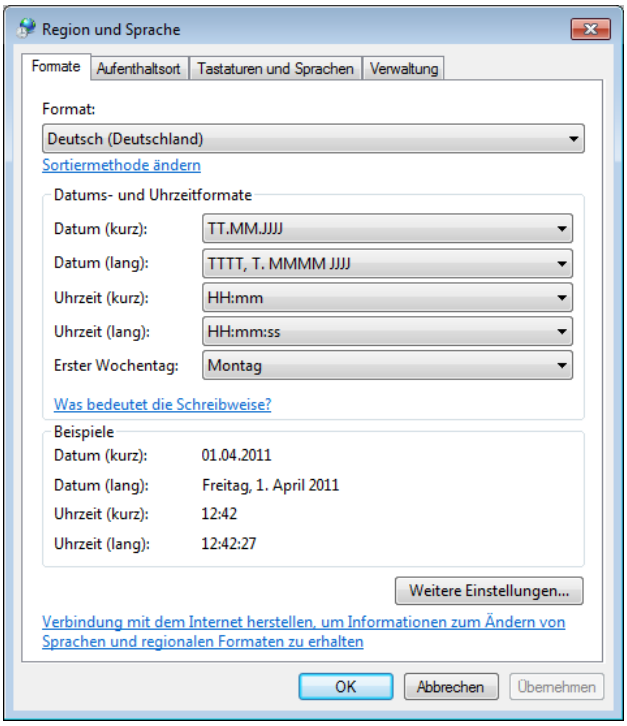


Bild 5: Datums- und Zeitformate in der Systemsteuerung

Sollten Sie im Dialog **Region und Sprache** auf die Schaltfläche **Weitere Einstellung** klicken und im folgenden Dialog **Format anpassen** unter **Datum** solche Einstellungen vorgenommen haben, die sich auf die verwendeten Trennzeichen auswirken, so schlägt sich dies auch auf die Standardausgabe unter Access nieder.

Benutzerdefinierte Ausgabeformate

Damit Access das Datum genau so ausgibt, wie Sie es sich wünschen, stellen Sie die Eigenschaft **Format** in Tabellen, Abfragen, Formularen oder Berichten entsprechend ein.

Sie können für die Eigenschaft einen der voreingestellten Werte aus der Liste auswählen (siehe Bild 7) oder einen eigenen Ausdruck eingeben.

Für das Formatieren von Datum und Uhrzeit stehen die folgenden Optionen zur Verfügung:

- Doppelpunkt: Trennzeichen, wird durch die in der Systemsteuerung angegebenen Standardtrennzeichen ersetzt
- **/**: Datumstrennzeichen
- **c**: Entspricht dem vordefinierten Format Standarddatum.

Formulare für die Dateneingabe

Validieren von Benutzereingaben

Für grundlegende Validierungsfunktion liefern Access-Tabellen und -Felder entsprechende Eigenschaften. Wer mehr will, lässt die Benutzereingabe in Formularen direkt bei der Eingabe prüfen. Wie dies funktioniert, erfahren Sie in diesem Artikel.

Die Validierung bei der Eingabe von Daten ist eine wichtige Funktion einer Formulars. Wenn Sie den Benutzer nach Lust und Laune Werte eingeben lassen, werden früher oder später nicht sinnvoll verwertbare Daten vorliegen – daher sollten Sie Ihren Anwendungen eine Validierung hinzufügen, die Werte nach der Eingabe prüft und gegebenenfalls auf Fehler hinweist (siehe Bild 1).

Validierung auf Tabellenebene

Der Artikel **Tabellen entwerfen - Teil IV: Felder und ihre Eigenschaften** hat bereits erläutert, wie Sie direkt im Tabellenentwurf festlegen, welche Werte für ein Feld zulässig sind und welche Meldung angezeigt werden soll, wenn der Benutzer eine ungültige Eingabe durchführt. Die dort vorgestellten Eigenschaften **Gültigkeitsregel** und **Gültigkeitsmeldung** gelten anwendungsweit.

Das bedeutet, dass diese auch bei Eingaben in die Steuerelemente von Formularen, die an die entsprechenden Felder gebunden sind, funktionieren. Genau wie bei der Eingabe in die Tabellen selbst wird die Gültigkeitsregel gleich nach dem Abschließen der Eingabe und vor dem Übernehmen des Wertes durchgeführt.

Dasselbe gilt für die beiden gleichnamigen Eigenschaften, die Sie im Tabellenentwurf für die Tabellenebene festlegen können (siehe **Tabellen entwerfen – Teil V: Eigenschaften von Tabellen**).

Hier verschiebt sich der Zeitpunkt, zu dem die Regeln geprüft werden, allerdings ein wenig nach hinten: Die Prüfung findet erst nach dem Abschließen der Eingabe des Datensatzes, aber noch vor dem Speichern des Datensatzes in der Tabelle statt.

Auch noch einige weitere Eigenschaften von Tabellen und ihrer Felder dienen prinzipiell der Validierung der Benutzereingaben:

- Der Datentyp lässt nur Eingaben nach bestimmten Vorgaben zu, beispielsweise **Zahl**, **Datum/Zeit** oder **Ja/Nein**.
- Die Eigenschaften **Eingabe erforderlich** oder **Leere Zeichenfolge** eines Feldes bestimmen, ob ein Wert oder eine Zeichenfolge eingegeben werden muss.
- Sogar die Eigenschaft **Indiziert** kann in die Validierung eingreifen, wenn Sie diese auf den Wert **Ja (Ohne Duplikate)** einstellen. Das Eingeben eines Wertes, der bereits in einem anderen Datensatz enthalten ist, würde dann das Speichern des Datensatzes verhindern

Sie können die Validierung jedoch auch direkt im Formular vornehmen, ohne Eigenschaften wie Gültigkeitsregel und Gültigkeitsmeldung zu verwenden. Manchmal muss man die durch diese Eigenschaften ausgelösten Effekte sogar beeinflussen, damit keine für den Benutzer unverständlichen Meldungen erscheinen.

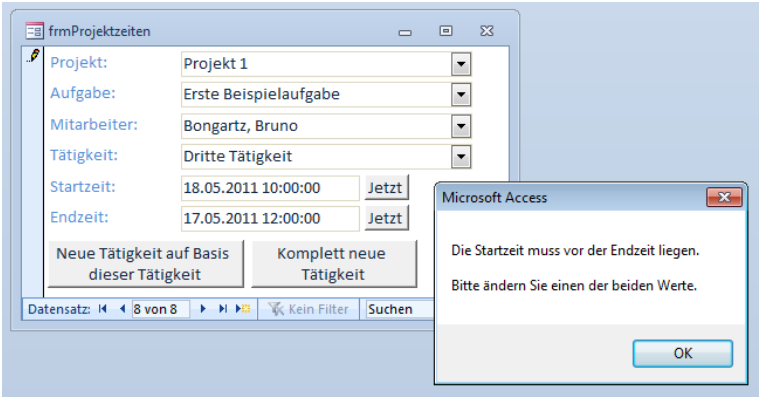


Bild 1: Beispiel für eine Validierungsmeldung

Validierung auf Tabellen- oder Formularebene?

Der Vorteil der Validierungen, die durch die Feld- und Tabelleneigenschaften ausgelöst werden, ist die Einfachheit. Sie legen fest, wann der Wert eines Feldes

gültig ist und welche Meldung bei der Eingabe ungültiger Werte angezeigt werden soll. Oder Sie lassen Access einfach entsprechende Meldungen ausgeben, wenn der Benutzer einen Wert in einem eindeutigen Feld doppelt eingibt oder einen Text in ein Zahlenfeld eintippt.

Im Gegensatz zu selbstdefinierten Gültigkeitsmeldungen sind diese Fehlermeldungen für Otto Normalverbraucher jedoch oft nicht leicht zu verstehen.

In vielen Fällen müssen Sie also ohnehin die eingebauten Meldungen abfangen und durch eigene Meldungen ersetzen (wie dies geschieht, erfahren Sie weiter unten). Dies erledigen Sie durch entsprechende Ereignisprozeduren im Formular. Wenn Sie gleichzeitig Gültigkeitsregeln auf Tabellen- oder Feldebene verwenden, bedeutet dies, dass Sie die Prüfung der Daten auf mehrere Bereiche der Datenbank aufteilen. Dies ist aus Gründen der Wartbarkeit der Anwendung nicht empfehlenswert. Sie sollten also entweder nur einfache Validierungen mit den Möglichkeiten des Tabellenentwurfs verwenden oder aber gleich die komplette Eingabeprüfung in die Formulare integrieren.

Wenn Sie die Validierung direkt im Formulare programmieren möchten, benötigen Sie eine oder mehrere Ereignisprozeduren, die vor oder nach der Eingabe der Daten durch den Benutzer ausgelöst werden. Grundsätzlich unterscheiden wir aber auch noch nach Ein gaben auf Feld- und Datensatzebene.

Wenn ein Benutzer beispielsweise einen Text in ein Datumsfeld eingibt, sollten Sie diesen direkt auf die fehlerhafte Eingabe hinweisen. Der Benutzer ist dann gedanklich noch in der Nähe des entsprechenden Feldes und kann eine Korrektur gleich vornehmen.

Die andere Variante ist die Prüfung der Eingabe erst vor dem Speichern des Datensatzes. Dort müssten Sie dann alle Daten auf Richtigkeit und Vollständigkeit prüfen und den Benutzer gegebenenfalls durch eine Meldung auf eventuell fehlende Daten hinweisen.

Optimalerweise kombiniert man beide Varianten. Fehleingaben, die an Ort und Stelle erkannt werden, sollten auch gleich gemeldet werden. Andere Fehleingaben kann man schlicht nicht direkt erkennen – auf ein leeres Feld etwa sollten Sie den Benutzer erst hinweisen, wenn dieser versucht, den Datensatz zu speichern.

Schließlich gibt es auch noch abhängige Kriterien bei der Validierung: Wenn beispielsweise ein Feld das Eintrittsdatum und ein anderes das Austrittsdatum markiert, darf das Eintrittsdatum nicht hinter dem Austrittsdatum liegen.

Solche Zustände können Sie erst prüfen, wenn der Benutzer beide Eingaben vorgenommen hat. Dies ist in der Regel erst beim Speichern des kompletten Datensatzes der Fall, wie Sie später sehen werden.

All diese Validierungen führen Sie vor dem Speichern des Datensatzes durch. Bei Prüfungen auf Feld- beziehungsweise Steuerelementebene erledigen Sie dies in einer Prozedur, die durch das Ereignis **Vor Aktualisierung** des entsprechenden Steuerelements durchgeführt wird. Validierungen auf Datensatzebene hingegen finden in der Ereignisprozedur **Vor Aktualisierung** des Formulars selbst statt.

Es gibt noch Sonderfälle wie etwa das Auftreten leerer Feldwerte, wenn die Feldeigenschaft **Eingabe erforderlich** den Wert **Ja** hat, wenn der Benutzer einen doppelten Wert in ein eindeutiges Feld eingibt et cetera. Die dadurch ausgelösten Fehler fangen Sie in der Ereignisprozedur **Bei Fehler** des Formulars ab – mehr dazu ebenfalls weiter unten.

Beispieltabelle und -formular

Für die folgenden Beispiele haben wir eine Tabelle namens **tblBeispieleValidierung** mit den folgenden Feldern vorbereitet (siehe Bild 2):

- **EinfacherText**: Einfaches Textfeld, dessen Werte wir nach bestimmten Kriterien validieren werden

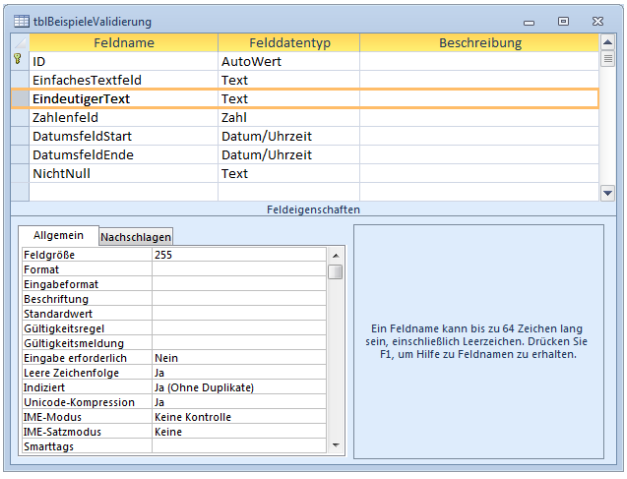


Bild 2: Das Feld **EindeutigerText** der Beispieltabelle hat einen eindeutigen Index.

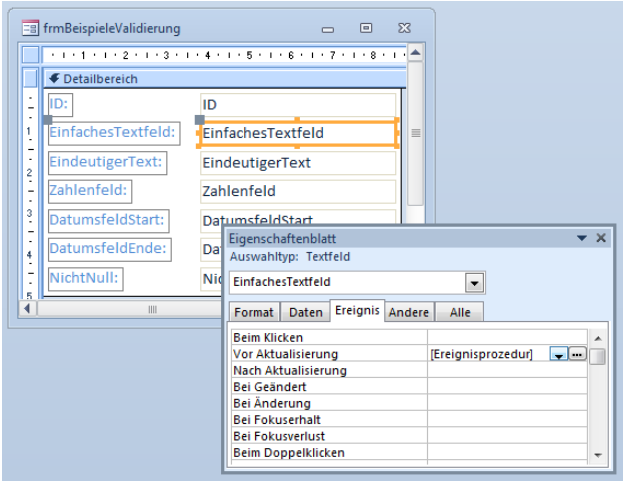


Bild 3: Anlegen der Ereignisprozedur, die nach der Eingabe ausgelöst werden soll

- **EindeutigerText**: Textfeld, dessen Eigenschaft **Indiziert** den Wert **Ja (Ohne Duplikate)** enthält
- **Zahlenfeld**: Feld zur Eingabe von Zahlenwerten
- **DatumsfeldStart** und **DatumsfeldEnde**: Felder zur Demonstrierung der Validierung von Datumsbereichen
- **NichtNull**: Textfeld, dessen Eigenschaft **Eingabe erforderlich** den Wert **Ja** enthält

Das Formular **frmBeispieleValidierung** enthält diese Tabelle als Datenherkunft. Damit das Formular alle in der Tabelle enthaltenen Felder anzeigt, haben wir diese in der Entwurfsansicht aus der Feldliste in den Detailbereich des Formulars gezogen.

Validierungsprozedur anlegen

Wenn die Validierung gleich nach dem Eingeben eines Wertes in ein Feld ausgelöst werden soll, hinterlegen Sie für die Ereigniseigenschaft **Vor Aktualisierung** eine entsprechende Prozedur. Um diese anzulegen, gehen Sie wie folgt vor:

- Öffnen Sie das Formular in der Entwurfsansicht.
- Markieren Sie das Steuerelement, für das Sie die Ereignisprozedur anlegen möchten.
- Wählen Sie für die Eigenschaft **Vor Aktualisierung** des Steuerelements den Eintrag **[Ereignisprozedur]** aus und klicken Sie auf die Schaltfläche mit den drei Punkten (siehe Bild 3).
- Der VBA-Editor zeigt nun den leeren Prozedurrumpf an (siehe Bild 4).

Für die folgenden Experimente fügen Sie nun zunächst eine einzige Anweisung hinzu, die dafür sorgt, dass beim Auslösen des Ereignisses ein Meldungsfenster erscheint:

```
Private Sub EinfachesTextfeld_BeforeUpdate(Cancel
                                    As Integer)
    MsgBox "Geändert!"
End Sub
```

Wechseln Sie dann in die Formularansicht des Formulars, geben Sie einen Text in das Feld ein und betätigen Sie die Eingabetaste. Wie erwartet erscheint das Meldungsfeld.

Sinn dieser Übung ist es nun, herauszufinden, zu welchen Gelegenheiten das Ereignis ausgelöst wird. Dies geschieht beispielsweise zu folgenden Anlässen, aber jeweils nur nach Eingabe mindestens eines Zeichens:

- Verlassen des Textfeldes mit der Tabulator- oder der Eingabetaste oder auch durch einen Mausklick in ein anderes Feld

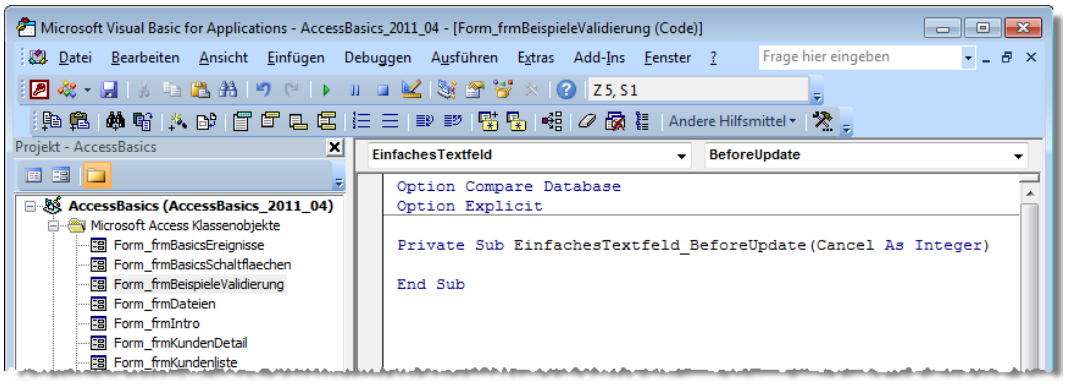


Bild 4: Anlegen der Ereignisprozedur, die nach der Eingabe ausgelöst werden soll

- Speichern des Datensatzes
- Schließen des Formulars

Übrigens ist entscheidend, ob Sie überhaupt ein Zeichen eingeben, nicht, ob der Inhalt geändert wurde. Wenn Sie ein Zeichen hinzufügen und dieses gleich wieder löschen, hat das Feld zwar nachher den gleichen Wert – das Ereignis **Vor Aktualisierung** wird aber trotzdem ausgelöst.

Validierung programmieren

Nun fügen wir die Zeilen Code hinzu, die den Inhalt des Steuerelements prüfen. Zunächst formulieren wir den Code allgemein und fügen anschließend die gewünschte Bedingung hinzu.

Die folgende Prozedur prüft die Bedingung **<Validierungsbedingung>**. Ist diese wahr, führt sie drei Anweisungen durch. Die erste zeigt die Meldung **<Validierungsmeldung>** an, die zweite stellt den Parameter **Cancel** auf den Wert **True** ein und die dritte beendet die aktuelle Prozedur:

```
Private Sub EinfachesTextfeld_BeforeUpdate(Cancel ↵
                                        As Integer)
    If <Validierungsbedingung> Then
        MsgBox "<Validierungsmeldung>"
        Cancel = True
        Exit Sub
    End If
End Sub
```

Eine wichtige Rolle spielt der Parameter **Cancel**. Wenn Sie diesem den Wert **True** übergeben, wird das Übernehmen des Wertes des aktuellen Steuerelements unterbrochen. Die Einfügemarke verbleibt dann im aktuellen Steuerelement – auch wenn Sie den Fokus auf ein anderes Steuerelement verschoben haben.

Die dritte Anweisung beendet die Validierungsprozedur. Warum das? Es kann sein, dass Sie mehrere Validierungsregeln für ein einziges Steuerelement festlegen. In diesem Fall sollte der Benutzer nicht gleich mehrere Meldungen hintereinander angezeigt bekommen, sondern nur die jeweils erste.

Validierungsregeln und -meldungen

Nun schauen wir uns einige Beispiele für Validierungsbedingungen an.

- **IsNull(Me!EinfachesTextfeld)**: Prüft, ob das Feld den Wert **Null** hat
- **IsNumeric(Me!EinfachesTextfeld)**: Prüft, ob das Feld einen numerischen Wert enthält.
- **Me!EinfachesTextfeld = "abc"**: Prüft, ob das Feld den Wert **abc** enthält.
- **Me!EinfachesTextfeld LIKE "A*"**: Prüft, ob das Feld mit **A** beginnt.
- **Len(Me!EinfachesTextfeld) = 5**: Prüft die Länge der im Feld enthaltenen Zeichenkette.
- **IsDate(Me!EinfachesTextfeld)**: Prüft, ob das Feld ein gültiges Datum enthält.
- **Me!DatumsfeldStart > "1.1.2011"**: Prüft, ob ein Datum größer als ein anderes Datum ist.

Mehrere Validierungen

Wenn es mehrere Regeln für die Eingabe eines Feldwertes gibt, sollten Sie diese auch in mehrere Prüfungen aufteilen. Sie können dem Benutzer so deutlich mitteilen, warum der Wert nicht gültig ist. In der Prüfung auf Datensatzebene, die wir uns gleich anschauen, benötigen Sie ohnehin Validierungen für mehrere Steuerelemente, die Sie nacheinander durchführen werden.

Validierung beim Speichern des Datensatzes

Einige Validierungen nehmen Sie am besten erst vor, wenn der Benutzer den Datensatz speichert. Dabei handelt es sich vor allem um solche Validierungen, bei denen die Werte zweier oder mehrerer Felder voneinander abhängen. Das Paradebeispiel sind Datumsangaben wie etwa in den beiden Feldern **DatumsfeldStart** und **DatumsfeldEnde** der Beispieltabelle. Das Startdatum darf natürlich nicht hinter dem Enddatum liegen.

Als erstes benötigen Sie wieder eine Ereignisprozedur, die wiederum durch das Ereignis **Vor Aktualisierung** ausgelöst wird. Allerdings gehört die entsprechende Ereigniseigenschaft diesmal zum Formular selbst und nicht zu einem der enthaltenen Steuerelemente. Das Ereignis **Vor Aktualisierung** des Formulars wird zum Beispiel durch folgende Aktionen ausgelöst – aber jeweils nur, wenn der Datensatz

Access-Funktionen

Teil III: Datum und Zeit

Datums- und Zeitangaben lassen sich unter Access hervorragend verwalten – wenn man weiß, wie es richtig funktioniert. Dieser Artikel zeigt, welche Funktionen Access zum Verarbeiten von Datums- und Zeitangaben in Ausdrücken und unter VBA bietet.

Zeitfunktionen

In den folgenden Abschnitten lernen Sie die folgenden Zeitfunktionen kennen (erste Angabe: deutscher Name, in Klammern: englischer Name der Funktion):

- **Datum** (**Date**): Gibt das aktuelle Systemdatum zurück.
- **Zeit** (**Time**): Gibt die aktuelle Systemzeit zurück.
- **Jetzt** (**Now**): Gibt das aktuelle Systemdatum und die aktuelle Systemzeit zurück.
- **IstDatum** (**IsDate**): Überprüft, ob ein Ausdruck in einen Wert vom Datentyp **Datum/Uhrzeit** konvertiert werden kann.
- **ZDate** (**CDate**): Wandelt eine gültige Datumseingabe in den Datentyp **Datum/Uhrzeit** um.
- **Jahr** (**Year**): Extrahiert die vierstellige Jahreszahl aus einem Datum.
- **Monat** (**Month**): Extrahiert den Monat (1-12) aus dem Datum.
- **Tag** (**Day**): Extrahiert den Tag (1-31) aus dem Datum.
- **Stunde** (**Hour**): Extrahiert die Stunde (0-23) aus dem Datum.
- **Minute** (**Minute**): Extrahiert die Minute (0-59) aus dem Datum.
- **Sekunde** (**Second**): Extrahiert die Sekunde (0-59) aus dem Datum.
- **DatTeil** (**DatePart**): Liefert einen durch einen Parameter bestimmten Teil des Datums.
- **Wochentag** (**Weekday**): Liefert den Wochentag des Datums.
- **DatSeriell** (**DateSerial**): Fügt aus den Komponenten **Jahr**, **Monat** und **Tag** einen Wert vom Datentyp **Datum/Uhrzeit** zusammen.
- **ZeitSeriell** (**TimeSerial**): Fügt aus den Komponenten **Stunde**, **Minute** und **Sekunde** einen Wert vom Datentyp **Datum/Uhrzeit** zusammen.
- **DatAdd** (**DateAdd**): Addiert ja nach Parameter Jahre, Monate, Tage, Stunden, Minuten oder Sekunden zum angegebenen Datum hinzu.
- **DatDiff** (**DateDiff**): Liefert die Anzahl der Zeitintervalle eines bestimmten Intervalls, die zwischen zwei Daten liegen.

Die deutsche Variante der Funktionen können Sie in der deutschsprachigen Version statt der englischen in Ausdrücken verwenden, also beispielsweise in Werten oder Kriterien in Abfragen oder als Eigenschaften von Formularen, Berichten oder Steuerelementen. Die englischen Varianten verwenden Sie in VBA-Code.

Sie können die englischen Bezeichnungen auch in Ausdrücken angeben, diese werden jedoch in der deutschen Version von Access in die deutschen Bezeichnungen umgewandelt. In Ausdrücken verwenden Sie das Semikolon als Trennzeichen zwischen den einzelnen Parametern, unter VBA das Komma.

Aktuelles Datum und aktuelle Zeit ermitteln

Die Funktionen **Datum**, **Zeit** und **Jetzt** liefern Informationen über die aktuelle Systemzeit. **Datum** ermittelt nur das Datum (**1.1.2011**), **Zeit** nur die Zeit (**13:23:10**) und **Jetzt** liefert beide Komponenten zusammen (**1.1.2011 13:23:10**).

Die Funktionen können Sie beispielsweise einsetzen, um den Zeitpunkt des Anlegens eines Datensatzes in einer Tabelle zu speichern. Dazu tragen Sie einfach die gewünschte Funktion für die Eigenschaft **Standardwert** des Feldes zum Speichern dieser Infor-

mation ein (siehe Bild 1). In diesem Fall soll die Funktion **Jetzt** gleich Datum und Uhrzeit gemeinsam in das Feld eintragen. Als Datentyp legen Sie optimalerweise **Datum/Uhrzeit** fest.

Die Funktionen **Datum**, **Zeit** und **Jetzt** werden auch gern in Berichten eingesetzt. Wenn Sie beispielsweise jeweils das aktuelle Datum im Seitenfuß der Berichts anzeigen möchten, öffnen Sie diesen in der Entwurfsansicht, fügen dem Seitenfußbereich ein neues Textfeld hinzu und tragen für seine Eigenschaft Steuerelementinhalt den Ausdruck **=Datum()** ein (siehe Bild 2).

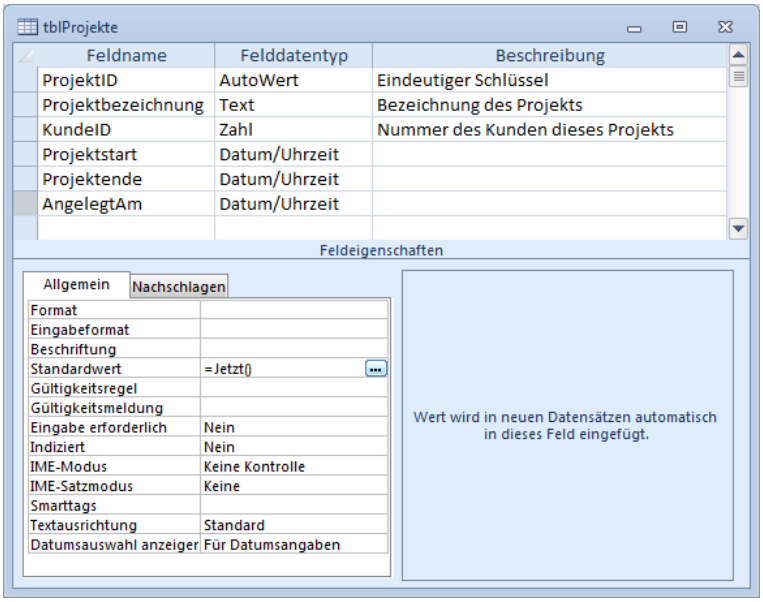


Bild 1: Anlegen einer Datumsfunktion als Standardwert eines Tabellenfeldes

Datum und Zeit als Parameter angeben

In den folgenden Abschnitten werden wir oft Datum- und Zeitangaben als Parameter verwenden. Die Frage ist: Welche Formate sind hier zulässig? Es gibt einige Standardformate, die in allen Umgebungen arbeiten und einige landesspezifische Formate.

In Deutschland können Sie ein Datum beispielsweise als **1.1.2011**, **1. Jan 2011** oder **1. Januar 2011** angeben.

Bei einem englischen Betriebssystem werden diese Angaben als Parameter von Datumsfunktionen zu Fehlern führen. Umgekehrt kommt ein deutsches System nicht mit englischen Formaten zurecht.

Nichts verkehrt machen Sie, wenn Sie ein allgemeingültiges Format verwenden. Dieses lautet beispielsweise **yyyy/mm/dd hh:nn:ss**, wobei **nn** für Minuten steht (die anderen Abkürzungen sollten selbsterklärend sein).

Prüfen, ob ein Wert ein gültiges Datum ist

Mit der Funktion **IstDatum** (**IsDate**) prüfen Sie, ob ein Datum ein gültiges Datum ist. Das zu prüfende Datum geben Sie dabei als Parameter an. Im Direktfenster des VBA-Editors (Tastenkombination **Strg + G**) prüfen Sie die Gültigkeit eines Datums beispielsweise mit der folgenden Anweisung:

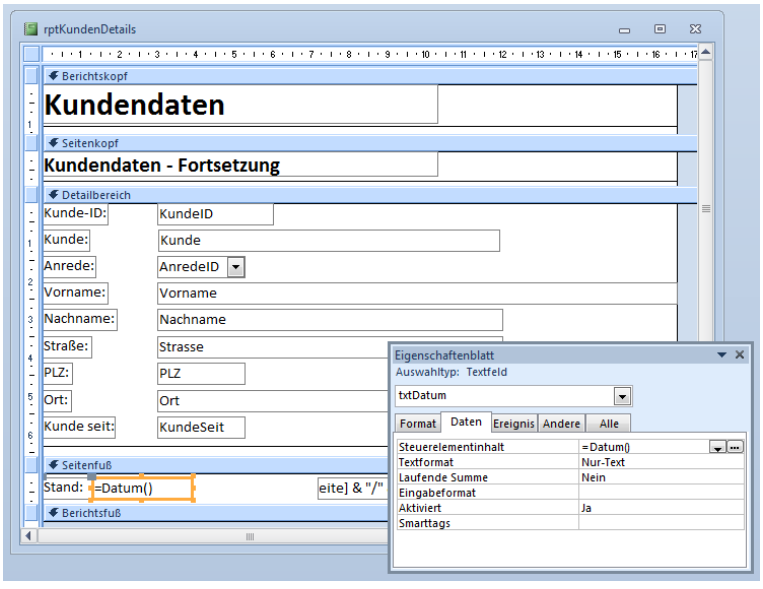


Bild 2: Verwenden einer Datumsfunktion im Seitenfuß eines Berichts

```
? IsDate("1.1.2010")
Wahr
```

Wenn der Parameter kein gültiges Datum liefert, gibt die Funktion den Wert **False** zurück. Das gilt sowohl für Daten im falschen Format als auch für ungültige Datumsangaben:

```
? IsDate("1. January 2010")
Falsch
? IsDate("29.2.2011")
Falsch
```

Werte in Datum umwandeln

Werte, die noch nicht im Datumsformat vorliegen, können Sie mit der Funktion

ZDate (**CDate**) in den Datentyp **Datum/Zeit** konvertieren.

ZDate liefert einen Fehler, wenn Sie versuchen, eine ungültige Angabe in ein Datum zu konvertieren (siehe Bild 3). Dabei ist die Funktion teilweise schwer einzuschätzen. Während die **IstDatum**-Funktion zum Beispiel eine Zahl wie **33333** nicht als Datum erkennt, wandelt die **ZDatum**-Funktion diesen Wert in ein Datum um:

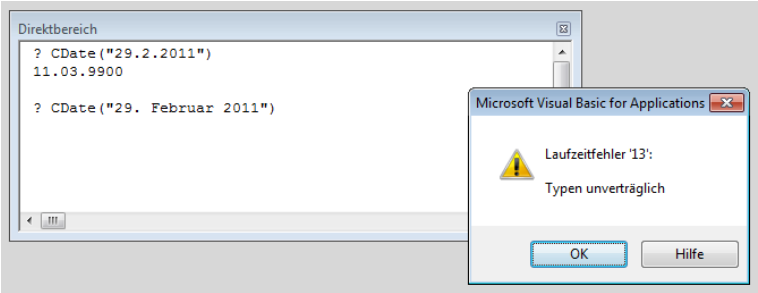


Bild 3: Das Konvertieren ungültiger Datumsangaben liefert eine Fehlermeldung.

```
? CDate(33333)
05.04.1991
```

Beim Datum **29.2.2011**, das ja eigentlich nicht existiert, liefert **IstDatum** den Wert **Falsch**. Die **ZDate**-Funktion hingegen erkennt hier ein Datum:

```
? CDate("29.2.2011")
11.03.9900
```

Hier werden schlicht die Punkte entfernt, die Zeichenkette in eine Zahl umgewandelt und schon hat man den 2.922.011-ten Tag nach dem 30.12.1899.

Tag, Monat und Jahr extrahieren

Die Funktionen **Tag** (**Day**), **Monat** (**Month**) und **Jahr** (**Year**) liefern Bestandteile des als Parameter angegebenen Datums.

Diese Funktionen haben wir in einer Beispielabfrage ohne Datenherkunft eingesetzt (**qryDatumExtrahieren**). Sie liefert pro Feld eines der Bestandteile des Datums als Ganzzahl (siehe Bild 4).

Ein praktischer Einsatzzweck ergibt sich, wenn Sie beispielsweise alle Kunde ausgeben möchten, die im folgenden Monat Geburtstag haben. Dabei verwenden Sie die Funktion **Monat()** gleich zwei Mal: Einmal zum Ermitteln des Monats, in dem der Kunde Geburtstag hat und einmal zum ermitteln der Nummer des kommenden Monats als Kriterium (siehe Bild 5).

Stunde, Minute und Sekunde extrahieren

Das Gleiche gelingt mit den Stunden, Minuten und Sekunden einer Zeitangabe. Die folgenden drei Aufrufe haben wir im Direktfenster des VBA-Editors abgesetzt:

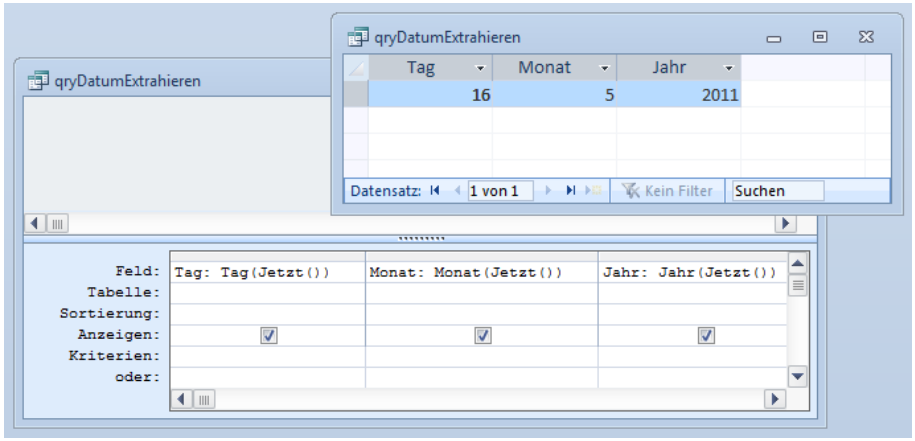


Bild 4: Extrahieren der Datumsbestandteile in einer Abfrage

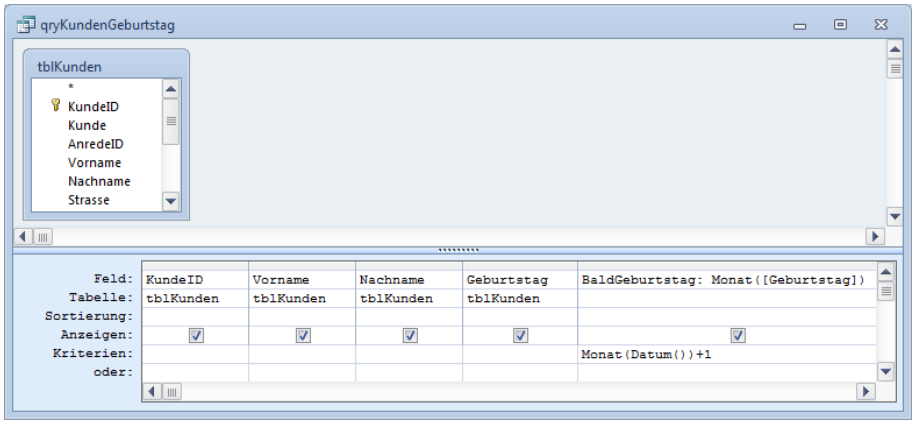


Bild 5: Welche Kunden haben nächsten Monat Geburtstag?

Abfragen für die Datenauswahl

Teil X: Datum und Zeit

Wenn Sie Datum- und Zeitangaben in einer Tabelle speichern, werden Sie auch in Abfragen über diese Informationen auf Datensätze zugreifen wollen. Damit beantworten Sie dann Fragen wie "Wieviele Artikel wurden in einem bestimmten Zeitraum verkauft?". Beim Zusammenstellen der Vergleichskriterien gibt es jedoch einiges zu beachten.

Einfache Vergleiche mit Datumswerten

Bereits beim einfachen Vergleichen bestehender Werte mit Datumskriterien fällt auf, dass Access als Kriterium angegebene Datums- und Zeitangaben gleich ändert.

Wenn Sie etwa wie in Bild 1 ein einfaches Datum eingeben, wandelt Access dieses Kriterium in den Ausdruck **#17.05.2011#** um.

Für die folgenden Beispiele haben wir eine kleine Tabelle namens **tblTermine** angelegt, die schlicht einige Datensätze mit verschiedenen Terminen enthält (siehe Bild 2).

Das Ergebnis der Abfrage nach Terminen mit dem Datum **17.5.2011** liefert erstaunlicherweise nur den Termin mit der ID **1** zurück, obwohl der folgende Datensatz als Datum auch den **17.5.2011** enthält (siehe Bild 3).

Der Grund erschließt sich, wenn Sie sich die Natur eines Datumsfeldes ansehen: Es enthält intern einen Double-Wert, dessen Stellen vor dem Komma die Anzahl der Tage nach dem 30.12.1899 darstellen und deren Nachkommastellen die Uhrzeit repräsentieren. Der erste Datensatz enthält also eigentlich den Wert **40680** im Feld **Termin**, während dieser Wert im zweiten Datensatz **40680,7155787037** lautet. Und genau diese beiden Werte vergleicht die Abfrage auch: Sie sind nicht gleich, und deshalb zeigt die Abfrage nur den ersten Wert an.

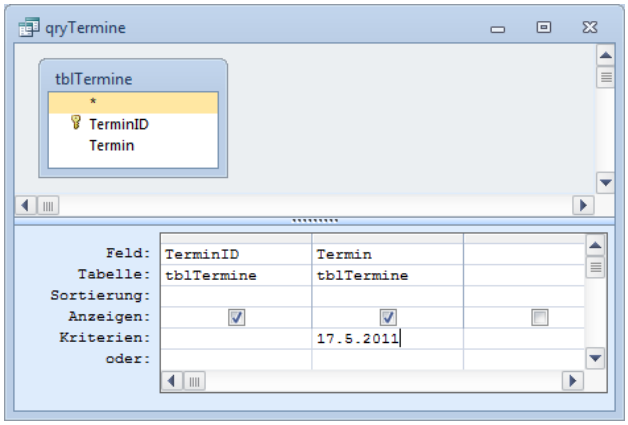
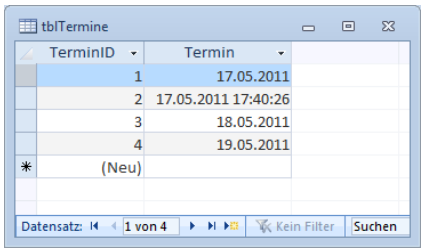


Bild 1: Eingabe eines einfachen Datums als Kriterium

Zeiträume abfragen

Wie also erhalten wir alle Datensätze, deren Termin auf den 17.5.2011 fällt? Access liefert für Abfragen die Zwischen-Klausel. Probieren wir es also mit dem Ausdruck **Zwischen #17.05.2011# Und #18.05.2011#** als Kriterium für das Feld **Termin** (siehe Bild 4). Das Ergebnis liefert diesmal zwar beide Termine, die am 17.5.2011 stattfinden, aber auch den vom 18.5.2011. Klar: Das Kriterium lautete ja prinzipiell auch **Zwischen 40680 Und 40681**, und dort fällt der 18.5.2011

Bild 2: Tabelle mit einigen Terminen



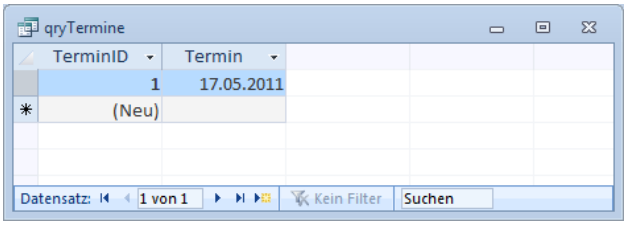


Bild 3: Sind das alle Termine am 17.5.2011?

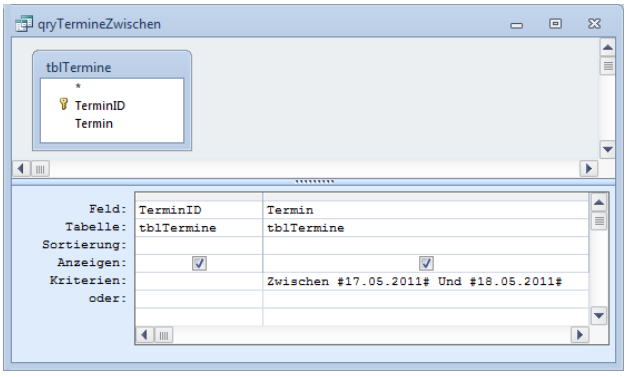


Bild 4: Abfrage mit einem Zwischen-Kriterium

VBA-Grundlagen

Schleifen in VBA

Wenn Sie in VBA einen Vorgang mehr als einmal durchführen möchten, verwenden Sie dazu eine sogenannte Schleife. Davon gibt es verschiedene Arten: Eine wird entsprechend einer vorgegebenen Anzahl durchlaufen, andere bis zum Eintreten einer bestimmten Bedingung. Dieser Artikel stellt die verschiedenen Schleifenarten vor.

Schleifen

Grundsätzlich sind Schleifen Code-Konstrukte und ähneln vom Aufbau her den VBA-Routinen. Sie besitzen einen Schleifenkopf und eine abschließende Anweisung, die allerdings in diesem Fall nur das Ende eines Durchlaufs der Anweisungen innerhalb der Schleife bedeutet.

Beispiele

Die Beispiele dieses Artikels finden Sie im Modul **mdlSchleifen** der Beispieldatenbank **Access_Basics_2011_04.mdb**.

For...Next-Schleife

Die **For...Next**-Schleife zeichnet sich dadurch aus, dass zu Beginn festgelegt wird, wie oft die enthaltenen Anweisungen durchlaufen werden.

Um nachzuhalten, wie oft die Anweisungen innerhalb der Schleife bereits durchlaufen wurden, wird eine sogenannte Laufvariable verwendet, die meistens **i** genannt wird.

Die erste Zeile der **For...Next**-Schleife enthält außerdem noch den ersten und den letzten Wert sowie gegebenfalls eine Schrittweite. Die Standardschrittweite lautet **1**. Man gibt nur in Ausnahmefällen eine alternative Schrittweite an – dazu später mehr.

Die letzte Zeile einer **For...Next**-Schleife enthält lediglich das **Next**-Schlüsselwort und optional den Namen der Laufvariable (dies dient ausschließlich der Übersicht). Insgesamt sieht eine **For...Next**-Schleife, die beispielsweise mit den Werten von **1** bis **10** für die Variable **i** durchlaufen werden soll, wie folgt aus:

```
Dim i As Integer
For i = 1 To 10
    Debug.Print i
Next i
```

Innerhalb der **For...Next**-Schleife wird der Wert von **i** im Direktfenster ausgeben. Dort erscheinen beim Aufrufen der Prozedur **ForNext_I** des Moduls **mdlSchleifen** der Beispieldatenbank also die Zahlen von **1** bis **10**. Der Ablauf der **For...Next**-Schleife wird auch vom Flussdiagramm in Bild 11 skizziert. Die Schleife startet mit dem Wert **1** für die Variable **i**.

Enthält **i** einen Wert kleiner oder gleich dem Endwert der Schleife, werden die Anweisungen des Schleifenkörpers ausgeführt und **i** um eins erhöht.

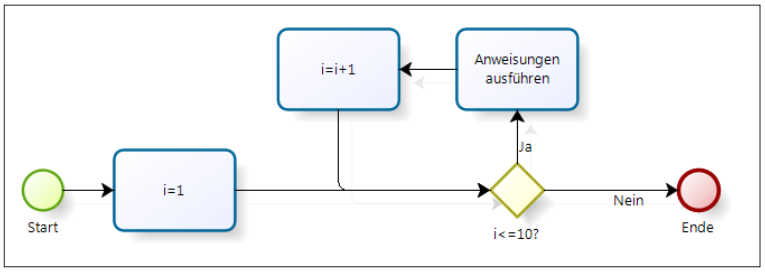


Bild 11: Flussdiagramm für eine **For...Next**-Schleife

Monate ausgeben

Die Laufvariable können Sie gleich sinnvoll nutzen. Das folgende Beispiel durchläuft etwa die Zahlen von 1 bis 12 und gibt die entsprechenden Monatsnamen aus:

```
For i = 1 To 12
    Debug.Print Format("1." & i & "." & Year(Date), _
                       "mmmm")
Next i
```

Schrittweite mit Step einstellen

Der **For...**-Zeile können Sie noch das Schlüsselwort Step mit der gewünschten Schrittweiter hinzufügen. Wenn Sie die Zahlen von **10** bis **1** rückwärts durchlaufen möchten, geben Sie als Startwert **10**, als Endwert **1** und für Step den Wert **-1** an:

```
For i = 10 To 1 Step -1
    Debug.Print i
Next i
```

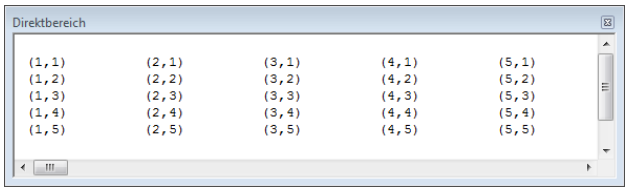


Bild 1: Ausgabe zweier verschachtelter **For...Next**-Schleifen

Exit For

Gelegentlich werden Sie eine **For...Next**-Schleife vorzeitig verlassen wollen. In diesem Fall können Sie die **Exit For**-Anweisung verwenden:

```
For i = 1 To 10
    Debug.Print i
    If i = 5 Then
        Exit For
    End If
Next i
```

Verschachtelte Schleifen

Möglicherweise möchten Sie einmal zwei oder mehr verschachtelte Schleifen verwenden. Dazu benötigen Sie entsprechend viele Laufvariablen, im folgenden Beispiel i und j:

```
Dim i As Integer
Dim j As Integer
For i = 1 To 5
    For j = 1 To 5
        Debug.Print "(" & j & "," & i & ")",
    Next j
    Debug.Print
Next i
```

Die obigen Schleifen geben eine Zahlenmatrix wie in Bild 1 aus. Damit jeweils fünf Einträge nebeneinander ausgegeben werden, beenden Sie die **Debug.Print**-Anweisung der inneren Schleife mit dem Komma-Zeichen.

Dies bedeutet, dass die folgende **Debug.Print**-Anweisung in die gleiche Zeile drucken soll – und zwar mit einem Tabulator-Schritt als Abstand. Sie können auch das Semikolon als letztes Zeichen angeben. Die Ausgabe wird dann unmittelbar hinter dem letzten Zeichen fortgesetzt.

Damit die Ausgabe nach fünf Elementen in der folgenden Zeile fortgesetzt wird, ruft die Prozedur nach der Abarbeitung der inneren Schleife eine **Debug.Print**-Anweisung ohne auszugebenden Text auf. Dies bewirkt lediglich den Sprung in die nächste Zeile.

Alternative Laufvariablen

Eine Laufvariable muss nicht zwingend eine Ganzzahl des Typs Integer oder Long sein. Sie können auch mit Dezimalzahlen arbeiten und mit dem Step-Schlüsselwort etwa Schritte von 0,1 verarbeiten:

```
Dim s As Single
For s = 0 To 1 Step 0.1
    Debug.Print s
Next s
```

Das Beispiel hat allerdings noch einen kleinen Schönheitsfehler, denn die Ausgabe im Direktfenster sieht so aus:

```
 0
 0,1
 0,2
 0,3
 0,4
 0,5
 0,6
 0,7
 0,8000001
 0,9000001
```

Hier fällt erstens auf, dass nach **0,7** ein Rundungsfehler auftritt. Außerdem fehlt der eigentlich erwartete Wert **1**. Dieses Problem ist typisch für Gleitkommazahlen. Im vorliegenden Fall sollten Sie also besser einen Festkomma-Datentyp wie **Currency** verwenden:

```
Dim c As Currency
For c = 0 To 1 Step 0.1
    Debug.Print c
Next c
```

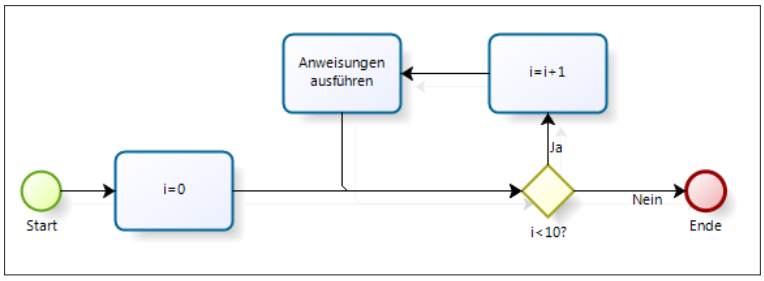


Bild 2: Flussdiagramm für eine **Do While...Loop**-Schleife

Werte zu Kombinationsfeldern hinzufügen

Kombinationsfelder bieten meist Daten aus Lookup-Tabellen zur Auswahl an. Das bedeutet, dass Sie damit etwa die Anrede oder den Titel einer Person festlegen können, wobei Anreden und Titel in separaten Tabellen gespeichert sind. Das ist schon hilfreich. Noch praktischer wäre es allerdings, wenn Sie neue Einträge für die separaten Tabellen direkt über das Kombinationsfeld eintragen könnten. Wie dies funktioniert, zeigt dieser Artikel.

Beispieldatenbank

Die Beispiele zu diesem Artikel finden Sie in der Datenbank **1105_KombifelderErweitern.mdb**.

Lookup-Tabellen

Wie Sie Lookuptabellen anlegen, haben Sie bereits in **Anreden und Co. mit Wertlisten oder Lookup-Tabellen verwalten** erfahren. Der Benutzer kann dort über ein Kombinationsfeld auf einfache Weise beispielsweise einen der Werte **Herr** oder **Frau** als **Anrede** für eine Person auswählen.Das ist natürlich kein besonders gutes Beispiel, wenn es darum geht, die Daten einer Lookup-Tabelle zu erweitern. Interessanter sind Kandidaten wie Abteilungen, Funktionen, Kategorien et cetera.

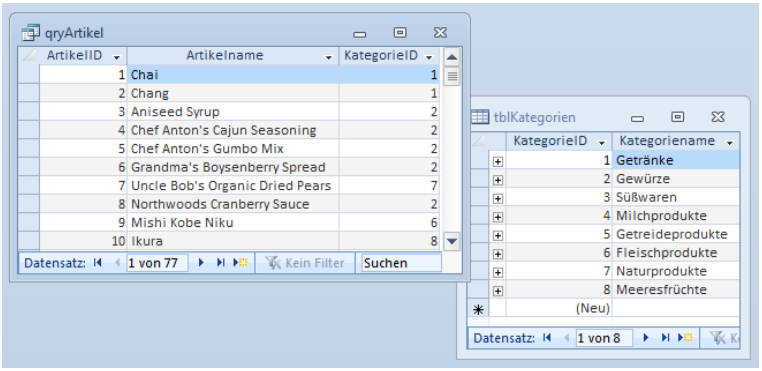


Bild 1: Artikel und Kategorien werden über das Fremdschlüsselfeld **KategorieID** verknüpft

Im Rahmen dieses Artikels kümmern wir uns um die Artikelkategorien der Südsturm-Datenbank und wollen diese über die Benutzeroberfläche erweitern. Dabei interessieren uns besonders die beiden Tabellen **tblArtikel** und **tblKategorien**. Die Tabelle **tblArtikel** enthält ein Feld namens **KategorieID**, mit dem der Primärschlüsselwert der passenden Kategorie der Tabelle **tblKategorien** festgelegt werden kann (siehe Bild 1).

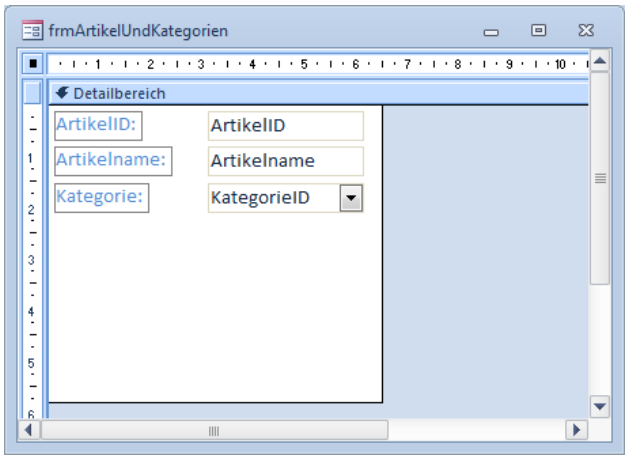


Bild 2: Das Beispielformular in der Entwurfsansicht

Damit der Benutzer die Kategorie einfach per Nachschlagefeld beziehungsweise Kombinationsfeld auswählen kann, wurde das Feld **KategorieID** wie in →*[basics]* **Nachschlagefeld für eine 1:n-Beziehung einrichten** beschrieben in ein Nachschlagefeld umgewandelt.

Formular zur Anzeige von Artikeln und Kategorien

Das Formular aus Bild 2 heißt **frmArtikelUndKategorien** und verwendet die Tabelle **tblArtikel** als Datenherkunft. Um das Beispiel einfach zu halten, haben wir einfach nur die drei Felder **ArtikelID**, **Artikelname** und **KategorieID** in den Detailbereich der Entwurfsansicht des Formulars gezogen. Dadurch, dass das Feld **KategorieID** bereits im Tabellenentwurf als Nachschlagefeld definiert wurde, wird es auch im Formular gleich als Kombinationsfeld ausgeführt. Wenn Sie nun in die Formularansicht wechseln, können Sie vorhandene Artikel bearbeiten oder neue Artikel eingeben und dabei die Kategorie bequem per Kombinationsfeld auswählen (siehe Bild 3).

Beim Anlegen der Steuerelemente, die an die Felder **ArtikelID**, **Artikelname** und **KategorieID** gebunden sind, vergibt Access eben diese Bezeichnungen als Steuerelementnamen. Damit wir später im Artikel besser zwischen den Feldern und Steuerelementen unterscheiden können, erweitern Sie die Namen der Steuerelemente um entsprechende Präfixe, also

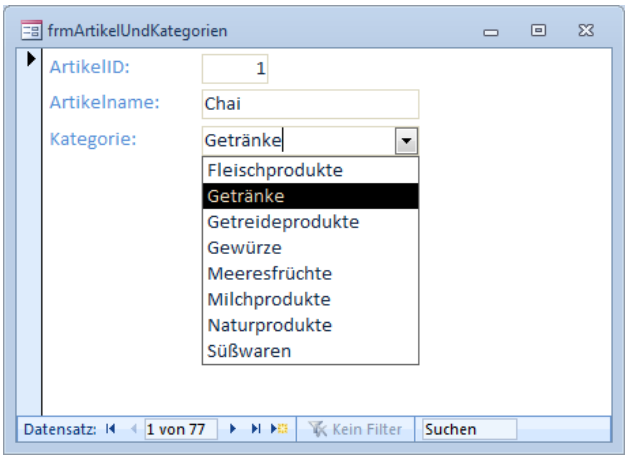


Bild 3: Kombinationsfeldeinträge auswählen

txtArtikelID, **txtArtikelname** und **cboKategorieID**. Mehr zu Präfixen erfahren Sie unter **Konventionen**.

Neue Kategorien eingeben

Was aber, wenn Sie nun einen Artikel anlegen, der sich in keine der vorhandenen Kategorien einsortieren lässt? Ganz klar: Der Benutzer öffnet die Tabelle **tblKategorien**, trägt die neue Kategorie ein und kann diese dann nach dem Aktualisieren des Formulars **frmArtikelUndKategorien** auswählen ... oder doch nicht? Nein! Der Benutzer soll auf gar keinen Fall direkt auf die Tabellen einer Anwendung zugreifen. Sie als Entwickler dürfen natürlich in Ausnahmefällen direkt Daten in Tabellen bearbeiten, aber dem Benutzer sollten Sie geeignete Formulare zur Verfügung stellen.

Es gibt nun zwei Möglichkeiten zur Eingabe von Kategorien:

- Sie legen ein neues Formular an, das an die Tabelle **tblKategorien** gebunden ist und erlauben darüber das Eingeben und Bearbeiten der Kategorien. Dieses Formular können Sie beispielsweise über eine Schaltfläche rechts vom Kombinationsfeld öffnen.

- Sie erlauben dem Benutzer, neben den vorhanden Kategorien auch neue Kategorien direkt in das Kombinationsfeld einzugeben. Diese werden dann, gegebenenfalls nach Rückfrage, in der Tabelle **tblKategorien** gespeichert.

Beide Varianten haben Vorteile: Die erste erlaubt gleichzeitig, die bestehenden Kategorien zu bearbeiten oder auch mal gleich mehrere neue Kategorien einzugeben. Die zweite ist direkter: Der Benutzer braucht nur den Kategorietext einzugeben und diese wird direkt für den aktuellen Artikel übernommen. In diesem Artikel betrachten wir die zweite Methode, die erste schauen wir uns in einem weiteren Artikel namens **Meldungsfenster anzeigen und auswerten** an.

Kombinationsfeld aufbohren

Ob der Benutzer neue Werte in ein Kombinationsfeld eingeben kann, hängt in erster Linie von der Eigenschaft **Nur Listeneinträge** ab (siehe Bild 4). Nun könnten Sie auf die Idee kommen, dass wir diese Eigenschaft für unsere Zwecke, nämlich im laufenden Betrieb direkt neue Kategorien in das Kombinationsfeld einzugeben, auf den Wert **Nein** einstellen müssen. Der Versuch scheitert: Die Eigenschaft darf laut

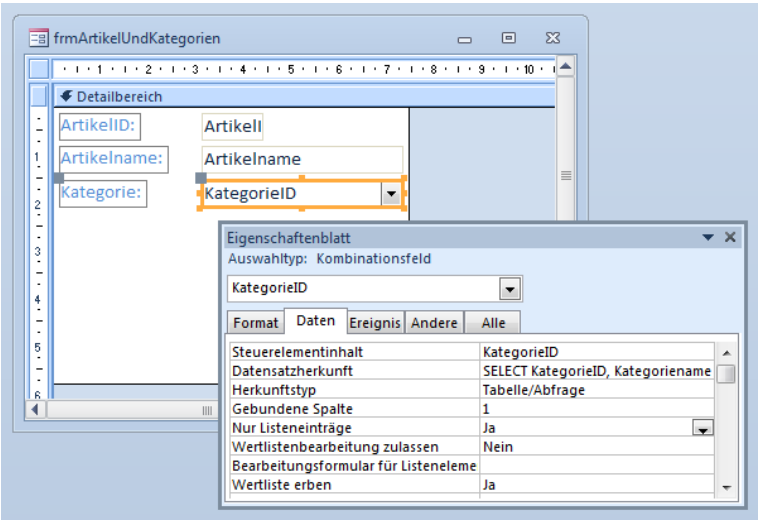


Bild 4: Die Eigenschaft **Nur Listeneinträge**

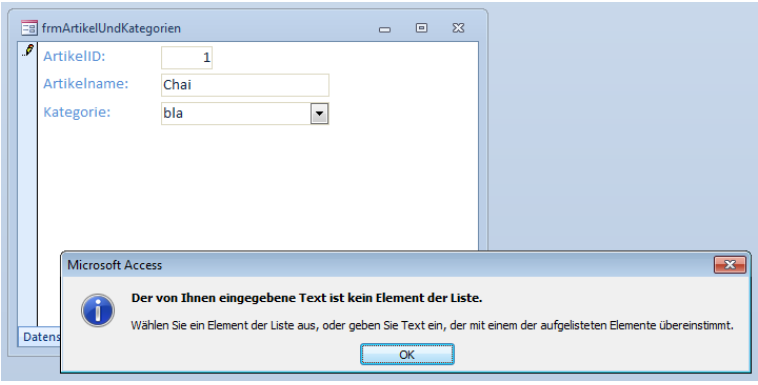


Bild 5: Diese Fehlermeldung erscheint, wenn die Eigenschaft **Nur Listeneinträge** den Wert **Ja** enthält.

der dabei erscheinenden Meldung nur auf **Nein** eingestellt werden, wenn die erste sichtbare Spalte die gebundene Spalte ist.

In unserem Fall enthält die erste Spalte der Datensatzherkunft des Kombinationsfeldes das Feld **KategorieID**, angezeigt wird aber die zweite Spalte **Kategoriename**. Die Eigenschaft **Nur Listeneinträge** könnten Sie nur auf **Nein** einstellen, wenn die **KategorieID** als erstes Feld im Kombinationsfeld angezeigt würde.

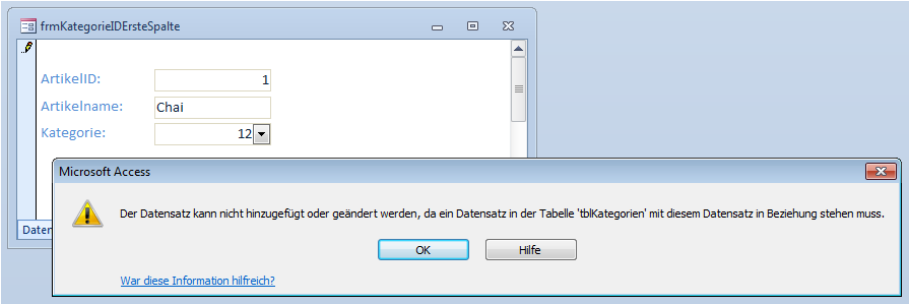


Bild 6: Fehler beim Einfügen eines neuen Wertes in ein Kombinationsfeld, das durch eine Beziehung mit referentieller Integrität nur vorhandene Werte oder **Null** aufnehmen kann.

Das lässt sich ändern, indem Sie die Eigenschaft **Spaltenbreiten** auf einen Wert wie **1cm** ändern – die Daten des Feldes **KategorieID** werden dann auf einer Breite von einem Zentimeter angezeigt, das Feld **Kategoriename** nimmt den verbleibenden Platz der aufgeklappten Liste ein (im Steuerelement selbst wird nun nur noch die **KategorieID** angezeigt).

Nun lässt sich die Eigenschaft **Nur Listeneinträge** auf den Wert **Nein** einstellen. Ist es nun auch möglich, neue Werte für die gebundene Spalte des Steuerelements, also für die **KategorieID**, einzugeben? Nein: Das Feld **KategorieID** ist als Fremdschlüsselfeld zum gleichnamigen Feld der Tabelle **tblKategorien** ausgelegt und gleichzeitig gibt es eine mit referentieller Integrität festgelegte Beziehung zwischen diesen beiden Feldern.

Dies besagt, dass das Fremdschlüsselfeld **KategorieID** nur den Wert **Null** oder einen der Werte des Feldes **KategorieID** der Tabelle **tblKategorien** aufnehmen darf. Sprich: Sie können nun zwar theoretisch Werte in das Kombinationsfeld **cboKategorieID** eingeben, aber dies löst einen Fehler aus, weil die entsprechende Beziehung nur vorhandene Werte oder **Null** als Wert des zugrunde liegenden Feldes erlaubt.

Wenn Sie dieses Verhalten reproduzieren möchten, schauen Sie sich das Formular **frmKategorieID-ErsteSpalte** an (siehe auch Bild 6).

Listeneinträge und mehr

Es gibt jedoch eine Konstellation, die es erlaubt, Listeneinträge auszuwählen und auch neue Einträge zu einem Feld hinzuzufügen. Dies wäre beispielsweise der Fall, wenn die Kategorie nicht aus einer separaten Tabelle ausgewählt wird, sondern einfach nur in ein Textfeld der Tabelle **tblArtikel** eingetragen würde.

Diese Variante beleuchtet der Beitrag **Kombinationsfeld als Eingabehilfe für Textfelder** in Ausgabe 6/2011 von **Access [basics]**.

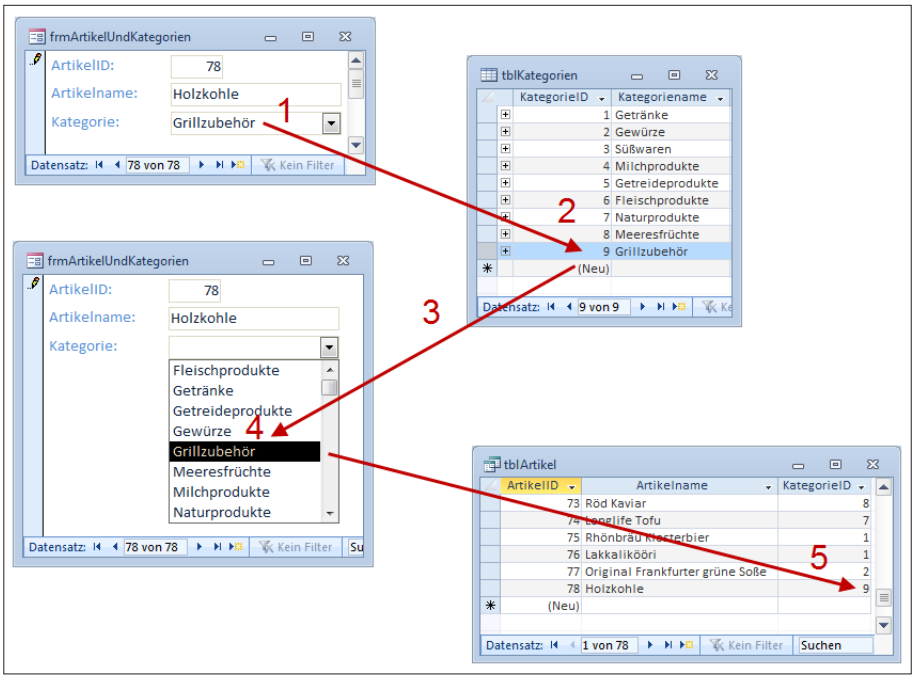


Bild 7: Ablauf beim Hinzufügen eines Eintrag zu einer Lookup-Tabelle per Kombinationsfeld

Aktionsabfragen per VBA ausführen

Aktionsabfragen lassen sich bequem mit der Entwurfsansicht für Abfragen zusammenstellen. Sie können damit Daten an Tabellen anfügen, bestehende Daten ändern oder löschen und sogar gleich die passende Tabelle zum Einfügen von Daten erstellen. Wie aber soll der Benutzer eine solche Abfrage aufrufen, ohne doppelt auf den entsprechenden Eintrag im Datenbankfenster oder im Navigationsbereich zu klicken? Dieser Artikel zeigt, wie Sie dies realisieren.

Beispieldatenbank

Die Beispiele zu diesem Artikel finden Sie in der Datenbank **1105_AktionsabfragenPerVBA.mdb**.

Aktionsabfragen

Wenn Sie eine Aktionsabfrage erstellen, gehen Sie wie in **Aktionsabfragen – Teil II: Löschabfragen** oder **Aktionsabfragen – Teil I: Anfügeabfragen** beschrieben vor. Weitere Artikel zu Aktualisierungsabfragen oder Tabellenerstellungsabfragen folgen in späteren Ausgaben.

Das Resultat beim Erstellen einer Aktualisierungsabfrage mit der Abfrageentwurfsansicht ist eine gespeicherte Abfrage. Diese führen Sie per Doppelklick auf den entsprechenden Eintrag im Datenbankfenster oder im Navigationsbereich aus.

Ein Beispiel hierfür ist die Löschabfrage aus Bild 1. Sie ist unter dem Namen **qryArtikelLoeschen** gespeichert und bildet die Grundlage für das folgende Beispiel.

Die Abfrage löscht genau einen Datensatz, und zwar den mit dem Wert **1** im Feld **ArtikelID**. Das gilt natürlich nur, wenn ein solcher Datensatz überhaupt vorhanden ist. Sonst bewirkt der Aufruf dieser Abfrage gar nichts.

Die Execute-Methode

Nun folgt der interessante Teil: Access bietet eine ganze Reihe von VBA-Objekten an, die wiederum praktische Methoden etwa für den Zugriff auf Daten offerieren. In unserem Fall geht es um das Objekt, dass die aktuelle Datenbank repräsentiert.

Es hat den Datentyp **Database** und wird als Objektvariable deklariert. Das alles ist für Einsteiger vielleicht noch etwas unverständlich – das ist aber kein Problem. Manche Dinge kann man zu Beginn einfach als gegeben hinnehmen. In diesem Fall geht es um die folgenden Codezeilen:

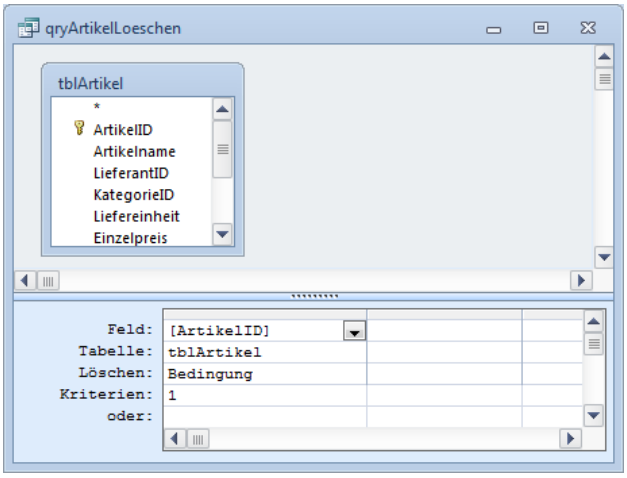


Bild 1: Entwurfsansicht einer Abfrage zum Löschen des Artikels mit dem Wert **1** im Feld **ArtikelID**.

```
Dim db As Database
Set db = CurrentDb
```

Die erste Zeile deklariert **db** als Objektvariable des Typs **Database**, die zweite füllt diese Variable mit dem Wert der Funktion **CurrentDb**. **CurrentDb** wiederum liefert einen Verweis auf die aktuelle Datenbank.

Dank dieser beiden Zeilen können Sie nachfolgend über die Variable **db** bestimmte Aktionen mit der Datenbank durchführen.

In unserem Fall soll eine Aktionsabfrage durchgeführt werden, genau genommen eine Löschabfrage.

Der dazu benötigte Befehl des mit **db** referenzierten Objekts heißt **Execute**. **Execute** erwartet als Parameter zumindest eine Variable oder Zeichenkette, die folgendes enthält:

- den Namen der Aktionsabfrage, wie Sie ihn auch im Datenbankfenster oder im Navigationsbereich finden (hier also beispielsweise **qryArtikelLoeschen**)
- den SQL-Code der Aktionsabfrage (wie Sie diesen ermitteln, erfahren Sie gleich).

Am einfachsten ist es natürlich, wenn Sie eine **DELETE**-Abfrage per Abfrageentwurf zusammenstellen und diese direkt per VBA aufrufen möchten. Die nötigen Anweisungen sehen dann so aus:

```
Private Sub EinfacherAufruf()
    Dim db As DAO.Database
    Set db = CurrentDb
    db.Execute "qryArtikelLoeschen"
    Set db = Nothing
End Sub
```

Wenn Sie das Beispiel nachvollziehen möchten, legen Sie ein neues Standardmodul an und fügen Sie den obigen Code dort ein (→*[basics]* **VBA-Modul anlegen**).

Wenn Sie als Parameter der **Execute**-Methode einfach nur den Namen einer Aktionsabfrage angeben, wird diese einfach aufgerufen. Allerdings müssen bereits alle Einstellungen in der Abfrage selbst vorgenommen werden – zum Beispiel, ob alle Datensätze der Tabelle gelöscht werden sollen oder nur einer. Im obigen Beispiel wird die Abfrage **qryArtikelLoeschen** ausgelöst, was den Artikel mit dem Wert **1** im Feld **ArtikelID** löscht.

Die Abfrage führen Sie beispielsweise aus, indem Sie die Einfügemarke irgendwo innerhalb der Prozedur platzieren und dann die Taste **F5** betätigen oder den Menübefehl **Ausführen|Sub/Userform ausführen** aufrufen.

Für den Anfang prüfen Sie nun per Sichtkontrolle, ob der Datensatz tatsächlich gelöscht wurde – und zwar, indem Sie die Tabelle **tblArtikel** öffnen und schauen, ob der Datensatz noch vorhanden ist.

Achtung: Keine Rückfrage!

Die **Execute**-Anweisung führt die angegebene Aktionsabfrage ohne Rückfrage aus. Wenn Sie also etwa einen Artikel erst löschen möchten, wenn der Benutzer dem explizit zugestimmt hat, müssen Sie die **Execute**-Anweisung noch in eine **If...Then**-Bedingung einfassen, die das Ergebnis einer **MsgBox**-Funktion prüft. Dies kann etwas allgemein formuliert etwa so aussehen:

```
If MsgBox("Datensatz löschen?", vbYesNo) = vbYes Then
    Set db = CurrentDb
    db.Execute "qryArtikelLoeschen"
End If
```

Aktionsabfrage als SQL-Ausdruck angeben

Manchmal kann es hilfreich sein, die Aktionsabfrage nicht als Abfrage zu speichern, sondern direkt auszuführen. Das bedeutet, dass Sie der **Execute**-Anweisung nicht den Namen der gespeicherten Abfrage, sondern einen entsprechenden SQL-Ausdruck angeben.

Fehlende SQL-Kenntnisse sind hier kein Problem: Access bietet die Möglichkeit, Abfragen in der SQL-Ansicht anzuzeigen. Und aus dieser können Sie den gewünschten SQL-Ausdruck einfach herauskopieren und in den VBA-Code einfügen.

Im Falle der Aktionsabfrage unseres Beispiels sieht das so aus:

- Öffnen Sie die Abfrage in der Entwurfsansicht.
- Klicken Sie mit der rechten Maustaste auf die Titelzeile und wählen Sie den Eintrag **SQL-Ansicht** aus (siehe Bild 2).

Fertig! Der SQL-Ausdruck zur aktuell geöffneten Aktionsabfrage wird nun wie in Bild 3 angezeigt.

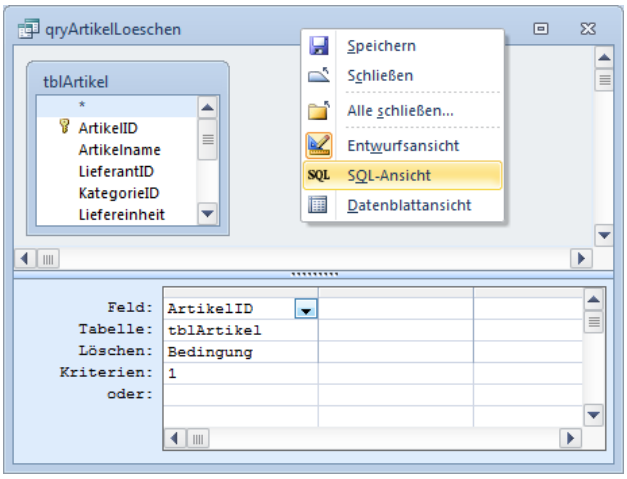


Bild 2: SQL-Ansicht einer Abfrage aktivieren

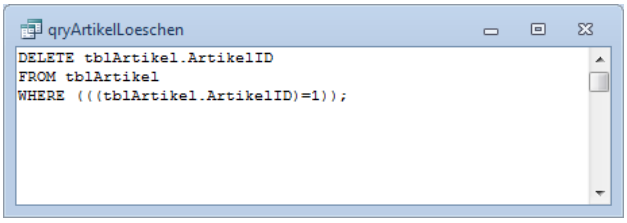


Bild 3: SQL-Ansicht einer Abfrage

Daten per Kombinationsfeld auswählen und löschen

Wenn Sie den Benutzern Ihrer Datenbank die Gelegenheit bieten möchten, Datensätze gezielt zu löschen, lässt sich das ganz leicht etwa mit einem Kombinationsfeld und einer Schaltfläche erreichen. Mit dem Kombinationsfeld wählt der Benutzer den zu löschenden Datensatz aus und die Schaltfläche startet den eigentlichen Löschvorgang.

Beispieldatenbank

Die Beispiele zu diesem Artikel finden Sie in der Datenbank **1105_DatenPerKombifeldLoeschen.mdb**.

Beispiel: Artikel löschen

Das Beispiel zu diesem Artikel soll ein Formular enthalten, in dem der Benutzer einen Artikel auswählt und diesen mit einem Klick auf eine Schaltfläche löscht. Die Auswahl des zu löschenden Artikels soll der Einfachheit halber über ein Kombinationsfeld erfolgen, dass die Felder **ArtikelID** und den **Artikelname** der Tabelle **tblArtikel** als Datensatzherkunft enthält.

Zum Nachvollziehen des Beispiels erstellen Sie ein neues Formular namens **frmArtikelLoeschen** und fügen ein Kombinationsfeld und eine Schaltfläche wie in Bild 1 hinzu. Das Kombinationsfeld nennen Sie **cboArtikelID** und die Schaltfläche **cmdArtikelLoeschen**. Damit das Kombinationsfeld die Artikel aus der Tabelle **tblArtikel** zur Auswahl anbietet, klicken Sie in den Eigenschaften des Kombinationsfeldes auf **Datensatzherkunft** und dann auf die rechts erscheinende Schaltfläche mit den drei Punkten. Es öffnet sich der Abfrageeditor. Dort fügen Sie aus der Liste der Tabellen den Eintrag **tblArtikel** hinzu und ziehen die beiden Felder **ArtikelID** und **Artikelname** in das Entwurfsraster. Stellen Sie außerdem die Sortierung für das Feld **Artikelname** auf **Aufsteigend** ein.

Schließen Sie den Abfrageentwurf und prüfen Sie, ob Access den folgenden Ausdruck in die Eigenschaft Datensatzherkunft übernommen hat – dies ist der SQL-Code der Abfrage:

```
SELECT tblArtikel.ArtikelID, tblArtikel.Artikelname
FROM tblArtikel;
```

Stellen Sie außerdem die beiden Eigenschaften **Spaltenanzahl** und **Spaltenbreiten** auf die Werte **2** und **0cm** ein. Auf diese Weise wird die erste Spalte mit den Werten des Feldes **ArtikelID** mit der Breite **0** angezeigt und somit ausgeblendet, das zweite Feld **Artikelname** nimmt den verbleibenden Platz ein. Nach einem Wechseln in die Formularansicht können Sie

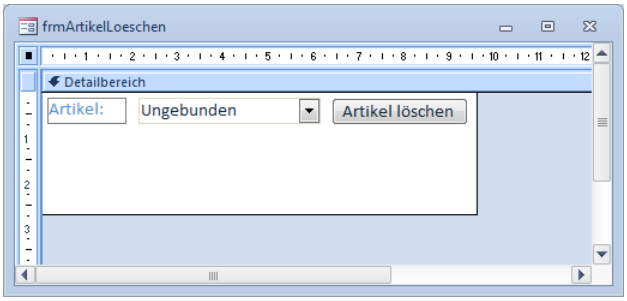


Bild 1: Entwurfsansicht des Formulars zum Löschen von Artikeln

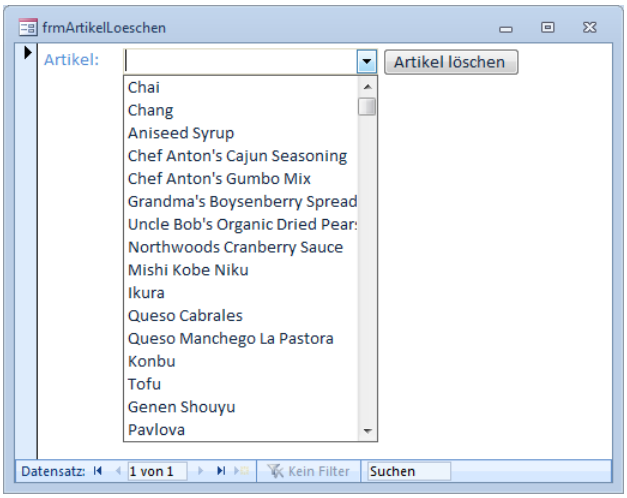


Bild 2: Auswahl eines Artikels per Kombinationsfeld

das Auswählen des zu löschenden Artikels bereits testen (siehe Bild 2).

Löschen per Mausklick

Fehlt noch die Schaltfläche **cmdLoeschen**: Diese soll nach dem Anklicken den aktuell ausgewählten Datensatz aus der Tabelle **tblArtikel** löschen. Dazu legen Sie zunächst eine leere Ereignisprozedur für das Ereignis **Beim Klicken** an. Wählen Sie dazu in der Entwurfsansicht des Formulars den Eintrag **[Ereignisprozedur]** für die Eigenschaft **Beim Klicken** der Schaltfläche aus und klicken Sie dann auf die Schaltfläche mit den drei Punkten. Der VBA-Editor zeigt nun die leere Prozedur an:

```
Private Sub cmdArtikelLoeschen_Click()
End Sub
```

Meldungsfenster anzeigen und auswerten

Das Meldungsfenster ist ein wichtiges und einfach zu bediendendes Element zur Interaktion mit dem Benutzer. Einfache Meldungsfenster zeigen einfach nur Meldungen an und werden durch einen Klick auf die OK-Schaltfläche wieder geschlossen. Sie können damit jedoch auch einfache Fragen stellen, die der Benutzer per Mausklick auf die unterschiedlichen Schaltflächen beantworten kann.

Beispieldatenbank

Die Beispiele zu diesem Artikel finden Sie in der Datenbank **1105_Meldungsfenster.mdb**.

Einfaches Meldungsfenster

Meldungsfenster zeigen Sie mit der **MsgBox**-Anweisung an. Auch wenn diese Anweisung mehrere Parameter anbietet, reicht der Einsatz des ersten Parameters in vielen Fällen schon aus. Dabei soll schlicht und einfach ein Text angezeigt werden:

```
MsgBox "Hallo"
```

Sie können diese Anweisung zum Ausprobieren einfach im Direktfenster des VBA-Editors eingeben. Dieses öffnen Sie am schnellsten mit der Tastenkombination **Strg + G**. Die besondere Eigenart des Meldungsfensters ist, dass die aufrufende Anwendung nicht fortgeführt wird, bis der Benutzer das Meldungsfenster wieder geschlossen hat.

Parameter der MsgBox-Anweisung

Das Meldungsfenster kann natürlich nicht nur einfache Texte anzeigen. Sie können auch die Titelzeile festlegen, eines von vier Icons auswählen und ein Set von Schaltflächen festlegen.

Dies sind die vier Parameter des **MsgBox**-Anweisung:

- **prompt**: Die soeben bereits verwendete Meldung.
- **buttons**: Gibt an, welche Schaltflächenkombination das Meldungsfenster anzeigen soll.

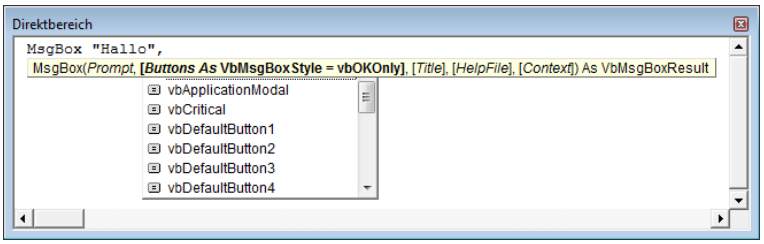


Bild 2: Der zweite Parameter offenbart mehr Optionen als sein Name besagt.

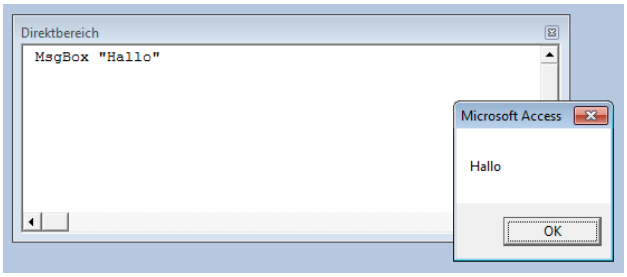


Bild 1: Die einfachste Variante eines Meldungsfensters

- **title**: Legt den Text in der Titelzeile des Meldungsfensters fest.

Die beiden übrigen Parameter **helpfile** und **context** werden in der Praxis so gut wie gar nicht eingesetzt und deshalb hier nicht beschrieben.

Meldungstext

Der Meldungstext kann ca. 1.024 Zeichen lang sein. Zeilenumbrüche erhalten Sie, indem Sie den Text auf zwei durch Anführungszeichen eingefasste Zeichenketten aufteilen und die Konstante **vbCrLf** wie folgt integrieren:

```
MsgBox "Erste Zeile" & vbcrlf & "Zweite Zeile"
```

Schaltflächen und Co.

Wenn Sie im Direktbereich mit der Eingabe der Parameter der **MsgBox**-Funktion experimentieren, stoßen Sie schnell darauf, dass der zweite Parameter offensichtlich nicht nur für die Darstellung der Schaltflächen zuständig ist. In der Tat bietet IntelliSense eine ganze Reihe von Werten für diesen Parameter an (siehe Bild 2) – und nur einige scheinen tatsächlich für Schaltflächenkombinationen zu stehen. Dabei handelt es sich um die folgenden Werte:

- **vbAbortRetryIgnore**: Zeigt die Schaltflächen **Abbrechen**, **Wiederholen** und **Ignorieren** an.
- **vbOKCancel**: Zeigt die Schaltflächen **OK** und **Abbrechen** an.

Daten in der Endlosansicht von Formularen anzeigen

Nicht immer soll ein Formular nur einen Datensatz gleichzeitig anzeigen. Manchmal wollen Sie dem Benutzer auch eine Liste mehrerer Datensätze einer Tabelle präsentieren. Dazu gibt es mehrere Möglichkeiten, zum Beispiel die Endlosansicht. Dieser Artikel zeigt Ihnen, wie Sie die Daten einer Tabelle übersichtlich in einem Formular darstellen.

Beispieldatenbank

Die Beispiele zu diesem Artikel finden Sie in der Datenbank **1105_Endlosformular.mdb**.

Varianten für die Anzeige mehrerer Datensätze

Grundsätzlich gibt es mehrere Möglichkeiten, Daten in Listenform in einem Formular anzuzeigen:

- Datenblattansicht in Formularen
- Endlosansicht in Formularen
- Listenfeld-Steuerelement
- nicht in Access eingebaute Steuerelemente wie etwa das ListView-Steuerelement

Die Datenblattansicht haben Sie bereits in einem anderen Zusammenhang kennengelernt: nämlich zur Anzeige von verknüpften Daten in Unterformularen (**Formulare für die Dateneingabe – Teil II: Formulare für Daten aus 1:n-Beziehungen**). Die Anzeige von Daten in Listenfeldern haben wir in **Steuerelemente – Teil II: Das Listenfeld** vorgestellt. Im vorliegenden Artikel schauen wir uns nun an, welche Möglichkeiten die Endlosansicht von Formularen bietet.

Artikel im Endlosformular

Für dieses Beispiel sollen die Datensätze einer Tabelle namens **tblArtikel** in der Endlosansicht eines Formulars angezeigt werden. Dazu führen Sie zunächst die folgenden Schritte durch:

- Legen Sie ein neues, leeres Formular an.
- Stellen Sie die Eigenschaft **Datenherkunft** auf **tblArtikel** ein.
- Ändern Sie die Eigenschaft **Standardansicht** auf **Endlosformular** ein (siehe Bild 1).

Das waren die Vorbereitungen. Nun geht es an den größten Teil der Arbeit: das Einfügen und Ausrichten der Steuerelemente.

Dazu müssen Sie grundsätzlich wissen, dass ein Formular in der Endlosansicht den Detailbereich und die darin enthaltenen Steuerelemente für jeden Datensatz einmal anzeigt. Wieviele Datensätze gleichzeitig sichtbar sind, ohne dass Sie scrollen müssen, hängt von mehreren Faktoren ab. Der erste ist die Höhe des für die Anzeige der Detailbereiche für die einzelnen Datensätze verfügbare Platz, der zweite die Höhe des Detailbereichs für einen Datensatz.

Die Höhe für die Detailbereiche entspricht der Höhe des Formulars minus der Höhe von Formularkopf und Formularfuß, sofern diese beiden Bereiche eingeblendet sind. Um den Formularkopf und -fuß einzublenden, klicken Sie mit der rechten Maustaste auf die Titelzeile des Detailbereichs (siehe Bild 2). Danach erscheinen die beiden Bereiche über- und unterhalb des Detailbereichs (siehe Bild 3).

Damit Sie das Zusammenspiel der einzelnen Bereiche genauer kennenlernen, erledigen Sie nun die folgenden Dinge:

- Fügen Sie im Formularkopf ein Beschriftungsfeld mit dem Text **Artikelübersicht** ein (die Schriftgröße können Sie gern auf einen größeren Wert einstellen).

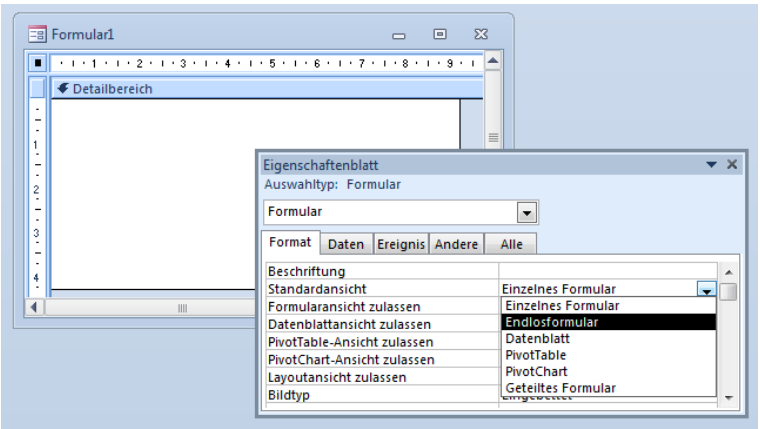


Bild 1: Die verschiedenen Icons im Meldungsfenster

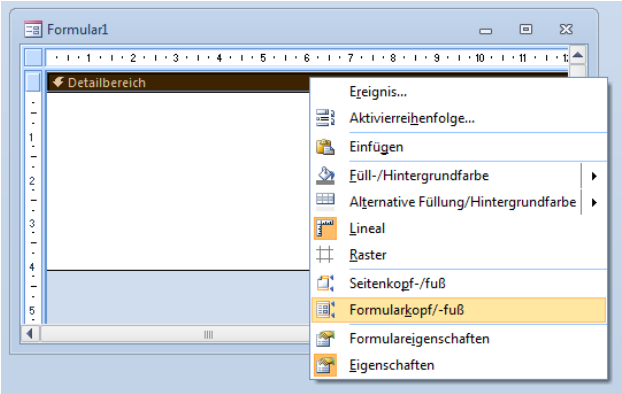


Bild 2: Einblenden von Formularkopf und -fuß

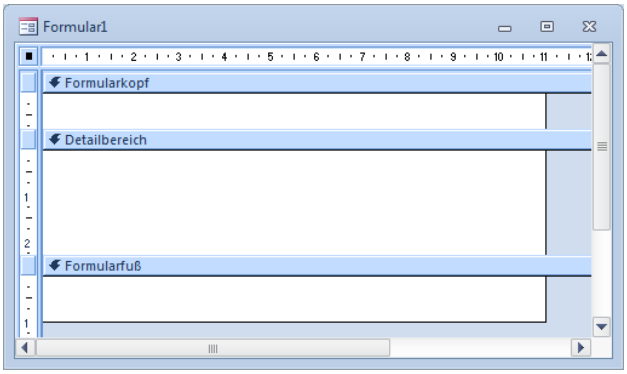


Bild 3: Formular mit Formularkopf und -fuß

- Legen Sie im Formularfuß eine Schaltfläche mit der Beschriftung **OK** und dem Namen **cmdOK** an.

Fügen Sie im Detailbereich vorerst zwei Felder der in der Datenherkunft angegebenen Tabelle hinzu. Dazu aktivieren Sie am einfachsten die Feldliste mit der Schaltfläche mit der Tastenkombination **Alt + F8** und ziehen die zwei Felder **ArtikelID** und **Artikel** in den Detailbereich (siehe Bild 4). Nach einem Wechsel in die Formularansicht sieht das Formular wie in Bild 5 aus. Die Inhalte von Formularkopf und Formularfuß werden oben und unten angezeigt, dazwischen erscheint der Detailbereich so oft, wie es der verbleibende Platz zulässt.

Wenn Sie einen Datensatz etwa zum Löschen markieren möchten, klicken Sie wie in der Datenblattansicht auf den Datensatzmarkierer links und betätigen die **Entfernen**-Taste. Zwischen den Datensätzen können Sie wie gewohnt mit den Navigationsschaltflächen unten navigieren.

Hier tritt auch gleich der entscheidende Unterschied zur Datenblattansicht zutage: Sie können die Steuerelemente zur Anzeige der Feldinhalte des Datensatzes beliebig innerhalb des Detailbereichs anordnen.

Und es wird noch besser: Sie können sogar Schaltflächen im Detailbereich unterbringen, die für jeden einzelnen Datensatz angezeigt werden. Damit können Sie dem Benutzer eine Löschen-Schaltfläche anbieten, mit der ein Datensatz ohne vorheriges Selektieren mit dem Datensatzmarkierer gelöscht werden kann.

Übrigens erreichen Sie die wechselnde Hintergrundfarbe nur unter Access 2007 und jünger, ältere Access-Versionen bieten dieses Feature noch nicht an.

Daten in Tabellenform im Endlosformular

Wenn Sie die übrigen Felder der Tabelle **tblArtikel** zum Detailbereich der Entwurfsansicht hinzufügen, wird es dort etwas unübersichtlich. Das ist logisch, denn die Beschriftungsfelder und die eigentlichen Steuerelemente zur Anzeige der Feldinhalte nehmen Platz weg. Zumindest die Beschriftungsfelder können wir aber viel besser anordnen – nämlich im Formularkopf.

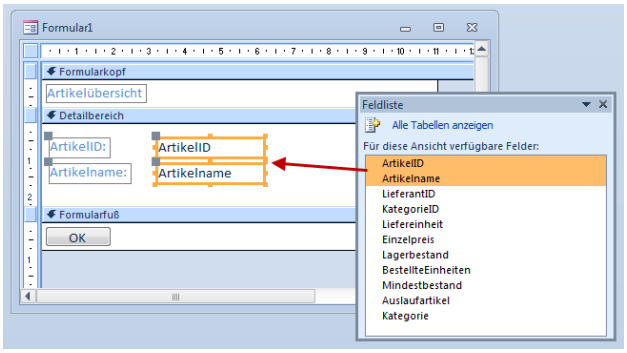


Bild 4: Hinzufügen der Felder zum Detailbereich der Entwurfsansicht

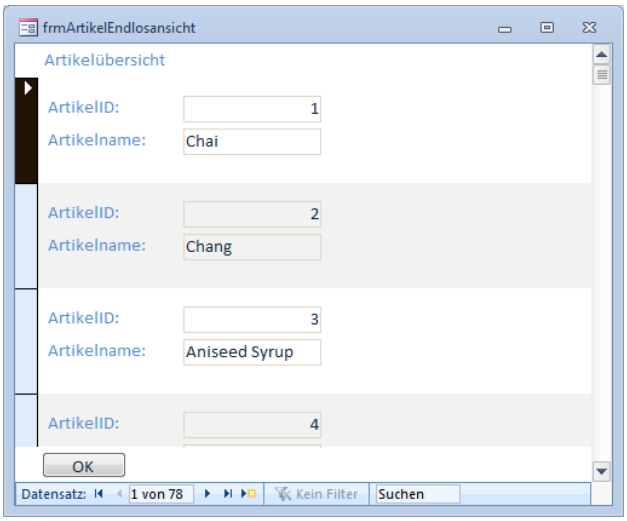


Bild 5: Ein Formular mit mehreren Datensätzen in der Endlosansicht

CSV-Export für Adressdaten

Der Export von Daten ist eine sehr wichtige Funktion unter Access. Ein Export ist schnell erledigt – Sie rufen einfach den Export-Assistenten auf, tragen die notwendigen Daten ein und schreiben die Daten dann in eine entsprechende Text-, CSV- oder Excel-Datei. Im vorliegenden Beitrag wollen wir uns ansehen, welche Anforderungen es in der Praxis gibt. Dazu werfen wir einen genauen Blick auf einen Export, der genauen Spezifikationen genügen muss.

Beispieldatenbank

Die Beispiele zu diesem Artikel finden Sie in der Datenbank **1106_CSVExport.mdb**.

Zum Götterboten

In diesem Artikel geht es um den Paketversand mit Hermes. Hier soll keine Schleichwerbung gemacht werden, aber Hermes bietet eine Online-Schnittstelle, der Sie eine CSV-Datei mit Adressdaten übergeben können. Diese Daten werden dann online gespeichert und können für den späteren Druck von Versandetiketten weiterverwendet werden.

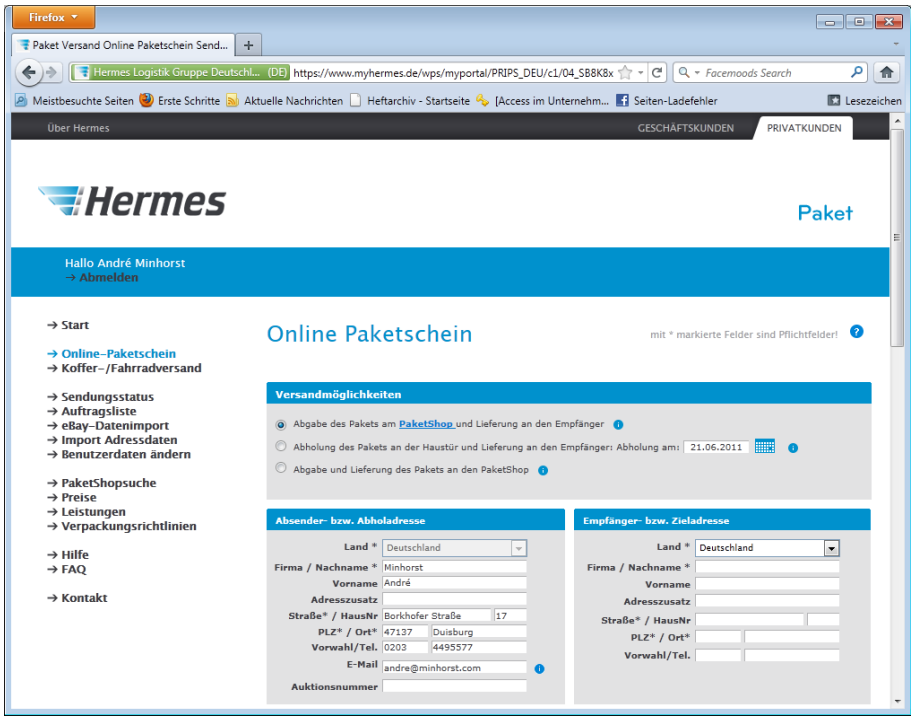


Bild 1: Beispiel für einen Online-Paketschein

Wie bei vielen anderen Anbietern können Sie bei Online-Beauftragung ein paar Cent oder gar einen Euro sparen. Sie geben dann die Empfänger- und die Absenderdaten online ein, drucken einen Paketschein aus, versehen Ihre Sendung damit und geben diese beim Postamt oder beim jeweiligen Dienstleister ab.

Im Beispiel ist dies nicht anders: Das Formular aus Bild 1 etwa ist bereits zur Hälfte vorausgefüllt, weil der Absender sich registriert und seine Adressdaten hinterlegt hat. Hier braucht man nur noch die Daten des Empfängers einzutragen, weiter unten im Formular die Paketgröße auszuwählen und zwei Häkchen zu setzen – etwa zum Akzeptieren der AGB.

Die vorliegenden Absenderdaten sparen schon einige Arbeit. Wenn Sie jedoch regelmäßig Sendungen an Ihre Kunden verschicken und deren Adressdaten in einer Datenbank vorliegen, macht es unnötig Arbeit, all diese Daten manuell in das Formular aus Bild 1 zu übertragen.

Ein Blick in die Navigationsleiste links offenbart jedoch einen Eintrag namens **Import Adressdaten**. Das hört sich gut an! Einen Klick weiter zeigt sich das Formular aus Bild 2. Hier können Sie eine Datei von Ihrer Festplatte angeben und diese zum Einlesen der enthaltenen Daten hochladen. Fehlt nur noch eine Information: Wie soll diese Datei aussehen und wie bekommen wir die vorliegenden Adressdaten aus der Datenbank dort hinein?

Import-Vorgaben

Normalerweise bieten Dienstleiter wie der aus unserem Beispiel eine Beschreibung der zu liefernden Daten an. Das diese nicht nach Gutdünken zusammengestellt werden können, liegt auf der Hand: Schließlich soll die Datei automatisch eingelesen werden und muss daher einem vorgegebenen Schema entsprechen. In diesem Fall bietet der Dienstleister zwei Varianten an – eine für nationale und eine für internationale Sendungen. Für nationale Sendungen sehen die Vorgaben so aus:

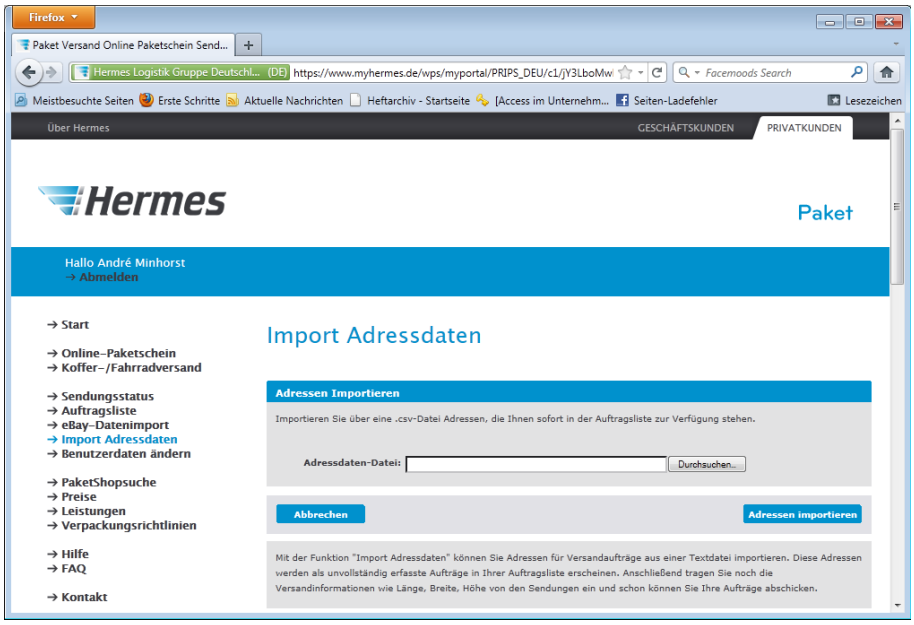


Bild 2: Schnittstelle zum Hochladen von CSV-Dateien mit Adressenlisten

- Jede Adresse landet in einer eigenen Zeile.
- Jede Zeile hat zehn Felder.
- Die Felder werden durch das Semikolon (;) getrennt, die Zeilen durch einen Zeilenumbruch.
- Es können maximal 150 Datensätze importiert werden.
- Die zehn Felder sind so aufgebaut (maximale Zeichenzahl in Klammern): **Vorname** (20), **Nachname** (25), **Adresszusatz** (20), **Straße** (27), **Hausnummer** (5), **PLZ** (5), **Ort** (25), **Tel. Vorwahl** (6), **Telefonnummer** (8), **E-Mail-Adresse** (80).

Als erstes fällt auf, dass Straße und Hausnummer getrennt angegeben werden sollen. Ups! In den bisherigen Beispieldatenbanken haben wir Straße und Hausnummer immer in einem einzigen Feld gespeichert! Aber keine Sorge: Straße und Hausnummer können auch zusammen in das Feld **Straße** eingetragen werden.

Auch für die Telefonnummer gibt es zwei Felder – aber diese ist wohl ohnehin nur den Fall gedacht, dass etwas Unvorhergesehenes mit der Lieferung geschieht und der Empfänger benachrichtigt werden muss.

Unter den zehn Feldern befindet sich keines, das einen Firmennamen aufnehmen kann – außer vielleicht das Feld **Adresszusatz**.

Ein Blick auf das Onlineformular zur Eingabe von Adressen zeigt jedoch, dass eine Firma gegebenenfalls mit im **Nachname**-Feld landet (**Firma/Nachname**).

Export vorbereiten

Die Daten zum Füttern der Online-Schnittstelle stammen aus der Tabelle **tblKunden** der Beispieldatenbank, deren Entwurf wie in Bild 3 aussieht. Das Problem bei dieser Tabelle ist, dass Sie nicht alle für den Export notwendigen Felder enthält. So können wir wie oben beschrieben nicht auf ein eigenes Feld für die Hausnummer zugreifen und auch das Feld **Adresszusatz** ist nicht vorhanden.

Außerdem sollen die Felder **Kunde** (als **Firma**) und **Nachname** in einem Feld der Exportdatei zusammengefasst werden – und das hier ebenfalls nicht der Fall. Die Lösung dieses Problems ist einfach: Sie

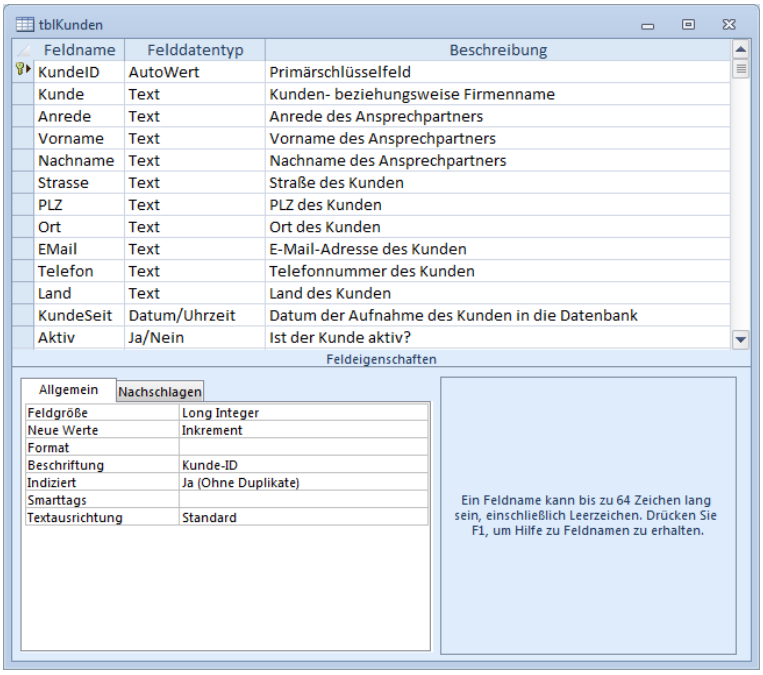


Bild 3: Schnittstelle zum Hochladen von CSV-Dateien mit Adressenlisten

erstellen eine Abfrage, welche die Felder der Tabelle **tblKunden** genau so ausgibt, dass daraus eine Exportdatei nach dem gewünschten Schema erstellt werden kann.

Gehen Sie dabei folgendermaßen vor:

- Erstellen Sie eine neue, leere Abfrage (**Abfrage erstellen**).
- Fügen Sie die Tabelle **tblKunden** als Datenherkunft hinzu.
- Legen Sie dann die benötigten Felder wie im Anschluss beschrieben an.

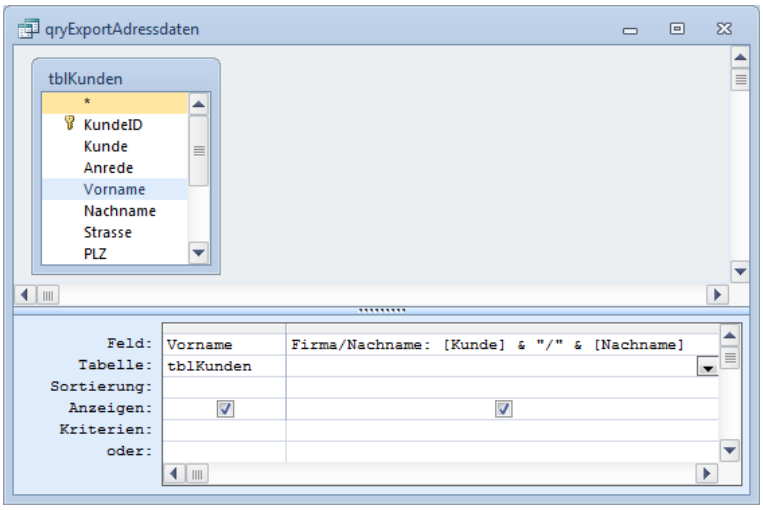


Bild 4: Entwurf der Abfrage zum Zusammenstellen der Adressdaten

Das Zusammenstellen der Felder ist die Hauptarbeit beim Export. Das erste Feld ist einfach: Den Vornamen liefert das Feld **Vorname**, also ziehen Sie dieses als erstes Feld in das Entwurfsraster.

Das Feld für **Firma/Nachname** ist gleich etwas schwieriger: Hier benötigen wir ein sogenanntes berechnetes Feld (siehe **Abfragen für die Datenauswahl – Teil VI: Berechnete Felder in Abfragen und Eigenschaften**). Zuerst legen Sie den Namen des Feldes fest, wie er in der Datenblattansicht der Abfrage erscheinen soll – in diesem Fall **Firma/Nachname**.

Dann folgt ein Doppelpunkt und schließlich der Ausdruck mit den benötigten Daten. Da die beiden Felder Kunde und Nachname zusammengeführt und durch einen Schrägstrich getrennt werden sollen, sieht der komplette Ausdruck so aus (siehe auch Bild 4):

```
Firma/Nachname: [Kunde] & "/" & [Nachname]
```

Ein Wechsel zur Datenblattansicht offenbart jedoch einige Schwächen des ersten Ansatzes (siehe Bild 5). Wenn **Kunde** und/oder **Nachname** fehlen, soll nämlich auch kein Schrägstrich angezeigt werden. Also verwenden wir die in **Access-Funktionen – Teil II: Bedingungen: Wenn(), Schalter() und Wahl()** vorgestellte **Wenn**-Funktion.

Der erste Parameter gibt den zu prüfenden Ausdruck an. Ist mindestens eines der beiden Felder **Kunde** oder **Nachname** leer? Die Prüfung sieht so aus: **IstNull([Kunde]) Oder IstNull([Nachname])**. Wenn die Bedingung eintritt, soll kein Schrägstrich ausgegeben werden, sonst ja. Also vervollständigen Sie den Ausdruck wie folgt:

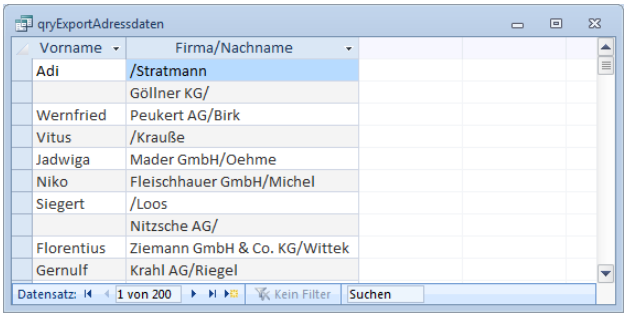


Bild 5: Der erste Versuch, Firma und Nachname zusammenzuführen

```
Firma/Nachname: [Kunde] & Wenn(IstNull([Kunde])
Oder IstNull([Nachname]);"";"/") & [Nachname]
```

Das lässt sich noch ein wenig kürzer formulieren. Dazu wenden Sie einen kleinen Trick an: Es gibt zum Verknüpfen von Texten nicht nur den Verknüpfungsoperator **&**, sondern auch das Plus-Zeichen (**+**). Wenn einer der beiden durch das Plus-Zeichen zusammengefassten Ausdrücke den Wert **Null** hat, wie es bei leeren Textfelder der Fall ist, dann ist auch die Verkettung beider Ausdrücke **Null**. Der Ausdruck sieht nun so aus:

```
Firma/Nachname: [Kunde] & Wenn(IstNull([Kunde]+[Nac
hname]);"";"/") & [Nachname]
```

Der Adresszusatz ist in unserem Fall immer leer. Dennoch muss dieses Feld im Export in Form eines leeren Feldes vorkommen. Es wird schlicht durch eine leere Zeichenkette zwischen zwei Semikola repräsentiert (**;;**). Das dritte Feld der Abfrage bezieht sich also nicht auf ein Feld der Herkunftstabelle, sondern verwendet schlicht den folgenden Ausdruck:

```
Adresszusatz: ""
```

Im- und Export-Spezifikationen definieren und verwenden

Wer oft Daten mit anderen Anwendungen austauscht und dabei die Import- und Export-Funktionen von Access nutzt, arbeitet meist mit den dafür vorgesehenen Assistenten. Dumm nur, wenn man die notwendigen Einstellungen für jeden Im- und Export neu vornehmen muss. Das muss nicht sein: Access bietet die Möglichkeit, die Einstellungen als Spezifikation zu speichern und diese später auf verschiedene Arten zu nutzen.

Beispieldatenbank

Die Beispiele zu diesem Artikel finden Sie in der Datenbank **1106_Spezifikationen.mdb**.

Spezifikationen speichern

Die Assistenten für den Import und Export von Daten kennen Sie bereits aus verschiedenen Artikeln. In **Daten importieren – Teil I: Daten aus CSV-Dateien** lernen Sie etwa den Import kennen, in **CSV-Export für Adressdaten** den Export von Daten.

Die Assistenten zum Im- und Exportieren besitzen eine unscheinbare, oft übersehene Schaltfläche namens **Erweitert...**.

Ein Klick darauf öffnet einen weiteren Dialog, der nicht nur alle Einstellungen übersichtlich darstellt, sondern der auch noch die Möglichkeit zum Speichern von Spezifikationen bietet, sondern auch eine zum erneuten Öffnen bereits gespeicherter Spezifikationen (siehe Bild 1).

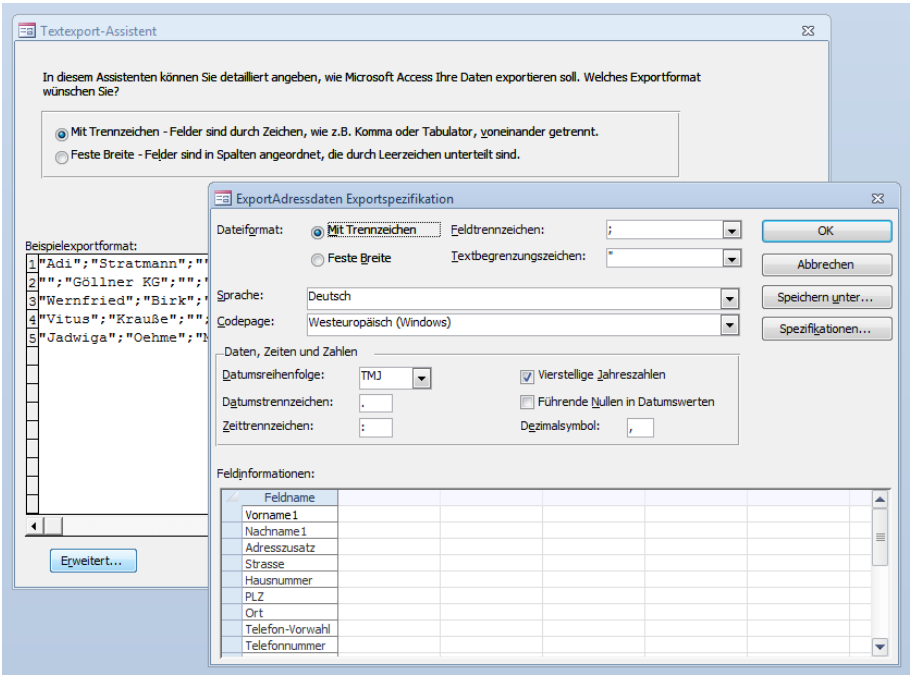


Bild 1: Der Dialog mit der Zusammenfassung der Einstellungen

Wenn Sie die gewünschten Einstellungen vorgenommen haben, klicken Sie auf die Schaltfläche **Speichern unter ...** und geben im nun erscheinenden Dialog **Import/Export-Spezifikation speichern** den Namen ein, unter dem Access die Spezifikation speichern soll.

Nachdem dies geschehen ist, klicken Sie auf die Schaltfläche **Spezifikationen ...**, um sich die Liste der verfügbaren Spezifikationen anzusehen (siehe Bild 2).

Import/Export nach Spezifikationen

Der Vorteil einer Spezifikation ist dieser: Sie können Daten damit importieren und exportieren, ohne jedesmal alle Einstellungen neu eingeben zu müssen. Ein kleiner Wermutstropfen bleibt dennoch: Sie können die Quell-/Zieldatei nicht mit der Spezifikation speichern, sondern müssen diese jedes Mal neu auswählen.

Das heißt, dass Sie beim Verwenden einer gespeicherten Spezifikation den Assistenten starten, die Quell-/Zieldatei festlegen, im Assistenten zur **Erweitert...**-Sektion wechseln und dort über die Schaltfläche Spezifikationen die gewünschte Spezifikation auswählen.

Spezifikationen anpassen

Ein weiterer Vorteil beim Einsatz von Spezifikationen ist, dass Sie diese nachträglich anpassen können. Dazu öffnen Sie diese ebenfalls über den Assistenten, laden die gewünschte Spezifikation, nehmen die Änderung vor und speichern die Spezifikation wieder.

Datenbanken aufteilen

Wer seine Daten mit einer Datenbank verwaltet, greift nicht unbedingt nur allein auf diese Daten zu. Es kommt auch vor, dass die enthaltenen Daten von mehreren Mitarbeiteren bearbeitet werden sollen – und dies unter Umständen auch noch gleichzeitig. Dieser Artikel liefert die Grundlagen zum Einsatz von Access als Datenlieferant im Mehrbenutzerbetrieb.

Beispieldatenbank

Die Beispiele zu diesem Artikel finden Sie in der Datenbank **1106_Multiuser.mdb**, **1106_Multiuser_FE.mdb** und **1106_Multiuser_BE.mdb**.

Wie alles beginnt

Es kommt nicht selten vor, dass Daten in einer Excel-Datei gespeichert werden, die von mehreren Personen bearbeitet wird. Dies erfordert bestimmte Voraussetzungen: Zunächst müssen alle beteiligten Mitarbeiter Zugriff auf die Excel-Tabelle erhalten. Dies kann man so regeln, dass diese per E-Mail von Mitarbeiter zu Mitarbeiter geschickt wird oder man legt die Excel-Datei einfach auf einem Rechner ab, der von den betroffenen Mitarbeitern erreichbar ist.

Die erste Lösung löst schnell Chaos aus: Wenn die Datei auf mehreren Rechnern liegt und hin- und hergeschickt wird, gibt es schnell parallel bearbeitete Dateien. Das Ablegen auf einem Server ist sinnvoll, wenn man direkt auf diese Datei zugreift. Sobald diese jedoch auf den eigenen Rechner kopiert wird, können theoretisch auch gleich mehrere Mitarbeiter verschiedene Versionen bearbeiten – auch das ist problematisch.

Die einzige Lösung für diesen Fall ist ein Dokumentenmanagementsystem wie etwa SharePoint: Hier darf nur jeweils ein Mitarbeiter ein Dokument auschecken und dieses bearbeiten. Erst wenn dieser das Dokument wieder eingecheckt hat, darf der nächste Mitarbeiter darauf zugreifen.

Wenn die Excel-Datei dann noch Daten enthält, auf die viele Mitarbeiter oft schreibend zugreifen müssen, sinkt die Produktivität schnell, weil immer nur eine Person gleichzeitig auf das Dokument zugreifen kann.

In solchen Fällen ist es sinnvoll, wenn irgendwo ein Access-Programmierer in der Nähe ist und den Ernst der Lage erkennt: Dies ist ein Einsatzbereich für Access! Dabei sei vorausgesetzt, dass sich die Anzahl der Benutzer und ihrer Zugriffe in einem gewissen Rahmen hält.

Nun kommt es darauf an, dass der Access-Programmierer sich auch wirklich auskennt und nicht nur ein Gelegenheits-Benutzer ist. In diesem Fall geschieht nämlich oft Folgendes: Die Daten werden aus der Excel-Datei in eine Tabelle einer Access-Datenbank übertragen.

Die Access-Datenbank landet auf einem für alle beteiligten Mitarbeiter zugänglichen Rechner, zum Beispiel einem Server. Die Mitarbeiter starten die Access-Datenbank durch einen Doppelklick auf diese Datei und sehen ihren Traum von der mehrbenutzerfähigen Datenbank verwirklicht (Access muss dazu auf allen Arbeitsplatzrechnern installliert sein).

Dies funktioniert, denn so können tatsächlich mehrere Benutzer gleichzeitig auf die Daten der Tabelle zugreifen (okay, wenn der gleiche Datensatz von mehreren Personen bearbeitet wird, gibt es je nach Einstellung Probleme – mehr dazu später). Allerdings müssen nicht nur die zu bearbeitenden Daten, sondern auch die Formulare, die Berichte und mehr auf den jeweiligen Arbeitsplatz übertragen werden.

Datenbank aufteilen

Also gehen Sie auch noch den letzten Schritt: Sie teilen die Datenbank in ein Frontend und ein Backend auf. Das Frontend enthält die Abfragen, Formulare, Berichte und Module/Makros.

Das Backend bewahrt nur noch die Tabellen mit den Daten auf. Damit das Frontend auf die im Backend gespeicherten Tabellen zugreifen kann, erstellen Sie sogenannte Verknüpfungen auf diese Tabellen. Darüber können Sie genauso auf die verknüpften Daten zugreifen als wenn Sie direkt mit den Tabellen arbeiten.

Zur Tat

Genug der Theorie: Im Download zu dieser Ausgabe finden Sie die Datenbank **1106_Multiuser.mdb**, die wir in den folgenden Schritten in eine Frontend- und eine Backend-Datenbank aufteilen werden (das Ergebnis finden Sie ebenfalls im Download).

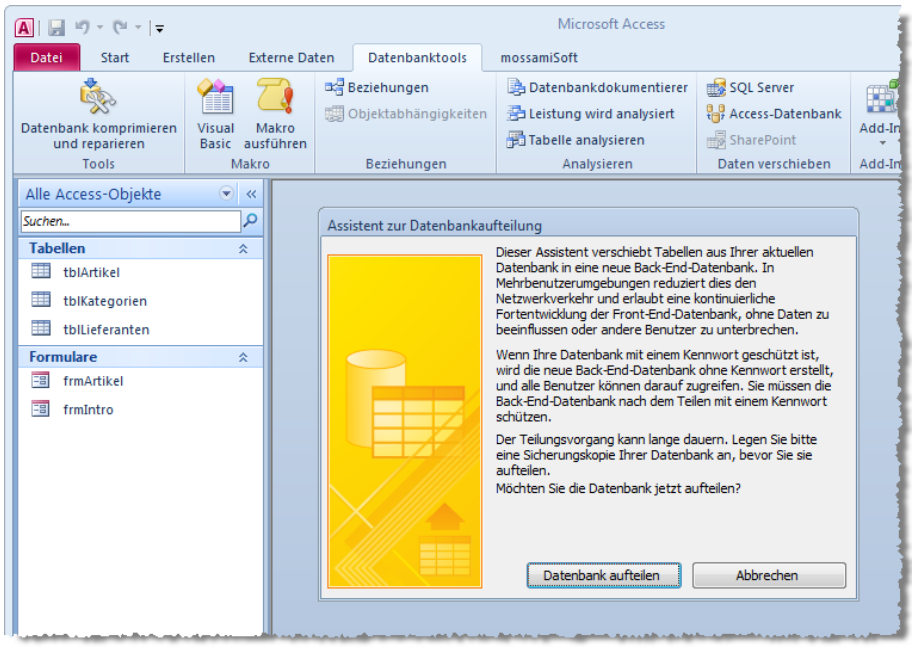


Bild 1: Start des Assistenten zur Datenbankaufteilung

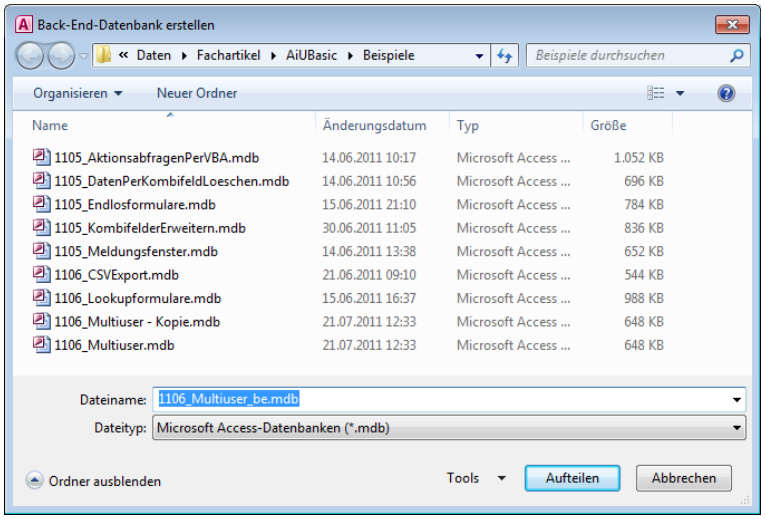


Bild 2: Festlegen des Namens der neuen Backend-Datenbank

Die Datenbank enthält die drei Tabellen **tblArtikel**, **tblKategorien** und **tblLieferanten** sowie ein Formular namens **frmArtikel** zur Anzeige dieser Daten.

Aufteilen per Assistent

Sie können die Aufteilung der Datenbank mit dem dafür vorgesehenen Assistenten durchführen oder aber selbst die notwendigen Schritte erledigen.

Den Assistenten finden Sie unter Access 2010 beispielsweise im Ribbon unter Datenbanktools|Daten verschieben|Access-Datenbank (siehe Bild 1). Unter Access 2003 und älter verbirgt sich diese Funktion hinter dem Menüeintrag **Extras|Datenbank-Dienstprogramme|Assistent zur Datenbankaufteilung**.

Der Assistent fragt im zweiten Schritt den Dateinamen der Datenbankdatei ab, welche die Tabellen aufnehmen soll, also den Part des Backends übernimmt (siehe Bild 2). In diesem Fall hängen Sie einfach die Zeichenfolge **_be** an den Dateinamen an. Die Datei soll zunächst im gleichen Verzeichnis gespeichert werden.

Das war es schon – je nach der Anzahl der Tabellen und der enthaltenen Daten meldet Access mehr oder weniger schnell Vollzug.

Aufteilen von Hand

Für spätere Arbeiten ist es unabdingbar, die einzelnen Schritte, die der Assistent im Hintergrund durchführt, zu kennen. Diese sehen kurz gefasst so aus:

- Importieren der Tabellen in einer neuen Backend-Datenbank
- Löschen der Tabellen aus der Frontend-Datenbank
- Erstellen von Verknüpfungen auf die Tabellen im Backend im Frontend

Quelle des manuell gesteuerten Aufteilens der Datenbank ist wiederum die Datei **1106_Multiuser.mdb**. Als erstes erstellen Sie eine neue Backend-Datenbank namens **1106_Multiuser_be.mdb**. Diese lassen Sie nach dem Erstellen geöffnet, um die Tabellen aus der Originaldatei zu importieren.

Den Import starten Sie je nach Access-Version wie folgt:

Access 2003 und älter: Betätigen Sie den Menüeintrag **Datei|Externe Daten|Importieren**. Wählen Sie im folgenden Dialog die Quelldatei aus, also **1106_Multiuser.mdb**. Im folgenden Dialog markieren Sie alle Tabellen, die in das Backend importiert werden

Datenbanken im Mehrbenutzerbetrieb

Wenn Sie eine Datenbank aufteilen und die Benutzer von mehreren Backends aus auf die Daten im Backend zugreifen, gibt es eine neue Herausforderung: Wie schnell sollen Änderungen an Daten auf den anderen Rechnern angezeigt werden? Was geschieht, wenn mehrere Benutzer gleichzeitig auf einen Datensatz zugreifen und diesen ändern? Diese und weitere Fragen beantwortet dieser Artikel.

Beispieldatenbanken

Die Beispieldatenbanken zu diesem Artikel heißen **1106_Multiuser_fe1.mdb**, **1106_Multiuser_fe2.mdb** und **1106_Multiuser_be.mdb**. Sie finden diese im Download zu diesem Artikel.

Die Datenbanken sind das Ergebnis der Aufteilung einer Datenbank in Frontend und Backend. Details hierzu finden Sie im Artikel **Datenbanken aufteilen**.

Mehrbenutzertest

Die Mehrbenutzerfähigkeiten einer Frontend-Backend-Konstellation können Sie leicht testen. Dazu brauchen Sie noch nicht einmal mehrere Rechner. Erstellen Sie einfach ein weiteres Frontend, indem Sie die Datei **1106_Multiuser.mdb** kopieren und unter einem anderen Namen wieder einfügen. Öffnen Sie nun beide Instanzen dieser Datenbank. Ändern Sie einen Datensatz in der ersten Instanz und betrachten Sie, was mit dem gleichen Datensatz in der zweiten Instanz geschieht. Es dauert eine Weile, aber dann wird die Änderung dort automatisch angezeigt.

Anzeigeaktualisierung

Je nach Anforderungen soll die Anzeige möglichst schnell aktualisiert werden. Dies können Sie über entsprechend Optionen beeinflussen:

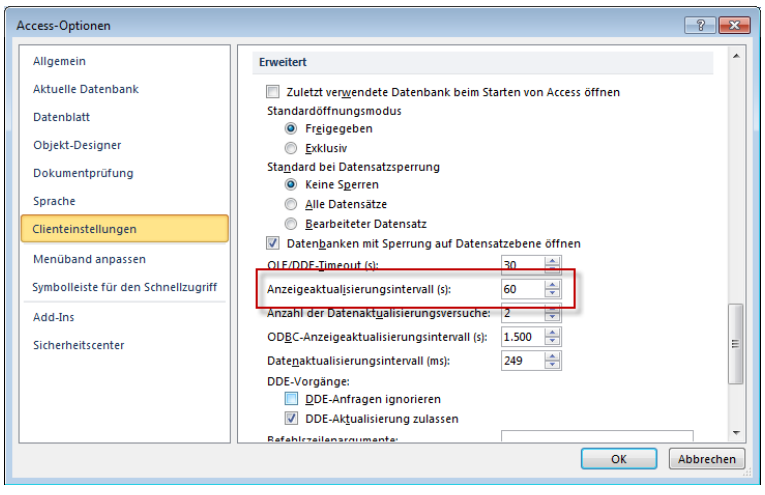


Bild 1: Einstellen des Aktualisierungsintervalls

Access 2003 und älter: Öffnen Sie den Optionen-Dialog über den Menüeintrag **Extras|Optionen** und stellen Sie die Eigenschaft **Weitere|Intervall für Anzeigeaktualisierung (s)** auf die Anzahl Sekunden ein, nach der die Daten erneut abgefragt werden sollen. Wenn Sie hier beispielsweise eine **1** eintragen, wird die Anzeige jede Sekunde aktualisiert.

Access 2007: Öffnen Sie den Optionen-Dialog über die Schaltfläche **Access-Optionen** des Office-Menüs. Die gesuchte Eigenschaft finden Sie unter **Erweitert|Erweitert|Anzeigeaktualisierungsintervall (s)**.

Access 2010: Hier öffnen Sie die Access-Optionen mit dem Ribbon-Eintrag **Datei|Optionen**. Die Option lautet **Clienteinstellungen|Erweitert|Anzeigeaktualisierungsintervall (s)**. Sie finden diese in Bild 1.

Diese und weitere Einstellungen, die Sie in den Access-Optionen vornehmen, wirken sich erst nach dem Schließen und dem erneuten Öffnen der aktuellen Datenbank aus.

Gleichzeitiges Bearbeiten

Sie können das Verhalten von Access beim gleichzeitigen Bearbeiten eines Datensatzes durch zwei Benutzer beeinflussen. Standardmäßig können mehrere Benutzer den gleichen Datensatz bearbeiten, ohne das etwas geschieht. Früher oder später könnte jedoch Folgendes geschehen:

- Benutzer 1 ändert ein Feld des Datensatzes, ohne diesen zu speichern.
- Benutzer 2 ändert ebenfalls ein Feld dieses Datensatzes. Auch er speichert die Änderungen nicht.
- Benutzer 1 speichert den Datensatz (beispielsweise durch den Wechseln zu einem anderen Datensatz).
- Benutzer 2 möchte seine Änderungen ebenfalls speichern. Er enthält allerdings die Meldung aus Bild 2.

Tabellenverknüpfungen pflegen

Wenn Sie eine Datenbank in Frontend und Backend aufgeteilt haben, greift das Frontend über eine Verknüpfung auf die Tabellen im Backend zu. Wo sich das Backend befindet, wird in einer Systemtabelle gespeichert. Diese wird beim Verschieben des Backends jedoch nicht automatisch aktualisiert, was zur Folge hat, dass das Frontend sein Backend und die enthaltenen verknüpften Tabellen nicht mehr findet. Dieser Artikel stellt Strategien vor, um die Verknüpfungen stets aktuell zu halten.

Beispieldatenbank

Die Beispieldatenbanken zu diesem Artikel heißen **1106_Verknuepfungen_FE.mdb** und **1106_Verknuepfungen_BE.mdb**. Sie finden diese im Download zu diesem Artikel.

Anwendungsfälle

Für das Aufteilen von Datenbanken gibt es verschiedene Anlässe. Die wichtigsten sind die bessere Möglichkeit der Wartung und der Mehrbenutzerzugriff. Eine aufgeteilte Datenbank lässt sich besser warten, weil man die Anwendungslogik, also Abfragen, Formulare, Berichte und VBA/Makros anpassen und anschließend einfach die neue Version des Frontends mit dem Backend verknüpfen kann.

Der Mehrbenutzerzugriff erlaubt es, das mehrere Benutzer mit jeweils einem auf ihrem Arbeitsplatz installiertem Frontend auf die im Backend auf dem Server gespeicherten Daten zugreifen können.

Natürlich lassen sich die verbesserten Wartungsmöglichkeiten auch in Mehrbenutzerumgebungen einsetzen.

Beim einfachen Aufteilen einer Datenbank zur besseren Wartung einer Einzelplatz-Anwendung liegen Frontend und Backend optimalerweise im gleichen Verzeichnis der lokalen Festplatte. Hier kann es vorkommen, dass Frontend- und Backend-Datenbank zusammen in ein anderes Verzeichnis oder sogar auf einen anderen Rechner übertragen werden.

Beim einer Mehrbenutzer-Datenbank liegen die Frontends jeweils auf den Arbeitsplätzen der Mitarbeiter, das Backend befindet sich auf einem Server-Rechner. Das entsprechende Netzwerkverzeichnis ist normalerweise auf dem jeweiligen Arbeitsplatzrechner gemappt. Dies geschieht entweder individuell, das heißt, das jeder Benutzer das Verzeichnis unter einem selbst gewählten Buchstaben mappt, oder jeder Benutzer verwendet standardmäßig den gleichen Pfad für den Zugriff auf den Server.

Verknüpfungen erneuern

Wann immer Sie bei einer Kombination aus Frontend- und Backend-Datenbank die Backend-Datenbank verschieben, müssen Sie die Verknüpfungen anpassen.

Daher ist Variante mit der individuellen Festlegung des Laufwerkbuchstabens die ungünstigste: Sie müssen dann auch für jedes Frontend die Verknüpfungen individuell erneuern.

Wenn alle Benutzer den gleichen Pfad verwenden, beispielsweise **e:\Datenbanken\Backend.mdb**, brauchen Sie die Verknüpfungen nur einmal auf Basis dieses Pfades zu erstellen und können die Frontend-Datenbank dann von jedem Arbeitsplatz aus problemlos einsetzen. Selbst bei einem Verschieben des Backends muss nur das Mapping aktualisiert werden.

Das Verschieben von Frontend- und Backend einer Einzelplatzlösung ist ein Zwischending: Auch hier müssen die Verknüpfungen erneuert werden, aber da beide Dateien immer im gleichen Verzeichnis liegen, lässt sich dies einfach realisieren.

Grundlagen der Verknüpfung

Die Daten einer Verknüpfung werden in der Systemtabelle **MSysObjects** gespeichert. Diese Tabelle ist standardmäßig nicht sichtbar. Um sie sichtbar zu machen, führen Sie in Abhängigkeit von der Access-Version Folgendes durch:

Access 2003 und älter: Klicken Sie auf den Menüeintrag **Extras|Optionen**. Dort finden Sie im Bereich Ansicht unter Anzeigen den Eintrag **Systemobjekte**. Aktivieren Sie diese Option und schließen Sie den Dialog wieder.

Access 2007 und jünger: Die richtige Option ist hier ein wenig schwieriger zu finden. Klicken Sie mit der rechten Maustaste auf die Titelleiste des Navigationsbereichs und wählen Sie dort den Eintrag **Naviga-**

tionsoptionen aus (siehe Bild 1). Im nun erscheinenden Dialog **Navigationsoptionen** finden Sie untern links den Eintrag **Systemobjekte** anzeigen, den Sie aktivieren (siehe Bild 2). Der Navigationsbereich beziehungsweise das Datenbankfenster zeigt nun einige Einträge in grauer Schrift an, darunter auch **MSysObjects**.

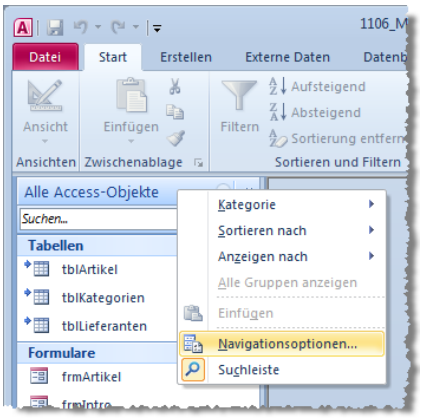


Bild 1: Aufrufen der Navigationsoptionen unter Access 2007 und jünger

Wenn Sie diese Tabelle öffnen und die enthaltenen Spalten ein wenig umstellen, erhalten Sie eine Ansicht wie in Bild 3. Wichtig sind die Spalten **Name**, **Type** und **Database**. **Name** zeigt den Namen des jeweiligen Objekts an, **Type** den Typ (**6** steht für verknüpfte Tabellen) und **Database** liefert den Pfad zu der Datenbank, welche die verknüpften Tabellen enthält.

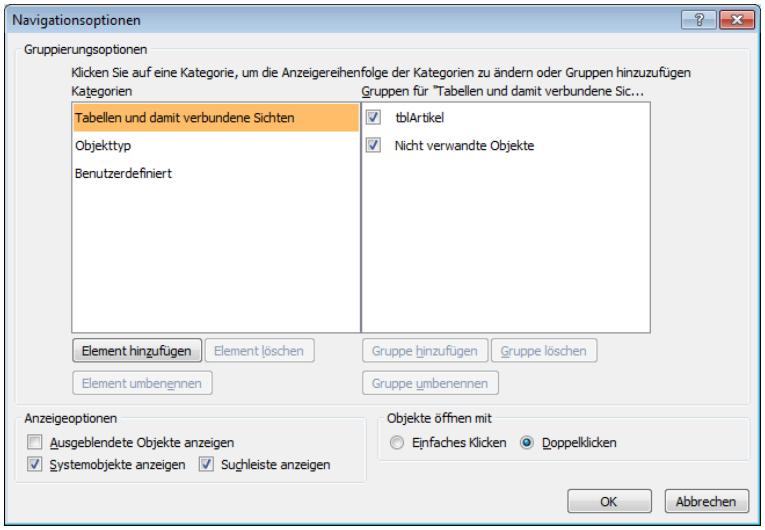


Bild 2: Aktivieren der Anzeige der Systemobjekte unter Access 2007 und jünger

Nun mögen Sie denken, dass Sie beim Verschieben der Backend-Datenbank einfach den Pfad in dieser Tabelle ändern können, damit die Datenbank wieder auf die verknüpften Tabellen zugreifen kann. Aber weit gefehlt: Es handelt sich um eine Systemtabelle, deren Daten nicht geändert werden können. Genau genommen lassen sich noch nicht einmal die Änderungen an der Reihenfolge der Spalten speichern.

Verknüpfung mit Backend im gleichen Verzeichnis

Befindet sich das Backend immer im gleichen Verzeichnis wie das Frontend, weil die Datenbank nur aus Gründen der besseren Wartbarkeit aufgeteilt wurde, können Sie dem Frontend eine kleine Prozedur hinzufügen, die bei jedem Start die Verknüpfung erneuert. Dies beugt dem Umbenennen der übergeordneten Verzeichnisse oder dem Verschieben von Frontend und Backend gemeinsam in ein anderes Verzeichnis vor. Die dazu verwendete Prozedur nutzt einige VBA-Befehle, die wir in **Access [basics]** noch nicht vorgestellt haben, aber im Rahmen dieser kleinen Lösung greifen wir einfach einmal den folgenden Ausgaben vor. Zum besseren Verständnis haben wir den grundsätzlichen Ablauf in Bild 4 skizziert. Die Prozedur legen Sie in einem Standardmodul an. Sie sieht so aus:

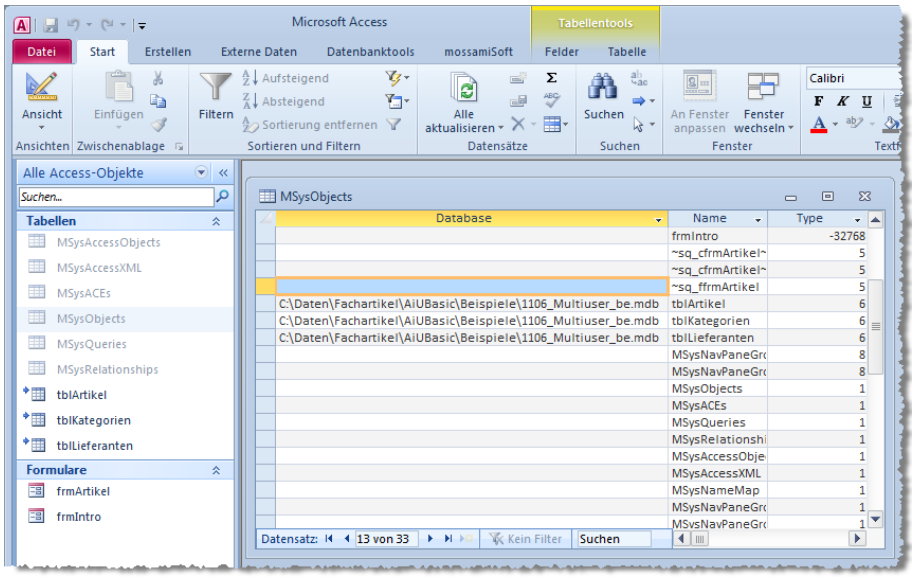


Bild 3: Die Tabelle **MSysObjects** zeigt den Pfad der verknüpften Tabellen an.

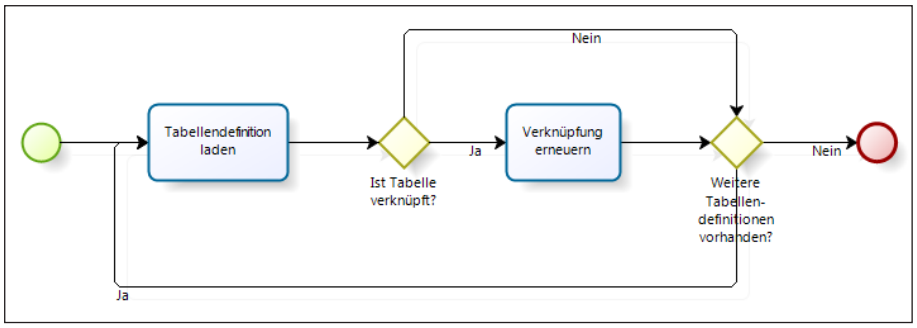


Bild 4: Ablauf beim Wiederverknüpfen von Tabellen eines Backends im gleichen Verzeichnis wie die Frontend-Datenbank

```
Public Sub VerknuepfungAktualisieren()
    Dim strDB As String
    Dim strPath  As String
    Dim strConnect As String
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    Set db = CurrentDb
    strPath = CurrentProject.Path
    For Each tdf In db.TableDefs
        strConnect = tdf.Connect
        If Len(strConnect) > 0 Then
            strDB = Mid(strConnect, 7
                        InStrRev(strConnect, "\"))
            tdf.Connect = ;7
                    ";database=" & strPath & strDB
            tdf.RefreshLink
        End If
    Next tdf
End Sub
```

Die Prozedur deklariert zunächst einige Variablen, deren Bedeutung wir im Folgenden erläutern. Als erstes wird die Objektvariable **db** über die Funktion **CurrentDB** mit einem Verweis auf die aktuelle Datenbank gefüllt. Das bezieht sich in diesem Fall auf die Frontend-Datenbank. Als zweites ermittelt die **Path**-Eigenschaft des **CurrentProject**-Objekts den Pfad zur aktuellen Datenbank, also das Verzeichnis, in dem sich nicht nur das Frontend, sondern auch das Backend befindet. Liegt die Datenbank in einem Verzeichnis wie **c:\Beispiele\Beispieldatenbank.mdb**, liefert diese Funktion **c:\Beispiele** zurück.

Dann beginnt eine **For Each**-Schleife, die in der folgenden Ausgabe von **Access [basics]** erläutert wird, mit dem Durchlaufen der Objekte der Auflistung **TableDefs** des aktuellen Datenbank-Objekts. Die **TableDefs**-Auflistung enthält wiederum Objekte des Typs **TableDef**. Ein **TableDef**-Objekt liefert Informationen über die Definition einer Tabelle, also beispielsweise zu den enthaltenen Feldern und deren Datentypen.

Beim Durchlaufen der Auflistung **TableDefs** wird nun bei jedem Durchlauf ein einziges **TableDef**-Objekt mit der Objektvariablen **tdf** referenziert. Innerhalb der Schleife greift die Prozedur dann darauf zu, um den Wert der Eigenschaft **Connect** in die String-Variable **strConnect** zu schreiben. Diese Eigenschaft liefert für eine verknüpfte Access-Datenbank einen Ausdruck, der aus der Zeichenkette **;database=** und dem Namen der Backend-Datenbank besteht, also beispielsweise

```
;database=c:\Beispiele\Beispieldatenbank.mdb
```

Danach folgen einige Zeichenketten-Funktionen. Die erste heißt **Len** und ermittelt die Länge des Inhalts der Variablen **strConnect**. Ist die Länge größer als **0**, enthält die **Connect**-Eigenschaft einen Wert und es handelt sich beim aktuellen **TableDef**-Objekt um eine verknüpfte Datenbank. Diese Unterscheidung ist wichtig, weil die Datenbank ja auch lokale Tabellen enthält – zum Beispiel die Systemtabellen oder auch benutzerdefinierte Tabellen.

Nachdem der Status des **TableDef**-Objekts geklärt ist, kann sich die Prozedur an das Aktualisieren der Verknüpfung heranwagen Die erste der drei Zeilen innerhalb der **If...Then**-Bedingung ermittelt den reinen Dateinamen des Wertes der **Connect**-Eigenschaft, bei **;database=c:\Beispiele\Beispieldatenbank.mdb** also beispielsweise **\Beispieldatenbank.mdb**. Dabei ermittelt die Funktion **InStrRev** zunächst die Position des letzten Backslash-Zeichens der Zeichenkette (****). Damit wird dann der zweite Parameter der **Mid**-Zeichenkette gefüttert, die alle Zeichen der im ersten Parameter angegebenen Zeichenkette ab der im zweiten Parameter angegebenen Position zurückliefert. Dabei wird das Backslash-Zeichen mit eingeschlossen.

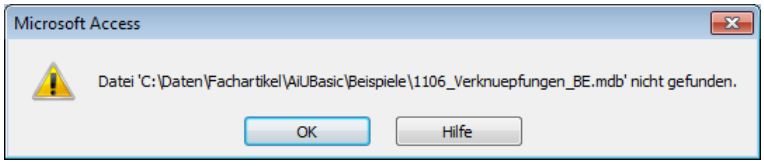


Bild 5: Diese Meldung erscheint, wenn Access die für eine Verknüpfung angegebene Datenbank nicht findet.

Aktion beim Start: Das Autoexec-Makro

Wenn Access direkt nach dem Öffnen einer Access-Anwendung automatisch eine VBA-Funktion aufrufen soll, finden Sie hier die richtige Lösung: Das Autoexec-Makro! Dieses Makro liefert die Möglichkeit, beliebige Funktionen aufzurufen, Formular zu öffnen und mehr.

Beispieldatenbank

Die Beispiele zu diesem Artikel finden Sie in der Datenbank **1107_Autoexec.mdb**.

Makros

Makros sind grundsätzlich nicht so flexibel wie VBA-Code. Es steht nur ein begrenzter Befehlsumfang bereit und auch die Wartbarkeit ist nicht besonders gut. Allerdings haben Makros dennoch eine Daseinsberechtigung, vor allem mit der Einführung der Webdatenbanken mit Access 2010. Webanwendungen lassen sich ausschließlich mit Makros programmieren.

Aber auch für ältere Access-Versionen lohnt sich ein Blick auf Makros, vor allem auf zwei spezielle Exemplare. Diese zeichnen sich zunächst durch ihre Namen aus: **Autoexec** und **Autokeys**. Ein Makro namens **Autoexec** wird immer automatisch beim Öffnen einer Datenbankdatei ausgelöst und mit dem **Autokeys**-Makro können Sie Tastenkombinationen beliebige Makrofunktionen zuweisen.

In diesem Artikel schauen wir uns allerdings zunächst das **Autoexec**-Makro an und konzentrieren uns darauf, eine vorhandene VBA-Funktion zu starten.

Die dabei verwendete Funktion muss sich in einem Standardmodul befinden und mit dem Schlüsselwort **Public** als öffentliche Funktion deklariert sein. Zu Beispielzwecken verwenden Sie die folgende Funktion:

```
Public Function AutoexecText()
    MsgBox "Per Autoexec aufgerufen"
End Function
```

Autoexec-Makro anlegen

Makros legen Sie je nach Version wie folgt an:

Access 2003 und älter: Wechseln Sie im Datenbankfenster zur Registerseite **Makros** und klicken Sie auf die Schaltfläche **Neu**.

Access 2007: Klicken Sie im Ribbon auf den Eintrag **Erstellen|Andere|Makro**.

Access 2010: Wählen Sie im Ribbon den Befehl **Erstellen|Makros und Code|Makro** aus.

Es erscheint der Makro-Editjor, der unter Access 2010 deutlich anders aussieht als unter den älteren Access-Versionen. Daher betrachten wir die Erstellung des **Autoexec**-Makros für diese Versionen getrennt.

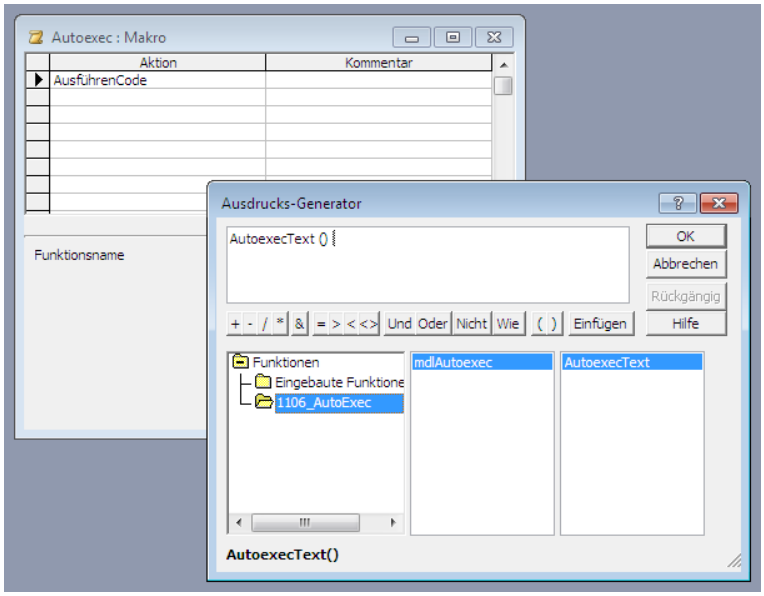


Bild 1: Anlegen des **Autoexec**-Makros unter Access 2007 und älter

Autoexec-Makro unter Access 2007 und älter anlegen

Wenn Sie den Makro-Editor geöffnet haben, erscheint dieser zunächst mit einer leeren Liste zum Eintragen der gewünschten Aktionen. Klicken Sie in die erste Zeile und wählen Sie aus dem Kombinationsfeld den Eintrag **AusführenCode** aus. Dieses Makro erwartet einen Parameter, den Sie unten unter Funktionsname eintragen können. Am einfachsten gelingt dies, wenn Sie auf die **Aufbauen**-Schaltfläche (...) klicken und den Ausdrucks-Generator verwenden. Hier klicken Sie zunächst doppelt auf den Eintrag **Funktionen** und dann auf den Namen des VBA-Projekts

Dialog zur Auswahl von Dateien anzeigen

Auch wenn Access eigentlich alles an Bord hat, was man zum Bearbeiten von Daten benötigt, so muss man sich doch das eine oder andere hinzuprogrammieren. Dazu gehört eine Funktion, mit der Sie einen Dialog zum Auswählen von Dateien anzeigen können. Dieser Artikel zeigt die einfachste versionsübergreifende Lösung dieser Aufgabe.

Beispieldatenbank

Die Beispieldatenbank zu diesem Artikel heißt **1107_DateiOeffnenDialog.mdb**. Sie finden diese im Download zu diesem Artikel.

Wozu Dateien auswählen?

Wenn Sie Anwendungen mit Access programmieren, werden Sie früher oder später eine Funktion zum Auswählen von Dateien anbieten müssen. Anlässe dazu gibt es genug:

- Der Benutzer soll eine Datei auswählen, deren Daten importiert werden sollen (Excel-Tabellen, Textdateien, XML-Dokumente).
- Sie möchten Bilder mit der Datenbank verwalten, ohne die Bilder in der Datenbank zu speichern. Dazu müssen Sie die Dateinamen samt Pfad in einer Tabelle speichern und diese dazu vorher auswählen.
- Sie möchten dem Benutzer die Möglichkeit bieten, nach dem Verschieben des Backends einer Datenbank selbst nach dieser Datei zu suchen (siehe **Aktion beim Start: Das Autoexec-Makro**).

Verschiedene Möglichkeiten

Es gibt die verschiedene Varianten, um einen Datei öffnen-Dialog anzuzeigen. Früher verwendete man dazu das Common Dialogs-Steuerelement, neuere Access-Versionen bieten die **FileDialog**-Klasse für diesen Zweck an. Es gibt jedoch auch eine Variante, die in allen Access-Versionen arbeitet. Diese steckt in einer verborgenen und nicht dokumentierten Klasse, ist aber sehr leicht aufzurufen.

Datei öffnen-Dialog der Wizhook-Klasse

Um diese Klasse namens **Wizhook** zu aktivieren, platzieren Sie in einer neuen Prozedur die folgende Anweisung:

```
WizHook.Key = 51488399
```

Danach fügen Sie eine Anweisung ein, die den Befehl **GetFileName** dieser Klasse aufruft. Wie Bild 1 zeigt, liefert IntelliSense schnell eine Liste aller verfügbaren Parameter.

Die Parameter haben folgende Bedeutung:

- **hwndOwnder**: Handle der Anwendung, von der aus der Dialog aufgerufen wird, hier Application.hWndAccessApp
- **AppName**: Name der Anwendung, hier Microsoft Access
- **DlgTitle**: Titel des Dateidialogs
- **OpenTitle**: Beschriftung der Schaltfläche zum Übernehmen der Auswahl
- **File**: Liefert die ausgewählte(n) Datei(en) als Zeichenkette
- **InitialDir**: Verzeichnis, das beim Öffnen angezeigt wird
- **Filter**: Liste der Dateifilter
- **FilterIndex**: Nummer des Filters, der beim Öffnen angewendet wird
- **View**: Dateiansicht

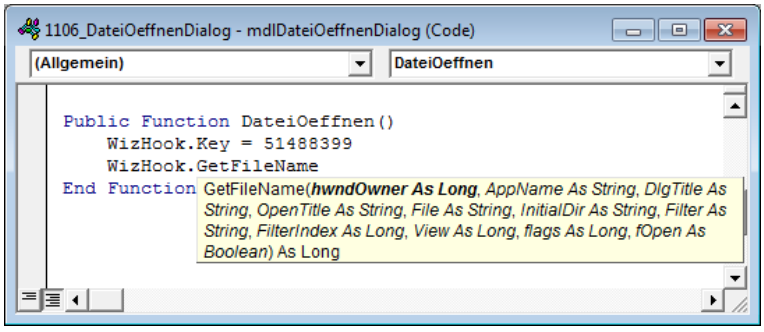


Bild 1: Parameter der Funktion **GetFileName**

Fehler unter VBA

Wer mit VBA arbeitet und damit seine Access-Anwendung automatisieren möchte, macht Fehler – gerade als Einsteiger. Da ist es gut zu wissen, welche Fehler passieren können und wie diese einzuordnen sind. Dieser Artikel stellt die verschiedenen Fehlerarten vor, denen Sie im Laufe Ihres Entwicklerlebens begegnen werden.

Fehlerarten unter VBA

Unter VBA gibt es verschiedene Fehler, die sich zu unterschiedlichen Anlässen bemerkbar machen:

- Syntaxfehler
- Kompilierzeitfehler
- Laufzeitfehler

Die folgenden Abschnitte stellen die Fehlerarten vor.

Syntaxfehler

Syntaxfehler sind die am einfachsten zu behandelnden Fehler: Sie entstehen gleich bei der Eingabe und werden entweder direkt beim Verlassen der Codezeile oder spätestens beim Ausführen gemeldet.

Ein Beispiel ist eine fehlende schließende Klammer wie in Bild 1. Der Fehler fällt dem VBA-Interpreter gleich auf, wenn Sie die Zeile verlassen. In diesem Falle wird das **Then**-Schlüsselwort markiert, weil es das erste ungültige ist, dem eigentlich die schließende Klammer hätte vorangehen müssen.

Syntaxfehler beheben

Nach dem Bestätigen der Fehlermeldungen mit **OK** hinterlegt der VBA-Editor die Kopfzeile der Routine übrigens gelb und will Ihnen damit sagen: Korrigieren Sie den Fehler und betätigen Sie die Taste **F5**, damit wir einen neuen Versuch unternehmen können.

Das können Sie tun, wenn es sich nur um eine kleine Ergänzung wie etwa das Hinzufügen einer schließenden Klammer oder einer **End If**-Zeile handelt – wenn Sie die Routine etwa als Ereignisprozedur über das Anklicken einer Schaltfläche eines Formulars gestartet haben, brauchen Sie nicht erneut auf die Schaltfläche zu klicken, sondern können direkt mit **F5** fortfahren.

Anzeige von Syntaxfehlern ausschalten

Irgendwann kennen Sie die typischen Syntaxfehler und möchten vielleicht auf die Anzeiger der Fehlermeldungen verzichten. Für diesen Fall hält der Optionen-Dialog des VBA-Editors eine passende Einstellung bereit.

Öffnen Sie den Optionen-Dialog mit dem Menü-Eintrag **Extras|Optionen** und deaktivieren Sie die Einstellung **Automatische Syntaxüberprüfung** (siehe Bild 2).

Wenn Sie nun einen Syntaxfehler produzieren, der bereits durch die Prüfung einer einzigen Zeile ermittelt werden kann, zeigt der VBA-Editor keine Fehlermeldung an, sondern markiert einfach nur die fehlerhafte Zeile rot.

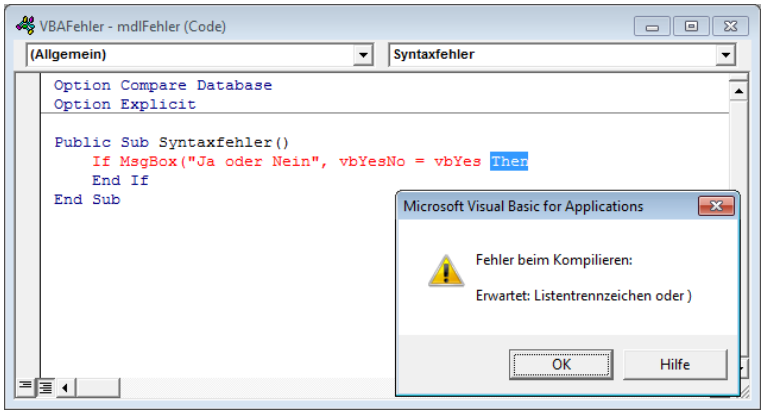


Bild 1: Syntaxfehler durch eine fehlende Klammer

Dabei kann es durchaus vorkommen, dass mehrere fehlerhafte Zeilen gleichzeitig markiert werden (siehe Bild 3). Für Einsteiger mag das nicht relevant sein, aber später werden Sie es möglicherweise zu schätzen wissen, wenn Sie etwa beim Importieren und Umwandeln von Code in anderen Programmiersprachen nicht ständig Fehlermeldungen wegklicken müssen, um den Code in Ruhe VBA-konform zu gestalten.

Und wenn wir gerade schon bei den Optionen zur Anzeige von Syntaxfehlern sind:

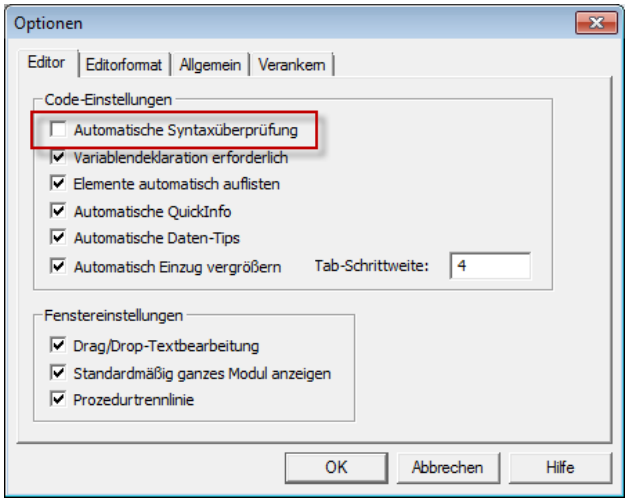


Bild 2: Deaktivieren der Anzeige von Syntaxfehlern

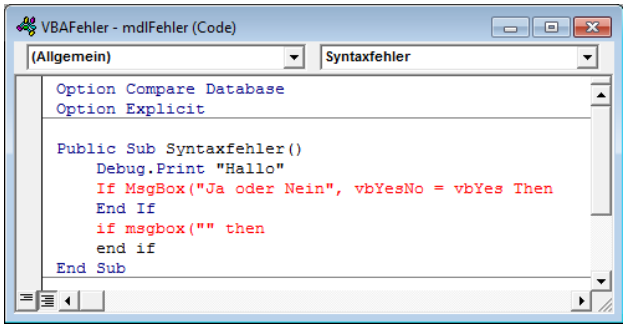


Bild 3: Ohne automatische Syntaxüberprüfung können auch mehrere fehlerhafte Zeilen farbig hinterlegt werden.

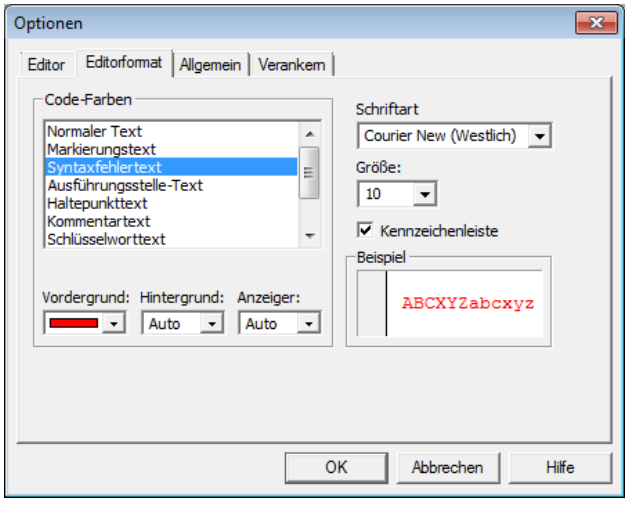


Bild 4: Einstellen der Textfarbe etwa für Zeilen mit Syntaxfehlern

Auf der zweiten Seite der Optionen können Sie unter Code-Farben einstellen, in welcher Farbe verschieden markierte Zeilen und Texte erscheinen sollen.

Wenn Sie also beispielsweise in der Schule zuviele rote Kommentare in Ihren Klassenarbeiten vorgefunden haben, möchten Sie fehlerhafte Zeilen vielleicht eher grün gefärbt vorfinden (siehe Bild 4).

Kompilierzeitfehler

Andere Syntaxfehler kann VBA nicht direkt beim Abschließen einer Zeile, sondern erst beim Ausführen der betroffenen Routine erkennen. Dabei handelt es sich um Fehler beim Aufbau von mehrzeiligen Konstrukten wie beispielsweise **If...Then**, **Do...Loop** oder **For...Next**.

Ein Beispiel zeigt Bild 5. VBA prüft die Syntax der Routine und stellt fest, dass sich vor dem Ende der Routine zwar ein **If...Then**, aber kein **End If** befindet. Das ist natürlich nicht erlaubt, also erscheint auch hier eine Fehlermeldung – allerdings erst, nachdem die Prozedur aufgerufen wurde.

Die Syntaxprüfung erfolgt vor dem Kompilieren der Routine, um diese in ausführbaren Code zu übersetzen. Treten dort Fehler auf, kann die Kompilierung nicht fortgesetzt werden und es erscheint eine entsprechende Meldung.

Die Kompilierung können Sie allerdings auch durchführen, ohne die Routine auszuführen. Den dazu notwendigen Befehl finden Sie im Menü unter **Debuggen|Kompilieren von <Projektname>**.

Dieser Befehl kompiliert immer das komplette VBA-Projekt, also alle in den Standard- und Klassenmodulen enthaltenen Deklarationen und Routinen.

Der VBA-Editor meldet beim Kompilieren immer nur einen Fehler gleichzeitig. Um weitere Fehler zu entdecken, müssen Sie diesen Fehler beheben und dann erneut kompilieren. Erst wenn kein Fehler mehr vorliegt, können Sie das komplette Projekt kompilieren.

Wenn Sie eine Routine aufrufen, wird diese auf jeden Fall kompiliert, andere unabhängige Routinen jedoch nicht. Wenn die Routine jedoch weitere Prozeduren oder Funktionen aufruft, werden zuerst alle Routinen kompiliert.

Beispiele für Kompilierzeitfehler

Beim Kompilieren können beispielsweise die folgenden Fehler auftreten:

Fehlerbehandlung unter VBA

Wer mit VBA arbeitet, wird früher oder später auf das Thema Fehlerbehandlung stoßen. Je komplexer Code wird und je mehr sein Ablauf von externen Faktoren beeinflusst wird, desto mehr sind sie auf eine Fehlerbehandlung angewiesen. Die Fehlerbehandlung sorgt dafür, dass der Benutzer nicht mit schwer interpretierbaren oder noch besser gar nicht mit Fehlermeldungen konfrontiert wird. Im Hintergrund hilft eine Fehlerbehandlung aber noch in vielen anderen Fällen, von denen der Benutzer gar nichts mitbekommt.

Wozu eine Fehlerbehandlung?

Eine perfekte Anwendung arbeitet ohne Fehler. Perfekte Anwendungen sind jedoch selten, also treten auch Fehler auf. Zumindest der Benutzer einer Anwendung sollte jedoch den Eindruck erhalten, dass er mit einer perfekten Anwendung arbeitet – oder zumindest mit einer, die Fehler, wenn sie denn auftreten, entsprechend professionell verarbeitet.

Sprich: Besser, eine Anwendung wird im schlimmsten Fell beim Auftreten eine Fehlers kontrolliert beendet, als wenn sie unkontrolliert abstürzt – und am Ende sogar noch Daten verloren gehen.

Wenn unter VBA ein Fehler auftritt, geht das in der Regel beispielsweise mit dem Leeren von Objektvariablen einher. Wenn Sie also gerade eine globale Objektvariable mit einem Verweis auf ein Objekt wie ein Formular gefüllt haben und dann ein Fehler auftritt, wird die Objektvariable geleert. Mit der Folge, dass ein Zugriff auf das mit dieser Variable referenzierte Objekt ebenfalls zu einem Fehler führt.

Wenn Sie in einem solchen Fall eine Fehlerbehandlung verwenden, verhindern Sie zumindest den Folgefehler. Und Sie präsentieren dem Benutzer damit gegebenenfalls eine aussagekräftige Fehlermeldung statt einer von Access/VBA generierten Fehlermeldung, mit welcher der Benutzer oft nichts anfangen kann.

Dieser Artikel zeigt, wie Sie die unter VBA auftretenden Fehler so behandeln, dass Ihre Anwendung stabil weiterläuft und der Benutzer gegebenenfalls erfährt, was warum schief gelaufen ist.

Fehler ohne Fehlerbehandlung

Tritt ein Fehler ohne Fehlerbehandlung auf, löst dieser eine entsprechende Meldung aus und zeigt die fehlerhafte Stelle im Code an. Dies gilt jedoch nur, wenn Sie die Vollversion von Access verwenden und die Datei im **.mdb**-, dem **.accdb**- oder einem anderen nicht geschützten Format vorliegt. Wenn die Datenbank als **.mde**- oder **.accde**-Datenbank kompiliert wurde, also ihr Code nicht mehr im Entwurf eingesehen werden kann, erhalten Sie nur die Fehlermeldung mit Nummer und Text, können aber nicht mehr den fehlerhaften Code einsehen. Bei Verwendung der Runtime-Version von Access (mehr dazu in einem späteren Artikel) erhalten Sie nur noch eine allgemeine Fehlermeldung. Deshalb ist gerade in diesem Fall eine benutzerdefinierte Fehlerbehandlung sehr wichtig!

Die Meldung sieht beispielsweise wie in Bild 1 aus, weitere Beispiele und Grundlagen zu den verschiedenen Fehlerarten erhalten Sie im Artikel **Fehler unter VBA**.

Fehlerbehandlung von Laufzeitfehler

Im Gegensatz zu Syntax- oder Kompilierfehlern können Sie auf Fehler zur Laufzeit mit entsprechenden Anweisungen reagieren. Die einfachste Möglichkeit ist es, den Fehler einfach zu ignorieren. Dazu tragen Sie die Anweisung **On Error Resume Next** innerhalb der Routine, aber vor der fehlerhaften Zeile ein. Ein Beispiel sieht so aus:

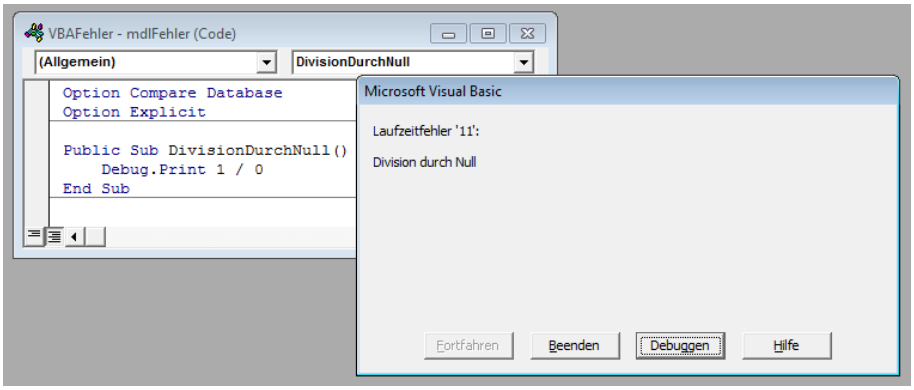


Bild 1: Fehlermeldung eines unbehandelten Fehlers

```
Public Sub DivisionDurchNullFehlerbehandlung()
    On Error Resume Next
    Debug.Print 1 / 0
End Sub
```

Das Resultat dieser Vorgehensweise ist, dass der in der Zeile **Debug.Print 1/0** auftretende Fehler einfach ignoriert wird. Dummerweise tut die Zeile aber auch nichts mehr.

Im folgenden Beispiel tritt in der Zeile **i = 32768** ein Überlauf auf (Fehler **6**). Die fehlerhafte Zeile wird nicht ausgeführt, **i** wird also nicht mit dem angegebenen Wert gefüllt. **Integer**-Variablen haben den Standardwert **0**, also gibt das Meldungsfenster den Wert **0** aus:

```
Public Sub Ueberlauf()
    Dim i As Integer
    On Error Resume Next
    i = 32768
    MsgBox i
End Sub
```

Das ist ein Problem, denn durch den übergangenen Fehler entsteht ein logischer Fehler: Obwohl **i** eigentlich den Wert **32768** enthalten sollte, liefert das Meldungsfenster den Wert **0**. Solche logischen Fehler entstehen oft, wenn man – teilweise aus Bequemlichkeit – einen Fehler mit **On Error Resume Next** unterbindet.

On Error Resume Next sollten Sie also nur dort einsetzen, wo es nicht anders geht – oder dort, wo Sie vielleicht sogar einen Fehler provozieren möchten (hierzu später mehr).

Ein Beispiel für den sinnvollen Einsatz ist etwa das Anlegen eines Verzeichnisses im aktuellen Datenbankverzeichnis – beispielsweise, um dort Daten etwa im Excel-Format zu exportieren. Die Anweisung **MkDir** legt ein mit dem einzigen Parameter angegebenes Verzeichnis an. Wenn Sie diesen Parameter wie folgt verwenden, entsteht dadurch etwa das Verzeichnis **Export** in dem Ordner, in dem sich auch die Datenbankdatei befindet (**CurrentProject.Path** liefert den Datenbankpfad):

```
MkDir CurrentProject.Path & "\Export"
```

Das Dumme ist nur: Wenn Sie die Prozedur, in der sich diese Anweisung befindet, ein zweites Mal ausführen, ist das Verzeichnis bereits vorhanden; der erneute Versuch, dieses anzulegen, führt zu einem Fehler (siehe Bild 2).

Fehlerbehandlung deaktivieren

Nun gibt es zwei Möglichkeiten: Entweder Sie prüfen zuvor, ob das Verzeichnis schon vorhanden ist, oder Sie ignorieren einfach den Fehler beim erneuten Anlegen, indem Sie zuvor die Anweisung **On Error Resume Next** ausführen. Letzteres ist kein Problem, wenn Sie sehr vorsichtig damit umgehen. Das zieht in diesem Fall vor allem nach sich, dass Sie die Fehlerbehandlung unmittelbar im Anschluss an die betroffene Anweisung wieder so einstellen, dass Fehlermeldungen angezeigt werden! Dies erledigen Sie mit der Anweisung **On Error Goto 0**. Wenn Sie die folgende Prozedur ausführen, löst das wiederholte Anlegen des Verzeichnisses keine Fehlermeldung aus, wohl aber die folgende Division durch **0**:

```
Public Sub VerzeichnisAnlegen()
    On Error Resume Next
    MkDir CurrentProject.Path & "\Export"
    On Error GoTo 0
    Debug.Print 1 / 0
End Sub
```

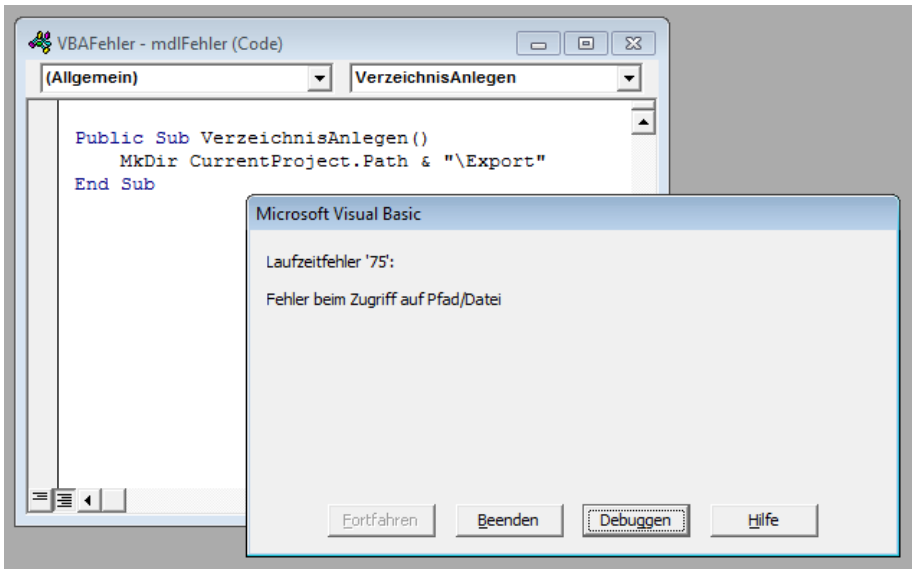


Bild 2: Ein bereits vorhandenes Verzeichnis lässt sich nicht nochmals anlegen.

Was also erledigt **On Error Goto 0** genau? Es deaktiviert eine eventuell zuvor aktivierte benutzerdefinierte Fehlerbehandlung, die im einfachsten Fall nur aus dem Ignorieren von Fehlern durch die Anweisung **On Error Resume Next** besteht.

Es setzt aber auch das Fehlerobjekt von VBA zurück – und dieses schauen wir uns im nächsten Abschnitt genauer an.

Das Fehlerobjekt Err

Auch wenn wir in **Access [basics]** noch nicht explizit über Objekte gesprochen haben: Bei der Fehlerbehandlung kommen wir nicht umhin, diesem Thema ein wenig vorzugreifen. Entscheidend für die Fehlerbehandlung ist nämlich, dass Sie zunächst einmal den Fehler erkennen können.

Dazu schreibt VBA (fast) alle notwendigen Informationen in das Fehlerobjekt **Err**. Wenn Sie beispielsweise nach dem Auftreten eines Fehlers seine Fehlernummer ermitteln möchten, erledigen Sie das wie im folgenden Codeschnipsel:

```
On Error Resume Next
Debug.Print 1 / 0
Debug.Print Err.Number
```

Die erste Anweisung schaltet die VBA-interne Fehlerbehandlung ab, die zweite löst den Fehler aus und die dritte gibt die Nummer des durch die zweite Anweisung ausgelösten Fehlers aus.

In diesem Fall gibt die dritte Anweisung den Wert **11** im Direktfenster aus. Vielleicht möchten Sie auch noch die Fehlerbeschreibung ausgeben? Kein Problem – fügen Sie einfach noch die folgende Zeile hinzu:

```
Debug.Print Err.Description
```

Im Direktfenster erscheint nun zusätzlich der Text **Division durch Null**.

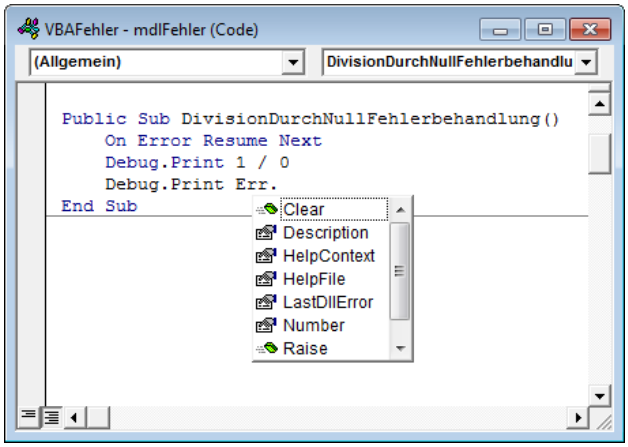


Bild 3: Der VBA-Editor bietet alle Eigenschaften und Methoden des Err-Objekts per IntelliSense zur Auswahl an.

Bei der Verwendung von eingebauten VBA-Objekten unterstützt der VBA-Editor, indem er die Eigenschaften und Methoden dieser Objekte direkt nach der Eingabe des Objektnamens und eines Punktes zur Auswahl anbietet (siehe Bild 3).

Zur Begriffsklärung: Eine Methode eines Objekts ist nichts weiter als ein Befehl, den Sie im Kontext dieses Objekts aufrufen können.

Die beiden Eigenschaften **Number** und **Description** sind im Rahmen einer einfachen Fehlerbehandlung auch schon die einzigen beiden interessanten Eigenschaften, wobei prinzipiell die Eigenschaft **Number** und die darauf gewonnene Fehlernummer für die Fehlerbehandlung völlig ausreicht.

On Error Goto 0 und das Err-Objekt

Nun haben wir weiter oben erwähnt, dass Sie mit **On Error Resume Next** die VBA-interne Fehlerbehandlung ausschalten und diese mit **On Error Goto 0** wieder aktivieren.

Bei der Anweisung **On Error Goto 0** ist besonders die Wirkung auf das **Err**-Objekt zu beachten: Dieses wird dadurch nämlich wieder in seinen Ausgangszustand versetzt, wodurch etwa die Eigenschaft **Number** den Wert **0** und **Description** bei erneuter Abfrage eine leere Zeichenkette liefert.

Sollten Sie also nach einem durch Einsatz von **On Error Resume Next** unterdrückten Fehler die Eigenschaften des **Err**-Objekts auswerten wollen, dürfen Sie die VBA-interne Fehlerbehandlung nicht zuvor wieder mit **On Error Goto 0** wieder einschalten.

Die richtige Vorgehensweise sähe dann so aus: Zuerst wird die Fehlerbehandlung mit **On Error Resume Next** deaktiviert, der Fehler wird ausgelöst, die Fehlerursache ausgegeben (hier per Meldungsfenster) und schließlich die Fehlerbehandlung von VBA wieder angeschaltet.

Den Code zeigt folgendes Listing, das Ergebnis Bild 4:

Lookup-Daten per Formular verwalten

Wenn Sie Daten wie Anreden, Funktionen, Abteilungen et cetera in Lookup-Tabellen speichern und diese Daten in weiteren Tabellen per Nachschlagefeld verfügbar machen, werden Sie in Formularen Kombinationsfelder zur Auswahl solcher Daten bereitstellen. Was aber, wenn der Benutzer von diesem Formular die Daten der Lookup-Tabelle bearbeiten möchte – etwa, um diese zu erweitern, zu ändern oder um nicht benötigte Einträge zu löschen? Dann stellen Sie ihm ein weiteres Formular bereit, das er über eine Schaltfläche direkt neben dem Kombinationsfeld öffnen kann. Wie das funktioniert, zeigt dieser Artikel.

Beispieldatenbank

Die Beispiele zu diesem Artikel finden Sie in der Datenbank **1107_Lookupformular.mdb**.

Ausgangsformular

Im Artikel **Werte zu Kombinationsfeldern hinzufügen** haben wir gezeigt, wie Sie neue Daten für Lookup-Tabellen direkt in das Kombinationsfeld zu ihrer Auswahl eingeben können. Der vorliegende Artikel verwendet die gleiche Grundkonfiguration, das heißt: Das Formular **frmArtikel** basiert auf der Tabelle **tblArtikel** als Datenherkunft. Das Kombinationsfeld **cboKategorien** ist an das Feld **KategorieID** dieser Tabelle gebunden und zeigt seinerseits Daten aus der Tabelle **tblKategorien** an. Ausführlichere Informationen etwa zum Erstellen des Formulars finden Sie im oben genannten Artikel.

Formular zum Bearbeiten von Lookup-Daten

In diesem Beispiel sollen die Daten der Tabelle **tblKategorien** bearbeitet werden, und davon nur die beiden Felder **KategorieID** und **Kategorie**. Dazu erstellen Sie ein Formular namens **frmKategorien**, das als Hauptformular dient:

- Das Formular soll ein Unterformular enthalten, das wir gleich im Anschluss erstellen.
- Außerdem weisen Sie diesem eine Schaltfläche namens **cmdOK** mit der Beschriftung **OK** zu.
- Da das Formular selbst keine Daten anzeigt, sondern diese im Unterformular dargestellt werden, legen Sie für die Eigenschaften **Datensatzzeiger**, **Navigationsschaltflächen**, **Bildlaufleisten** und **Trennlinien** den Wert **Nein** fest. Außerdem soll die Eigenschaft **Automatisch zentrieren** den Wert **Ja** erhalten.

Speichern Sie das Formular unter dem Namen **frmKategorien** und legen Sie ein weiteres Formular an, dass Sie unter dem Namen **sfmKategorien** speichern:

- Weisen Sie der Eigenschaft **Datenherkunft** des Formulars den Wert **tblKategorien** zu.
- Fügen Sie die beiden Felder **KategorieID** und **Kategorie** aus der Feldliste zum Formularentwurf hinzu.
- Stellen außerdem die Eigenschaft **Standardansicht** auf **Datenblatt** ein. Die Steuerelemente brauchen Sie unter dieser Einstellung nicht mühevoll anzuordnen, in der Datenblattansicht geschieht dies automatisch.

Das Unterformular **sfmKategorien** sieht nun wie in Bild 1 aus. Damit die Schaltfläche **cmdOK** das Formular schließt, fügen Sie der Eigenschaft **Beim Klicken** der Schaltfläche den Wert **[Ereignisprozedur]** hinzu, klicken auf die Schaltfläche mit den drei Punkten (...) und ergänzen die nun im VBA-Editor angezeigte Ereignisprozedur wie folgt:

```
Private Sub cmdOK_Click()
    DoCmd.Close acForm, Me.Name
End Sub
```

Nun fügen Sie das Unterformular **sfmKategorien** zum Detailbereich des Hauptformulars **frmKategorien** hinzu – wie das am einfachsten gelingt, erfahren Sie in **Unterformular zu Formular hinzufügen**.

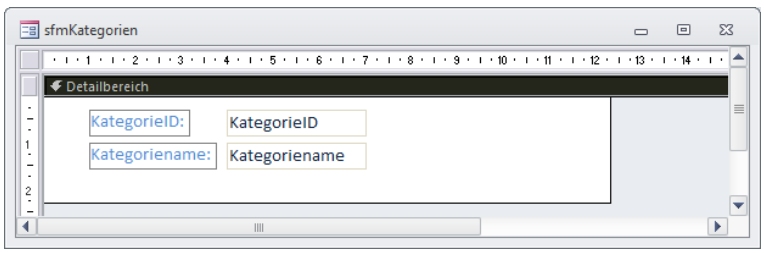


Bild 1: Das Unterformular zur Anzeige der Kategorien

Das Ergebnis sieht im Entwurf etwa wie in Bild 2 aus.

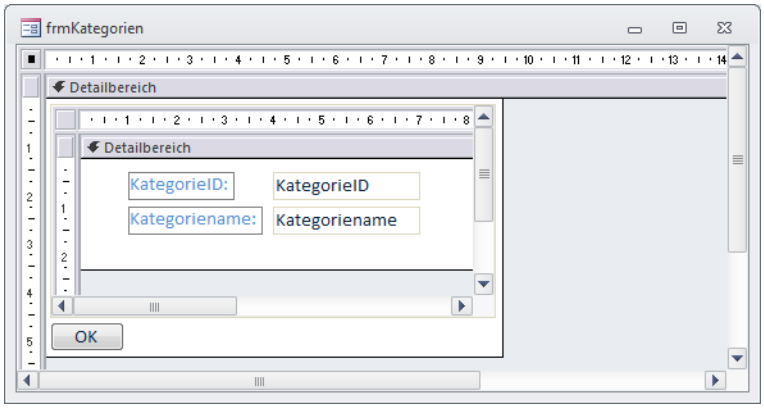


Bild 2: Haupt- und Unterfomular in vereinigter Form

Kategorie-Aktionen

Es gibt mehrere Aktionen in Zusammenhang mit den Kategorien, die einem Artikel zugeordnet werden können:

- Hinzufügen einer neuen Kategorie
- Bearbeiten der Liste von Kategorien
- Löschen von Kategorien
- Zusammenführen von Kategorien

Hinzufügen einer neuen Kategorien

Dieses Thema ist bereits erledigt: Das Hinzufügen einer neuen Kategorie erfolgt schlicht durch das Eintragen des Kategorienamens und der Bestätigung der Rückfrage durch den Benutzer – genau so, wie wir es bereits in **Werte zu Kombinationsfeldern hinzufügen** beschrieben haben.

Bearbeiten und Löschen der Kategorien

Das Bearbeiten und Löschen der Kategorien kann nicht direkt im Artikel-Formular erfolgen. Wenn der Benutzer hier die Bezeichnung einer Kategorie ändert, erkennt die Prozedur, die durch das Ereignis **Bei nicht in Liste** ausgelöst wird, dass hier eine neue Kategorie vorliegt und fragt, ob diese in der Kategorientabelle angelegt werden soll – siehe **Werte zu Kombinationsfeldern hinzufügen**.

Zum Bearbeiten der Bezeichnungen von Kategorien soll vielmehr das Formular **frmKategorien** erscheinen und dem Benutzer die Eingabe der neuen Bezeichnung ermöglichen.

Wie aber öffnen wir das Formular und wie zeigen wir dem Benutzer, dass er überhaupt die Möglichkeit hat, die Bearbeitung der Kategorien direkt vom Artikel-Formular aus zu starten?

Hier gibt es wiederum zwei Möglichkeiten:

- Sie fügen eine offensichtliche Methode hinzu, beispielsweise eine Schaltfläche rechts neben dem Kombinationsfeld. Als Beschriftung können Sie einfach drei Punkte (...) verwenden (siehe Bild 3).
- Oder Sie erlauben dem Benutzer, das Kategorien-Formular durch einen Doppelklick in das Kombinationsfeld **cboKategorieID** zu öffnen.

Zum Löschen von Kategorien sollten Sie ebenfalls das Detailformular heranziehen – mehr dazu weiter unten.

Kategorie-Formular öffnen

Zum Öffnen des Kategorie-Formulars fügen Sie also eine Schaltfläche namens **cmdKategorienBearbeiten** rechts nebem dem Kombinationsfeld **cboKategorieID** ein (siehe Bild 4). Weisen Sie der Eigenschaft **Beim Klicken** den Wert **[Ereignisprozedur]** zu und klicken Sie auf die daraufhin erscheinende Schaltfläche mit den drei Punkten (...) neben der Eigenschaft. Der VBA-Editor zeigt nun die durch das Ereignis **Beim Klicken** ausgelöste Prozedur an, die Sie wie folgt ergänzen:

```
Private Sub cmdKategorienBearbeiten_Click()
    DoCmd.OpenForm "frmKategorien", _
        WindowMode:=acDialog, DataMode:=acFormEdit
End Sub
```

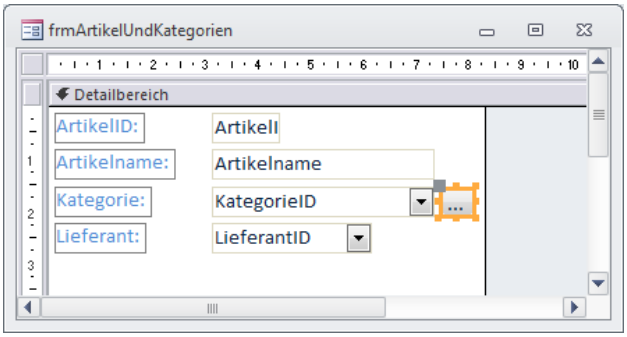


Bild 3: Schaltfläche zum Öffnen des Formulars **frmKategorien**

Die Prozedur öffnet das Formular **frmKategorien** wie in Bild 4. Sie können hier bereits mit Bordmitteln Kategorien ändern (durch einfaches Bearbeiten), hinzufügen (durch Einfügen von Datensätzen) oder löschen (durch markieren und betätigen der **Entf**-Taste).

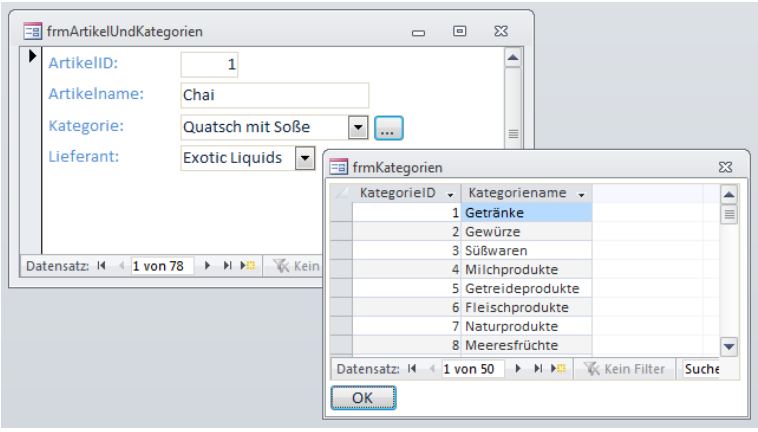


Bild 4: Öffnen der Kategorie-Übersicht per Schaltfläche

Das Löschen gelingt allerdings nur, wenn die Kategorie noch keinem Artikel-Datensatz zugewiesen ist. Dafür sorgt die für die Beziehung zwischen den Tabellen **tblArtikel** und **tblKategorien** festgelegte referentielle Integrität. Sollten Sie dennoch versuchen, einen bereits referenzierten Datensatz der Tabelle **tblKategorien** im Formular **frmKategorien** zu löschen, rufen Sie damit die Meldung aus Bild 5 hervor.

Außerdem kann es geschehen, dass Sie einer Kategorie einen Namen geben, der schon für eine andere Kategorie vergeben wurde. Auch dies löst einen Fehler aus, da das Feld **Kategoriename** der Tabelle **tblKategorien** mit einem eindeutigen Index versehen ist.

Sie sehen: Schon ein einfaches Formular zur Bearbeitung der Daten einer Lookup-Tabelle kann einige Probleme beziehungsweise Fehlermeldungen mit sich bringen. Wie Sie um die Anzeige dieser Meldungen herumkommen und stattdessen benutzerdefinierte Meldungen liefern, erfahren Sie im Artikel **Fehlerbehandlung in Formularen**.

Schließen und aktualisieren

Doch wir wollen uns noch um die eigentlichen Formularfunktionen kümmern. Eine davon ist das Aktualisieren des Kombinationsfeldes nach dem Schließen des Formulars zum Bearbeiten der Kategorien. Wenn der Benutzer beispielsweise den Namen der aktuell ausgewählten Kategorie ändert, soll dieser natürlich auch im Kombinationsfeld direkt entsprechend angezeigt werden. Dazu fügen Sie der Prozedur, welche das Kategorien-Formular öffnet, eine weitere Anweisung hinzu (hier fett gedruckt):

```
Private Sub cmdKategorienBearbeiten_Click()
    DoCmd.OpenForm "frmKategorien", _
        WindowMode:=acDialog, DataMode:=acFormEdit
    Me!cboKategorieID.Requery
End Sub
```

Diese Anweisung aktualisiert die Datensatzherkunft des Kombinationsfeldes, liest also den Inhalt der für die Eigenschaft **Datensatzherkunft** angegebenen Tabelle/Abfrage. Wie aber kann es sein, dass diese Anweisung nicht direkt nach dem Öffnen des Formulars **frmKategorien** ausgeführt wird, sondern erst nach dem Schließen dieses Formulars? Dies liegt am Parameter **WindowMode** der **DoCmd.OpenForm**-Methode zum Öffnen des Formulars, der hier auf **acDialog** eingestellt wird. Dies sorgt dafür, dass der Code der aufrufenden Prozedur erst fortgesetzt wird, wenn das geöffnete Formular entweder geschlossen oder zumindest unsichtbar gemacht wird.

Das Schließen des Formulars **frmKategorien** ruft also die Methode **Requery** des Kombinationsfeldes auf den Plan. Was bewirkt der zweite Parameter **acFormEdit**? Er sorgt schlicht dafür, dass die Datensätze beim Öffnen mit diesem Parameter bearbeitet oder neue Datensätze angelegt werden können.

Bild 5: Diese Fehlermeldung erscheint, wenn Sie versuchen, einen verknüpften Datensatz der Tabelle **tblKategorien** zu löschen.

Aktuelle Kategorie beim Öffnen anzeigen

Schauen wir uns noch eine mögliche Verbesserung an: Das Formular **frmKategorien** soll gleich beim Öffnen die im Kombinationsfeld

Recordsets in der Datenblattansicht

Recordsets sind ein elementarer Bestandteil beim Umgang mit Daten in Access. Ein Recordset, zu deutsch Datensatzgruppe, enthält einen oder mehrere Datensätze. Ein Datensatz entspricht etwa der Zeile einer Tabelle, kann aber auch eine Zeile eines Abfrageergebnisses enthalten. Recordsets liefern die Daten für die Anzeige in Formularen und Berichten und in Steuerelementen wie dem Kombinationsfeld oder dem Listenfeld. Die Datenblattansicht von Tabellen, Abfragen oder auch in Formularen liefert ein gutes Abbild einer Datensatzgruppe. Dieser Artikel vermittelt Begriffe und grundlegenden Techniken für den Einsatz von Recordsets in der Datenblattansicht.

Recordsets im Datenblatt

Für die Anzeige der Daten einer Tabelle oder eines Abfrageergebnisses in einem Datenblatt ist ein Recordset verantwortlich. Ein Recordset besteht aus einem oder mehreren Datensätzen, kann aber auch keinen Datensatz enthalten – ein leeres Recordset erhalten Sie beispielsweise, wenn Sie eine Abfrage ausführen, die keine passenden Datensätze findet. Ein Beispiel zeigt Bild 1. Die Datenblattansicht ist ein hervorragendes Mittel, um direkt mit den Daten eines Recordsets zu arbeiten. Eine Zeile der Datensatzgruppe entspricht dabei einem Record, also einem Datensatz. Der aktuelle Datensatz wird dabei immer optisch hervorgehoben – hier ist es der Datensatz mit dem Wert **74** im Feld **ArtikelID**. Das bedeutet allerdings nicht unbedingt, dass dies auch der 74. Datensatz ist – die Werte des Feldes **ArtikelID** müssen nicht unbedingt durchnummeriert sein.

Datensatzzeiger

Um den aktuellen Datensatz zu kennzeichnen, gibt es den sogenannten Datensatzzeiger. Diesen können Sie bewegen, um anderen Datensätze als den zur Zeit markierten zum aktuellen Datensatz zu machen. Das geht in der Datenblattansicht auf verschiedene Arten: Sie können einfach mit der Maus auf einen anderen Datensatz klicken oder auch mit den Steuerelementen der Navigationsleiste im unteren Bereich des Datenblatts durch die Datensätze navigieren. Auch das Löschen eines Datensatzes führt dazu, dass ein anderer Datensatz zum aktuellen Datensatz wird.

Die Position des Datensatzzeigers können Sie unten im Navigationsbereich ablesen. Bei älteren Access-Versionen erscheint dort ein Pfeil, bei neueren Versionen wird das entsprechende Kästchen farbig markiert (siehe Bild 3). Die Nummerierung der Datensätze beginnt dabei mit Eins und erfolgt immer durchlaufend – abhängig von der jeweiligen Sortierung. Dabei wird immer die Position des Datensatzes in der Datensatzgruppe wiedergegeben! Wenn die Datensätze nach dem Wert des Feldes **ArtikelID** sortiert sind, nimmt beispielsweise der Datensatz mit dem Wert **1** im Feld **ArtikelID** auch die erste Position ein. Sortiert man die Datensätze hingegen nach dem Feld **Artikelname**, findet sich der gleiche Datensatz möglicherweise an einer ganz anderen Stelle wieder.

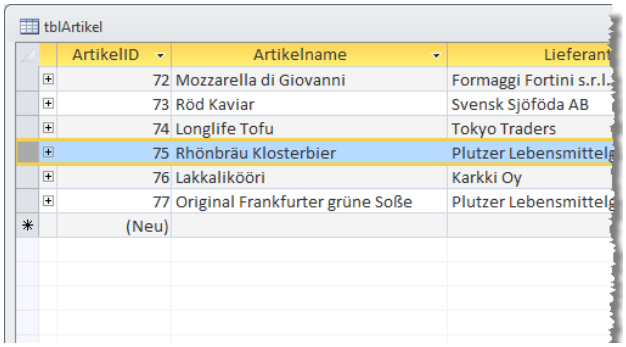


Bild 1: Ein Recordset in der Datenblattansicht einer Tabelle

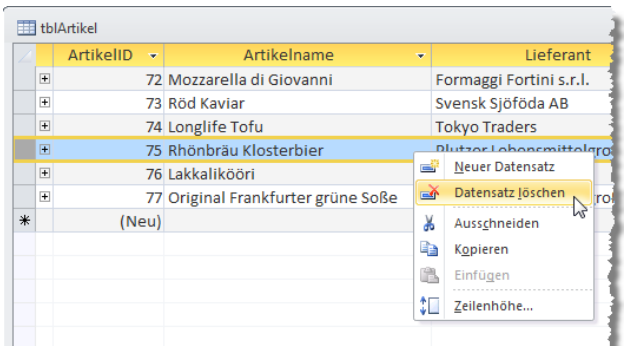


Bild 2: Löschen eines Records

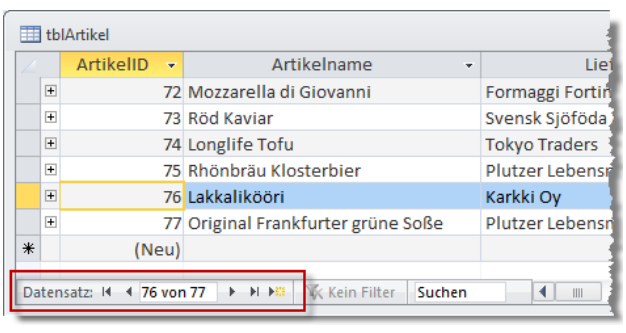


Bild 3: Navigationsleiste mit Datensatznummer

Datensatz löschen

Im Datenblatt löschen Sie einen Datensatz, indem Sie den kompletten Datensatz durch einen Klick auf den sogenannten Datensatzmarkierer, also den grauen Bereich links von den Feldern, markieren und dann die **Entfernen**-Taste oder den entsprechenden Eintrag in der Symbolleiste/dem Ribbon oder dem Kontextmenü betätigen (siehe Bild 2).

Es reicht nicht, einfach nur die Einfügemarke in einem Feld des zu löschenden Datensatzes zu platzieren – damit löschen Sie höchstens Teile des jeweiligen Feldinhalts. Nach dem Löschen wird der Datensatzzeiger auf den folgenden Datensatz verschoben. Wurde der letzte Datensatz der Datensatzgruppe gelöscht, wird der Datensatzzeiger auf einen neuen, leeren Datensatz gesetzt.

Achtung: Das Löschen eines Datensatzes kann nicht rückgängig gemacht werden. Wenn Sie sich die Möglichkeit vorbehalten möchten, den Datensatz wieder herzustellen, sollten Sie diesen ausschneiden (zum Beispiel mit der Tastenkombination **Strg + X**).

Sie können diesen dann durch Anklicken des Datensatzmarkierers eines neuen Datensatzes und Betätigen der Tastenkombination **Strg + C** wieder einfügen. Allerdings wird bei dieser Methode ein neuer Wert im Primärschlüsselfeld vergeben, wenn dieses als Autowert ausgelegt wurde.

Felder im Datensatz

Jeder Datensatz enthält eines oder mehrere Felder. Die Felder besitzen einen Datentyp, der vorgibt, mit welcher Art von Daten das Feld gefüllt werden darf. Die Felder liegen in einer Auflistung vor, die in der Datenblattansicht durch Spalten repräsentiert wird.

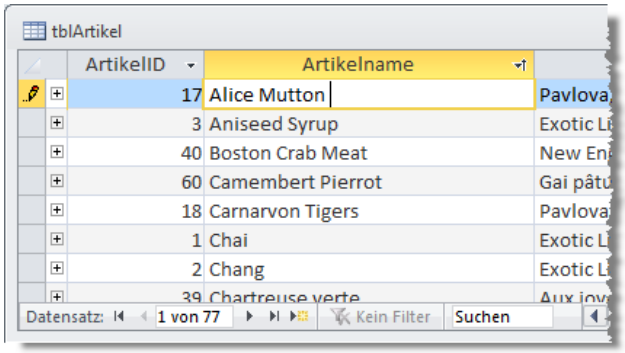


Bild 4: Bearbeiten eines Datensatzes

Navigieren im Datenblatt

Die Navigationsleiste eines Datenblatts ermöglicht die folgenden Schritte, wobei diese auch mit entsprechenden Tastenkombinationen durchgeführt werden können:

- Aktivieren des ersten Datensatzes (**Strg + Pos1**)
- Aktivieren des vorherigen Datensatzes (**Nach oben**)
- Aktivieren des folgenden Datensatzes (**Nach unten**)
- Aktivieren des letzten Datensatzes (**Strg + Ende**)
- Aktivieren eines neuen Datensatzes (**Strg + +**)
- Aktivieren des Datensatzes mit einer bestimmten Nummer (Eingabe der Nummer in das Textfeld)
- Zum nächsten Feld springen (**Tabulator**)
- Zum vorherigen Feld springen (**Umschalt + Tabulator**)

Daten bearbeiten

Wenn das in der Datenblattansicht angezeigte Recordset aktualisierbar ist (dies ist nicht immer der Fall), können Sie die Daten direkt innerhalb dieser Ansicht bearbeiten.

Dazu klicken Sie einfach in das Feld, das Sie bearbeiten möchten und geben den gewünschten Wert ein beziehungsweise wählen diesen aus, wenn es sich um ein Kombinationsfeld handelt. Ist eines der Felder bearbeitet, erscheint im Datensatzmarkierer links ein **Bearbeiten**-Symbol (siehe Bild 4).

Die Änderungen an einem einzelnen Feld können Sie mit der Escape-Taste solange rückgängig machen, bis Sie ein anderes Feld des gleichen Datensatzes bearbeiten. Dann können Sie die Änderung zwar immer noch rückgängig machen, aber nur für den kompletten Datensatz: Betätigen Sie einmal die Escape-Taste, um die letzte Feldänderung zu revidieren und drücken Sie ein zweites Mal auf diese Taste, um die Änderungen am allen Feldern rückgängig zu machen. Dies gelingt allerdings nur, bis Sie den Datensatz gespeichert haben – durch eine der folgenden Aktionen, die alle darin resultieren, dass das

Formular- und Berichtsvorlagen

Unter Access können Sie eine ganze Reihe von Einstellungen vornehmen, die sich beim Entwickeln von Datenbanken bemerkbar machen und eine Menge Zeit sparen können. Einige davon finden Sie in den Access-Optionen, für andere sind ein paar Handgriffe notwendig. Dieser Artikel zeigt, wie Sie einige Zeit beim Anlegen von Formularen, Berichten und den enthaltenen und Steuerelementen sparen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **11_08_FormularUndBerichtsvorlagen.mdb**.

Das Formular Normal

Je nach Anwendung werden Sie einige grundlegende Formulartypen einsetzen: eines für die Anzeige von Daten, in denen der Benutzer blättern kann, eines für die Anzeige von Detaildaten, das nur jeweils einen einzigen Datensatz darstellen soll oder auch Unterformulare, die ihre Daten grundsätzlich in der Datenblattansicht darstellen.

Für all diese Formular legen Sie normalerweise beim Anlegen die wichtigsten Einstellungen fest. Bei einem Formular, dass die Detaildaten zu einem Datensatz anzeigt und entweder zum Anlegen oder zum Bearbeiten eines einzigen Datensatzes eingesetzt wird, werden Sie beispielsweise die Eigenschaft **Automatisch zentrieren** auf **Ja** und die Eigenschaften **Datensatzmarkierer**, **Navigationsschaltflächen**, **Trennlinien** und **Bildlaufleisten** auf den Wert **Nein** einstellen. Außerdem können Sie sogar festlegen, welche Bereiche beim Anlegen des Formulars aktiviert sein sollen und weitere Eigenschaften wie etwa die Hintergrundfarbe festlegen. Ein Beispiel zeigt Bild 1.

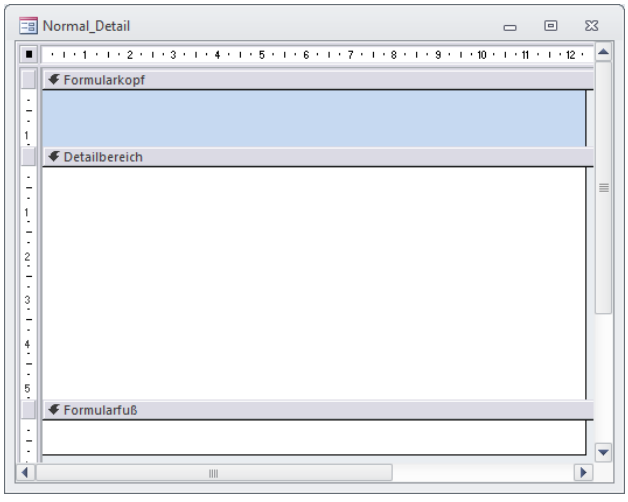


Bild 1: Einstellungen für eine Formularvorlage

Für all diese Formulartypen präferieren Sie wahrscheinlich eine gemeinsame Schriftart und -größe in Steuerelementen wie Beschriftungsfeldern, Textfeldern, Kombinationsfeldern, Listenfeldern, Schaltflächen und Umschaltflächen. Und auch in der Datenblattansicht passt Ihnen die von Access voreingestellte Schriftgröße möglicherweise nicht.

All dies lässt sich zumindest für einen Formulartyp beheben, indem Sie ein Formular namens **Normal** anlegen und für dieses die gewünschten Eigenschaften einstellen. Gehen wir davon aus, dass Sie gerade speziell Detailformulare entwerfen. Dann stellen Sie die oben erwähnten Eigenschaften **Datensatzmarkierer**, **Navigationsschaltflächen**, **Trennlinien** und **Bildlaufleisten** auf **Nein** und **Automatisch zentrieren** auf **Ja** ein.

Wozu nun diese ganze Arbeit? Nun, schließen Sie das Formular und erstellen Sie ein neues Formular. Legen Sie nun ein neues Formular an, erhält dieses genau die Eigenschaften, die Sie für das Formular namens **Normal** angegeben haben! Dass das als Vorlage verwendete Formular **Normal** heißt, ist übrigens kein Zufall: Sie können den Namen des Formulars und auch des Berichts, das als Vorlage für neue Formulare und Berichte dienen soll, in den Access-Optionen einstellen. Im Falle von Access 2007 und jünger finden Sie die entsprechenden Eigenschaften beispielsweise unter **Objekt-Designer|Entwurfsansicht für Formulare und Berichte** (siehe Bild 2).

Steuerelementeigenschaften voreinstellen

Sie können jedoch nicht nur Formular- und Berichtseigenschaften festlegen, sondern auch bestimmte Eigenschaften von Steuerelementen.

Ein gutes Beispiel sind die Eigenschaften, die das Aussehen der Schrift beeinflussen, also etwa **Schriftart**, **Schriftgrad**, **Unterstriche**, **Kursiv**, **Fett**, **Textfarbe** und so weiter. Zumindest für einige werden Sie eine feste Vorstellung haben – zum Beispiel für die Schriftgröße. Ab Access 2007 erscheinen Texte in Formularen standardmäßig in **Schriftgrad 11** – dies mag für den einen

oder anderen zu groß sein. Wenn Sie dies möchten, dass Access neue Steuerelemente direkt etwa mit dem Wert **9** für die Eigenschaft **Schriftgrad** anlegt, führen Sie gleich nach dem Erstellen eines neuen Formulars die folgenden Schritte durch:

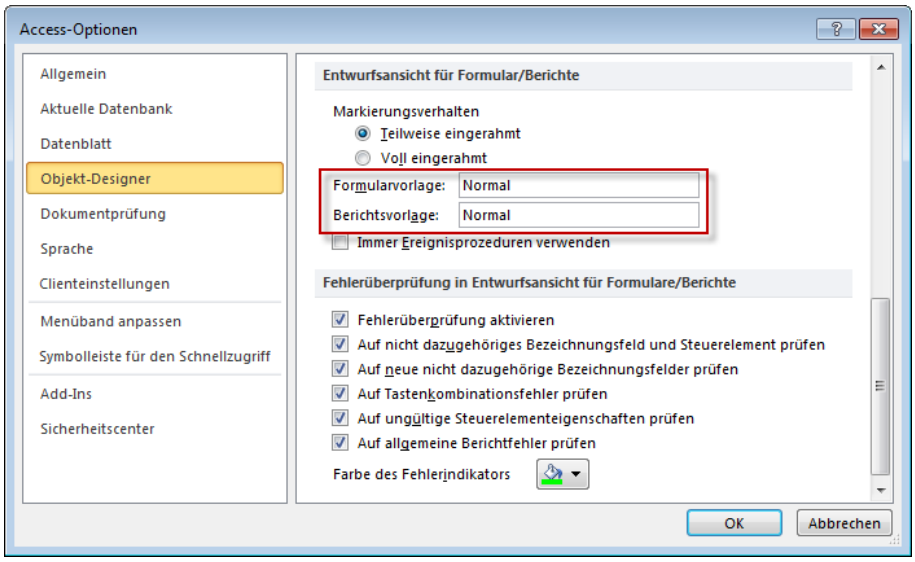


Bild 2: Einstellen der Namen des Standardformulars beziehungsweise -berichts

- Öffnen Sie das Formular in der Entwurfsansicht.
- Klicken Sie in der Symbolleiste oder im Ribbon die Schaltfläche zum Hinzufügen eines Steuerelements an, beispielsweise für das Textfeld. Fügen Sie das Steuerelemente aber nicht ein!
- Aktivieren Sie das Eigenschaftsfenster (Taste **F4**) Dieses zeigt nun dort, wo sonst der Steuerelementname erscheint, einen Text wie **Standard: Textfeld** an – je nachdem, welchen Steuerelementtyp Sie gerade ausgewählt haben. In Bild 3 sehen Sie beispielsweise, dass die sonst bei Textfeldern üblichen Eigenschaften **Links** und **Oben** fehlen – kein Wunder, denn für ein Standardsteuerelement können Sie eben maximal **Höhe** und **Breite** vordefinieren.
- Stellen Sie die Eigenschaft **Schriftgrad** auf **9** oder einen anderen gewünschten Wert ein. Nehmen Sie nach Wunsch weitere Änderungen vor.
- Jetzt folgt der wichtigste Schritt: Speichern Sie die aktuelle Einstellung entweder mit der Tastenkombination **Strg + S** oder über die **Speichern**-Schaltfläche in der Symbolleiste beziehungsweise im Ribbon.
- Legen Sie nun ein neues Steuerelement des soeben bearbeiteten Typs an und prüfen Sie, ob die geänderten Eigenschaften des Standardsteuerelements übernommen wurden – was der Fall sein dürfte.

Wenn Sie die etwa die Eigenschaft **Schriftgrad** anpassen möchten, sollten Sie dies durchgängig für alle Steuerelemente erledigen, die Texte anzeigen – also für Textfelder, Bezeichnungsfelder, Listenfelder, Kombinationsfelder, Schaltflächen und Umschaltflächen.

Wenn Sie nun ein neues Formular anlegen und hoffen, dass die Änderungen der Standardeigenschaften sich dort auswirken, werden Sie enttäuscht werden: Die Änderungen gelten nur jeweils für das Formular, in dessen Kontext diese gespeichert wurden. Aber es gibt eine Lösung: Stellen Sie die Standardeigenschaften einfach für das Formular **Normal** ein! Wenn Sie dann ein neues Formular erstellen, erzeugt Access dieses auf Basis des Formulars **Normal** und übernimmt auch die für die verschiedenen Steuerelemente vorliegenden Standardeigenschaften.

Formatierung der Datenblattansicht

Wenn Sie ein Formular in der Datenblattansicht anzeigen möchten, besteht ebenfalls die Wahrscheinlichkeit, dass dieses die Daten nicht standardmäßig

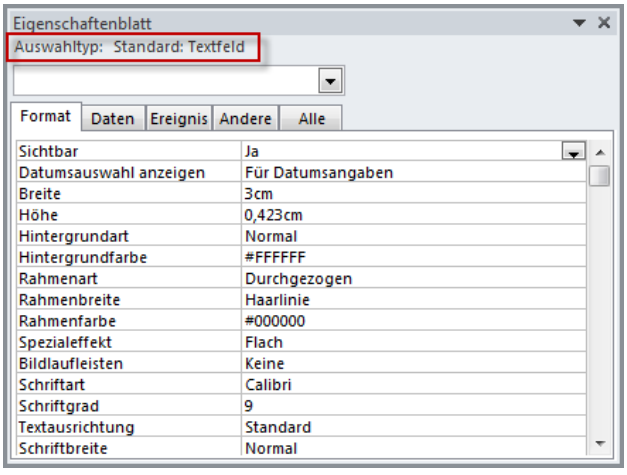


Bild 3: Einstellen von Standardeigenschaften für ein Steuerelement

Datensätze in Unterformularen anlegen

Wie Sie Haupt- und Unterformulare zur Darstellung und Bearbeitung von Daten aus Tabellen mit 1:n-Beziehung erstellen, lesen Sie in jedem Grundlagenbuch und auf jeder halbwegs relevanten Internetseite zum Thema Access. Dass der Benutzer bei der Dateneingabe jedoch leicht auf die Nase fallen kann, lassen die meisten außer acht: Wer nämlich ohne Datensatz im Hauptformular Daten in das Unterformular eingibt, schickt seine Datensätze ins Beziehungsnirvana. Dieser Artikel zeigt, wie Sie dies verhindern.

Beispieldatenbank

Die Beispieldatenbank zu diesem Artikel heißt **11_08_DatensaetzeInUnterformularen.mdb**.

Problem beim Eingeben von Daten in Unterformularen

Wenn Sie zwei Tabellen namens **tblKunden** und **tblProjekte** verwenden, bei der die Datensätze der Tabelle **tblProjekte** über das Fremdschlüsselfeld **KundeID** mit dem Primärschlüsselfeld der Tabelle **tblKunden** verknüpft sind, sieht die Vorgehensweise zum Anzeigen dieser Daten in zwei Formularen namens **frmKunden** und **sfmProjekte** so aus:

- Sie legen ein Hauptformular namens **frmKunden** an, dass auf einer Tabelle wie **tblKunden** als Datenherkunft basiert und deren Felder anzeigt.

- Sie erstellen ein Formular namens **sfmProjekte**, dass die Tabelle **tblProjekte** als Datenherkunft nutzt.

- Sie fügen das Formular **sfmProjekte** als Unterformular in das Hauptforular **frmKunden** ein. Dabei stellt Access normalerweise direkt die beiden Eigenschaften **Verknüpfen von** und **Verknüpfen nach** des Unterformularsteuerelements so ein, dass das Unterformular nur die Projekte zum jeweiligen Kunden des Hauptformulars anzeigt (siehe Bild 1). Beim Anlegen neuer Projekte im Unterformular wird das Fremdschlüsselfeld **KundeID** der Tabelle **tblProjekte** gleich mit dem Primärschlüsselwert des Datensatzes gefüllt, der im Hauptformular angezeigt wird.

Dies reicht auch normalerweise völlig aus, um verknüpfte Daten per Haupt- und Unterformular einzugeben. Dummerweise aber kann es auch passieren, dass der Benutzer erst ein Projekt einträgt und erst dann die Kundendaten. Dazu springt er zunächst auf einen leeren Kundendatensatz (oder er arbeitet mit einer neuen, leeren Datenbank – dann gibt es ohnehin noch keinen Kundendatensatz). Er trägt im Unterformular das Projekt ein (siehe Bild 2), wechselt dann zum Hauptformular und legt einen Kunden an. Das sieht auf den ersten Blick sogar noch gut aus. Sobald Access jedoch merkt, dass da ein neuer Datensatz angelegt werden soll (in der Regel direkt nach der Eingabe des ersten Zeichens), verschwindet der Projektdatensatz aus dem Unterformular! Was geschieht hier? Nun: Durch das Einstellen der beiden Eigenschaften **Verknüpfen von** und **Verknüpfen nach** des Unterformular-Steuerelements legen Sie zwei Dinge fest:

- Das Unterformular soll nur Datensätze anzeigen, die mit dem Datensatz des Hauptformulars verknüpft sind.

- Beim Anlegen neuer Datensätze soll das Fremdschlüsselfeld im Unterformular automatisch mit dem aktuellen Primärschlüsselwert im Hauptformular gefüllt werden.

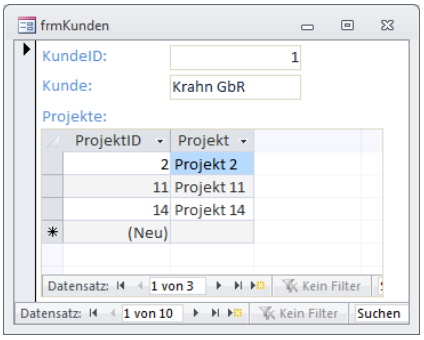


Bild 1: Normalerweise zeigt das Unterformular die verknüpften Datensätze an.

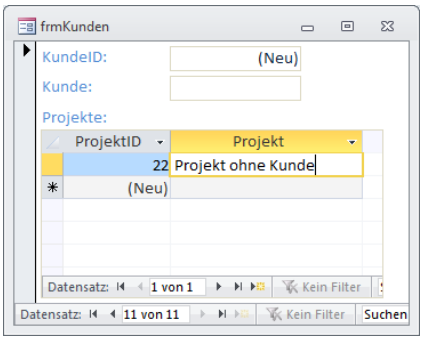


Bild 2: Was aber, wenn man ein Projekt eingibt, ohne dass das Hauptformular einen Kunden anzeigt?

Fehlerbehandlung in der Praxis

Wer die technischen Grundlagen zur Fehlerbehandlung im Artikel Fehlerbehandlung unter VBA gelesen hat, braucht nun ein paar Praxisbeispiele. Kein Problem: Der vorliegende Artikel ergänzt noch ein paar Kleinigkeiten und zeigt dann, wie Sie die Befehle zur Fehlerbehandlung sinnvoll in Ihre eigenen Anwendungen integrieren. Dabei erfahren Sie, dass eine Fehlerbehandlung nicht nur unvorhergesehene Fehler abfangen, sondern auch ganz gezielt eingesetzt werden kann.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Beispieldatenbank **11_08_FehlerbehandlungVBAII.mdb**.

Fehler als Option

Einer der wichtigsten Anwendungsfälle neben dem gesteuerten Behandeln von Laufzeitfehlern ist das Provozieren von Fehlern. Gut, eigentlich soll kein Fehler provoziert werden, aber das Auftreten eines Fehlers bei einer bestimmten Aktion kann hier als Option behandelt werden.

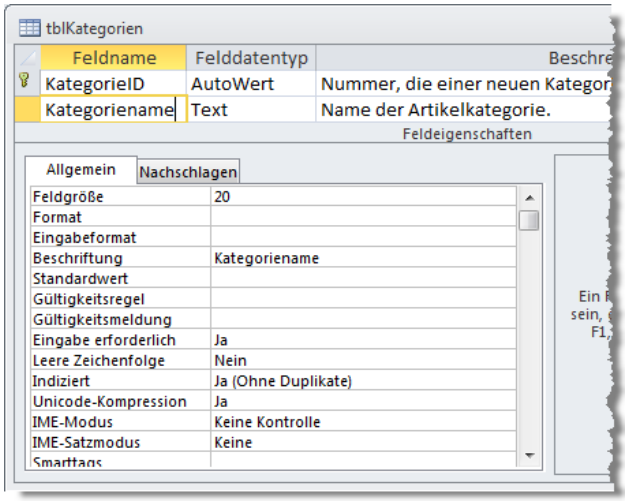


Bild 1: Das Hinzufügen doppelter Werte in dieses Feld löst einen Fehler aus.

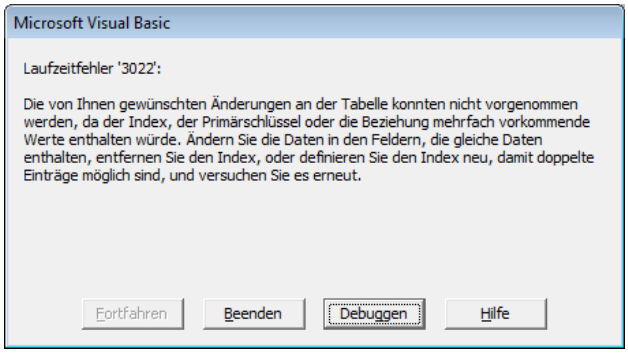


Bild 2: Fehlermeldung beim Anlegen doppelter Werte im eindeutigen Feld

Stellen Sie sich vor, Sie hätten eine kleine Prozedur, die einen Eintrag in eine Lookup-Tabelle vornehmen soll – zum Beispiel den Namen einer Kategorie in der Tabelle **tblKategorien**. Das Feld **Kategoriename** ist dabei durch den Wert **Ja (Ohne Duplikate)** für die Eigenschaft **Indiziert** als eindeutiges Feld markiert (siehe Bild 1). Das wiederholte Anlegen des gleichen Wertes schon in der Datenblattansicht der Tabelle führt dabei zu einer Fehlermeldung. Das Gleiche geschieht, wenn Sie einen Wert wie mit der folgenden VBA-Prozedur einfügen:

```
Public Sub KategorieHinzufuegen(strKategorie _
                                As String)
    Dim db As DAO.Database
    Set db = CurrentDb
    db.Execute "INSERT INTO _
          tblKategorien(Kategoriename) VALUES('" _
              & strKategorie & "')", dbFailOnError
    Set db = Nothing
End Sub
```

Es erscheint die Fehlermeldung aus Bild 2.

Wie können Sie diesen Fehler umgehen? Dazu gibt es zwei Varianten. Die erste prüft zuvor, ob es bereits eine Kategorie mit dem angegebenen Namen gibt:

```
If IsNull(DLookup("KategorieID", "tblKategorien", _
      "Kategoriename = '" & strKategorie & "'")) Then
    db.Execute "INSERT INTO _
          tblKategorien(Kategoriename) VALUES('" _
              & strKategorie & "')", dbFailOnError
Else
    MsgBox "Die Kategorie '" & strKategorie _
                      & "' ist bereits vorhanden."
End If
```

Die zweite Variante fragt gar nicht groß nach, sondern legt den Datensatz einfach an. Da vorher mit **On Error Resume Next** die eingebaute Fehlerbehandlung deaktiviert wird, löst die **Execute**-Methode keinen Fehler aus, wenn der anzulegende Datensatz bereits vorhanden ist. Dafür wird aber nachher geprüft, ob nicht vielleicht doch ein Fehler aufgetreten ist –

beispielsweise der mit der Nummer **3022**. In diesem Fall zeigt die in der **If...Then**-Bedingung enthaltene Anweisung eine entsprechende Meldung an:

```
On Error Resume Next
db.Execute "INSERT INTO ⤵
          tblKategorien(Kategoriename) VALUES('" ⤵
              & strKategorie & "')", dbFailOnError
If Err.Number = 3022 Then
    MsgBox "Die Kategorie '" & strKategorie ⤵
                      & "' ist bereits vorhanden."
End If
```

Wirkungsbereich einer Fehlerbehandlung

Eine Fehlerbehandlung aktivieren Sie durch die **On Error...**-Anweisung aktiviert und sie und endet spätestens mit dem Ende der Prozedur. Sie müssen allerdings zwei Besonderheiten beachten:

- Sie können die Fehlerbehandlung auch vor dem Prozedurende mit der Anweisung **On Error Goto 0** deaktivieren.

- Wenn die Prozedur eine weitere Prozedur oder Funktion aufruft, während die Fehlerbehandlung aktiviert ist, gilt die Fehlerbehandlung auch hier als aktiviert. Das Gleiche gilt auch für weitere Verschachtelungen.

Wenn Sie also in einer Prozedur wie folgt eine Fehlerbehandlung aktivieren, von dort eine weitere Prozedur aufrufen, in der dann schließlich ein Fehler auftritt, wird der Fehler von der Fehlerbehandlung der aufrufenden Prozedur behandelt. Die aufrufende Prozedur mit der Fehlerbehandlung sieht so aus:

```
Public Sub VerschachtelteFehlerbehandlung()
    On Error GoTo Fehler
    Call WeitereProzedur
Ende:
    Exit Sub
Fehler:
    MsgBox "Fehler " & Err.Number & ", '" ⤵
                          & Err.Description & "'"
    GoTo Ende
End Sub
```

Dies ist die aufgerufene Prozedur:

```
Public Sub WeitereProzedur()
    Debug.Print 1 / 0
End Sub
```

Auch wenn der Fehler erst in der Prozedur **WeitereProzedur** ausgelöst wird, wird er dennoch in der aufrufenden Prozedur verarbeitet.

Wenn Sie jedoch in der aufgerufenen Prozedur gleich zu Beginn die Anweisung **On Error Goto 0** einfügen, deaktiviert dies die Fehlerbehandlung nicht! Um dies zu testen, haben wir an verschiedenen Stellen **Debug.Print**-Anweisungen zur Ausgabe von Zahlen in der Reihenfolge des erwarteten Ablaufs hinzugefügt. Bei folgendem Beispiel werden **1** und **2** ausgegeben, dann erscheint die Fehlermeldung der Fehlerbehandlung:

```
Public Sub VerschachtelteFehlerbehandlung()
    On Error GoTo Fehler
    Debug.Print 1
    Call WeitereProzedur
    Debug.Print 4
    Debug.Print 1 / 0
    Debug.Print 5
Ende:
    Exit Sub
Fehler:
    MsgBox "Fehler " & Err.Number & ", '" ⤵
                          & Err.Description & "'"
    GoTo Ende
End Sub

Public Sub WeitereProzedur()
    On Error GoTo 0
    Debug.Print 2
    Debug.Print 1 / 0
    Debug.Print 3
End Sub
```

Das heißt, dass eine durch die aufrufende Prozedur aktivierte Fehlerbehandlung in einer Unterprozedur nicht deaktiviert wird. Wenn Sie hingegen die Zeile **On Error Goto 0** in der aufrufenden Prozedur unmittelbar vor der Zeile **Call WeitereProzedur** einfügen,

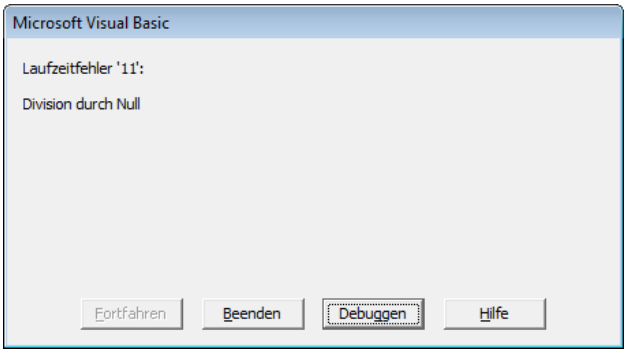


Bild 3: Eingebaute Fehlermeldung

Fehlerbehandlung in Formularen

In den Artikeln Fehler unter VBA und Fehlerbehandlung unter VBA (Teil I und II) haben Sie fast schon alles über Fehler und ihre Behandlung erfahren. Es gibt jedoch ein paar Fälle, in denen zwar Fehlermeldungen erscheinen, Sie aber keine Chance haben, auf diese zu reagieren. Dies geschieht beispielsweise, wenn Sie per Formular einen Wert zwei Mal in ein Feld mit eindeutigem Index eingeben. In diesem Artikel erfahren Sie, wie Sie solche Fehler ebenfalls abfangen und eine eigene Fehlerbehandlung dafür implementieren – gegebenenfalls auch mit benutzerdefinierter Fehlermeldung.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **11_08_FehlerbehandlungInFormularen.mdb**.

Benutzerdefinierte Fehlermeldungen

Nichts ist ärgerlicher, als wenn der Benutzer einer Anwendung Fehlermeldungen angezeigt bekommt, die er nicht interpretieren kann, weil diese schlicht nicht aussagekräftig genug ist. In solchen Fällen sollten Sie immer eine benutzerdefinierte Meldung bereithalten und entsprechend auf den Fehler reagieren (gegebenenfalls gelingt dies ja auch, ohne dass der Benutzer überhaupt eine Fehlermeldung zu Gesicht bekommt ...). Wie Sie dies für Fehler durchführen, die durch VBA-Anweisungen in den Ereignisprozeduren ausgelöst werden, erfahren Sie unter **Fehlerbehandlung unter VBA** und **Fehlerbehandlung in der Praxis**.

Die Fehler, von denen wir hier sprechen, sind in den meisten Fällen keine Fehler, sondern lediglich Meldungen, die auf Verletzungen etwa der Datenintegrität hinweisen – meist werden diese durch die Jet-Engine verursacht, die für den Zugriff und das Bearbeiten der in den Tabellen gespeicherten Daten verantwortlich ist. Beispiele hierfür sind die folgenden:

- Eingabe eines Textes in ein Zahlenfeld,
- Speichern eines Datensatzes, der noch nicht gefüllte Pflichtfelder enthält,
- Gültigkeitsmeldungen oder
- Eingabe eines bereits vorhandenen Wertes in ein Feld, das als eindeutiges Feld indiziert ist.

Restriktionen durch den Datentyp

Das erste Beispiel zeigt die grundlegende Vorgehensweise beim Abfangen von Fehlern in Formularen, die nicht durch den Code selbst ausgelöst werden. Das Beispielformular heißt **frmArtikel** und verwendet die Tabelle **tblArtikel** als Datenherkunft. Diese Tabelle enthält beispielsweise ein Feld namens **Einzelpreis**, das den Datentyp **Währung** aufweist. Dementsprechend darf der Benutzer nur Zahlenwerte eingeben. Wenn er hier etwa einen Text eingibt, weil er der Ansicht ist, man müsste neben der Preisangabe noch die Zeichenkette **EURO** ausschreiben, löst dies die Meldung aus Bild 1 aus.

Der Meldungstext ist eigentlich aussagekräftig, aber wir können ihn dennoch durch eine eigene Meldung ersetzen. Dazu verwenden Sie eine spezielle Ereignisprozedur namens **Bei Fehler**, die nur bei Fehlern ausgelöst wird, die nichts mit dem Code des Formulars zu tun haben. Aktivieren Sie dazu in der Entwurfsansicht des Formulars das Eigenschaftsfensters und wechseln Sie zur Registerseite **Ereignis**. Dort finden Sie, sofern das Formular selbst aktiviert ist, den Eintrag **Bei Fehler**. Wählen Sie hier den Wert **[Ereignisprozedur]** aus (siehe Bild 2) und klicken

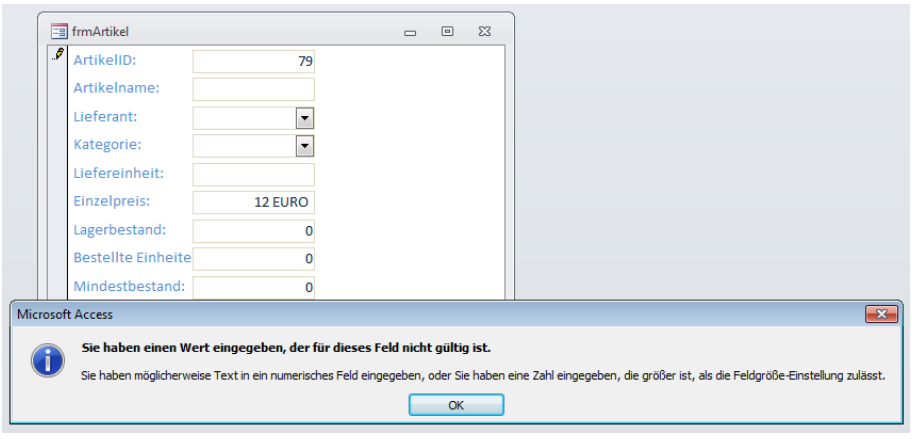


Bild 1: Dieser Meldung erscheint beim Eingeben von Text in numerische Felder.

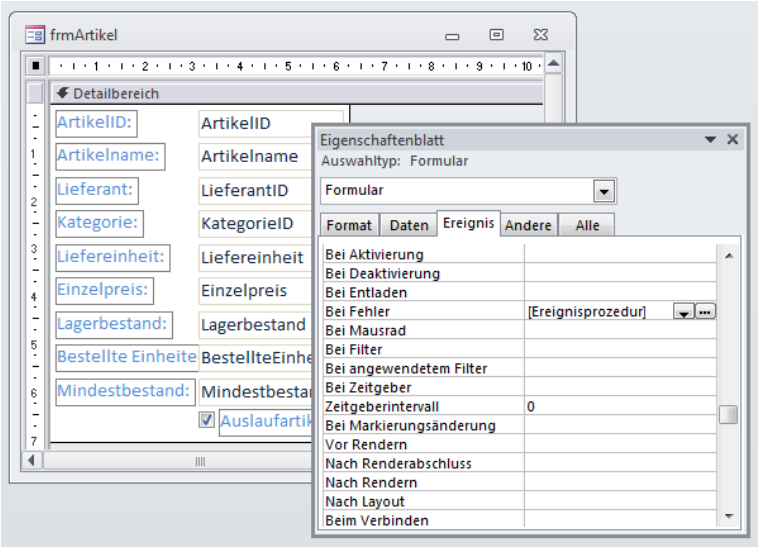


Bild 2: Anlegen der Ereignisprozedur, die bei Fehlern im Formular ausgelöst wird

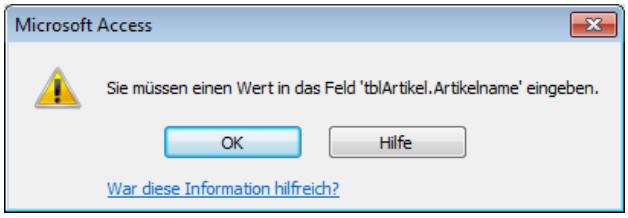


Bild 3: Fehlermeldung beim Speichern eines Datensatzes mit leeren Pflichtfeldern

Sie auf die Schaltfläche mit den drei Punkten. Es erscheint der VBA-Editor, der das Grundgerüst der Prozedur zeigt, mit der Sie die oben erwähnten Fehler abgreifen können:

```
Private Sub Form_Error(DataErr As Integer, _
                       Response As Integer)
End Sub
```

Diese Ereignisprozedur besitzt zwei Parameter:

- Der erste heißt **DataErr** und ist mit der Eigenschaft Number des **Err**-Objekts identisch – er liefert also die Nummer des Fehlers, durch den die **Form_Error**-Ereignisprozedur ausgelöst wurde. Durch diesen Parameter können Sie auf die unterschiedlichen Fehler entsprechend reagieren.

- Der zweite Parameter namens **Response** erwartet in dieser Prozedur einen von zwei Werten als Rückgabewert: **acDataErrContinue** bedeutet, dass die normalerweise von Access an dieser Stelle angezeigte Meldung nicht erscheinen soll, **acDataErrDisplay** legt fest, dass Access die Fehlermeldung wie gewohnt anzeigen soll.

Der Standardwert für **Response** lautet **acDataErrDisplay**. Das bedeutet, dass ohne weiteres Zutun Ihrerseits nach dem Anlegen der Ereignisprozedur einfach nur wie üblich die Access-Meldung erscheint. Soll dies nicht geschehen, stellen Sie **Response** auf den Wert **acDataErrContinue** ein:

```
Response = acDataErrContinue
```

Dadurch wird nun die Fehlermeldung unterdrückt. Der Benutzer kann das Feld aber nicht verlassen, ohne einen gültigen Wert einzugeben. Die folgende Anweisung teilt ihm dies mit:

```
MsgBox "Bitte geben Sie einen " _
    & "Zahlenwert als Einzelpreis ein."
```

Dadurch wird nun eine entsprechende Meldung angezeigt. Den offensichtlich ungültigen Wert könnten Sie mit folgender Anweisung löschen:

```
Me!Einzelpreis.Undo
```

Fertig – wenn der Benutzer nun einen ungültigen Wert für das Feld **Einzelpreis** eingibt, erscheint eine benutzerdefinierte Meldung. Die eigentliche Meldung unterbleibt und das Feld wird zurückgesetzt.

Fehler differenzieren

Fast perfekt, oder? Nein, nicht ganz: Was geschieht denn, wenn ein anderer Fehler auftritt – beispielsweise beim Versuch, den Datensatz zu speichern, ohne das Pflichtfeld **Artikelname** zu füllen? Ganz einfach: Es erscheint genau die gleiche Fehlermeldung wie beim Eingeben eines Textes in das Währungsfeld **Einzelpreis**. Eigentlich sollte aber die Meldung erscheinen, die in Bild 3 abgebildet ist.

Moment – war da nicht was mit dem ersten Parameter **DataErr**? Der liefert doch für jeden Fehler eine entsprechende Fehlernummer! Also bauen wir diesen in die Ereignisprozedur **Form_Error** ein, um die jeweilige Fehlermeldung zu ermitteln.

Als erstes finden wir die Nummer des Fehlers heraus, der ausgelöst wird, wenn der Benutzer einen Text in ein numerisches Feld eingibt. Dazu tragen Sie als erste Zeile der Prozedur die folgende Anweisung ein:

```
MsgBox DataErr
```

Dann geben Sie **10 Euro** in das Feld **Einheitspreis** ein und erhalten per Meldungsfenster die Fehlernummer für diesen Fehler, nämlich **2113**. Damit können wir arbeiten und bauen ein **Select Case**-Konstrukt auf, dass diesen Fehler mit den oben bereits ermittelten Befehlen behandelt:

```
Select Case DataErr
    Case 2113 'Ungültige Zahl
        MsgBox "Bitte geben Sie einen Zahlenwert " _
            & "als Einzelpreis ein."
        Me!Einzelpreis.Undo
        Response = acDataErrContinue
    Case Else
        MsgBox DataErr
End Select
```

Die **MsgBox**-Anweisung verschieben Sie in den **Case Else**-Zweig, damit jeder weitere, nicht behandelte Fehler per **MsgBox** ausgeben wird, bevor seine eigentliche Fehlermeldung erscheint.

Leere Pflichtfelder

Beim Speichern des Artikel-Datensatzes mit nicht gefülltem Feld **Artikelname** löst dies beispielsweise den Fehler **3314** aus. Darauf reagieren wir ebenfalls mit einem eigenen **Select Case**-Zweig:

```
Case 3314 'Pflichtfeld leer
    MsgBox "Bitte geben Sie einen Wert für " _
        "'Artikelname' ein."
    Me!txtArtikelname.SetFocus
    Response = acDataErrContinue
```

Neben der Meldung und dem Deaktivieren der eingebauten Fehlermeldung setzt dies auch noch den Fokus auf das Steuerelement **txtArtikelname**, damit der Benutzer gleich weiß, in welchem Steuerelement etwas schiefgelaufen ist. Dies ist nötig, weil Restriktionen wie etwa das Einstellen der Eigenschaft **Eingabe erforderlich** für ein Feld auf den Wert **Ja** erst beim Speichern des kompletten Datensatzes wirken.

Aber ist dieser Ansatz sinnvoll? In diesem Falle ausnahmsweise ja, denn Artikelname ist das einzige Feld, das nicht leer bleiben darf. Wenn Sie jedoch noch ein weiteres Pflichtfeld in der Tabelle hätten, könnten Sie beim Auslösen des Fehlers 3314 beim Speichern eines unvollständigen Datensatzes nur mit weiterem Code-Einsatz ermitteln, welches Feld denn nun unerlaubterweise leer geblieben ist. Besser wäre es an dieser Stelle vermutlich, für das Ereignis **Vor Aktualisierung** eine entsprechende Prüfung unterzubringen. Diese könnte beispielsweise so aussehen:

```
Private Sub Form_BeforeUpdate(Cancel As Integer)
    If IsNull(Me!txtArtikelname) Then
        MsgBox "Bitte geben Sie einen Wert für " _
            & "'Artikelname' ein."
        Me!txtArtikelname.SetFocus
        Cancel = True
    End If
End Sub
```

Dummerweise wird das Ereignis **Bei Fehler** vor dem Ereignis **Vor Aktualisierung** ausgelöst. Das ist aber auch nicht schlimm – Sie sorgen einfach dafür, dass die eingebaute Meldung beim Speichern eines Datensatzes nicht angezeigt wird und überlassen der benutzerdefinierten Validierung die Prüfung, ob die Pflichtfelder gefüllt wurden.

Der **Case**-Zweig für diesen Fall sieht dann einfach so aus:

```
Case 3314 'Pflichtfeld leer
    Response = acDataErrContinue
```

Aber Vorsicht: Sie müssen dann in der Validierung alle Pflichtfelder prüfen und entsprechende Meldungen anzeigen, da der Benutzer sonst nicht erfährt, warum sein Datensatz nicht gespeichert werden kann.

Gültigkeitsregeln aushebeln

Wenn Sie einmal eine Gültigkeitsregel in der **Bei Fehler**-Ereignisprozedur ausheben wollen, können Sie dies auch tun. Die Tabelle **tblArtikel** besitzt eine Gültigkeitsregel, die verhindert, dass etwa für das Feld **Lagerbestand** negative Zahlen eingegeben werden. Die Verletzung einer Gültigkeitsregel löst den Fehler **3316** aus. Behandeln Sie diesen ebenfalls mit einem entsprechenden **Case**-Zweig:

```
Case 3316
    MsgBox "Bitte geben Sie eine positive Zahl " _
        "für den Lagerbestand ein."
    Me!txtLagerbestand.Undo
    Response = acDataErrContinue
```

Schuldiges Feld ermitteln

Manchmal können mehrere Felder für einen Fehler verantwortlich sein. In der Tabelle **tblArtikel** dürfen

Kombinationsfeld als Eingabehilfe für Textfelder

Kombinationsfelder haben Sie in Access [basics] bisher als Möglichkeit kennengelernt, Verknüpfungen zu Daten aus anderen Tabellen herzustellen. Das ist auch der meistgenutzte Anwendungszweck von Kombinationsfeldern. Wenn Sie den Benutzern Ihrer Anwendung jedoch das Leben erleichtern möchten, können Sie es auch an anderer Stelle einsetzen – beispielsweise, um bereits eingegebene Werte eines Feldes zur Auswahl anzubieten. Das ist vor allem hilfreich, wenn das Feld Werte enthält, die mehrfach vorkommen, aber dennoch nicht in eine Lookup-Tabelle ausgegliedert wurden.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1109_KombinationsfeldAlsEingabehilfe.mdb**.

Hilfe bei PLZ und Ort

Ein Beispiel für diesen Fall sind PLZ und Ort bei Adressdaten. Normalerweise gibt der Benutzer Postleitzahlen und Orte immer komplett ein – ohne jegliche Eingabehilfe. Was aber, wenn die einzugebenden Adressen, beispielsweise von Kunden, immer wieder gleich sind, weil die Kunden alle aus dem direkten Umkreis stammen? Da wäre es doch sinnvoll, dem Benutzer die eine oder andere Erleichterung zu bieten. In diesem Fall verwenden Sie einfach ein Kombinationsfeld statt eines Textfeldes, dass sowohl die Eingabe neuer Werte als auch die Auswahl bereits vorhandener Werte ermöglicht – und somit auch die automatische Ergänzung beim Eingeben der ersten paar Zeichen einer PLZ oder eines Ortes.

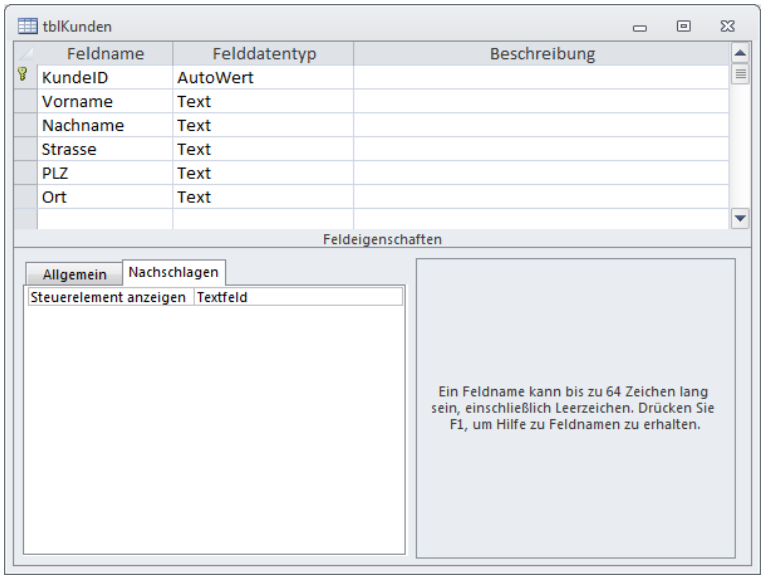


Bild 1: Beispieltabelle **tblKunden**

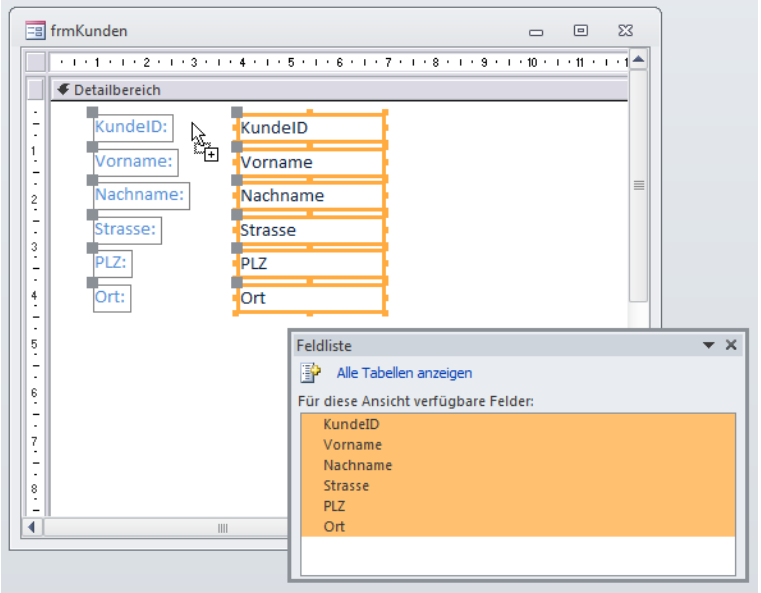


Bild 2: Hinzufügen der Felder der Tabelle **tblKunden** zum Formular **frmKunden**

Als Beispiel dienen die Felder **PLZ** und **Ort** der Tabelle **tblKunden** der Beispieldatenbank (siehe Bild 1).

Das Formular **frmKunden** verwendet diese Tabelle als Datenherkunft. Ziehen Sie in der Entwurfsansicht des Formulars einfach alle Felder aus der Feldliste in den Detailbereich. Sie erhalten dort zunächst eine Reihe Textfelder, auch für die beiden Felder **PLZ** und **Ort** (siehe Bild 2). Das ist kein Wunder, denn diese Felder sind ja im Entwurf der Tabelle nicht anders definiert worden. Damit sich dies ändert, klicken Sie zunächst mit der rechten Maustaste auf das Textfeld **PLZ**. Wählen Sie aus dem Kontextmenü den Eintrag **Ändern zu|Kombinationsfeld** aus. Führen Sie den gleichen Schritt für das Textfeld **Ort** aus. Benennen Sie die beiden Steuerelemente dann in **cboPLZ** und **cboOrt** um.

Optionsgruppen und Optionsfelder

Wenn sich für ein Feld einer Tabelle mehrere Werte einstellen lassen, liegen diese Werte meist in einer Lookup-Tabelle und werden per Nachschlagefeld beziehungsweise per Kombinationsfeld ausgewählt. Manchmal ist die Anzahl der verfügbaren Werte jedoch begrenzt und wird vor allem nicht erweitert. Und wenn die Werte dann auch noch übersichtlich im Formular dargestellt werden sollen, ist eine Optionsgruppe mit entsprechenden Optionsfeldern die richtige Wahl.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1109_Optionsgruppen.mdb**.

Wert auswählen

Kombinationsfelder, Listenfelder, Kontrollkästchen und Optionsgruppen sind allesamt Steuerelemente, mit denen Sie mindestens einen von mehreren Werten auswählen können – unabhängig davon, ob das Steuerelement an ein Feld einer Datenherkunft gebunden ist oder nicht. Während Kombinationsfelder meist zur Auswahl eines Wertes aus einer verknüpften Tabellen dienen, Sie mit Listenfeldern einen oder mehrere Werte markieren können und Kontrollkästchen meist Ja/Nein-Werte markieren, können Sie mit einer Optionsgruppe und den enthaltenen Optionsfeldern einen Wert aus einer limitierten Anzahl von Werten auslesen.

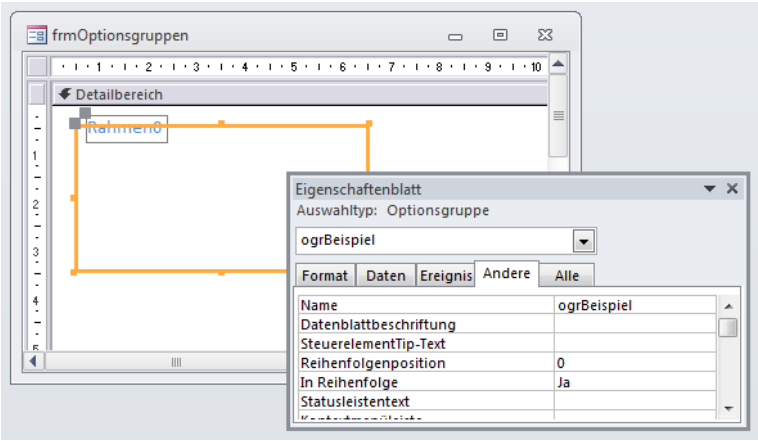


Bild 1: Einstellen des Namens einer frisch angelegten Optionsgruppe

Da die Optionsfelder einer Optionsgruppe statisch im Entwurf eines Formulars angelegt werden, kommen Sie nicht für die Auswahl von Daten infrage, die sich im Laufe der Zeit ändern können, weil Daten hinzukommen oder wegfallen. Sie dienen eher der Festlegung von Werten wie Klein/Mittel/Groß, von Farben, alternativ zum Kontrollkästchen für Ja/Nein-Werte und Artverwandte oder auch für andere Eigenschaften, für die zwei oder mehr Werte zur Auswahl stehen. Eine Optionsgruppe kann an ein Feld einer Tabelle gebunden sein oder auch für sich allein stehen – etwa, um ein zu öffnendes Formular oder einen Bericht auszuwählen, dass danach per Schaltfläche geöffnet wird.

Ungebundene Optionsgruppen

Im ersten Beispiel legen wir einfach eine Optionsgruppe mit drei Optionsfeldern an, um die wichtigsten Eigenschaften dieser Steuerelemente kennenzulernen. Erstellen Sie dazu ein neues Formular in der Entwurfsansicht und fügen Sie diesem zunächst eine Optionsgruppe hinzu. Dazu klicken Sie in der Liste der Steuerelemente auf das Optionsgruppe-Steuerelement und ziehen dann im Entwurf des Formulars einen Rahmen mit der gewünschten Größe auf.

Die Optionsgruppe hat zunächst eine wichtige Eigenschaft: den Namen (siehe Bild 1). Diesen stellen Sie auf einen aussagekräftigen Wert ein (in diesem Fall **ogrBeispiel**). Der Name ist nicht zu verwechseln mit der Beschriftung: Diese kommt mit einem angehefteten Bezeichnungsfeld. Auch diese sollten Sie direkt anpassen, indem Sie zuerst auf das Bezeichnungsfeld klicken und dann seine Eigenschaft **Beschriftung** ändern. Den Namen des Bezeichnungsfeldes müssen Sie indes nicht anpassen, Sie werden im Normalfall nicht per VBA darauf zugreifen.

Optionsfelder hinzufügen

Wenn Sie nun in der Liste der Steuerelemente auf ein Optionsfeld klicken und dann mit der Maus über die Optionsgruppe fahren, wissen Sie, wann Sie loslassen müssen – nämlich dann, wenn der Hintergrund der Optionsgruppe schwarz wird (siehe Bild 2). Achtung: Wenn Sie das Optionsfeld irgendwo außerhalb der Optionsgruppe platzieren und es später in die Optionsgruppe ziehen, wird es nicht mit der Optionsgruppe verknüpft, sondern steht für sich allein! Die

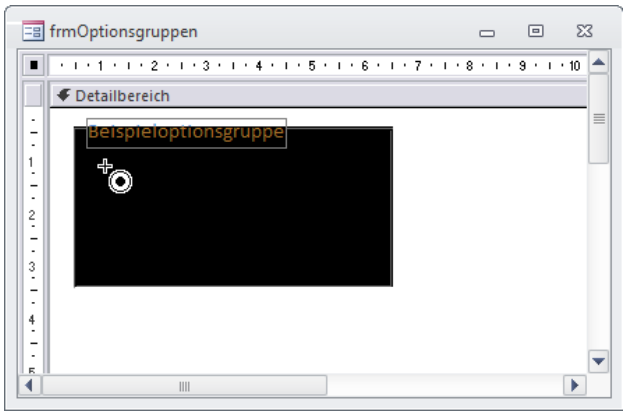


Bild 2: Hinzufügen eines Optionsfeldes zu einer Optionsgruppe

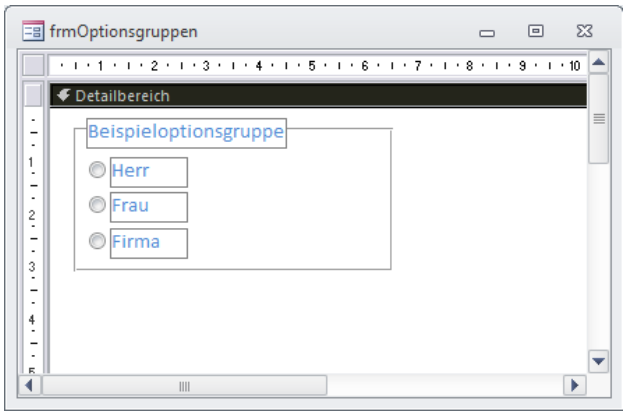


Bild 3: Hinzufügen weiterer Optionsfelder

einzige Möglichkeit, ein solches Optionsfeld noch zur Optionsgruppe hinzuzufügen, ist das Ausschneiden (zum Beispiel mit **Strg + X**), Markieren der Optionsgruppe und Einfügen (**Strg + V**). Fügen Sie auf diese Weise drei Optionsfelder zur Optionsgruppe hinzu. Die Optionsfelder bestehen wiederum aus dem eigentlichen Steuerelement und einem daran gebundenen Bezeichnungsfeld.

Stellen Sie die Beschriftung der drei Optionsfelder auf **Herr**, **Frau** und **Firma** ein (siehe Bild 3). Die Beschriftungen sind, bezogen auf die Funktion der angelegten Steuerelemente, Schall und Rauch. Sie dienen lediglich dazu, dem Benutzer zu zeigen, welche Bedeutung die einzelnen Optionen haben. Tatsächlich wichtig ist der Wert, den Sie für die Eigenschaft **Wert** der drei Optionsfelder vergeben. Wenn Sie die Optionsfelder alle einzeln über die Steuerelement-Liste angelegt haben, hat Access die Eigenschaft **Wert** automatisch vorbelegt, und zwar mit den Werten **1**, **2** und **3**. Bild 4 zeigt die Vorbelegung dieser Eigenschaft mit dem Wert **1**. Sie können die Optionsfelder auch hinzufügen, indem Sie das erste auf herkömmlichem Wege über die Steuerelement-Liste beziehen und dieses dann entsprechend der benötigten Anzahl von Optionsfeldern kopieren. Diese Vorgehensweise zaubert zwar schnell einige Optionsfelder in die Optionsgruppe, übernimmt aber leider auch den Wert des Originaloptionsfeldes. Das heißt, dass Sie die Werte der übrigen Felder später manuell anpassen müssen. Das ist natürlich kein Problem, wenn Sie ohnehin andere Werte als **1**, **2**, **3** und so weiter benötigen.

Standardwert festlegen

Beim Öffnen des Formulars soll üblicherweise gleich eine Option ausgewählt sein. Dies erreichen Sie, indem Sie für die Eigenschaft **Standardwert** des Optionsgruppen-Steuerelements einen der für die Optionsfelder vergebenen Werte eintragen. Im Beispiel aus Bild 5 handelt es sich um das oberste Optionsfeld mit dem Wert **1**. Wenn Sie keinen Standardwert auswählen und der Benutzer keine Option auswählt, hat die Optionsgruppe den Wert **Null**. Da dies in den wenigsten Fällen erwünscht, sollten Sie dem mit einer entsprechenden Validierung oder der Vorgabe eines Standardwerts vorbeugen.

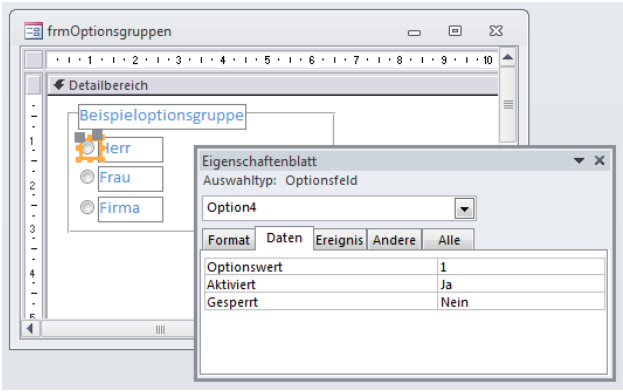


Bild 4: Wert eines der Optionsfelder

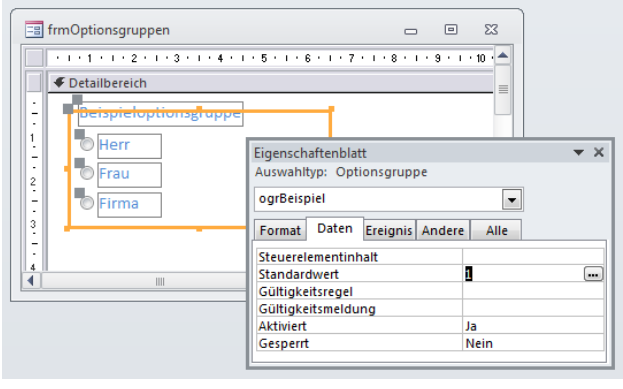


Bild 5: Standardwert der Optionsgruppe festlegen

Wert auslesen

Auf den Wert einer Optionsgruppe greifen Sie über die **Wert**-Eigenschaft der Optionsgruppe zu. Wenn Sie testweise den aktuellen Wert der Optionsgruppe sehen möchten, fügen Sie ein Textfeld zum Formular hinzu und stellen seine Eigenschaft **Steuerelementinhalt** auf den folgenden Wert ein:

```
=[ogrBeispiel].[Wert]
```

Da die Eigenschaft **Wert** die Standardeigenschaft dieses Steuerelements ist, reicht auch der folgende Ausdruck:

```
=[ogrBeispiel]
```

Wenn Sie nun in die Formularansicht wechseln, zeigt das Textfeld direkt nach dem Auswählen jeweils den Wert des aktuell ausgewählten Optionsfeldes an (siehe Bild 6).

Per VBA auf eine Änderung reagieren

Wenn Sie beim Ändern der ausgewählten Option eine VBA-Prozedur starten möchten, die eine Aktion auf Basis der ausgewählten Option durchführt, ist das auch kein Problem. Markieren Sie die Optionsgruppe in der Entwurfsansicht des Formulars und tragen Sie für die Eigenschaft **Nach Aktualisierung** den Wert **[Ereignisprozedur]** ein. Klicken Sie dann auf die Schaltfläche mit den drei Punkten rechts neben der Eigenschaft und ergänzen Sie die nun im VBA-Editor erscheinende Prozedur wie folgt:

```
Private Sub ogrBeispiel_AfterUpdate()
    MsgBox "Der aktuelle Wert der Optionsgruppe ⤵
                lautet " & Me!ogrBeispiel & "."
End Sub
```

Der Wert der Optionsgruppe wird hier einfach durch das Referenzieren des entsprechenden Steuerelements über seinen Namen realisiert (**Me!ogrBeispiel**), der Wert in einem Meldungsfenster ausgeben. Wenn Sie lieber die für das Optionsfeld angegebene Beschriftung ausgeben möchten, kommen Sie um ein wenig Schreibarbeit nicht herum. Das Ergebnis sieht so aus:

```
Private Sub ogrBeispiel_AfterUpdate()
    Select Case ogrBeispiel
        Case 1
            MsgBox "Gewählter Wert: Herr"
        Case 2
            MsgBox "Gewählter Wert: Frau"
        Case 3
            MsgBox "Gewählter Wert: Firma"
    End Select
End Sub
```

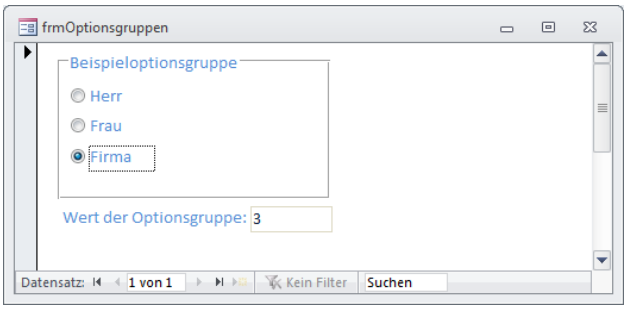


Bild 6: Ausgabe des aktuellen Werts einer Optionsgruppe

Gebundene Optionsgruppen

Sie können Optionsgruppen auch an ein Feld der Datenherkunft eines Formulars binden. Die einzelnen Optionsfelder der Optionsgruppe repräsentieren dann die Werte, die das zugrunde liegende Feld annehmen kann. Im Beispiel wollen wir die Anrede eines Kunden auswählen. Das Formular binden Sie an die Tabelle **tblKunden** der Beispieldatenbank. Die Beispieldatenbank besitzt eine weitere Tabelle namens **tblAnreden**. Diese enthält die Lookupdaten für das Feld **AnredeID** der Tabelle **tblKunden**, die wir mit der Optionsgruppe abbilden wollen (siehe Bild 7). Die Tabelle **tblAnreden** enthält die drei Werte **Herr**, **Frau** und **Firma**, denen die Primärschlüsselwert **1**, **2** und **3** zugewiesen wurden. Diese werden gleich noch benötigt! Erstens soll natürlich in das Fremdschlüsselfeld **AnredeID** der Tabelle **tblKunden** einer der drei Werte **1**, **2** oder **3** eingetragen werden, um dem Kunden die entsprechende Anrede zuzuweisen. Zweitens müssen wir die Optionsfelder der Optionsgruppe mit genau diesen Werten ausstatten. Legen Sie also ein neues Formular an und legen Sie die Tabelle **tblKunden** als Datenherkunft fest. Ziehen Sie die drei Felder

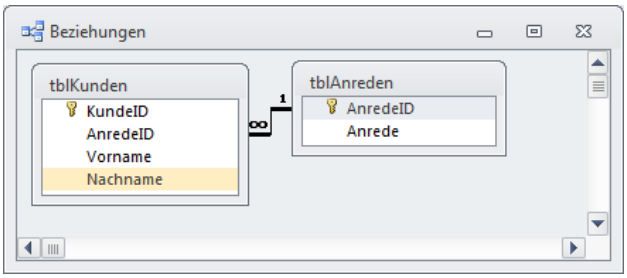


Bild 7: Beispieltabellen zur Demonstration einer gebundenen Optionsgruppe

Berichtsbereiche

Ein Bericht bietet einige Bereiche mehr als ein Formular. Damit können Sie auch komplexere Daten strukturiert in einem Bericht darstellen. Neben den Berichtsbereichen, die nur einmal pro Bericht angezeigt werden (Berichtskopf und -fuß) sowie den seitenweise erscheinenden Bereichen (Seitenkopf und -fuß) gibt es noch die Kopf- und Fußbereiche der einzelnen Gruppierungen - um diese kümmert sich ein weiterer Artikel. Dieser Artikel stellt die einzelnen Bereiche vor und erklärt, was Sie bei deren Einsatz beachten müssen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1109_Berichtsbereiche.mdb**.

Standardbericht

Wenn Sie einen neuen Bericht in der Entwurfsansicht öffnen, zeigt dieser standardmäßig drei Bereiche an: den Seitenkopf, den Detailbereich und den Seitenfuß (siehe Bild 1). Wenn Sie diesen Bereichen einen Text hinzufügen, um diese später identifizieren zu können, sieht die Seitenansicht wie in Bild 2 aus. Die wichtigste Erkenntnis, die wir hier gewinnen ist die folgende: Der Seitenfuß wird immer ganz unten angeordnet. Es spielt keine Rolle, wieviele Detaildatensätze sich auf der Seite befinden, er rutscht niemals nach oben.

Damit Sie die einzelnen Berichtsbereiche unterscheiden können, haben wir Bezeichnungsfelder mit dem jeweiligen Bereichsnamen integriert. Wenn Sie später Überschriften et cetera zum Berichts- und/oder Seitenkopf hinzufügen, verwenden Sie am besten ebenfalls Bezeichnungsfelder.

Bereiche ein- und ausblenden

Vielleicht brauchen Sie Seitenkopf und -fuß nicht, oder Sie benötigen zusätzlich einen Berichtskopf und -fuß: Dann blenden Sie diese Bereiche am einfachsten über das Kontextmenü eines der Bereichsköpfe ein oder aus (siehe Bild 3). Diese Bereiche können Sie auch über entsprechende Einträge der Menüleiste (Access 2003 und älter) beziehungsweise des Ribbons (Access 2007 und jünger) beein-

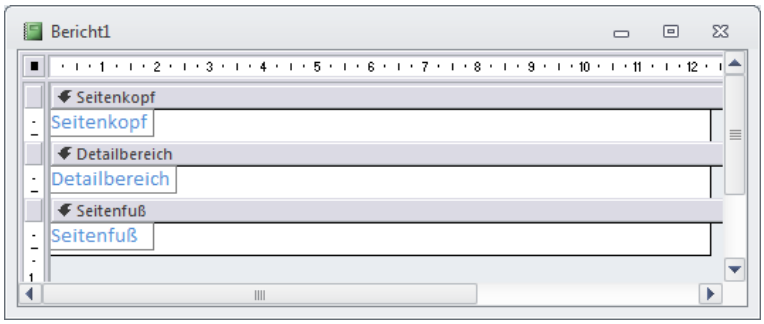


Bild 1: Standardmäßig angezeigte Berichtsbereiche

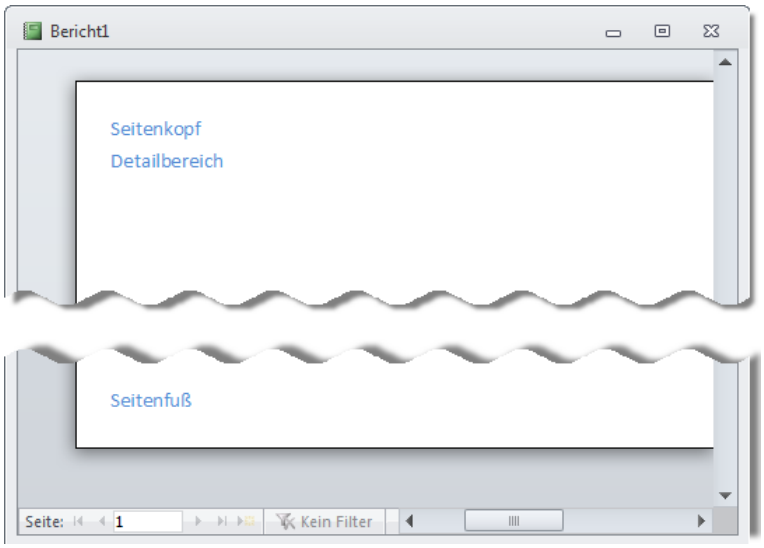


Bild 2: Berichtsbereiche in der Seitenansicht

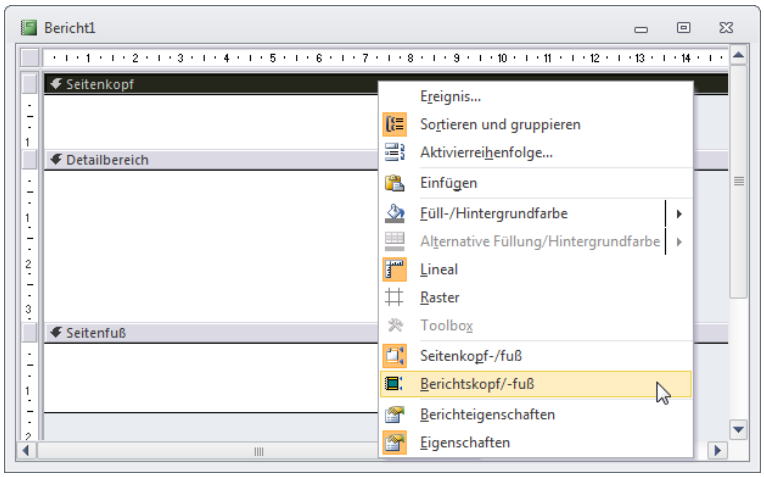


Bild 3: Ein- und Ausblenden von Berichtsbereichen

Sortieren in Berichten

In Tabellen stellen Sie die Sortierung in der Datenblattansicht ein, in Abfragen legen Sie diese im Entwurfsraster fest und in Formularen und den enthaltenen Steuerelementen wie Kombinationsfeldern und Listenfeldern gibt die für die Datenherkunft festgelegte Reihenfolge den Takt vor. Wie aber sieht es in Berichten aus? Um es vorweg zu nehmen: ganz anders. Dieser Artikel erläutert, wie Sie die in Berichten anzuzeigenden Daten sortieren.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1109_SortierenInBerichten.mdb**.

Sortieren und ...

Es ist sicher einer der weitverbreitetsten Anfängerfehler: Sie stellen die in einem Bericht anzuzeigenden Daten mithilfe einer entsprechenden Abfrage zusammen und weisen diese dem Bericht als Datenherkunft zu. Sie füllen den Bericht mit Steuerelementen, wechseln in die Seitenansicht und ... was ist das? Warum sortiert der Bericht die Daten nicht wie in der zugrunde liegenden Abfrage vorgegeben?

Der Grund für dieses Verhalten ist: Berichte haben einen eigenen Dialog, mit dem Sie die Reihenfolge der anzuzeigenden Daten festlegen. Die Sortierreihenfolge der als Datenherkunft verwendeten Tabellen oder Abfragen wird unter Umständen schlicht ignoriert, auch wenn Sie im Bericht gar keine explizite Sortierreihenfolge angeben. Das bedeutet: Es kommt vor, dass die in der Abfrage angegebene Sortierreihenfolge berücksichtigt wird, aber dies geschieht nicht zuverlässig. Das ist aber auch kein Problem, denn Berichte bringen ihre eigene Funktion zum Festlegen der Sortierung fest, ja sogar einen eigenen Dialog.

Diesen öffnen Sie am einfachsten, indem Sie in der Entwurfsansicht des Berichts mit der rechten Maustaste auf einen der Bereichsköpfe klicken und den Eintrag **Gruppieren und sortieren** aus dem Kontextmenü auswählen (siehe Bild 1).

Es erscheint der Dialog aus Bild 2, mit dem Sie die Sortierreihenfolge einstellen

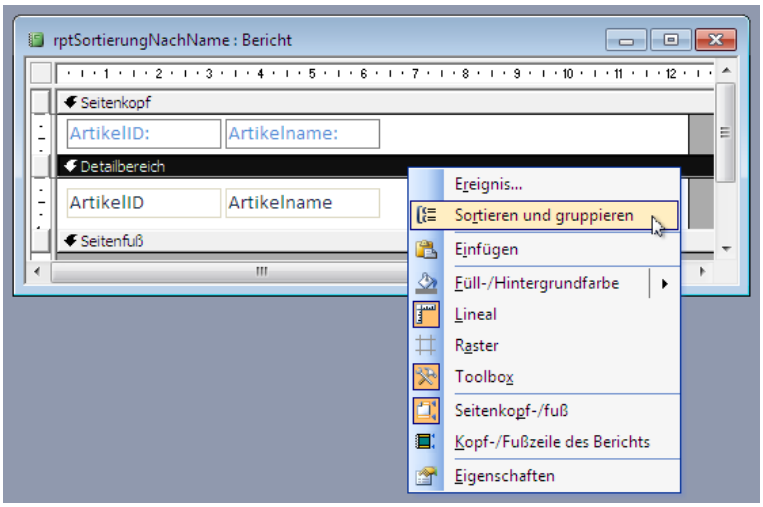


Bild 1: Aufruf des Dialogs zum Angeben einer Sortierung für die Daten des Berichts

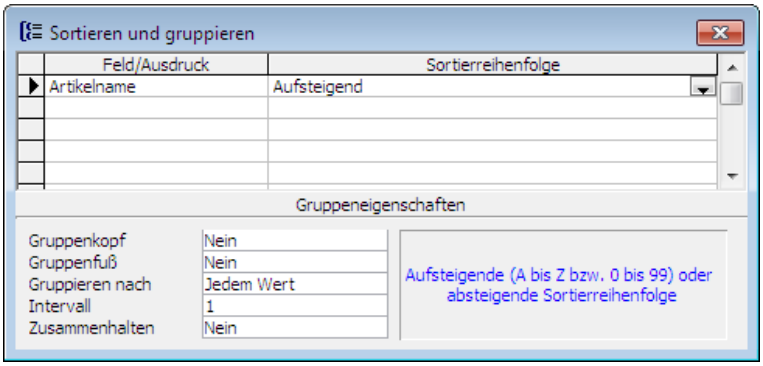


Bild 2: Angeben der Sortierreihenfolge eines Berichts

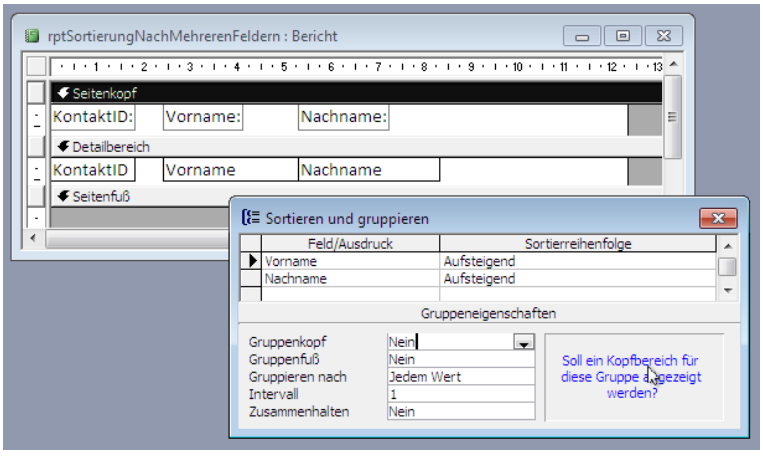


Bild 3: Bericht mit zweifacher Sortierung

weitere Befehle, unter anderem auch **Exklusiv öffnen** (variiert mit den verschiedenen Versionen).

Nach dem Schließen und erneuten Öffnen der Datenbank zeigt diese eine Inputbox zur Eingabe des Kennwort an.

Die Funktionen zum Ändern oder Entfernen des Kennworts finden Sie an gleicher oder ähnlicher Stelle wie die zum Anlegen des Kennworts.

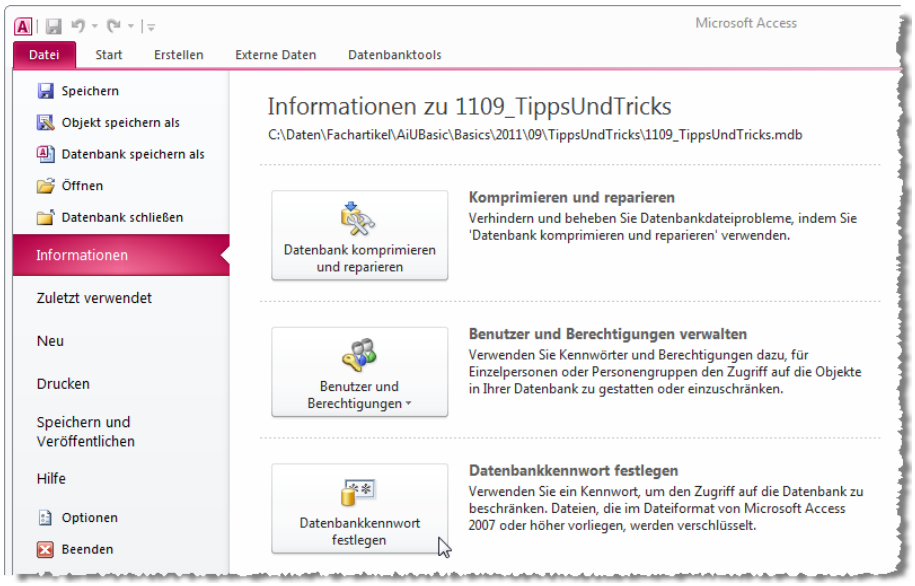


Bild 10: Festlegen des Datenbank-Kennworts unter Access 2010

Textbausteine verwenden

Wenn Sie zum Beispiel Anschreiben oder ähnliche Textdokumente mit Access als Bericht zusammenfügen, werden Sie diese Texte gelegentlich auch manuell eintippen. Dabei wäre es hilfreich, Standardfloskeln wie **Mit freundlichen Grüßen** et cetera durch Abkürzungen wie **mfg** in das entsprechende Textfeld eingeben zu können.

Das notwendige Werkzeug heißt **Autokorrektur** und wird von allen Office-Anwendungen eingesetzt, auch in Access. Das bedeutet, dass Eingaben in Textfeldern von der Autokorrektur geprüft und gegebenenfalls korrigiert werden.

Wenn Sie also etwa einen Punkt im Anschluss an ein Wort eintippen und das nächste Wort klein beginnen, wird die Autokorrektur aktiv. Sie korrigiert den vermeintlich fehlerhaften Text automatisch und teilt dies durch ein kleines Symbol mit. Mit einem Klick auf die beim Überfahren mit dem Mauszeiger erscheinende Schaltfläche zeigen Sie die Optionen der soeben durchgeführten Korrektur an (siehe Bild 11).

Mit einem Klick auf den Eintrag **Autokorrektur-Optionen steuern** öffnen Sie den Dialog aus Bild 12. Wenn Sie die Autokorrektur schlicht nicht mögen, deaktivieren Sie einfach alle Optionen dieses Dialogs.

Wenn Sie die Autokorrektur hingegen nutzen möchten, um Kürzel wie **mfg** automatisch in Texte wie **Mit freundlichen Grüßen** umzuwandeln, können Sie dies leicht bewerkstelligen. Geben Sie einfach unter **Ersetzen:** den Text **mfg** ein und unter **Durch:** den Text

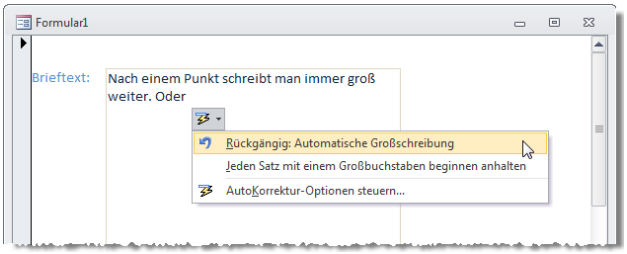


Bild 11: Die Autokorrektur in Aktion

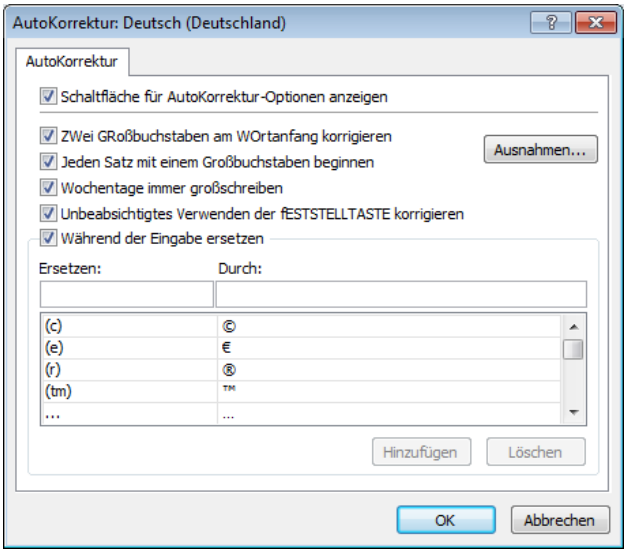


Bild 12: Optionen der Autokorrektur

Mit freundlichen Grüßen und klicken Sie auf **Hinzufügen** – schon ist die neue Floskel gespeichert.

Geben Sie diesen Text nun in ein Textfeld eines Access-Formulars ein, ersetzt Access diesen Text automatisch.

Datenblattansicht im Griff

Die Datenblattansicht ist die flexibelste Ansicht, wenn es darum geht, Daten tabellarisch anzuzeigen. Nicht nur der Entwickler, sondern auch der Benutzer kann die Daten mit dieser Ansicht nach Belieben darstellen. Dieser Artikel stellt alle Möglichkeiten der Datenblattansicht unter Access vor.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1110_Datenblattansicht.mdb**.

Datenblätter überall

Die Datenblattansicht können Sie gleich für die Darstellung dreier Objekttypen verwenden: Tabellen, Abfragen und Formulare. Gleichwohl sollten Sie spätestens bei Übergabe an den Anwender sicherstellen, dass dieser nur über Formulare auf die Daten zugreift und nicht direkt über Tabellen oder Abfragen – auf diese Weise nehmen Sie sich selbst die Möglichkeit, Features wie etwa eine Validierung zu nutzen. Wir konzentrieren uns hier also auf Formulare in der Datenblattansicht – und hier speziell auf Unterformulare.

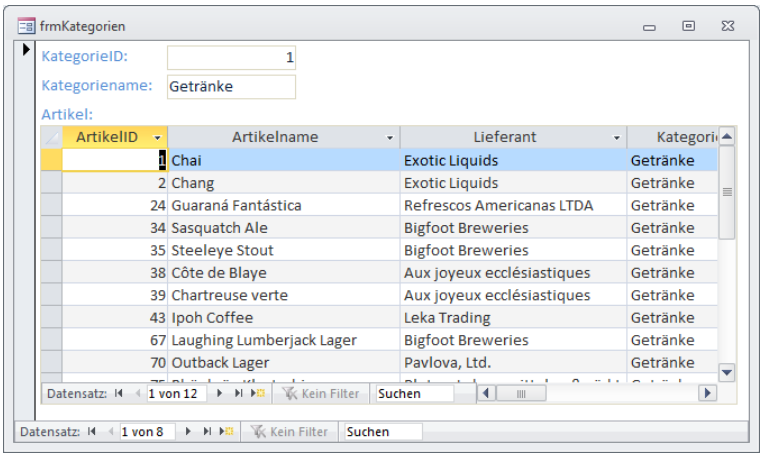


Bild 1: Unterformular in der Datenblattansicht

Unterformular als Datenblatt

Die Darstellung von Daten aus 1:n-Beziehungen erfolgt oft in Haupt- und Unterformular, wobei das Unterformular die verknüpften Daten in der Datenblattansicht anzeigt. Die folgenden Beispiele beziehen sich auf die beiden Formulare **frmKategorien** und **sfmArtikel**, wobei das Unterformular **sfmArtikel** die Tabelle **tblArtikel** als **Datenherkunft** verwendet. Außerdem besitzt die Eigenschaft **Standardansicht** des Unterformulars den Wert **Datenblatt** (siehe Bild 1).

Features der Datenblattansicht

Die Datenblattansicht zeigt Daten in tabellarischer Form an, also in je einer Spalte pro Feld. Sie können damit folgende Aktionen durchführen:

- Ändern der Reihenfolge der Spalten/Felder
- Spaltenbreite und Zeilenhöhe einstellen
- Felder ein-/ausblenden
- Sortieren nach Spalten/Feldern
- Filtern nach Werten in Spalten/Feldern
- Fixieren der ersten n Spalten
- Ändern des Aussehens (Schriftart und -größe)
- Ändern der Hintergrundfarbe
- Ändern des Gitterrasters
- Einstellen alternierender Hintergrundfarben (ab 2007)
- Bedingte Formatierung festlegen

Schriftart und -größe

Unter Access 2007 und jünger ist die Schriftgröße mit 11 Punkten für viele Zwecke etwas zu groß eingestellt, also schauen wir uns zunächst an, wie wir dies ändern – und vielleicht haben Sie auch in älteren Access-Versionen das Bedürfnis, die Schrift anzupassen.

- Access 2003 und älter zeigen beim Aktivieren des Datenblatts eine Symbolleiste an, mit der Sie die Schrift einstellen können.
- Unter Access 207 und jünger finden Sie im Ribbon-Tab **Start** einen Bereich, mit dem Sie die Schrifteigenschaften einstellen können. Sie müssen lediglich das Datenblatt aktivieren und dann die gewünschten Änderungen durchführen.

Im Gegensatz zu älteren Versionen, wo Änderungen automatisch übernommen werden, fragen Access 2007 und jünger nach Änderungen der Schriftart beim Speichern, ob die Änderungen gespeichert werden sollen. Bei **.mde**- und **.accde**-Datenbanken werden gar keine Änderungen am Entwurf gespeichert, also auch keine Einstellungen des Datenblatts.

Spaltenbreite einstellen

Access stellt die Spaltenbreiten in der Datenblattansicht nicht automatisch immer so ein, dass alle Texte komplett lesbar sind. Sie können die Spaltenbreiten jedoch manuell einstellen:

- Klicken Sie auf den schmalen vertikalen Strich zwischen zwei Spalten und ziehen Sie diesen bei gedrückter Maustaste nach links oder rechts (siehe Bild 2).

- Klicken Sie doppelt auf diesen Strich, um automatisch die optimale Spaltenbreite bezogen auf alle aktuell sichtbaren Inhalte einzustellen.

- Klicken Sie doppelt auf den Kopf der anzupassenden Spalte und wählen Sie aus dem Kontextmenü den Eintrag **Feldbreite** aus. Es erscheint ein Dialog, in den Sie die Breite eingeben können.

All diese Varianten können Sie auch nach vorherigem Markieren mehrerer zusammehängender Spalten (etwa durch Anklicken der Spalten bei gedrückter Umschalttaste) durchführen. Es werden dann alle Spalten angepasst, die Sie zuvor markiert haben. Interessanterweise erfolgt etwa beim Einstellen von Spaltenbreiten keine Rückfrage, ob die Änderungen gespeichert und beim nächsten Öffnen wieder hergestellt werden sollen.

Zeilenhöhe einstellen

Die Zeilenhöhe stellen Sie genauso ein, nur dass Sie dabei den horizontalen Strich ganz links im Bereich des Datensatzmarkierers zwischen zwei Zeilen nach oben oder unten verschieben. Ein wichtiger Unterschied zum Einstellen der Spaltenbreite ist: Sie stellen damit immer gleichzeitig die Höhe aller Zeilen ein, nicht für einzelne Datensätze.

Felder ein-/ausblenden

Wenn Sie mit der rechten Maustaste auf eine oder mehrere zuvor markierte Spalten klicken, erscheint

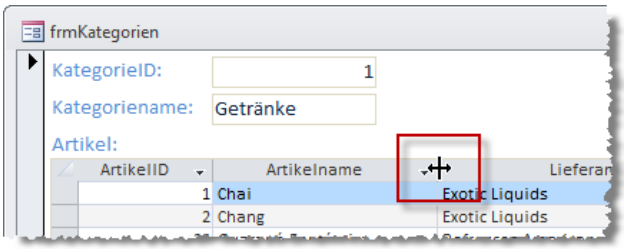


Bild 2: Spaltenbreite einstellen

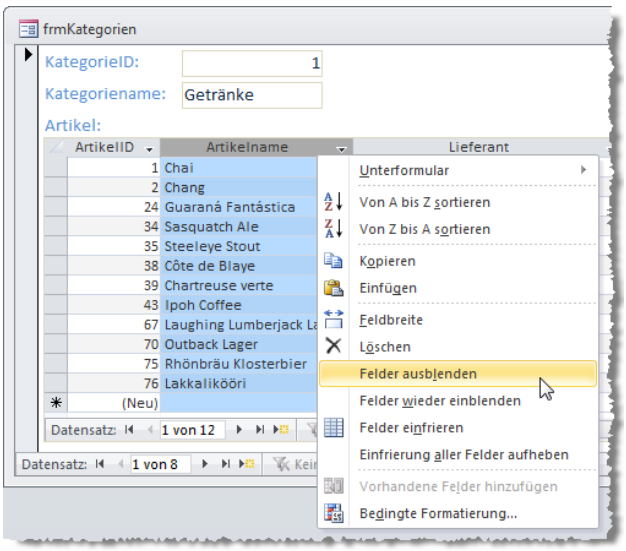


Bild 3: Spalten ausblenden

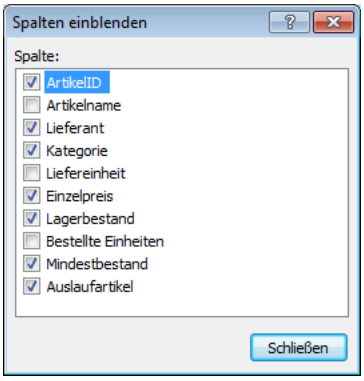


Bild 4: Spalten ein- und ausblenden

das Kontextmenü und zeigt unter anderem den Befehl **Felder ausblenden** an. Mit diesem Befehl blenden Sie die aktuell markierten Felder aus (siehe Bild 3).

Um diese wieder einzublenden, wählen Sie den Befehl **Felder wieder einblenden** des Kontextmenüs aus.

Es erscheint der Dialog **Spalten einblenden** (siehe Bild 4). Genaugenommen können Sie damit auch Spalten ausblenden – komfortabler sogar als über den entsprechenden Befehl des Kontextmenüs.

Aktuellen Datensatz kopieren

Es gibt verschiedene Gelegenheiten, zu denen Sie den aktuell im Formular angezeigten Datensatz kopieren möchten. Meist soll die Kopie des Datensatzes als Ausgangspunkt für einen neuen Datensatz dienen, damit Sie diesen nicht komplett neu anlegen müssen, sondern nur noch die gewünschten Felder anzupassen brauchen. Es gibt erstaunlich viele Techniken, wie Sie dies durchführen können. Dieser Artikel stellt eine einfache Variante vor.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1110_Datensatzkopieren.mdb**.

Handarbeit

Wann immer die Tabelle, die Abfrage oder das Formular mit dem zu duplizierenden Datensatz einen Datensatzmarkierer anbietet, können Sie den folgenden einfachen Weg beschreiten:

- Klicken Sie links auf den Datensatzmarkierer, um den kompletten Datensatz zu markieren (siehe Bild 1).
- Kopieren Sie den Datensatz – weiter unten erfahren Sie, wie das funktioniert.
- Klicken Sie auf den Datensatzmarkierer des neuen Datensatzes ganz unten in der Tabelle, der Abfrage oder dem Formular (siehe Bild 2).
- Fügen Sie den Inhalt der Zwischenablage, also den kopierten Datensatz, ein.

Fertig – der neue Datensatz enthält ein Duplikat des vorherigen Datensatzes. Der einzige Unterschied ist ein neuer Wert im Primärschlüsselfeld. Das gilt aber auch nur, wenn dieser als Autowertfeld ausgelegt wurde – sonst meldet Access beim Versuch, den Datensatz zu verlassen, einen Fehler wegen eines bereits vorhandenen Wertes im Primärschlüsselfeld.

Zum Kopieren eines bereits markierten Datensatzes stehen unter Access zum Beispiel die folgenden Methoden bereit:

- Betätigen der Tastenkombination **Strg + C**,
- Auswählen des Kontextmenü-Eintrags **Kopieren** oder

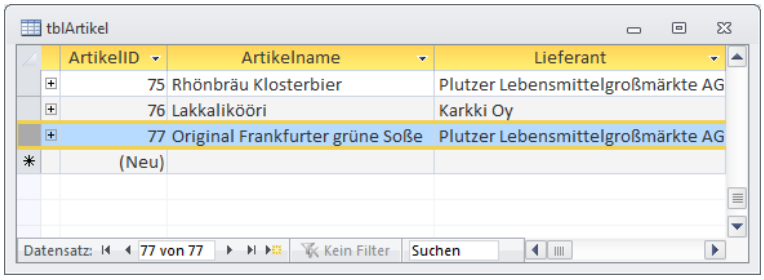


Bild 1: Markieren eines kompletten Datensatzes

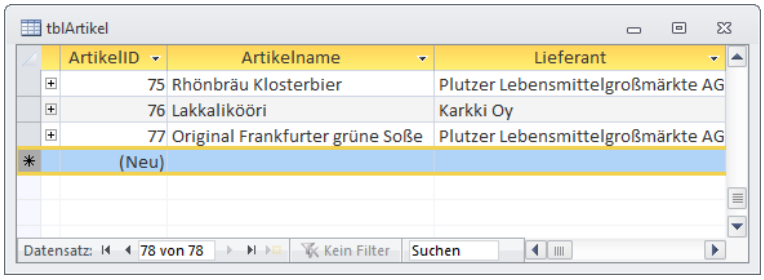


Bild 2: Markieren des Zieldatensatzes, in diesem Fall eines neuen Datensatzes

- Anwenden des entsprechenden Eintrags der Menü-/Symbolleiste namens **Bearbeiten|Kopieren** (unter Access 2003 und älter) oder des Ribbon-Befehls **Start|Zwischenablage|Kopieren** (unter Access 2007 und jünger).

Zum Einfügen des Datensatzes aus der Zwischenablage verwenden Sie entsprechend

- die Tastenkombination **Strg + V**,
- den Kontextmenü-Eintrag **Einfügen** oder
- den Menübefehl **Bearbeiten|Einfügen** (Access 2003 und älter) beziehungsweise den Ribbon-Eintrag **Start|Zwischenablage|Einfügen** (Access 2007 und jünger).

Datensatz per Mausklick kopieren

Wenn Ihnen das zu kompliziert ist oder Sie für den Benutzer Ihrer Datenbank eine einfachere Lösung benötigen, programmieren Sie einfach ein paar Zeilen VBA-Code.

Formulare und Steuerelemente referenzieren

Je mehr Sie VBA zur Steuerung von Formularen und Steuerelementen einsetzen, desto öfter greifen Sie dabei per Code auf die Inhalte oder Eigenschaften dieser Elemente zu – beispielsweise, um den Wert eines Feldes auszulesen oder einen Eintrag eines Kombinationsfeldes auszuwählen. Dieser Artikel liefert eine Übersicht, wie Sie von verschiedenen Stellen aus Formulare und Steuerelemente referenzieren.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1110_FormulareReferenzieren.mdb**.

Global referenzieren

Wir beginnen mit den Verweisen, die Sie von beliebigen Stellen aus nutzen können. Damit ist das Testen für den Anfang am einfachsten: Sie können die folgenden Anweisungen einfach in das Direktfenster des VBA-Editors eingeben, dass Sie mit **Strg + G** sichtbar machen.

Beginnen wir mit dem nackten Formular. Ein Formular referenzieren Sie mit einem Verweis auf ein Element der **Forms**-Auflistung.

Wenn Sie beispielsweise den Namen eines Formulars ermitteln möchten, können Sie die folgende Anweisung absetzen:

```
Debug.Print Forms.Item("frmBeispiel").Name
```

Kürzer und gebräuchlicher sind jedoch diese Versionen (siehe Bild 1):

```
Debug.Print Forms("frmBeispiel").Name
Debug.Print Forms!frmBeispiel.Name
```

Mit der Ausrufezeichen-Syntax greifen Sie auf alle Elemente von Auflistungen zu, die benutzerdefiniert sind, die Sie also selbst erstellt haben. Das gilt im Folgenden auch für Steuerelemente.

Wenn das Formular gerade das einzige geöffnete Formular ist, können Sie auch diese Syntax nutzen:

```
Debug.Print Forms(0).Name
```

Beachten Sie, dass dies immer funktioniert, wenn das betroffene Formular geöffnet ist – egal, in welcher Ansicht. Sie können also auch Eigenschaften von Formularen in der Entwurfsansicht über diese Anweisung ausgeben.

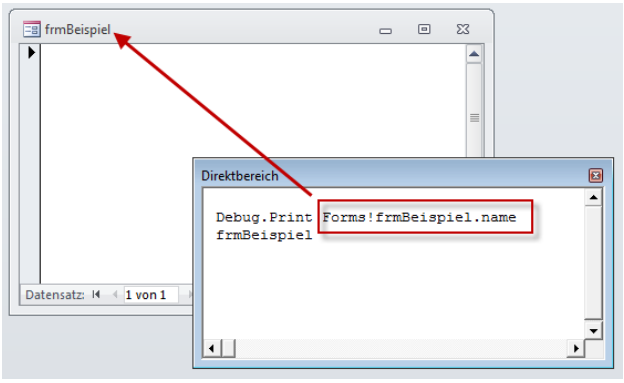


Bild 1: Referenzieren eines Formulars

Beispiel: Filter

Sie können so von außen, also vom Direktfenster oder auch von einem anderen Formular aus, Eigenschaften lesen und einstellen. Wenn Sie beispielsweise einen Filter für ein Formular festlegen möchten, verwenden Sie diese beiden Anweisungen:

```
Forms!frmBeispiel.Filter = "Artikelname = 'Chang'"
Forms!frmBeispiel.FilterOn = True
```

Um den Filter zu deaktivieren, stellen Sie die Eigenschaft **Filter** auf den ursprünglichen Wert ein, also eine leere Zeichenkette:

```
Forms!frmBeispiel.Filter = ""
```

FilterOn wird dabei automatisch auf **False** eingestellt, was sich leicht prüfen lässt:

```
? Forms!frmBeispiel.FilterOn
Falsch
```

Referenz speichern

Wenn Sie wie im Beispiel oben mehrfach auf ein referenziertes Objekt zugreifen, können Sie die Referenz auch in einer Variablen des entsprechenden Objekttyps speichern und auf die Variable zugreifen, statt immer wieder neu zu referenzieren. Für ein Formular deklarieren Sie die Variable wie folgt:

```
Dim frm As Form
```

Sobald Sie mit Variablen arbeiten, fügen Sie die Anweisungen praktischerweise in eine Prozedur ein, die Sie wie in folgendem Beispiel in einem Standardmodul anlegen (**mdlFormReferenzen**):

```
Public Sub FormReferenz()
    Dim frm As Form
    Set frm = Forms!frmBeispiel
    frm.Filter = "Artikelname = 'Chang'"
    frm.FilterOn = True
End Sub
```

Sich selbst referenzieren

Wenn Sie innerhalb des Klassenmoduls eines Formulars auf Formular-Eigenschaften zugreifen möchten, können Sie dies ebenfalls über die oben vorgestellten Referenzierungen durchführen.

Die Schaltfläche **cmdNameAusgeben** löst bei Mausklick eine Prozedur aus, die den Namen des Formulars **frmBeispiel** ausgibt:

```
Private Sub cmdNameAusgeben_Click()
    MsgBox Forms!frmBeispiel.Name
End Sub
```

Für den Zugriff auf ein Formular-Objekt innerhalb des Klassenmoduls des Formulars gibt es jedoch eine Vereinfachung.

Statt **Forms!<Formularname>** verwenden Sie einfach das Schlüsselwort **Me**. Die obige Prozedur könnte also auch so aussehen:

```
Private Sub cmdNameAusgeben_Click()
    MsgBox Me.Name
End Sub
```

Diese Schreibweise ist nicht nur kürzer, sondern hat noch einen weiteren wichtigen Vorteil gegenüber der absoluten Referenzierung des Formulars über den Namen: Sie können den Namen des Formulars einfach ändern, das Schlüsselwort **Me** bezieht sich immer auf das Formular selbst.

Noch besser: Sie können Code, der das Formular mit **Me** referenziert, problemlos in die Klassenmodule anderer Formulare übertragen. Sie wirken sich dann in gleicher Weise auf das neue Formular aus. Manchmal ist dies natürlich nicht gewünscht: Wenn Sie eine Schaltfläche zum Filtern eines Formulars auf ein anderes Formular verschieben, aber immer noch das Ursprungsformular filtern möchten, müssen Sie **Me** wieder in **Forms!<Formularname>** ändern.

Beispiel: Formular schließen

Ein oft verwendetes Beispiel für den Bezug eines Formulars auf sich selbst ist das Schließen mit einer **OK**-Schaltfläche. Die Ereignisprozedur **Beim Klicken** der Schaltfläche **cmdOK** des Formulars **frmBeispiel** versehen Sie dazu mit folgendem Code:

```
Private Sub cmdOK_Click()
    DoCmd.Close acForm, Me.Name
End Sub
```

Unterformulare referenzieren

Viele Entwickler tun sich beim Zugriff auf Daten in Unterformularen schwer. Dabei gibt es gar nicht so viel zu beachten – das Unterformular ist ein ganz normales Formular, dass allerdings nicht über die Forms-Auflistung erreichbar ist, sondern über das Unterformular-Steuerelement, in dem es sich befindet.

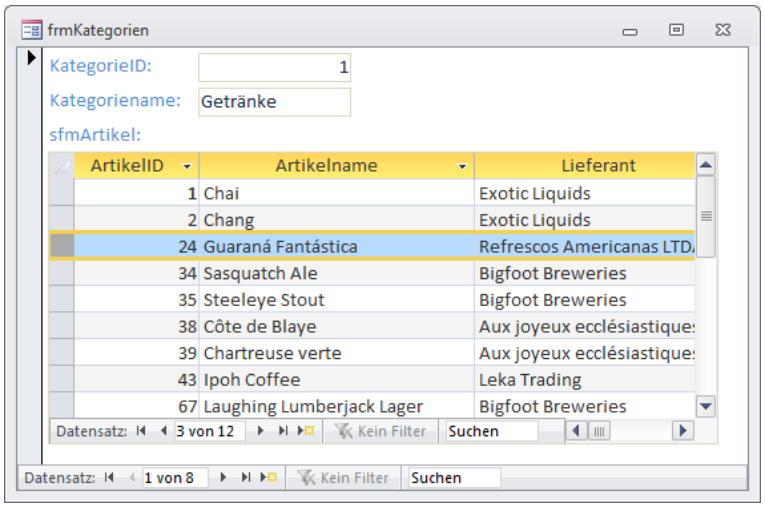


Bild 2: Referenzieren eines Unterformulars

Die folgenden Beispiele beziehen sich auf das Formular **frmKategorien**, in dem sich ein Unterformular-Steuerelement namens **sfm** befindet, welches das Formular **sfmArtikel** anzeigt (siehe Bild 2). Wenn Sie ein Formular per Drag and Drop aus dem Datenbankfenster oder dem Navigationsbereich als Unterformular in ein anderes Formular einfügen, stellt Access den Na-

Berichte öffnen

Wer mit Access arbeitet, möchte in der Regel auch die enthaltenen Daten hübsch aufbereitet als Bericht präsentieren. Damit der Bericht erscheint, ist allerdings noch ein wenig Handarbeit nötig: Schließlich wollen Sie dem Benutzer Ihrer Datenbank ja hoffentlich nicht das Datenbankfenster oder den Navigationsbereich als Steuerzentrale anbieten, sondern das Öffnen von Berichten und anderen Elementen per Menü, Ribbon oder schlicht von einem Formular aus ermöglichen. Dieser Artikel liefert die nötigen Grundlagen zum Anzeigen von Berichten.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1109_BerichteOeffnen.mdb**.

Bericht, öffne Dich!

Zum Öffnen von Formularen und Berichten gibt es im Wesentlichen zwei Möglichkeiten: per Doppelklick auf den entsprechenden Eintrag im Datenbankfenster (bis Access 2003) oder im Navigationsbereich (ab Access 2007) oder mit dem VBA-Befehl **DoCmd.OpenForm** beziehungsweise **DoCmd.OpenReport**. Dieser Artikel dreht sich um letztere Anweisung, die einen Bericht auf verschiedene Arten öffnet – und auf verschiedene Benutzeraktionen hin. Zum Testen öffnen Sie einfach den VBA-Editor, aktivieren das Direktfenster mit der Tastenkombination **Strg + G** und tippen dort einfach folgenden Befehl ein:

```
DoCmd.OpenReport "<Berichtname>"
```

Uups – schon sendet Access den Bericht an den Drucker. Da der durchschnittliche Access-Entwickler in seiner Laufbahn wohl mindestens zehn Mal mehr Formulare als Berichte öffnet, vergisst er gern, dass **DoCmd.OpenReport** nur mit dem Berichtsnamen als Parameter den Bericht direkt druckt. Damit der Bericht nicht gedruckt, sondern in der Seitenansicht angezeigt wird, fügen Sie einen weiteren Parameter namens **View** hinzu und vergeben für diesen den Wert **acViewPreview**:

```
DoCmd.OpenReport "rptArtikel", View:=acViewPreview
```

Das funktioniert: Der Bericht erscheint in der Seitenansicht (siehe Bild 1). Wenn Sie die Seitenansicht eines Berichts etwa über die Schaltfläche eines Formulars öffnen möchten, erstellen Sie zunächst ein Formular mit einer Schaltfläche namens **cmdSeitenansicht**. Wählen Sie für die Eigenschaft **Beim Klicken** den Wert **[Ereignisprozedur]** aus und klicken Sie auf die Schaltfläche mit den drei Punkten (...) rechts neben der Eigenschaft.

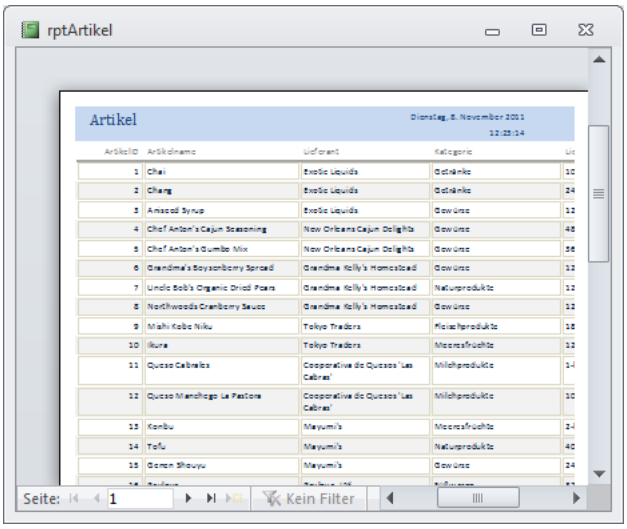


Bild 1: Ein in der Seitenvorschau geöffneter Bericht

Die nun im VBA-Editor angezeigte leere Ereignisprozedur füllen Sie wie folgt:

```
Private Sub cmdSeitenansicht_Click()
    DoCmd.OpenReport "rptArtikel", _
        View:=acViewPreview
End Sub
```

Wenn Sie nun in die Formularansicht wechseln und die Schaltfläche betätigen, erscheint der Bericht wie oben beschrieben (siehe Formular **frmBerichteOeffnen**, Bericht **rptArtikel**).

Weitere Berichtsansichten

Neben der Seitenansicht, die Sie mit dem Wert **acViewPreview** für den Parameter **View** erhalten, gibt es noch einige weitere Ansichten:

- **acViewDesign**: Entwurfsansicht, dient der Anpassung des Entwurfs des Berichts.
- **acViewLayout**: Layoutansicht. Wurde mit Access 2007 eingeführt und erlaubt das Anpassen des Entwurfs, während der Bericht Daten anzeigt.
- **acViewNormal**: Druckt den Bericht.

- **acViewPivotChart**, **acViewPivotTable**: Werden als Paramter angeboten, funktionieren aber nicht.

- **acViewPreview**: Öffnet die Seitenansicht des Berichts.

- **acViewReport**: Öffnet die Berichtsansicht (seit Access 2007). Die Berichtsansicht erlaubt interaktiven Zugriff auf einen Bericht.

Angezeigte Daten einschränken

Es gibt zwei Möglichkeiten, die in einem Bericht anzuzeigenden Daten anzuzeigen. Beide legen Sie über einen Parameter der **DoCmd.OpenForm**-Methode fest:

- **Filtername**: Erwartet den Namen einer Abfrage, deren **WHERE**-Abschnitt als Filter verwendet wird.

- **WhereCondition**: Erwartet die Angabe eines SQL-Ausdrucks wie **Artikelname LIKE 'A*'**.

Die Eigenschaft **Filtername** ist also eher für Benutzer gedacht, die SQL-Bedingungen lieber mit der Entwurfsansicht von Abfragen definieren, während **WhereCondition** die direkt Eingabe der SQL-Bedingung erlaubt.

Schauen wir uns zunächst die Methode mit dem Parameter Filtername genau an. Wir öffnen wiederum den Bericht **rptArtikel** der Beispieldatenbank in der Seitenansicht. Dieser soll nur diesmal nicht alle Datensätze, sondern nur diejenigen anzeigen, deren Feld **Artikelname** einen Wert enthält, der mit A beginnt. Die für den Parameter **Filtername** anzugebende Abfrage bezieht sich optimalerweise auf die gleiche Tabelle, die auch in der Datenherkunft des Berichts verwendet wird, also **tblArtikel**. Der Bericht entnimmt der Abfrage lediglich die Bedingung. Sie brauchen also nicht etwa alle Felder in das Entwurfsraster zu ziehen, die auch im Bericht erscheinen, sondern nur diejenigen, die in der Bedingung verwendet werden. Wir wollen erstmal nur alle Datensätze anzeigen, deren Artikelname mit **A** beginnt und erstellen daher eine Abfrage wie in Bild 2. Speichern Sie die Abfrage unter dem Namen **qryArtikelMitA** und rufen Sie den Bericht dann mit folgender Anweisung auf:

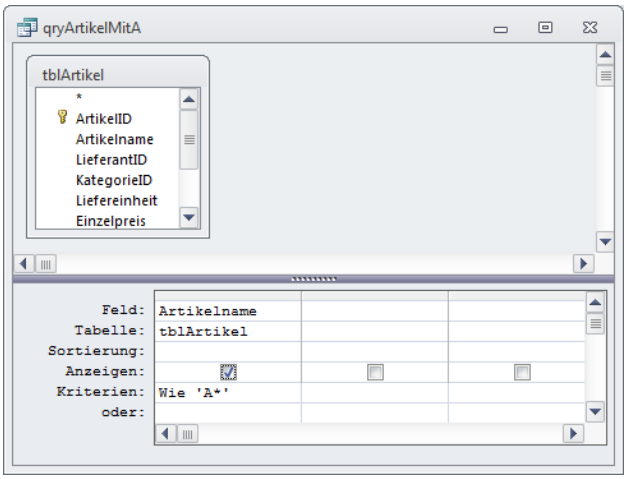


Bild 2: Diese Abfrage dient als Filter eines Berichts und wird mit dem Parameter **Filtername** übergeben.

```
DoCmd.OpenReport "rptArtikel", View:=acViewPreview,
Filtername:="qryFilterNurArtikelMitA"
```

Das Ergebnis sehen Sie in Bild 3. Es funktioniert – der Bericht zeigt genau die richtigen Datensätze an. Fragt sich, was geschieht, wenn Sie die Datensätze der Abfrage anderweitig einschränken, etwa durch ein Kriterium für ein Feld einer verknüpften Tabelle. Ein Beispiel finden Sie in Bild 4: Dort sollen alle Artikel geliefert werden, die von Kunden gekauft wurden, deren Firmenname mit **A** beginnt. Öffnen wir den Bericht mit folgender Anweisung:

```
DoCmd.OpenReport "rptArtikel", View:=acViewPreview,
Filtername:="qryArtikelVonKundeA"
```

Das Ergebnis sehen Sie in Bild 5: Der Bericht verlangt nach einen Wert für ein Feld namens **tblKun-**

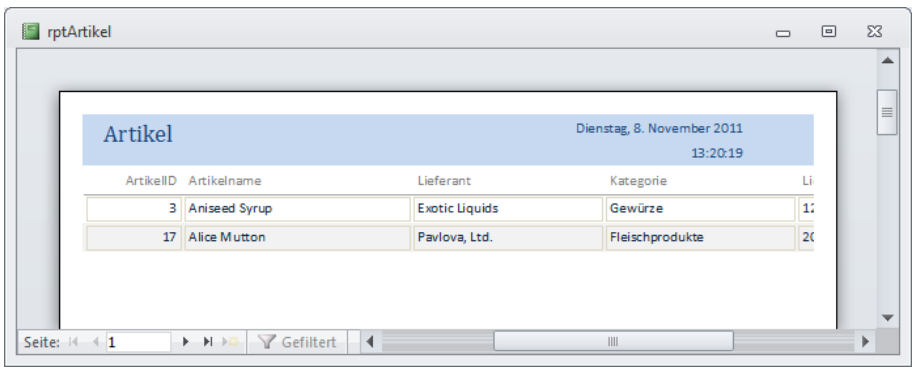


Bild 3: Dieser Bericht wurde mithilfe einer für das Kriterium **Filter** angegebenen Abfrage gefiltert.

DAO: Datenbank im Griff mit dem Database-Objekt

Wenn Sie mit VBA auf Informationen zur aktuellen Datenbank zugreifen möchten, bietet die DAO-Bibliothek mit dem Database-Objekt die perfekte Hilfe an: Dieses Objekt lässt sich mit einem Verweis auf die aktuell geöffnete, aber auch auf andere Datenbanken füllen und erlaubt dann nicht nur den Zugriff auf ihre Eigenschaften, sondern stellt auch eine ganze Reihe Funktionen bereit. Dieser Artikel zeigt, wie Sie mit dem Database-Objekt auf die aktuell geöffnete und auch auf andere Datenbanken zugreifen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1110_DAO_Database.mdb**.

Zugriff auf eine Datenbank

Warum sollte man überhaupt per VBA auf eine Datenbank und ihre Eigenschaften und Methoden zugreifen? Ganz einfach: Weil früher oder später die Methoden der Benutzeroberfläche und von Formularen ohne Einsatz von VBA-Ereignisprozeduren dazu führen werden, dass Sie Abläufe in Ihrer Access-Datenbank automatisieren werden – und zwar mit VBA.

Dabei werden Sie auch auf Informationen etwa zur aktuell geöffneten Datenbank angewiesen sein, beispielsweise um den Pfad herauszufinden, in dem sich die aktuelle Datenbank befindet. Oder Sie gehen einen Schritt weiter und möchten gleich noch Daten aus einer der in der Datenbank gespeicherten Tabellen auslesen – dies geht jedoch einen Schritt weiter und wird in einem weiteren Artikel beschrieben (**DAO-Recordsets öffnen und lesen**).

Wenn Sie auf die aktuelle Datenbank zugreifen möchten, brauchen Sie einen Verweis beziehungsweise eine Referenz auf diese Datenbank. Die Datenbank wird durch ein Objekt der Bibliothek DAO repräsentiert: dem **Database**-Objekt. Dieses Objekt muss nicht zwangsläufig auf die aktuell geöffnete Datenbank verweisen, aber dies wird in den meisten Fällen so sein. Einen Verweis auf dieses **Database**-Objekt erhalten Sie mit einer speziellen Funktion der Access-Bibliothek namens **CurrentDb**.

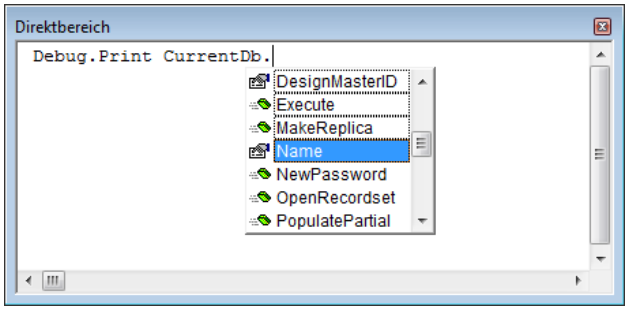


Bild 1: Eigenschaften und Methoden des mit **CurrentDb** gelieferten **Database**-Objekts

Da **CurrentDb** einen Verweis auf ein **Database**-Objekt liefert, können Sie direkt über die Funktion auf das **Database**-Objekt zugreifen. Das bedeutet, dass Sie die Eigenschaften und Methoden des Database-Objekts nutzen können. Öffnen Sie zum Beispiel das Direktfenster des VBA-Editors der aktuellen Datenbank mit **Strg + G** und tippen Sie die folgende Anweisung ein:

```
Debug.Print CurrentDb.Name
```

Bereits nach dem Eingeben des Punktes nach **CurrentDb** bietet IntelliSense eine Liste aller Eigenschaften und Methoden des mit der **CurrentDb**-Funktion gelieferten **Database**-Objekts an (siehe Bild 1). Nach Abschluss der Eingabe mit der Eingabetaste liefert die Name-Eigenschaft den kompletten Pfad mit Dateiname der aktuell geöffneten Datenbank, zum Beispiel:

```
C:\Daten\Fachartikel\AiUBasic\Basics\2011\09\DAO_
Database\1109_DAO_Database.mdb
```

Verweis auf die aktuelle Datenbank speichern

Wenn Sie mehr als einen Aufruf mit der **CurrentDb**-Funktion absetzen möchten, speichern Sie den damit gewonnenen Verweis auf die Datenbank in einer Variablen, die Sie wie folgt deklarieren:

```
Dim db As DAO.Database
```

Im folgenden Beispiel deklarieren Sie das Objekt, füllen es mit der **CurrentDb**-Funktion, geben Pfad und Dateiname der Datenbank aus und leeren die Objektvariable **db** wieder:

```
Public Sub Database_Beispiele()
    Dim db As DAO.Database
    Set db = CurrentDb
    Debug.Print db.Name
    Set db = Nothing
End Sub
```

For Each-Schleifen

Die For...Next-Schleife ist Ihnen längst bekannt. Etwas weniger oft nutzt man die For Each-Schleife. Der Hauptgrund ist, dass diese nur mit Auflistungen zusammenarbeitet, also mit Objekten, die meiste mehrere gleichartige Objekte zusammenfassen. Beispiele sind die Formulare einer Datenbank oder die Steuerelemente eines Formulars. Anhand dieser Beispiele lernen Sie in diesem Beitrag die Verwendung von For Each-Schleifen kennen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1110_ForEach.mdb**.

Jeder nur einmal!

Die **For Each**-Schleife ist bestens dazu geeignet. Elemente von Auflistungen unter VBA zu durchlaufen. Auflistungen? Was für Auflistungen? Nun: In objektorientierten Sprachen, wie es auch VBA teilweise ist, gibt es nicht nur Objekte mit bestimmten Eigenschaften (wie Formulare, Steuerelemente, Datenbanken oder Tabellen), sondern auch Objekte, die den Zugriff auf solche Elemente etwa über die Angabe eines Indexes erlauben.

Offene Formulare durchlaufen

Wenn Sie beispielsweise einmal wissen möchten, welche Formulare aktuell in Ihrer Datenbank geöffnet sind, hilft Ihnen eine **For Each**-Schleife weiter. Die Auflistung **AllForms** des CurrentProject-Objekts liefert die notwendigen Daten. Mit der folgenden Anweisung können Sie etwa im Direktfenster die Anzahl der Formulare der aktuellen Datenbank abfragen:

```
? CurrentProject.AllForms.Count
 4
```

Nun wollen Sie aber alle Formulare durchlaufen und deren Namen ausgeben. Um dies innerhalb einer **For Each**-Schleife durchzuführen, brauchen Sie zunächst einmal eine Objektvariable als Laufvariable. In diesem Fall verwenden Sie dazu ein Objekt des Typs **AccessObject**:

```
Dim frm As AccessObject
```

Wie finden Sie den Typ heraus? Ganz einfach: Öffnen Sie den Objektkatalog von Access mit der Tastenkombination **F2**, suchen Sie dort nach der Auflistungsklasse (hier **AllForms**) und klicken Sie auf die Eigenschaft, die so aussieht, als ob sie einen Verweis auf eines der enthaltenen Elemente zurückliefern

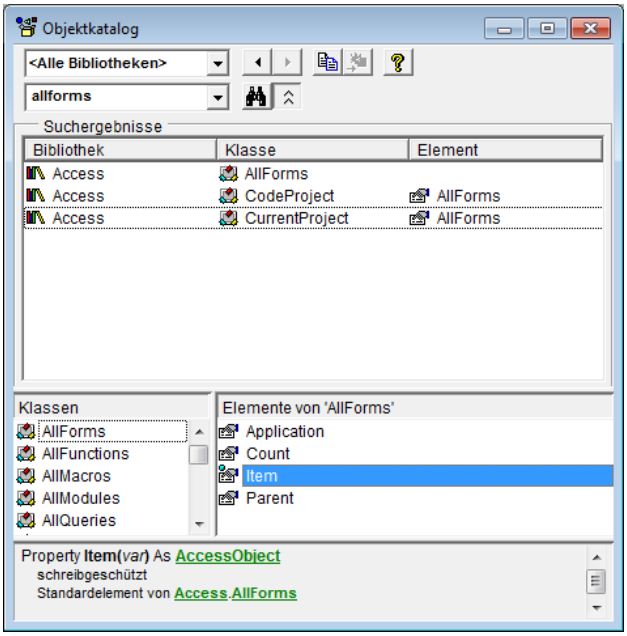


Bild 1: Der Objektkatalog hilft, den Typ eines Auflistungsobjekts zu ermitteln.

könnte – in diesem Fall also **Item**. Ganz unten erfahren Sie dann, dass **Item** den Typ **AccessObject** besitzt (siehe Bild 1).

Damit können Sie dann die **For Each**-Schleife zusammensetzen, die wie folgt aussieht:

```
For Each frm In CurrentProject.AllForms
    Debug.Print frm.Name
Next frm
```

Diese Schleife durchläuft alle Elemente der **AllForms**-Auflistung und weist das jeweils aktuelle Element der Variablen **frm** zu. Damit können Sie dann innerhalb der Schleife anstellen, was Sie möchten. Gut, es gibt eine Ausnahme: das Löschen der enthaltenen Elemente geht auf diese Weise schief.

Geöffnete Formulare durchlaufen

Viele Objekte können Sie auf verschiedene Arten durchlaufen. Wenn Sie beispielsweise nur die aktuell geöffneten Elemente referenzieren möchten, nutzen

Sie die **Forms**-Auflistung und durchlaufen alle darin enthaltenen Objekte des Typs **Form**. Das sieht dann so aus:

```
Dim frm As Form
For Each frm In Forms
    Debug.Print frm.Name
Next frm
```

Alternative mit For...Next

Das Gleiche gelingt auch mit einer **For...Next**-Schleife – die **For Each**-Schleife ist nur etwas eleganter. Mit **For...Next** sieht das Durchlaufen der Formulare wie folgt aus:

```
Dim i As Integer
For i = 0 To Forms.Count - 1
    Debug.Print Forms(i).Name
Next i
```

Und wenn Sie auch hier ein **Form**-Objekt nutzen möchten, füge Sie noch ein paar Zeilen hinzu:

```
Dim frm As Form
Dim i As Integer
For i = 0 To Forms.Count - 1
    Set frm = Forms(i)
    Debug.Print frm.Name
Next i
```

Warum bei 0 beginnen? Die meisten Auflistungen verwenden einen 0-basierten Index, das heißt, dass Sie das erste Element über den Wert 0 für den Index erhalten. Dementsprechend führt das Referenzieren des Elements mit dem Index, welcher der Anzahl entspricht (also beispielsweise **Forms.Count**), zu einem Fehler: Eine Auflistung mit vier Elementen verwendet, wenn Sie 0-basiert ist, die Index-Werte **0**, **1**, **2** und **3**. Der Verweis auf **Forms(4)** funktioniert dementsprechend nicht. Gelegentlich gibt es auch Elemente, deren Indexwerte bei **1** beginnen. Hier ist dann der Wert der **Count**-Eigenschaft gleichzeitig der Index-Wert des letzten Elements der Auflistung. Und dies ist ein Vorteil der **For Each**-Schleife: Sie schert sich nicht um den Index, sondern liefert einfach alle Werte der Auflistung zurück.

Elemente löschen

Mit der Methode **DeleteObject** des **DoCmd**-Objekts können Sie Access-Objekte wie Tabellen, Abfragen, Formulare et cetera löschen. Wenn wir das in eine **For Each**-Schleife einbauen, um beispielsweise alle Formulare der aktuellen Datenbank zu löschen, sähe das so aus:

```
Dim frm As AccessObject
For Each frm In CurrentProject.AllForms
    DoCmd.DeleteObject acForm, frm.Name
Next frm
```

Das führt aber zu einem Fehler, was sich am Beispiel der beiden Formulare **frm1** und **frm2** leicht nachvollziehen lässt. Im ersten Durchlauf wird die Variable **frm** mit einem Verweis auf **frm1** gefüllt. Dieses Formular wird mit der **DeleteObject**-Methode gelöscht. Dann versucht die **For Each**-Schleife, zum zweiten Objekt der Auflistung zu springen. Das gelingt allerdings nicht, weil nur noch ein Objekt da ist, das Objekt frm jedoch das zweite Element der Auflistung referenzieren will!

Auch in einer **For...Next**-Schleife wird es nicht besser. Im ersten Durchlauf würde von zwei Formularen **frm1** und **frm2** wiederum **frm1** gelöscht, im zweiten Durchlauf versucht die Prozedur jedoch, auf das zweite Element der **AllForms**-Auflistung zuzugreifen. Da diese jedoch nur noch ein Element enthält, scheitert auch diese Variante:

```
Dim frm As AccessObject
Dim i As Integer
For i = 0 To CurrentProject.AllForms.Count - 1
    Set frm = CurrentProject.AllForms(i)
    DoCmd.DeleteObject acForm, frm.Name
Next i
```

Wenn Sie also alle Elemente einer Auflistung löschen wollen, müssen Sie eine **For...Next**-Schleife verwenden, welche die Liste der Formulare rückwärts durchläuft – vom höchsten Index (**CurrentProject.AllForms.Count**) zum niedrigsten (**0**). Um bei unserem Beispiel der zwei Formulare **frm1** und **frm2** zu bleiben: Im ersten Durchlauf wird das zweite Elemente der Auflistung **AllForms** gelöscht, im zweiten das erste – und dies gelingt, weil nach dem Löschen des zweiten Elements nur noch das erste vorhanden ist:

```
Dim frm As AccessObject
Dim i As Integer
For i = CurrentProject.AllForms.Count - 1 To 0 Step -1
    Set frm = CurrentProject.AllForms(i)
    DoCmd.DeleteObject acForm, frm.Name
Next i
```

Bedingte Formatierung

Wenn es darum geht, Daten in Abhängigkeit ihrer Werte hervorzuheben, bietet Access mit der bedingten Formatierung ein interessantes Werkzeug. Damit legen Sie die Bedingung fest, unter denen eine Formatierung erfolgt, und bestimmen, wie die formatierten Werte aussehen. Dabei stehen vor allem Zeichenformatierungen und die Hintergrundfarbe zur Verfügung. Und mit Access 2010 wurde sogar noch die Möglichkeit eingeführt, Zahlenwerte durch entsprechende Balken zu repräsentieren – und es wurde eine ärgerliche Beschränkung aufgehoben.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1111_BedingteFormatierung.mdb**.

Bedingte Formatierung von Version zu Version

Die bedingte Formatierung ist keinesfalls von Beginn an Bestandteil von Access gewesen. Sie wurde erst mit Version 2000 eingeführt und mit Version 2010 erheblich erweitert. Seit Access 2000 können Sie damit drei verschiedene Bedingungen und die entsprechenden Formate je Ansicht festlegen, mit Access 2010 wurde die Anzahl der möglichen Bedingungen auf 50 erweitert.

Die bedingte Formatierung ist natürlich vorrangig dort interessant, wo mehrere Datensätze gleichzeitig erscheinen, um Datensätze mit bestimmten Eigenschaften herausfiltern zu können – beispielsweise solche, deren Umsatz einen bestimmten Wert unterschreitet. Die bedingte Formatierung kann jedoch auch dort interessant sein, wo es etwa um die Markierung von Fälligkeitsdaten geht. Beides hilft auch in der Detailansicht einzelner Datensätze, schnell bestimmte Datensätze aufzufinden.

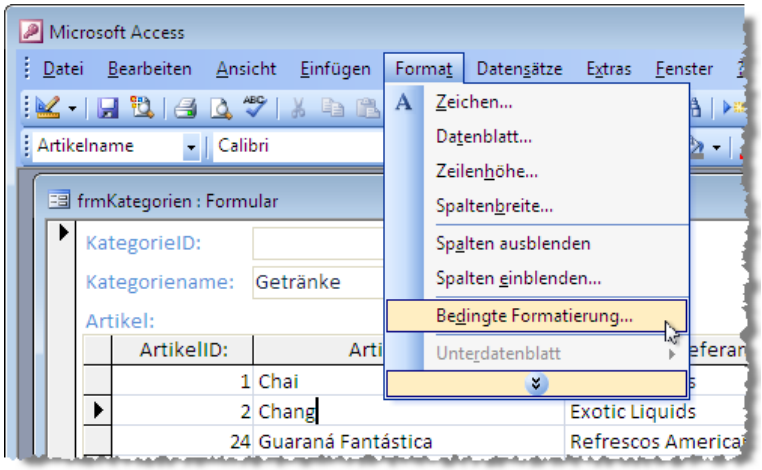


Bild 1: Aufrufen des Dialog zum Einstellen der bedingten Formatierung

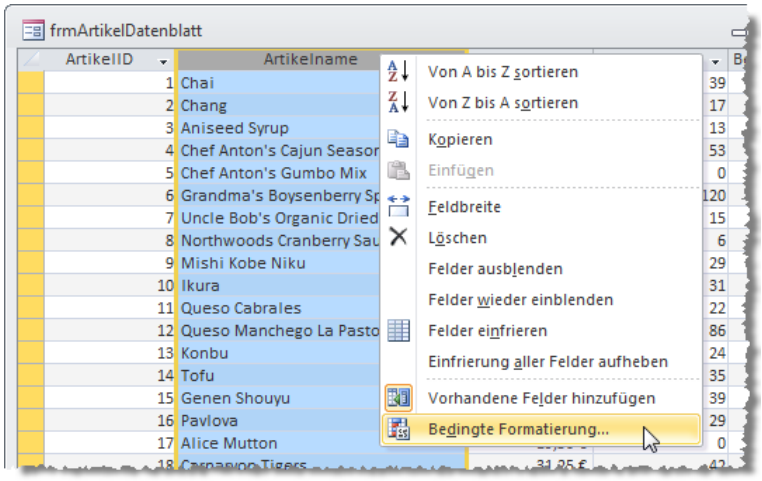


Bild 2: Unter Access 2010 öffnen Sie den Dialog einfach über das Kontextmenü der Spaltenüberschrift im Datenblatt.

Die bedingte Formatierung legen Sie sowohl in der Entwurfs- als auch in der Datenblatt- oder Formularansicht fest. Das heißt, Sie können auch in Echtzeit Bedingungen festlegen und die entsprechenden Formatierungen definieren.

Unter Access 2003 und älter rufen Sie den Dialog zum Festlegen der bedingten Formatierungen am einfachsten über den Menübefehl **Format|Bedingte Formatierung** auf (siehe Bild 1). Unter Access 2007 und jünger finden Sie den Befehl im Kontextmenü der Spaltenüberschrift im Datenblatt (siehe Bild 2) oder der einzelnen Felder in der Entwurfsansicht des Formulars.

Bedingte Formatierung anlegen

Wenn Sie etwa die Preise aller Artikel hervorheben möchten, deren Preis höher als 10 Euro liegt, ver-

wenden Sie dazu eine bedingte Formatierung für das Feld **Einzelpreis**. Markieren Sie dieses Feld beispielsweise in der Entwurfsansicht des Formulars, in dem es sich befindet, und öffnen Sie den Dialog zum Bearbeiten der bedingten Formatierungen. Dieser sieht wie in Bild 3 aus und bietet zunächst drei Optionen an:

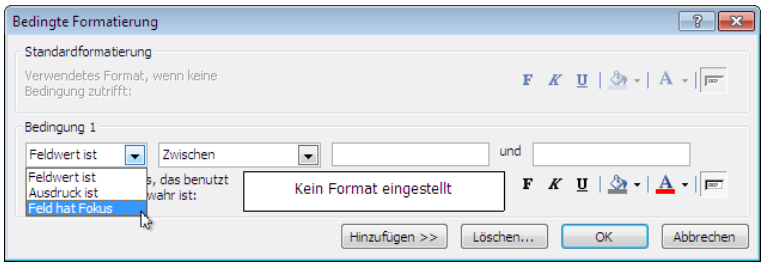


Bild 3: Bedingte Formatierung unter Access 2003 und älter anlegen

- **Feldwert ist**: Legt eine Formatierung fest, wenn der Wert des Feldes selbst eine bestimmte Bedingung erfüllt.

- **Ausdruck ist**: Legt eine Formatierung fest, die nicht nur vom Wert des aktuellen Feldes, sondern auch von anderen Feldern oder gar von ganz anderen Faktoren abhängen kann.

- **Feld hat Fokus**: Legt eine Formatierung fest, die angezeigt wird, wenn das Feld den Fokus erhält.

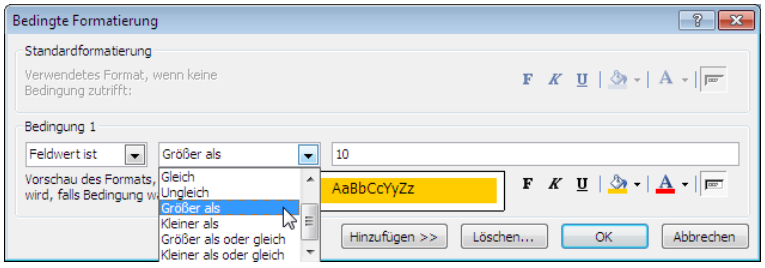


Bild 4: Vergleichsoperator der Bedingung

In unserem Fall benötigen wir zunächst die erste Option. Das zweite Auswahlfeld bietet dann die üblichen Vergleichsoperatoren an. Je nach Auswahl erscheinen rechts davon ein oder mehrere Felder zur Eingabe der Vergleichswerte. Im ersten Beispiel soll einfach nur geprüft werden, ob der Feldwert größer als 10 ist. Für diesen Fall legen Sie eine Formatierung fest, bei welcher der Hintergrund des Formulars orange dargestellt wird (siehe Bild 4).

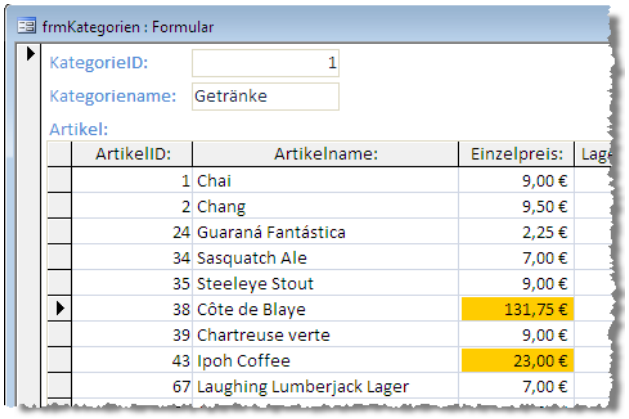


Bild 5: Bedingte Formatierung in Aktion

Das Ergebnis finden Sie in Bild 5. Alle betroffenen Einzelpreise werden mit der entsprechenden Farbe hinterlegt.

Wenn Sie nun alle Felder dieses Datensatzes orange hinterlegen möchten, verwenden Sie die zweite Option im linken Auswahlfeld einer bedingten Formatierung. Legen Sie etwa eine bedingte Formatierung für das Feld **Artikelname** an. Wählen Sie links **Ausdruck ist** aus und stellen Sie als Ausdruck den Wert **[Einzelpreis] > 10** ein. Legen Sie die gleiche Formatierung wie für das Feld **Einzelpreis** fest (siehe Bild 6). Als Resultat werden nun beide Felder für Datensätze mit Einzelpreisen größer als 10 Euro farbig hinterlegt.

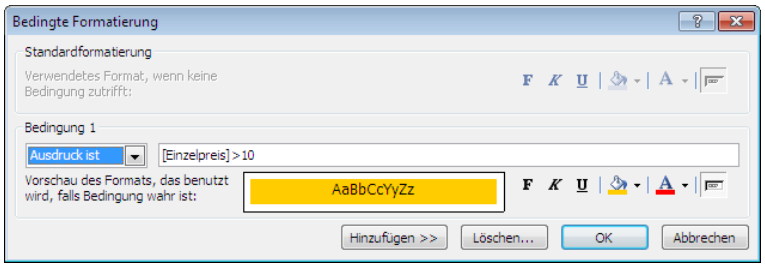


Bild 6: Ausdruck als Bedingung einer bedingten Formatierung

Bleibt noch die Variante mit dem Fokus. Damit können Sie den Benutzer beim Auffinden des Feldes helfen, das aktuell den Fokus besitzt. Wählen Sie für eine solche Bedingung den dritten Wert im linken Kombinationsfeld aus und stellen Sie die gewünschte Formatierung ein (beispielsweise einen grauen Hintergrund). Eine solche Formatierung sieht etwa wie in Bild 7 aus.

Damit eine solche Formatierung Sinn macht, müssen Sie natürlich alle Text- und Kombinationsfelder eines Formulars mit dieser Formatierung ausstatten. Da-

Mehrspaltige Berichte

Daten landen meist in tabellarischer Form in Berichten (Ausnahmen wie Rechnungen et cetera bestägigen die Regel). Oft braucht eine solche Liste nicht annähernd die komplette Seitenbreite. Zum Glück bietet Access die Möglichkeit, Berichte mehrspaltig auszugeben. Dieser Artikel zeigt, wo Sie die Spaltenanzahl eines Berichts einstellen und was Sie beachten müssen, damit Ihre Daten genau nach Wunsch auf die verschiedenen Spalten aufgeteilt werden.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1111_MehrspaljtigeBerichte.mdb**.

Spalte anlegen

Wenn Sie die Daten eines Berichts auf mehrere Spalten aufteilen möchten, müssen Sie die Breite des Berichts entsprechend anpassen. Während ein normaler Bericht die Breite des Zielmediums – also etwa eines DIN A4-Blatts – minus Seitenrändern einnimmt, ziehen Sie bei mehreren Spalten die Seitenränder ab und teilen die Breite dann durch die Anzahl der gewünschten Spalten.

Bei einer Breite von 21 Zentimetern und Seitenrändern von 2 Zentimetern bleiben also 17 Zentimeter übrig. Je nach Anzahl der Spalten müssen Sie aber auch noch den Zwischenraum zwischen den Spalten berücksichtigen. Wenn Sie also etwa zwei Spalten anlegen möchten, lassen Sie beispielsweise 0,5 Zentimeter Platz zwischen den Spalten und berücksichtigen für die Spaltenbreite jeweils 8,25 Zentimeter.

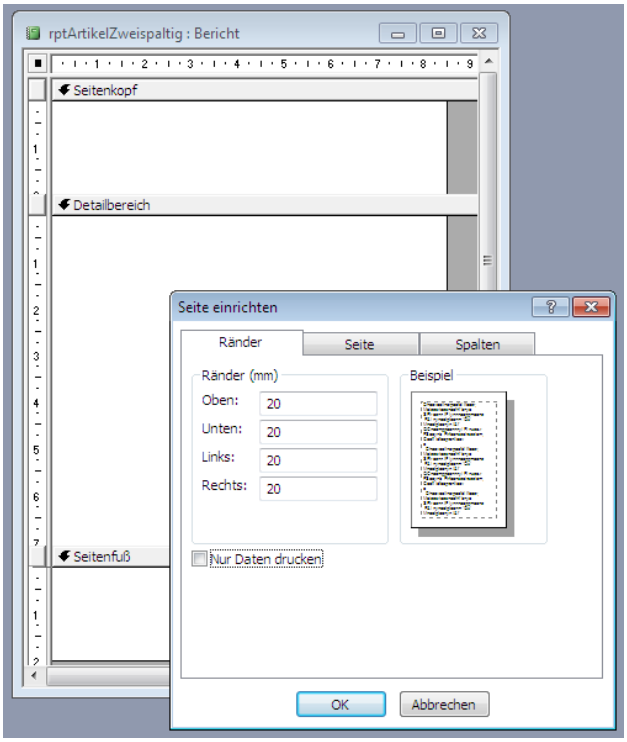


Bild 1: Einstellen der Seiten-Eigenschaften

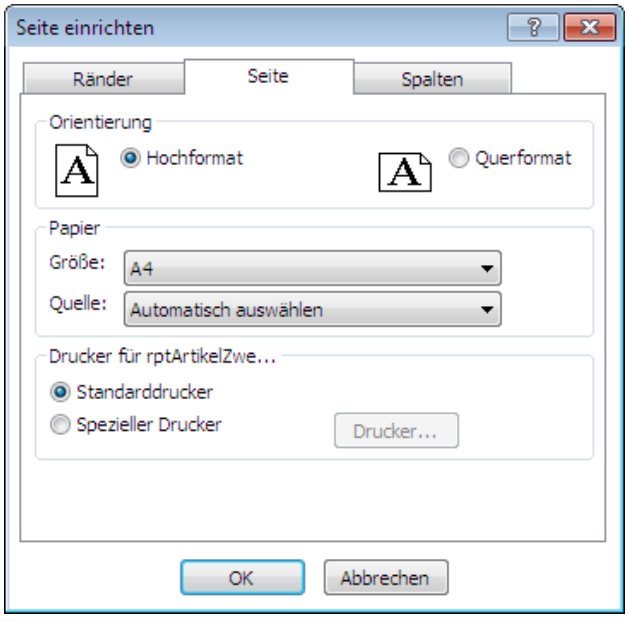


Bild 2: Weitere Seiteneigenschaften

Seite einrichten

Die für die mehrspaltige Ansicht wichtigsten Eigenschaften finden sich interessanterweise nicht im herkömmlichen Eigenschaftsfenster, sondern in einem separaten Dialog. Diesen rufen Sie unter Access 2003 und älter mit dem Menübefehl **Datei|Seite einrichten...** und unter Access 2007 und jünger mit dem Ribbon-Eintrag **Seite einrichten|Seitenlayout|Seite einrichten** auf, der allerdings nur bei in der Entwurfsansicht geöffnetem Bericht angeboten wird (unter Access 2003 und älter reicht es, wenn der Bericht im Datenbankfenster markiert ist). Auf die Registerseite Spalten dieses Dialogs gelangen Sie ab Access 2007 auch über den Ribbon-Eintrag **Seite einrichten|Seitenlayout|Spalten**.

Der Dialog sieht wie in Bild 1 aus und bietet zunächst Steuerelemente zum Einstellen der Seitenränder des aktuellen Berichts an. Auf der zweiten Seite bietet

sich die Möglichkeit, die Orientierung des Berichts einzustellen sowie die Größe des Papiers festzulegen. Außerdem geben Sie hier an, welcher Drucker zum Ausdrucken dieses Berichts verwendet werden soll (siehe Bild 2).

Die interessanten Eigenschaften finden sich schließlich auf der Registerseite **Spalten** (siehe Bild 3). Hier legen Sie zunächst fest, in wieviele Spalten der Bericht aufgeteilt werden soll.

Unser Beispielbericht soll einige Felder der Datensätze der Tabelle **tblArtikel** in zwei Spalten anzeigen. Dazu weisen Sie der Eigenschaft **Datenherkunft** des Berichts den Wert **tblArtikel** zu. Ziehen Sie dann einige Felder der Datenherkunft aus der Feldliste in den Detailbereich des Berichtsentwurfs und richten Sie diese wie inBild 4 aus.

Wichtig ist, dass die Summe aus Spaltenbreiten, Spaltenabständen und Seitenrändern nicht größer als die angegebene Seitenbreite wird, da sonst eine Meldung wie in Bild 5 erscheint. Richten Sie den Bericht also gleich in entsprechender Breite ein.

Spaltenlayout festlegen

Im Bereich **Spaltenlayout** des Dialogs **Seite einrichten** legen Sie fest, ob zunächst die erste Spalte von oben nach unten mit Datensätzen gefüllt wird und erst dann die zweite Spalte begonnen wird oder ob immer jeweils eine Zeile gefüllt werden soll, bevor weitere Datensätze in der folgenden Zeile landen.

In den meisten Fällen ist die erste Variante angezeigt. Wenn Sie beispielsweise nach dem Alphabet sortierte Artikel oder Adressen ausgeben, können Sie diese einfacher von oben nach unten und erst dann von links nach rechts durchsehen als umgekehrt.

Das Ergebnis dieses Berichtsentwurfs in der Seitenansicht finden Sie in Bild 6.

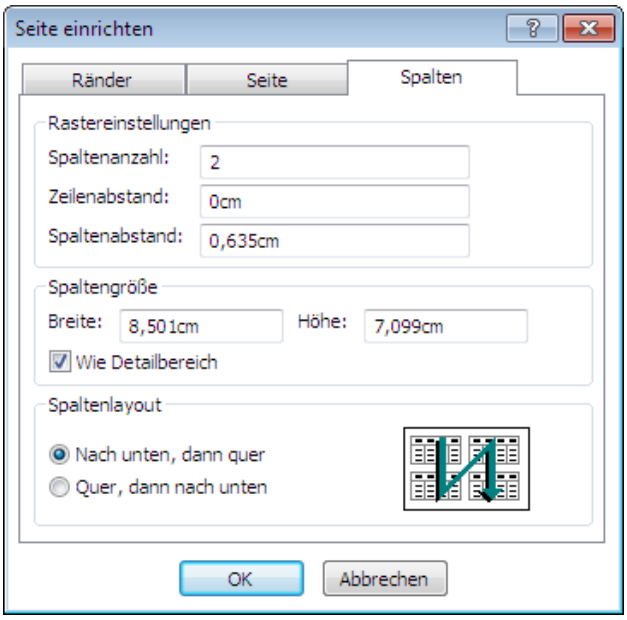


Bild 3: Spalten-Einstellungen des Berichts

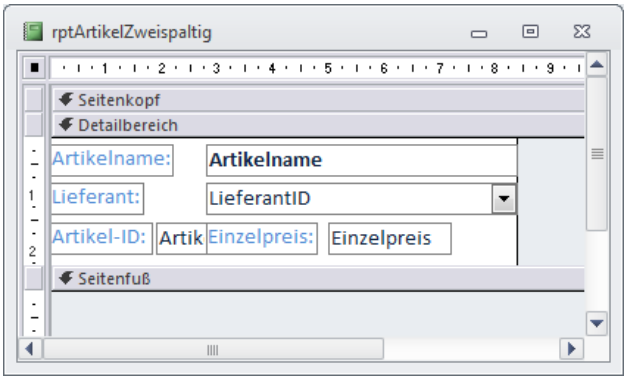


Bild 4: Entwurfsansicht des zweispaltigen Berichts

Seiten-/Berichtskopf über die gesamte Breite

Wenn Sie dem Bericht nun einen Seitenkopf hinzufügen, der über die gesamte Breite eine Überschrift anzeigen soll, können Sie das im Entwurf ohne Probleme bewerkstelligen – dies sieht schließlich wie in Bild 7 aus. Allein beim Wechsel in die Seitenansicht (und natürlich auch vor dem Drucken) wird wieder-

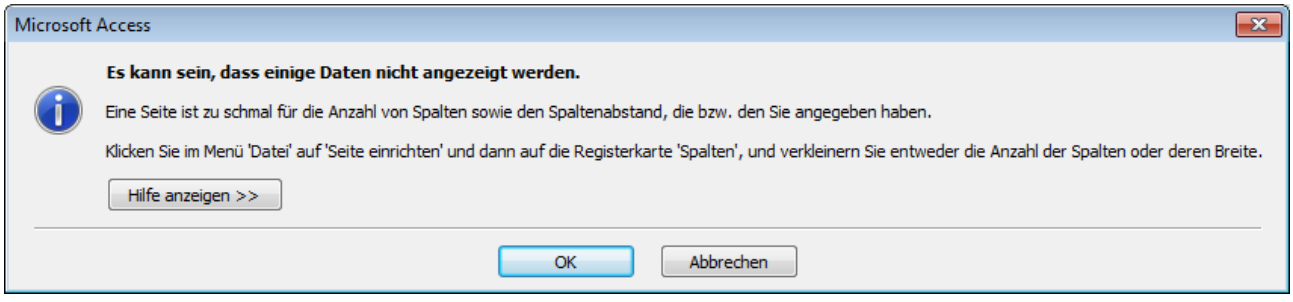


Bild 5: Diese Meldung erscheint, wenn der Bericht zu schmal für den anzuzeigenden Inhalt ist.

Zeichenketten-Funktionen

Die Arbeit mit Zeichenketten ist ein elementarer Bestandteil beim Umgang mit Access. Dies reicht vom einfachen Zusammensetzen einzelner Texte zu einer Zeichenkette über das Ermitteln bestimmter Textelemente bis hin zur buchstabenweisen Analyse von Texten. VBA bietet dazu einige Funktionen, die dieser Artikel vorstellt – neben den grundlegenden Techniken unter VBA und an anderen Orten wie Eigenschaften, Abfragen und Steuerelementen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1111_Zeichenkettenfunktionen.mdb**.

Einsatzzwecke

Die nachfolgend vorgestellten Zeichenkettenfunktionen können Sie wie die anderen bereits vorgestellten Access/VBA-Funktionen in manchen Tabelleneigenschaften, als Werte oder Kriterien in Abfragen oder als Eigenschaften oder Feldinhalte in Formularen, Berichten und Steuerelementen einsetzen. Darüberhinaus treten die nachfolgend vorgestellten Funktionen natürlich in VBA-Routinen in Erscheinung.

Zeichenketten

Zeichenketten kommen entweder als Literal oder beispielsweise als Wert einer Variablen, eines Feldes oder einer Eigenschaft vor. Folgendes könnte eine Zeichenkette liefern:

- **"André"**: Literal
- **""**: Leere Zeichenkette
- **strVorname**: String-Variable
- **Me!txtVorname.Value**: Wert eines Textfeldes
- **rst!Vorname**: Wert eines Feldes eines Recordsets

Während beim Literal, bei einer leeren Zeichenkette und bei einer gefüllten String-Variable keine Gefahr droht, können ein Steuerelement oder ein Feld durchaus den Wert **Null** enthalten. Dies ist im Gegensatz zur leeren Zeichenkette eigentlich keine Zeichenkette. Wenn Sie im Rahmen des Umgangs mit Zeichenketten auf einen **Null**-Wert stoßen, müssen Sie damit auf besondere Weise umgehen. Sie können beispielsweise mit einer leeren oder auch mit einer bestimmten anderen Zeichenkette weiterarbeiten, wenn Sie auf den Null-Wert treffen. Manchmal ist der Nullwert auch nützlich, mehr dazu später. Die einfachste Zeichenkettenfunktion ist das Verketten zweier Zeichenketten mit dem Kaufmanns-Und-Operator **(&)**. Bild 1 und Bild 2 zeigen, wie Sie ein Literal und den Wert einer VBA-Funktion für die Eigenschaft **Steuerelementinhalt** eines Textfeldes eines Formulars (oder wahlweise eines Berichts) eingeben und anzeigen.

Statt in die Eigenschaft **Steuerelementinhalt** können Sie den Ausdruck in der Entwurfsansicht des Formulars auch gleich in das Textfeld eingeben. In der Formularansicht wird der Ausdruck dann ausgewertet und angezeigt.

Damit Sie gleich sehen, dass dies auch per VBA funktioniert, legen Sie gleich darunter ein weiteres Textfeld namens **txtZeichenketteVBA** an und erstellen die folgende Ereignisprozedur, die durch das

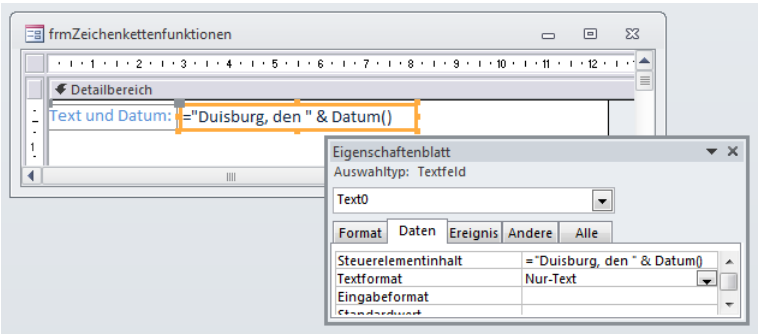


Bild 1: Zusammensetzen eines Literals und einer VBA-Funktion

Bild 2: Anzeige einer zusammengesetzten Zeichenkette

Ereignis **Beim Laden** des Formulars ausgelöst wird (Eigenschaft **Beim Laden** auf **[Ereignisprozedur]** einstellen, auf die Schaltfläche mit den drei Punkten klicken und die nun im VBA-Editor erscheinende Prozedur erweitern):

```
Private Sub Form_Load()
    Me!txtZeichenketteVBA = "Duisburg, den " & Date
End Sub
```

Die Prozedur bewirkt das Gleiche wie das Füllen der Eigenschaft **Steuerelementinhalt**: Es trägt einen Text wie **Duisburg, den 3.12.2011** in das Textfeld ein.

Der wichtigste Unterschied zwischen dem Einsatz als Eigenschaft und unter VBA ist, dass Sie bei einer Eigenschaft die deutsche Version der VBA-Funktionen verwenden (manchmal können Sie auch die englische Variante eingeben, diese wird dann übersetzt).

Damit Sie die folgenden Beispiele jeweils schnell ausprobieren können, formulieren wir diese in VBA – Sie geben diese dann testweise in das Direktfenster des VBA-Editors ein (**Strg + C**) und führen diese durch Betätigen der Eingabetaste aus.

& und +

Gerade eben haben Sie den **&**-Operator kennengelernt, mit dem Sie zwei Zeichenketten verknüpfen können:

```
Debug.Print "a" & "b"
  ab
```

Dies gelingt auch mit dem +-Operator:

```
Debug.Print "a" + "b"
  ab
```

Der wesentliche Unterschied liegt in der Behandlung von **Null**-Werten. Wenn einer der zu verknüpfenden Ausdrücke den Wert **Null** hat, behandelt der **&**-Operator diesen als leere Zeichenkette:

```
Debug.Print "a" & Null
  a
```

Bei Verwendung des **+**-Operator wird jedoch der komplette Ausdruck zu **Null**, wenn einer der Teilausdrücke den Wert **Null** aufweist:

```
Debug.Print "a" + Null
  Null
```

Länge von Zeichenketten ermitteln

Die Länge einer Zeichenkette ermitteln Sie mit der **Len**-Funktion. Sie erwartet lediglich die Zeichenkette als Parameter, die Sie als Literal oder in Form einer Variablen beziehungsweise eines Ausdrucks übergeben können. Beispiel:

```
Debug.Print Len("abc")
  3
```

In Ausdrücken verwenden Sie die deutsche Version, also beispielsweise **=Länge([txtZeichenkette])** (siehe Beispielformular **frmZeichenkettenfunktionen**).

Teile von Zeichenketten ermitteln

Mit den drei Funktionen **Left**, **Right** und **Mid** können Sie Teilzeichenketten einer Zeichenkette ermitteln:

- **Left**: Liefert die ersten **n** Zeichen einer Zeichenkette zurück. Der Aufruf **Left("abc",2)** liefert also beispielsweise die Zeichenkette **ab** zurück (die ersten beiden Zeichen des Ausdrucks **abc**).
- **Mid**: Liefert eine Teilzeichenkette einer Zeichenkette zurück, die nach einer bestimmten Anzahl Zeichen beginnt und eine festgelegte Länge hat. Der Aufruf **Mid("abc",2,1)** liefert also beispielsweise die Teilzeichenkette zurück, die mit dem zweiten Zeichen beginnt und ein Zeichen lang ist – also das Zeichen **c**.
- **Right**: Liefert die letzten **n** Zeichen einer Zeichenkette zurück. Der Aufruf **Right("abc",2)** liefert also

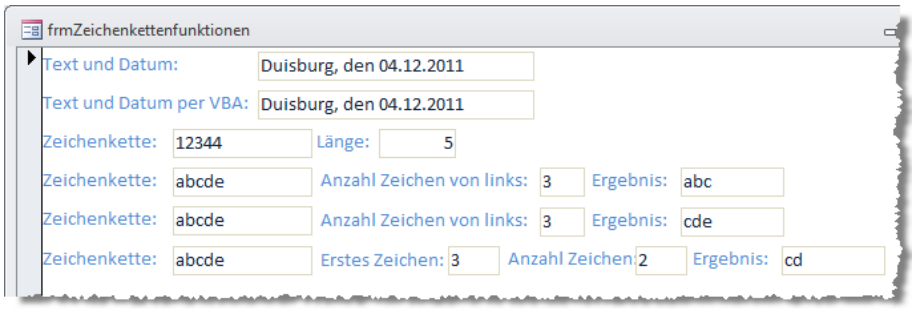


Bild 3: Beispiele für Zeichenkettenfunktionen in Textfeldern

DAO: Recordsets öffnen, lesen und bearbeiten

Die DAO-Bibliothek liefert alle für den Zugriff auf und die Manipulation von Daten in den Tabellen einer Access-Datenbank. Sie können damit Datensätze einer Tabelle oder Abfrage durchlaufen und dabei Informationen auslesen, Datensätze anlegen, bearbeiten oder löschen. Für all diese Aktionen benötigen Sie ein DAO-Recordset – ein Objekt, dass den Zugriff auf die benötigten Daten ermöglicht. Dieser Artikel stellt die Techniken vor, die zum Öffnen und Durchlaufen von Datensatzgruppen mittels VBA nötig sind.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1111_DAO_Recordsets.mdb**.

Es gibt diesmal ausschließlich Code-Beispiele. Wenn Sie diese ausprobieren möchten, legen Sie in einer Datenbank ein neues Standardmodul an, geben die Listings ein, platzieren die Einfügemarke irgendwo im Code der auszuführenden Routine und betätigen die Taste **F5** (oder **F8** für den Einzelschrittmodus).

Records und Recordsets

Ein Record entspricht einem Datensatz, ein Recordset einer Datensatzgruppe. Dabei kann eine Datensatzgruppe allerdings aujch keinen oder auch nur einen Datensatz enthalten.

Unter VBA greifen Sie auf Datensätze zu, um diese zu zählen, neu zu erstellen, zu bearbeiten, zu löschen oder einfach um die enthaltenen Werte zu untersuchen.

Recordset öffnen

Das Öffnen eines Recordsets erfordert das vorherige Deklarieren und Instanzieren eines **Database**-Objekt, das einen Verweis auf die Datenbank mit dem zu öffnenden Objekt enthält. Dies geht auf die Schnelle mit den folgenden beiden Zeilen:

```
Dim db As DAO.Database
Set db = CurrentDb
```

Weitere Informationen hierzu finden Sie im Artikel **DAO: Datenbank im Griff mit dem Database-Objekt**. Das auf diese Weise verfügbar gemacht **Database**-Objekt bietet die Funktion **OpenRecordset** an, mit der Sie die zu öffnende Datensatzgruppe definieren und die einen Verweis auf die zu erstellende Datensatzgruppe zurückliefert.

Für den besseren Überblick steigen wir gleich mit einem größeren Beispiel ein:

```
Public Sub RecordsetOeffnen()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("tblArtikel", _
        dbOpenDynaset)
    Do While Not rst.EOF
        Debug.Print rst!ArtikelID, rst!Artikelname
        rst.MoveNext
    Loop
End Sub
```

Diese Prozedur deklariert neben dem **Database**- auch ein **Recordset**-Objekt namens **rst**, dass den Verweis auf die zu erstellende Datensatzgruppe aufnehmen soll. Über diesen Verweis greifen Sie später auf die Daten der Datensatzgruppe zu. Dann erstellt die **OpenRecordset**-Methode ein Recordset, dass alle Felder und alle Datensätze der Tabelle **tblArtikel** enthält. Dazu brauchen Sie nur den Namen der betroffenen Tabelle als ersten Parameter anzugeben. Zur Bedeutung des zweiten Parameters kommen wir später; vorerst können Sie den Wert **dbOpenDynaset** als Standard verwenden.

Nach dem Ausführen dieser Anweisung macht **rst** prinzipiell den kompletten Inhalt der angegebenen Datenherkunft, hier der Tabelle **tblArtikel** verfügbar. Danach folgt ein Konstrukt, dass Sie noch ganz oft einsetzen werden: Das Durchlaufen der Datensätze einer Datensatzgruppe. Dies geschieht innerhalb einer **Do While**-Schleife. Diese prüft jeweils zu Beginn den Wert der Eigenschaft **EOF** der Recordset-Objekts **rst**.

Hierzu zur Erläuterung: Der Datensatzzeiger kann sich vor dem ersten Datensatz einer Datensatzgruppe befinden, aber auch hinter dem letzten Datensatz. Wenn **EOF** den Wert **True** zurückgibt, befindet sich der Datensatzzeiger hinter dem letzten Datensatz. In diesem Fall sollen die in der **Do While**-Schleife enthaltenen Anweisungen nicht mehr ausgeführt werden.

Dies wird jedoch zu Beginn nicht der Fall sein. Die Beispieltabelle enthält 77 Datensätze, die erst ein-

mal durchlaufen werden wollen. Damit der Datensatzzeiger, der beim Öffnen einer Datensatzgruppe standardmäßig auf den ersten Datensatz zeigt, sich zum nächsten Datensatz der Datensatzgruppe bewegt, rufen Sie die Methode **MoveNext** des **Recordset**-Objekts **rst** auf.

Vorher jedoch soll eine weitere Anweisung ein paar Daten des aktuellen Datensatzes im Direktfenster ausgeben, und zwar die Werte der Felder **ArtikelID** und **Artikelname**. Dies gelingt auch auf Anhieb, wie Bild 1 zeigt. Die **Debug.Print**-Anweisung gibt die beiden Werte eines Datensatzes jeweils in einer eigenen Zeile aus.

Auf Daten zugreifen

Im Beispiel haben Sie bereits eine Methode kennengelernt, auf die Feldinhalte des aktuellen Datensatzes zuzugreifen: die Ausrufezeichen-Syntax (**rst!Artikelname**). Insgesamt gibt es die folgenden Möglichkeiten, auf den Inhalt eines Feldes zuzugreifen:

```
rst.Fields("Artikelname").Value
rst.Fields("Artikelname")
rst.Fields(1).Value
rst.Fields(1)
rst(1).Value
rst(1)
rst("Artikelname").Value
rst("Artikelname")
rst!Artikelname.Value
rst!Artikelname
```

Die erste Variante verwendet explizit die **Fields**-Auflistung des **Recordset**-Objekts. Mit dieser können Sie auf alle Felder zugreifen, entweder über den Namen eines Feldes oder über den Index (**0** für das erste Feld, **1** für das zweite Feld und so weiter).

Mit **Value** greifen Sie explizit auf den Wert zu, ohne **Value** wird der Wert der Standardeigenschaft des **Field**-Objekts ausgegeben – und diese lautet auch **Value**.

Fields ist wiederum die Standardauflistung des **Recordset**-Objekts, sodass Sie hier **Fields(1)** einfach durch **(1)** ersetzen können. Alternativ verwenden Sie auch hier den Feldnamen, also **Fields("Artikelname")** oder **("Artikelname")**. Schließlich gibt es die Ausrufezeichen-Syntax, wie Sie auch im ersten Beispiel zum Einsatz kam.

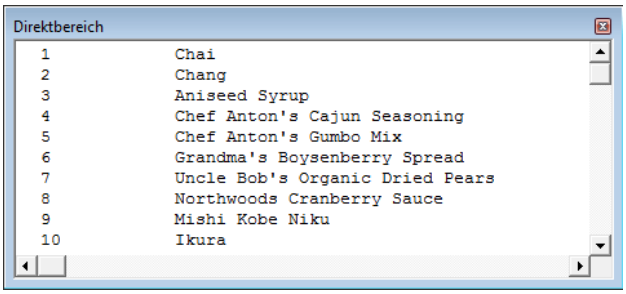


Bild 1: Ausgabe von Daten im Direktfenster

Recordset und Database schließen und leeren

Nach der Verwendung sollten Sie das **Recordset**-Objekt schließen und die Objektvariable leeren, die **Database**-Variable brauchen Sie nur zu leeren. Die drei notwendigen Anweisungen sehen wie folgt aus:

```
rst.Close
Set rst = Nothing
Set db = Nothing
```

Da diese immer gleich sind, führen wir sie in den folgenden Beispielen nicht explizit auf.

Vor und zurück

Eine der Methoden zum Bewegen des Datensatzzeigers haben Sie mit **MoveNext** bereits kennengelernt. Es gibt noch weitere ähnliche Methoden:

- **MovePrevious**: Bewegt den Datensatzzeiger zum vorherigen Datensatz.
- **MoveFirst**: Positioniert den Datensatzzeiger auf den ersten Datensatz.
- **MoveLast**: Positioniert den Datensatzzeiger auf den letzten Datensatz.

Zu weit bewegt?

Genauso wie die oben bereits vorgestellte Eigenschaft **EOF** die Position hinter dem letzten Datensatz markiert, können Sie mit der Eigenschaft **BOF** prüfen, ob der Datensatzzeiger sich vor dem ersten Datensatz befindet.

Dies ist wichtig, wenn Sie die Daten einer Datensatzgruppe mit den oben genannten Methoden durchlaufen. Sollte sich der Datensatzzeiger etwa bereits vor dem ersten Datensatz befinden, löst ein erneuter Aufruf der Methode **MovePrevious** einen Fehler

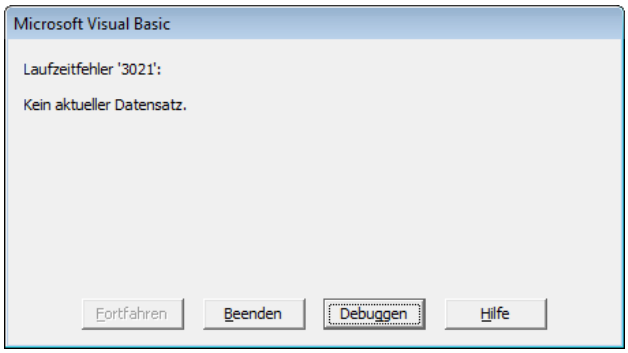


Bild 2: Fehler beim Bewegen des Datensatzzeigers

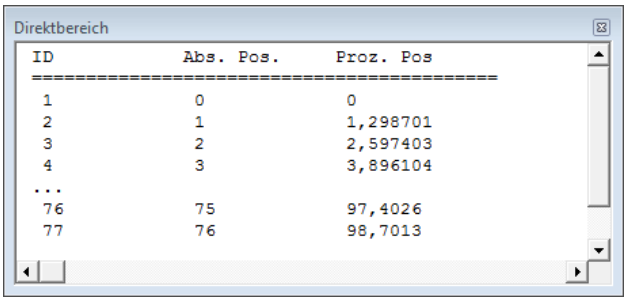


Bild 3: Ausgabe der absoluten und relativen Position des Datensatzzeigers

aus – genau wie der Aufruf von **MoveNext**, wenn der Datensatzzeiger sich bereits hinter dem letzten Datensatz befindet (siehe Bild 2).

Position des Datensatzzeigers

Manchmal kann es hilfreich sein, die Position des Datensatzzeigers zu kennen. Das **Recordset**-Objekt bietet dazu zwei Möglichkeiten an:

- **AbsolutePosition**: Liefert die genaue Position, wobei der Wert **0** dem ersten Datensatz entspricht.
- **PercentPosition**: Liefert die prozentuale Position.

Die Position können Sie beispielsweise beim Durchlaufen der Datensatzgruppe ausgeben lassen – hier mit absoluter und prozentualer Position:

```
Do While Not rst.EOF
    Debug.Print rst!ArtikelID, _
        rst.AbsolutePosition, rst.PercentPosition
    rst.MoveNext
Loop
```

Die Ausgabe für dieses Beispiel finden Sie in Bild 3. Sie können den Datensatzzeiger mit diesen beiden Eigenschaften auch positionieren. Wenn Sie etwa Informationen zum 50. Datensatz einer Datensatzgruppe benötigen, verwenden Sie folgende Anweisungen.

Wichtig ist, dass Sie wie beim Zählen der Datensätze mit **RecordCount** zuerst einmal zum letzten Datensatz springen müssen, um alle Datensätze zu erfassen:

```
rst.MoveLast
rst.AbsolutePosition = 50
Debug.Print rst!ArtikelID, rst.AbsolutePosition, _
     rst.PercentPosition
```

Punktlandung

Wer genau weiß, wieviele Datensätze er den Datensatzzeiger nach vorn oder hinten verschieben möchte, kann auch die **Move**-Methode einsetzen.

Diese erwartet zwei Parameter:

- Anzahl der zu überspringenden Datensätze
- Ausgangsdatensatz (wenn leer, wird die aktuelle Position verwendet)

Mit Kenntnis der **AbsolutePosition**-Eigenschaft lässt sich hier leicht experimentieren. Die folgende Prozedur springt zunächst 10 Sätze nach vorn, dann 5 zurück und schließlich 100 nach vorn.

Bei nur 77 Datensätzen in der betroffenen Tabelle gibt **rst.AbsolutePosition** den Wert **-1** aus und die **EOF**-Eigenschaft ist **True**.

```
Public Sub Springen()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("tblArtikel", _
        dbOpenDynaset)
    rst.Move 10
    Debug.Print rst.AbsolutePosition
    rst.Move -5
    Debug.Print rst.AbsolutePosition
    rst.Move 100
    Debug.Print rst.AbsolutePosition, rst.EOF
    rst.Close
    Set rst = Nothing
    Set db = Nothing
End Sub
```

Mehrfachauswahl in Listenfeldern auslesen

Listenfelder bieten im Gegensatz zur Datenblattansicht einen Vorteil: Sie können damit nicht nur einen, sondern auch mehrere Datensätze gleichzeitig auswählen. Dies bringt natürlich auch mehr Aufwand bei der Auswertung mit sich. Dieser Artikel zeigt, welche Auswahlmöglichkeiten Listenfelder liefern und wie Sie die ausgewählten Elemente per VBA-Code identifizieren.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1112_Mehrfachauswahl.mdb**.

Einsatzzwecke für die Mehrfachauswahl

Listenfelder unterscheiden sich von Kombinationsfelder dadurch, dass Sie mehrere Datensätze beziehungsweise Einträge gleichzeitig anzeigen können.

Warum diesen Vorteil nicht nutzen und auch gleich noch die Auswahl mehrerer Einträge gleichzeitig anbieten? Die Anwendungszwecke teilen sich auf zwei Bereiche auf, und meistens sind m:n-Beziehungen im Spiel:

- Auswahl eines oder mehrere Einträge, mit denen eine bestimmte Aktion durchgeführt werden soll, zum Beispiel zum Hinzufügen oder Entfernen von Einträgen aus einer Verteilerliste. Dabei zeigt ein Listenfeld die ausgewählten Einträge an, das andere die nicht ausgewählten Einträge. Die Auswahl der hinzuzufügenden oder zu entfernenden Einträge erfolgt mit der Maus, das Verschieben mit einem Klick auf eine entsprechende Schaltfläche.

- Darstellung der Elemente einer m:n-Beziehung: Hier kommt nur ein Listenfeld zum Einsatz. Dieses zeigt alle Datensätze der einen Seite der m:n-Beziehung an und aktiviert diejenigen Einträge, die aktuell verknüpft sind. Wenn Sie beispielsweise alle Länder auswählen möchten, in die Sie ein bestimmtes Produkt versenden, wäre dies ein geeignetes Mittel. Voraussetzung ist, dass die Menge der Einträge überschaubar ist, damit auch die markierten Einträge immer sichtbar sind.

Beispieltabellen

Im ersten Beispiel schauen wir uns die Mehrfachauswahl anhand der Tabelle **tblPublikationen** an, die lediglich aus den beiden Feldern **PublikationID** und **Publikation** besteht. Sie enthält einige Beispieldatensätze (siehe Bild 1).

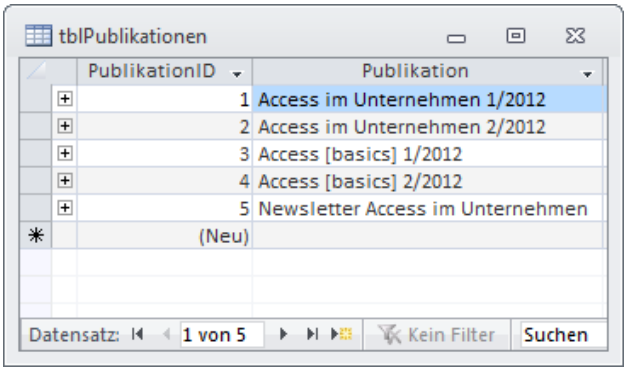


Bild 1: Tabelle **tblPublikationen** mit Beispieldaten

Das Formular **frmPublikationen** enthält ein Listenfeld namens **lstPublikationen**. Die Datensatzherkunft des Listenfeldes ist auf die Tabelle **tblPublikationen** eingestellt.

Da alle Felder und alle Datensätze dieser Tabelle angezeigt beziehungsweise als gebundene Spalte verwendet werden sollen, brauchen wir nicht extra eine Abfrage anzulegen, die Felder und/oder anzuzeigende Datensätze einschränkt.

Damit die erste Spalte mit den Werten des Feldes **PublikationID** nicht angezeigt wird, stellen Sie die Eigenschaften **Spaltenanzahl** und **Spaltenbreiten** auf die Werte **2** und **0cm** ein. Die erste Spalte wird somit automatisch ausgeblendet und die zweite über die volle Breite des Listenfeldes angezeigt (siehe Bild 2).

Mehrfachauswahl aktivieren

Die wichtigste Einstellung jedoch steckt im Bereich Andere des Eigenschaftsfensters und heißt **Mehrfachauswahl** (siehe Bild 3). Sie bietet die folgenden drei Einstellungsmöglichkeiten:

- **Keine**: Es kann nur jeweils ein einziger Eintrag markiert werden. Klickt man auf einen anderen Eintrag, wird die Markierung des bis dahin aktivieren Eintrags aufgehoben.

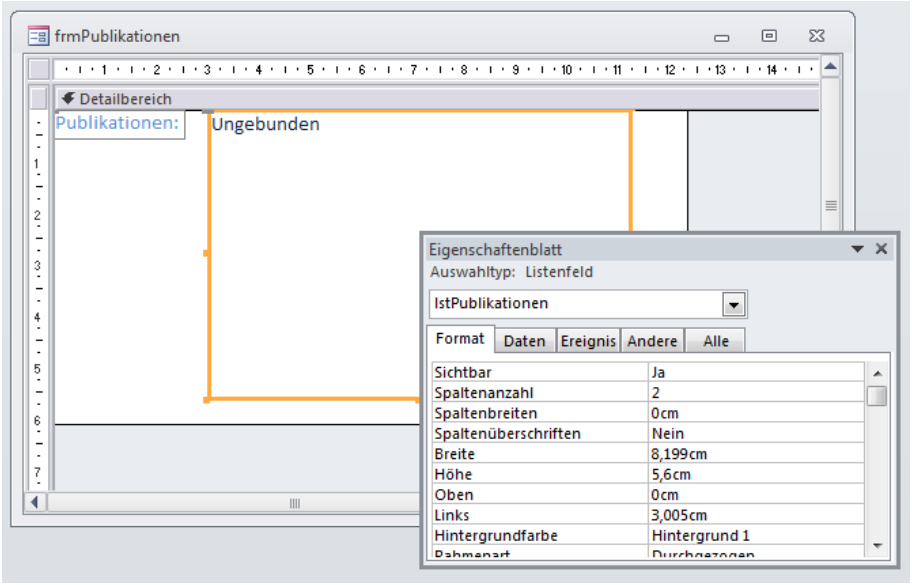


Bild 2: Das Formular **frmPublikationen** mit dem Listenfeld

Wenn Sie für die Eigenschaft **Mehrfachauswahl** den Eintrag **Einzeln** auswählen, gelingt dies jedoch nicht mehr.

In diesem Fall zeigt das Textfeld schlicht keinen Wert mehr an (siehe Textfeld **txtMehrfachauswahlEinzeln** und **lstMehrfachauswahlEinzeln** im Beispielformular **frmPublikationen**).

Auch bei der Option **Erweitert** für die Eigenschaft **Mehrfachauswahl** liefert das Listenfeld keinen Wert mehr zurück.

- **Einzeln**: Einträge werden durch Anklicken ausgewählt und auch wieder abgewählt. Es kann immer nur ein Eintrag zur gleichen Zeit aus- und abgewählt werden.

- **Erweitert**: Funktioniert beim einfachen Anklicken wie bei der Option **Keine**: Es wird immer nur ein Eintrag markiert.Erst wenn Sie die Steuerungs- oder die Umschalttaste hinzunehmen, passiert mehr: Bei gedrückter Steuerungstaste wählen Sie Einträge wie bei der Option **Einzeln** aus oder ab. Bei gedrückter Umschalttaste und dem Anklicken zweier Einträge werden alle Einträge vom zuerst bis zum zuletzt markierten Eintrag markiert. Die Option **Erweitert** liefert also praktisch das gleiche Verhalten wie bei der Ordner- und Dateiauswahl im Windows-Explorer.

Auswerten der Mehrfachauswahl

Interessant wird nun die Auswertung der mit der Mehrfachauswahl festgelegten Einträge. Bei der Einfachauswahl war dies einfach: Dort brauchten Sie nur den Wert des Steuerelements zu referenzieren.

Wenn Sie also in einem Textfeld namens **txtPublikation** den Wert des gebundenen Feldes **PublikationID** anzeigen wollten, brauchten Sie der Eigenschaft Steuerelementinhalt dieses Textfeldes einfach nur den Wert **=[lstPublikationen]** hinzuzufügen. Der Inhalt dieses Feldes wird gleichzeitig mit dem Listenfeld aktualisiert (siehe Bild 4).

Hier sind ein paar Zeilen VBA-Code erforderlich, mit denen Sie die markierten Einträge des Listenfeldes durchlaufen und diese beispielsweise zu einer Liste zusammensetzen. Diese Prozedur soll jeweils beim Aus- oder Abwählen eines jeden Eintrags ausgelöst werden.

Das Anklicken eines Eintrags löst sowohl das Ereignis **Nach Aktualisierung** als auch **Beim Klicken** aus. Wir nutzen das Ereignis **Nach Aktualisierung**, um die ausgewählten Daten zu ermitteln. Dazu legen Sie eine entsprechende Ereignisprozedur an: Wäh-

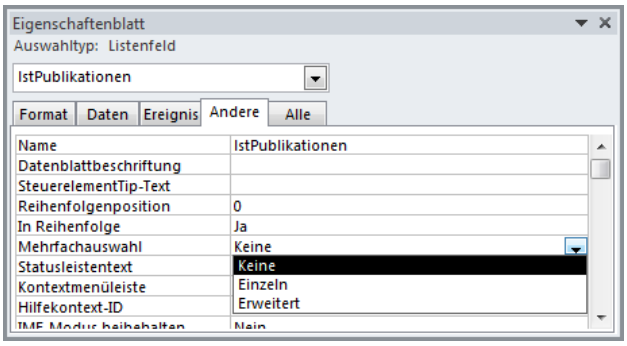


Bild 3: Einstellen der Eigenschaft **Mehrfachauswahl**

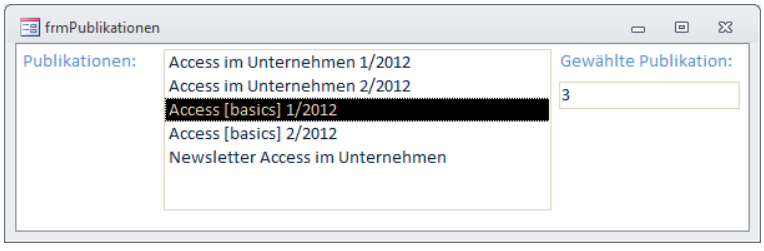


Bild 4: Anzeige der ausgewählten Publikation im Textfeld

DAO: Recordsets filtern, durchsuchen und sortieren

Mit Recordsets können Sie die Daten aus Tabellen und Abfragen für die Bearbeitung und Betrachtung unter VBA verfügbar machen. Dieser Artikel zeigt, wie Sie ein Recordset gezielt mit den benötigten Daten füllen, wie Sie diesen Datenbestand durchsuchen und wie Sie die Daten sortieren und filtern.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1112_DAO_Recordsets_Suchen.mdb**.

Es gibt diesmal ausschließlich Code-Beispiele. Wenn Sie diese ausprobieren möchten, legen Sie in einer Datenbank ein neues Standardmodul an, geben die Listings ein, platzieren die Einfügemarke irgendwo im Code der auszuführenden Routine und betätigen die Taste **F5** (oder **F8** für den Einzelschrittmodus).

Recordset gezielt füllen

Bereits mit der Angabe der Datenherkunft eines Recordsets beim Öffnen mit der **OpenRecordset**-Funktion können Sie die benötigten Daten nach Belieben einschränken. Dabei gelten die gleichen Regeln, die auch für Abfragen gelten: Sie können sowohl gezielt eines oder mehrere Felder als auch einen oder mehrere Datensätze in das **Recordset**-Objekt einlesen. Dabei verwenden Sie natürlich eine oder mehrere Tabellen als Datenherkunft, die typischerweise miteinander verknüpft sind.

Zur Festlegung der Datenherkunft haben Sie die folgenden drei Möglichkeiten:

- Sie geben einfach den Namen einer Tabelle an. Das Recordset enthält alle Datensätze mit allen Feldern der angegebenen Tabelle.
- Sie definieren die Daten für das Recordset mithilfe einer Abfrage, in der Sie die Tabellen, die Felder und die Kriterien für die aufzunehmenden Daten festlegen.
- Sie legen einen SQL-Ausdruck fest, der angibt, welche Daten aus welchen Tabellen und Feldern angezeigt werden sollen – und dort legen Sie ebenfalls die Kriterien fest.

Später lernen Sie, dass man Recordsets auch auf Basis anderer Recordsets öffnen kann.

Tabellen und Felder vorauswählen

Wenn Sie nicht alle Felder aller Datensätze einer Tabelle im Recordset verfügbar möchten, verwenden Sie am einfachsten eine Abfrage, die nur die benötigten Felder zurückliefert. In dieser Abfrage können Sie auch mehrere, normalerweise miteinander verknüpfte Tabellen kombinieren und Felder aus mehreren Tabellen zusammenführen.

Wenn Sie in einem Recordset beispielsweise auf eine Liste von Personen samt Anrede zugreifen möchten, wobei die Anreden sich in einer Lookup-Tabelle befinden, mit der die Personen-Tabelle per Fremdschlüsselfeld verknüpft ist, legen Sie zuerst eine Abfrage wie in Bild 1 an.

Wenn Sie diese Daten nun ausgeben möchten, verwenden Sie eine Prozedur wie die folgende:

```
Public Sub PersonenUndAnreden()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("qryPersonen", _
        dbOpenDynaset)
    Do While Not rst.EOF
        Debug.Print rst!Anrede, rst!Vorname, _
            rst!Nachname
        rst.MoveNext
    Loop
    rst.Close
```

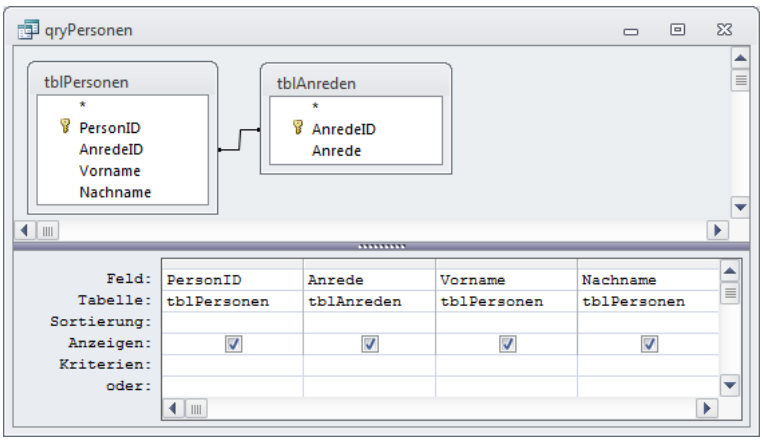


Bild 1: Zusammenführen von Personen und Anreden für ein Recordset

```
    Set rst = Nothing
    Set db = Nothing
End Sub
```

Sie sehen: Das Ausgeben der Daten erfolgt genauso, als ob diese sich in einer einzigen Tabelle befinden würden.

Filtern und Sortieren beim Öffnen des Recordsets

Wenn Sie nicht alle Datensätze einer Tabelle in ein Recordset aufnehmen möchten, können Sie diese ebenfalls mithifle einer Abfrage filtern. Dazu legen Sie einfach ein entsprechendes Kriterium für die betroffenen Felder fest. Wie dies gelingt, erfahren Sie beispielsweise im Artikel **SQL: SELECT-Abfragen**.

Nachträgliches Filtern eines Recordsets

Wenn Sie ein Recordset bereits geöffnet haben und die enthaltenen Datensätze auf Basis eines bestimmten Kriteriums weiter filtern möchten, können Sie die **Filter**-Eigenschaft einsetzen.

Die **Filter**-Eigenschaft erwartet eine gültige SQL-Bedingung als Ausdruck, also beispielsweise **Anrede = 'Frau'**. Nachdem sie die Filter-Eigenschaft gesetzt hat, öffnet die folgende Prozedur ein zweites Recordset-Objekt namens **rstNurFrauen** auf Basis des ersten Recordsets **rstAlle** und wendet den Filter darauf an:

```
Public Sub PersonenNachAnrede()
    Dim db As DAO.Database
    Dim rstAlle As DAO.Recordset
    Dim rstNurFrauen As DAO.Recordset
    Set db = CurrentDb
    Set rstAlle = db.OpenRecordset("qryPersonen", _
        dbOpenDynaset)
    rstAlle.Filter = "Anrede = 'Frau'"
    Set rstNurFrauen = rstAlle.OpenRecordset
    Do While Not rstNurFrauen.EOF
        Debug.Print rstNurFrauen!Anrede, _
            rstNurFrauen!Nachname
        rstNurFrauen.MoveNext
    Loop
    rstNurFrauen.Close
    Set rstNurFrauen = Nothing
    rstAlle.Close
    Set rstAlle = Nothing
    Set db = Nothing
End Sub
```

Es ist kein guter Programmierstil, aber Sie kommen bei dieser Technik auch mit einer einzigen Recordset-Variablen aus:

```
Public Sub PersonenNachAnrede_II()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("qryPersonen", _
        dbOpenDynaset)
    rst.Filter = "Anrede = 'Frau'"
    Set rst = rst.OpenRecordset
    Do While Not rst.EOF
        Debug.Print rst!Anrede, rst!nachname
        rst.MoveNext
    Loop
    rst.Close
    Set rst = Nothing
    Set db = Nothing
End Sub
```

Nachträgliches Sortieren eines Recordsets

Mit der gleichen Vorgehensweise können Sie ein Recordset auf nachträglich sortieren. Dazu stellen Sie mit der **Sort**-Eigenschaft des ersten Recordsets die Sortierung ein.

Dazu verwenden Sie den Namen des Feldes und eines der Schlüsselwörter **ASC** oder **DESC** die Reihenfolge der Sortierung. Wenn gleich nach mehreren Feldern sortieren möchten, verwenden Sie mehrerer solcher Ausdrücke und trennen diese durch Kommata.

Nach dem Festlegen der **Sort**-Eigenschaft erstellen Sie wiederum ein neues **Recordset**-Objekt auf Basis des ersten Recordsets. In diesem Fall speichern wir den Verweis darauf direkt in der gleichen Recordset-Variablen:

```
Public Sub PersonenSortiert()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("qryPersonen", _
        dbOpenDynaset)
    rst.Sort = "Vorname ASC, Nachname ASC"
    Set rst = rst.OpenRecordset
    Do While Not rst.EOF
        Debug.Print rst!Vorname, rst!nachname
        rst.MoveNext
    Loop
    rst.Close
```

SQL: SELECT-Abfragen, Teil I

Früher oder später ist es soweit: Sie formulieren Ihre erste eigene SQL-Abfrage. Der Abfrage-Editor kann den Zeitpunkt aufschieben, aber irgendwann wollen Sie beispielsweise für den Einsatz in einem DAO-Recordset mal eben selbst eine SELECT-Anweisung zusammenstellen. Glücklicherweise ist SQL kein Buch mit sieben Siegeln. Die Grundlagen zum Auswählen von Daten mit der SELECT-Anweisung lernen Sie in diesem Artikel kennen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1112_SQLSELECT.mdb**.

SQL – was ist das?

SQL steht für Structured Query Language, was wiederum Strukturierte Abfragesprache bedeutet. SQL kommt in mehreren Dialekten, die sich von Hersteller zu Hersteller unterschieden – Microsoft's SQL Server, MySQL und andere Systeme verwenden alle Abfragesprachen mit kleinen Unterschieden. So auch Access: Es setzt Access-SQL ein, das weitgehend mit ANSI-89 Level 1 kompatibel ist. Weitgehend bedeutet hier, dass Access-SQL einige Elemente hinzufügt (wie etwa das Referenzieren von Formularen und Steuerelementen aus der SQL-Anweisung heraus), andere Features aber nicht konsequent übernimmt. Tatsache ist, dass grundlegende SQL-Kenntnisse anderer Dialekte bei Verwendung von Access-SQL nicht schaden können.

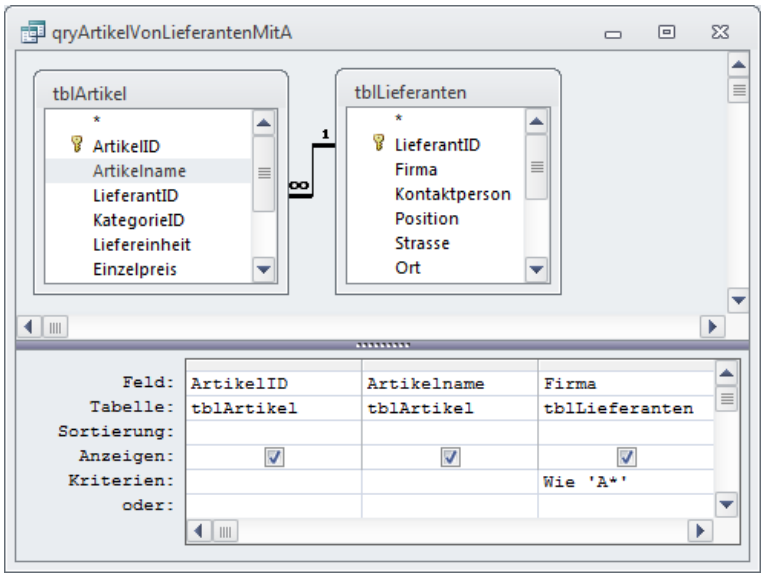


Bild 1: Abfrage mit zwei Tabellen und einem Kriterium in der Abfrage-Entwurfsansicht ...

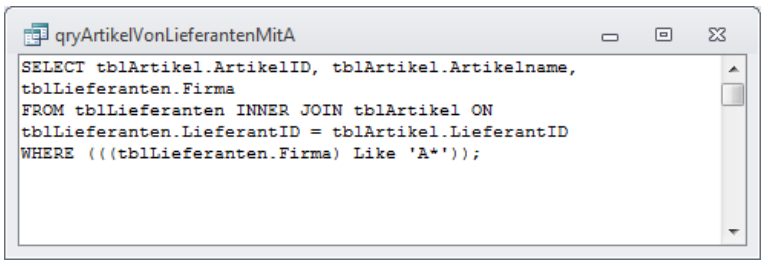


Bild 2: ... und in der SQL-Ansicht.

Access-SQL besteht im wesentlichen aus zwei Teilen: **DDL** (**Data Definition Language**) enthält Anweisungen, mit denen Sie Datenbanken, Tabellen, Felder und Indizes definieren. **DML** (**Data Manipulation Language**) wiederum liefert die Techniken zum Abfragen und zum Ändern von Daten, also zum Anlegen, Bearbeiten oder Löschen von Datensätzen.

Den ersten Teil der **Data Manipulation Language**, also das Abfragen von Daten, deckt weitgehend der **SELECT**-Befehl ab. Mit diesem können Sie alle Abfragen mit SQL definieren, die sich auch mit der Abfrage-Entwurfsansicht erstellen lassen.

Ein Beispiel ist die Abfrage aus Bild 1. Die Abfrage enthält zwei Tabellen und ein Kriterium. Wenn Sie im Kontextmenü der Titelleiste des Abfragefensters den Eintrag **SQL-Ansicht** auswählen, erhalten Sie die Ansicht aus Bild 2. Das sieht für jemanden, der noch nie eine SQL-Anweisung gesehen hat, möglicherweise etwas kompliziert aus. In der Tat enthält dieser kleine Text alle Informationen, die Sie über die Abfrage-Entwurfsansicht eingetragen haben.

Wohin damit?

SQL-Ausdrücke, die in einfachen Fällen etwa **SELECT * FROM tblArtikel** lauten und alle Datensätze der Tabelle **tblArtikel** zurückliefern, können Sie an sehr vielen Stellen einsetzen.

Wenn Sie beispielsweise eine UNION-Abfrage wie in **Daten zusammenführen mit UNION-Abfragen** be-

schrieben erstellen, kommen Sie um den Einsatz der SQL-Ansicht einer Abfrage nicht herum.

Wenn Sie die Datenherkunft für Formulare oder Berichte festlegen oder die Datensatzherkunft für Kombinations- und Listenfelder, wird diese ebenfalls als SQL-Ausdruck gespeichert (siehe Bild 3). Erst wenn Sie die Abfrage unter einem eigenen Namen speichern, sodass diese im Datenbankfenster oder im Navigationsbereich sichtbar wird, enthält auch die entsprechende Eigenschaft diesen Namen.

Auch die Eigenschaft **Datensatzherkunft** von Tabellenfeldern, für die Sie mit dem Nachschlagefeld-Assistenten ein Nachschlagefeld eingerichtet haben, wird mit dem entsprechenden SQL-Ausdruck gefüllt (siehe Bild 4).

Sollten Sie mit DAO arbeiten, um per VBA-Code auf die Daten in den Tabellen Ihrer Datenbank zuzugreifen, können Sie ebenfalls sowohl den Namen von Tabellen oder Abfragen, aber auch einen expliziten SQL-Ausdruck angeben (mehr dazu beispielsweise im Artikel **DAO: Recordsets öffnen, lesen und bearbeiten**).

Für Aktionsabfragen wie die Anfüge-, die Lösch-, die Aktualisierungs- und die Tabellenerstellungsabfrage gibt es sogar noch weitere Möglichkeiten:

- Sie können solche Abfragen direkt als Parameter der **RunSQL**-Methode des **DoCmd**-Objekts angeben.
- Auch die **Execute**-Methode des **Database**-Objekts ist für das Ausführen von Aktionsabfragen im SQL-Format prädestiniert.
- Und schließlich bieten auch Makros eine Möglichkeit, direkt SQL-Ausdrücke auszuführen – nämlich mit der Makroaktion **AusführenSQL** (siehe Bild 5).

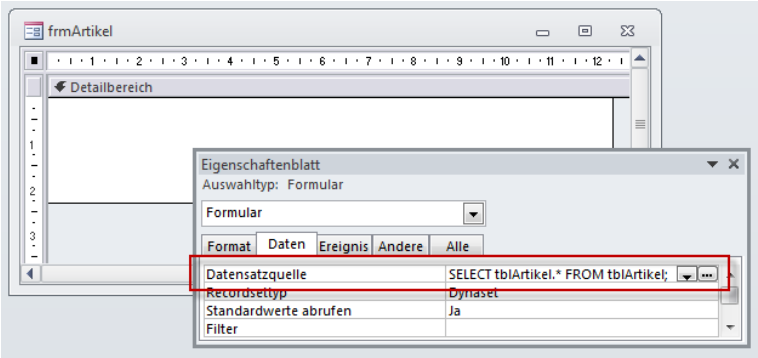


Bild 3: SQL-Ausdruck als Wert der Eigenschaft **Datenherkunft** eines Formulars

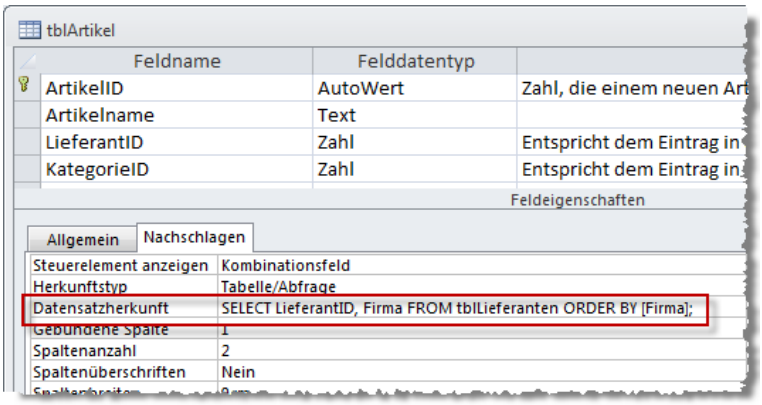


Bild 4: SQL-Ausdrücke in Nachschlagefeldern

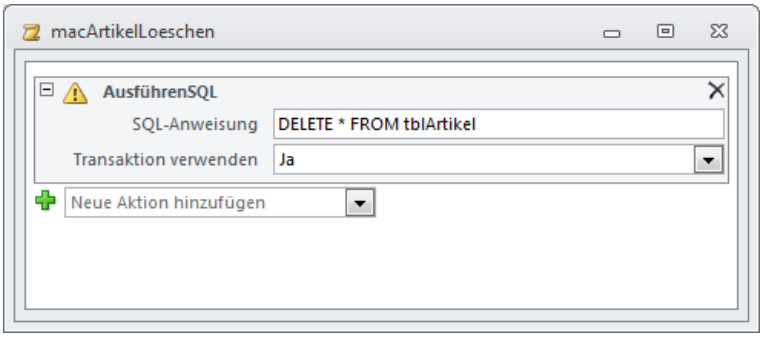


Bild 5: Aktionsabfrage per Makro

SQL-Ansicht anzeigen

Die SQL-Ansicht können Sie überall dort aktivieren, wo die Abfrage auch in der Entwurfsansicht erscheint. Je nach Access-Version gibt es dafür verschiedene Einträge im Menü beziehungsweise im Ribbon.

Daher verwenden Sie am einfachsten die versionsübergreifende Methode: den bereits erwähnten Eintrag **SQL-Ansicht** des Kontextmenüs der Titelzeile einer Abfrage.

Alle Daten einer Tabelle ermitteln

Wenn Sie einer Datenherkunft eines Formulars oder Berichts, der Datensatzherkunft eines Kombinations- oder Listenfeldes oder einer **OpenRecordset**-Methode alle Datensätze einer Tabelle

übergeben möchten, reicht es eigentlich aus, den Namen der Tabelle anzugeben. Die entsprechende SQL-Anweisung, die alle Felder für alle Datensätze der angegebenen Tabelle zurückliefert, besteht aus dem Tabellennamen und vorangestelltem **SELECT * FROM**:

```
SELECT * FROM tblArtikel
```

Dies ist jedoch nur die Kurzform. Der Vollständigkeit halber sollten Sie dem Sternchen als Platzhalter für alle Felder noch den Tabellennamen voranstellen:

```
SELECT tblArtikel.* FROM tblArtikel
```

Nur bestimmte Felder ausgeben

Wenn Sie nicht alle Felder der Tabelle ausgeben möchten, sondern nur bestimmte Felder, geben Sie diese als komma-separierte Liste anstelle des Sternchens an. Dies sieht für die beiden Felder **ArtikelID** und **Artikelname** etewa so aus:

```
SELECT ArtikelID, Artikelname FROM tblArtikel;
```

Wenn Sie diese Abfrage mit der Abfrage-Entwurfsansicht anlegen (siehe Bild 6) und dann zur SQL-Ansicht wechseln, sieht der SQL-Ausdruck so aus:

```
SELECT tblArtikel.ArtikelID, tblArtikel.Artikelname
FROM tblArtikel;
```

Zwingend notwendig wird dies, wenn die Abfrage Daten aus mehreren Tabellen anzeigen soll und Felder gleichen Namens in mehr als einer Tabelle vorkommen.

Ein Beispiel ist die folgende Abfrage (die Details zu den Schlüsselwörtern **INNER JOIN** et cetera klären wir später):

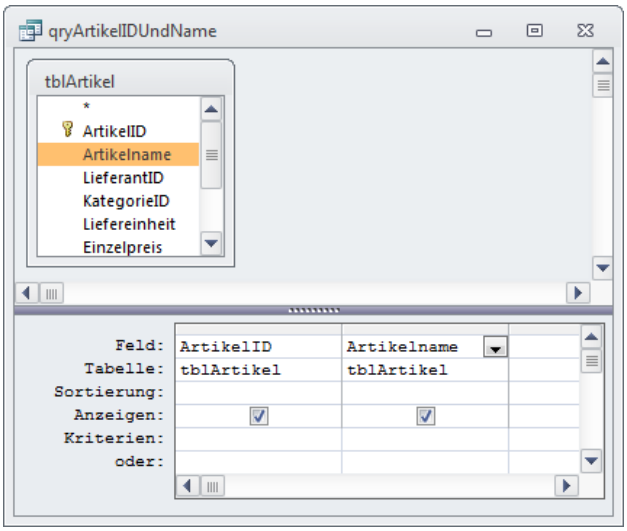


Bild 6: Abfrage mit nur zwei Feldern der Quelltabelle

```
SELECT ArtikelID, Artikelname, LieferantID
FROM tblLieferanten
INNER JOIN tblArtikel
ON tblLieferanten.LieferantID =
tblArtikel.LieferantID;
```

Diese Abfrage liefert beim Versuch, in die Datenblatt- oder Entwurfsansicht zu wechseln, die Fehlermeldung aus Bild 7. Um dies zu verhindern, reicht es aus, Felder mit mehrfach vorkommenden Feldnamen durch Voranstellen des Tabellennamens zu kennzeichnen.

Feldnamen mit Sonderzeichen

Auch wenn es normalerweise nicht der Fall sein sollte: Gelegentlich werden Sie auf Feldnamen treffen, die Sonderzeichen wie das Leerzeichen, Minuszeichen et cetera enthalten. Wenn also beispielsweise das Feld **ArtikelID** den Namen **Artikel-ID** erhält (wie in der Tabelle **tblArtikelAlt** der Beispieldatenbank) und Sie eine Abfrage wie folgt ausführen, treffen Sie auf Probleme – und zwar in Form der Abfrage eines

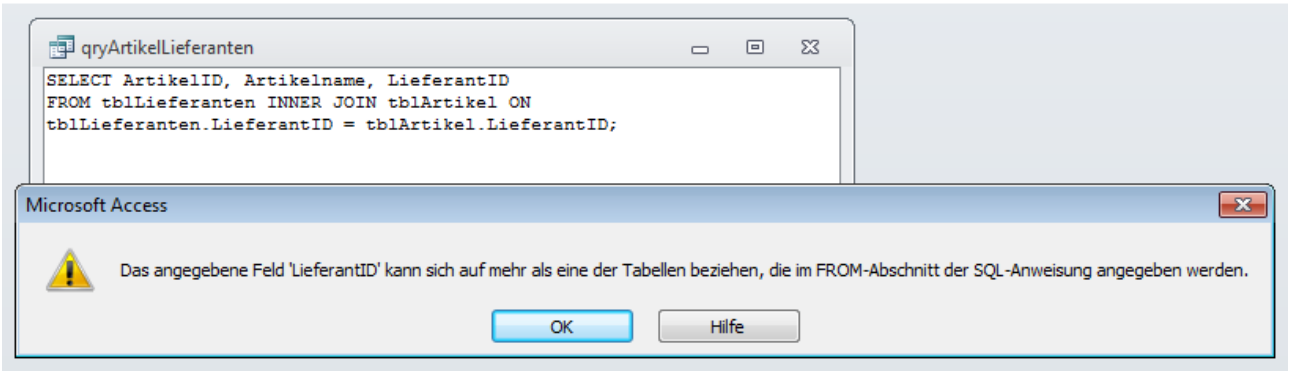


Bild 7: Fehlermeldung beim Einsatz einer Abfrage mit Feldern, die in mehreren beteiligten Tabellen vorkommen

Kunden per Bezeichnung verwalten

Beim Umgang mit Kundendaten benötigen Sie oft eine allgemeine Bezeichnung eines Kunden. Diese soll beispielsweise als Überschrift im Formular zur Verwaltung der Stammdaten eines Kunden dargestellt werden oder als Wert zur Auswahl in einem Kombinationsfeld angeboten werden. Praktisch wäre es, diesen Ausdruck etwa in einem Abfragefeld in der Form <Firma> (<Nachname>, <Vorname>) oder ähnlich abzubilden. Das gelingt jedoch nicht so leicht, weil ein Kunde mal mit, mal ohne Firma kommt. Und was geschieht überhaupt, wenn zwei Kunden den gleichen Namen haben? Diese und weitere Fragen beantwortet dieser Artikel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1112_KundePerBezeichnung.mdb** und **1112_SQLSELECT_Extern.mdb**.

Kundenbezeichnung erwünscht

Ein Einsatzzweck für eine Kundenbezeichnung, die sich aus der Firma des Kunden und dem Namen des Ansprechpartners zusammensetze, wäre etwa die Darstellung in einem Kombinationsfeld zur Auswahl eines Kunden etwa in einem Bestellformular. Normalerweise werden solche Kombinationsfelder immer mit der Kundentabelle als Datensatzherkunft und dem Nach- und dem Vornamen als anzuzeigendem Wert dargestellt.

Dabei fällt allerdings die Firmenbezeichnung weg. Wenn man die Firmenbezeichnung, die aber in der Regel nur bei einem Teil der Kunden vorliegt, hinzufügen möchte – etwa in der Form **Minhorst, André (Redaktionsbüro Minhorst)** oder **Redaktionsbüro Minhorst (Minhorst, André)**, muss man einen **Wenn**-Ausdruck in der Abfrage verwenden.

Andersherum kann es auch sein, dass zu einem Kunden gar kein Ansprechpartner vorliegt, sondern nur der Firmenname. Gegebenenfalls möchten Sie auch die Kundennummer in den Ausdruck für ein Kombinationsfeld oder für die Titelzeile des Detailformulars integrieren.

Richtig unpraktisch wird es, wenn zwei Kunden den gleichen Namen haben. Dann können Sie, wenn nur Vorname und Nachname angezeigt werden, anhand des Eintrags im Kombinationsfeld nicht auf die Schnelle entscheiden, ob Sie auch den richtigen Kunden auswählen.

Und da in diesem Fall der Ausdruck dynamisch etwa durch Feld einer Abfrage ermittelt wird, lässt sich auch nichts an diesem Ausdruck ändern – es sei denn, Sie ändern die zugrunde liegenden Daten. Das ist wiederum unmöglich – es sei denn, Sie möchten die verschiedenen Michael Müllers in Ihrem Kundenstamm mit **Müller1**, **Müller2** und **Müller3** betiteln. Spätestens beim Erstellen von Schriftverkehr auf Basis dieser Daten kommt es zu Problemen.

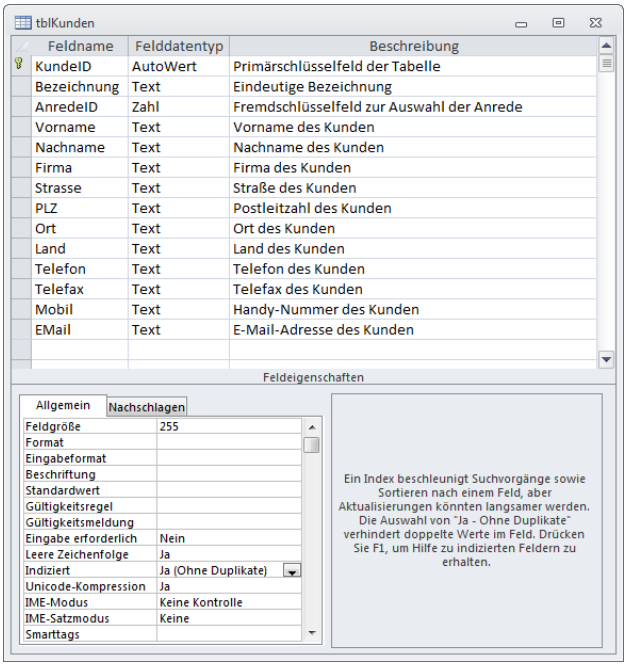


Bild 1: Kundentabelle mit Bezeichnungsfeld

Separates Bezeichnungsfeld

Die Tabelle **tblKunden** der Beispieldatenbank verfügt neben den üblichen Feldern über ein Feld namens Bezeichnung.

Dieses Feld soll eine eindeutige Bezeichnung eines jeden Kunden gewährleisten und wurde deshalb mit einem eindeutigen Index versehen – dies erreichen Sie beispielsweise durch das Einstellen der Eigenschaft **Indiziert** auf den Wert **Ja (Ohne Duplikate)**. Bild 1 zeigt den Entwurf der Tabelle mit dem Feld **Bezeichnung**.

Bezeichnung automatisch anpassen

Zur Anzeige der Daten der Tabelle **tblKunden** erstellen Sie ein Formular namens **frmKunden**, dass die Tabelle **tblKunden** als Datenherkunft verwendet. Ziehen Sie alle Felder der Tabelle in den Detailbereich des Formularentwurfs und ordnen Sie die Felder nach Wunsch an – beispielsweise so wie in Bild 2.

Typischerweise liegen schon Kundendaten vor, für die Sie die Bezeichnung nachträglich hinzufügen müssen, für neue Datensätze soll die Bezeichnung beim Speichern des Datensatzes eingetragen werden.

Wie aber soll die Bezeichnung nun aussehen? Dies ist letztlich egal, die paar Zeilen Code, welche die Bezeichnung zusammenstellen, können Sie leicht an die eigenen Bedürfnisse anpassen.

Wir entscheiden uns für das folgende Schema:

```
<KundeID> - <Firma> (<Nachname>, <Vorname>)
```

Für neue Datensätze soll die Bezeichnung zunächst beim Speichern des Datensatzes angelegt werden.

Damit noch nachträgliche Änderungen notwendig sind, bringen wir die notwendigen Codezeilen in der Ereignisprozedur unter, die durch das Ereignis Vor Aktualisierung ausgelöst wird.

Legen Sie diese Prozedur zunächst an, indem Sie in die Entwurfsansicht des Formulars wechseln und im Eigenschaftsfenster für den Eintrag **Vor Aktualisierung** den Wert **[Ereignisprozedur]** auswählen. Danach klicken Sie auf die Schaltfläche mit den drei Punkten (...) und füllen die nun im VBA-Editor angezeigte Prozedur wie folgt:

```
Private Sub Form_BeforeUpdate(Cancel As Integer)
    Me!Bezeichnung = Me!KundeID & " - " _
        & Me!Firma & " (" & Me!Nachname & ", " _
        & Me!Vorname & ")"
End Sub
```

Wenn Sie nun einen neuen Datensatz anlegen und diesen speichern (entweder durch Verlassen des Datensatzes, durch Betätigen der Tastenkombination **Strg + S** oder durch einen Mausklick auf den Datensatzmarkierer), wird die Bezeichnung wie in Bild 3 angepasst. Die einzige Anweisung dieser Ereignisprozedur fügt den Inhalt einiger Felder und einige Zeichen wie das Minuszeichen und die öffnende und schließende Klammer zusammen.

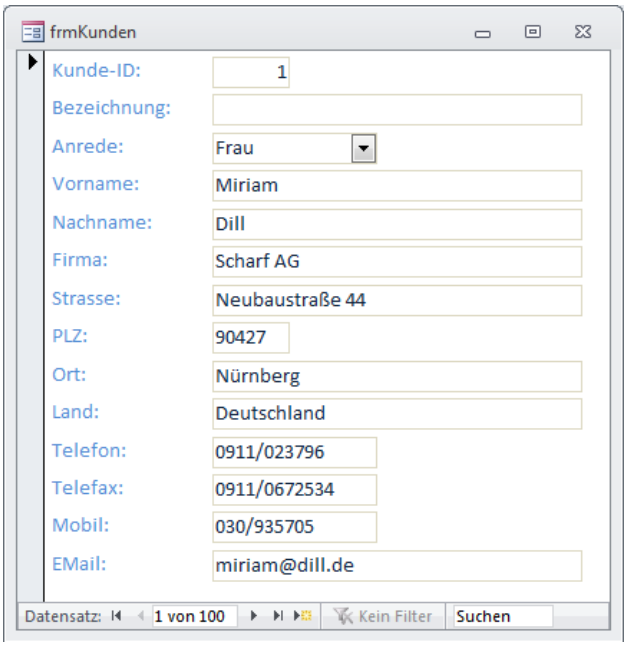


Bild 2: Formular zur Anzeige der Kundendaten

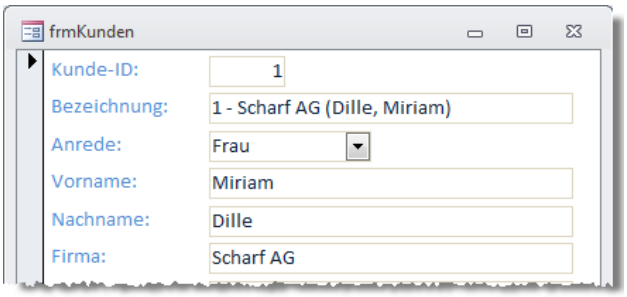


Bild 3: Kunde mit angepasster Bezeichnung

Bestehende Daten mit Bezeichnung versehen

Wenn Sie bereits bestehende Kundendaten um eine Bezeichnung ergänzen wollen, verwenden Sie am einfachsten eine Aktualisierungsabfrage. Diese verwendet etwa den gleichen Ausdruck wie die VBA-Prozedur von oben und sieht wie in Bild 4 aus. Nach dem Ausführen der Abfrage erhalten Sie das Ergebnis aus Bild 5 – natürlich nur im Optimalfall, nämlich wenn alle Felder der Datensätze gefüllt sind (dazu später mehr).

Bezeichnung einsetzen

Sie können die Bezeichnung nun beispielsweise in der Titelleiste des Detailfensters zur Anzeige von Kunden einsetzen. Damit diese immer die Daten zum

aktuellen Kunden anzeigt, legen Sie eine Ereignisprozedur an, die beim Anzeigen eines Datensatzes ausgelöst wird und die dementsprechend **Beim Anzeigen** heißt. Die Ereignisprozedur enthält nur eine einzige Zeile, die wie folgt aussieht:

```
Private Sub Form_Current()
    Me.Caption = _
        Me!Bezeichnung
End Sub
```

Das Formular zeigt die Bezeichnung nun in der Titelleiste an (siehe Bild 6).

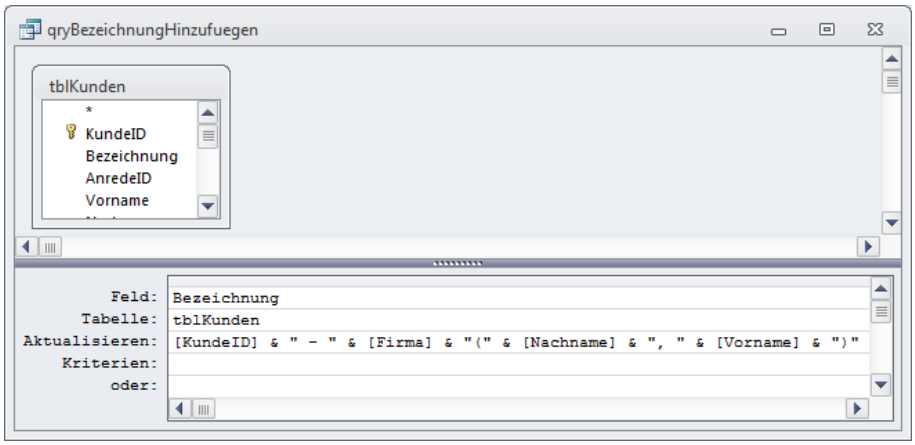


Bild 4: Aktualisieren aller Bezeichnungen der Kundentabelle

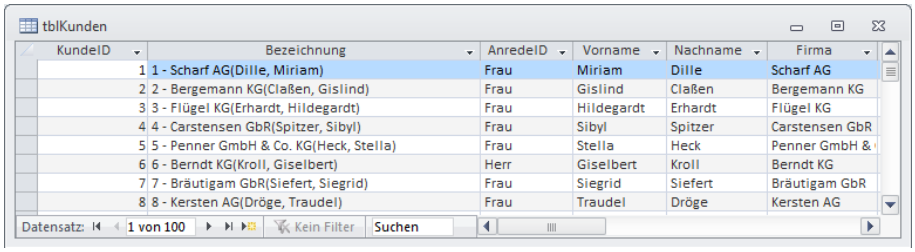


Bild 5: Kunden mit Bezeichnung

Bezeichnung in Kombinationsfeldern

Außerdem können Sie die Bezeichnung nutzen, wenn Sie einen Kunden zu irgendeinem Zweck per Kombinationsfeld auswählen möchten. Diesem weisen Sie dann als Datensatzherkunft die Abfrage aus Bild 7 zu.

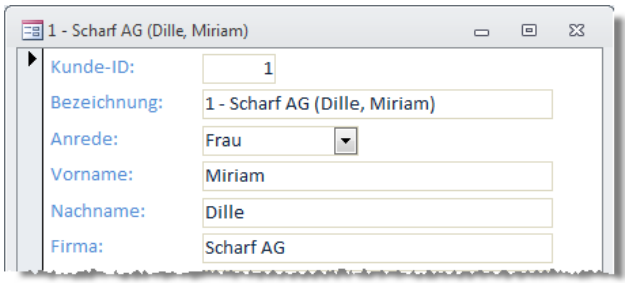


Bild 6: Anzeige der Bezeichnung im Formulartitel

Stellen Sie die Eigenschaften **Spaltenanzahl** und **Spaltenbreiten** auf die Werte **2** und **0cm** ein, damit das Formular das gebundene Feld, also **KundeID**, ausblendet und nur den Inhalt des Feldes **Bezeichnung** anzeigt.

In der Abfrage haben wir eine Sortierung nach dem Firmennamen eingestellt. Dies können Sie natürlich nach Ihren eigenen Vorstellungen anpassen.

Danach können Sie die Kunden mit einem Kombinationsfeld wie in Bild 8 auswählen.

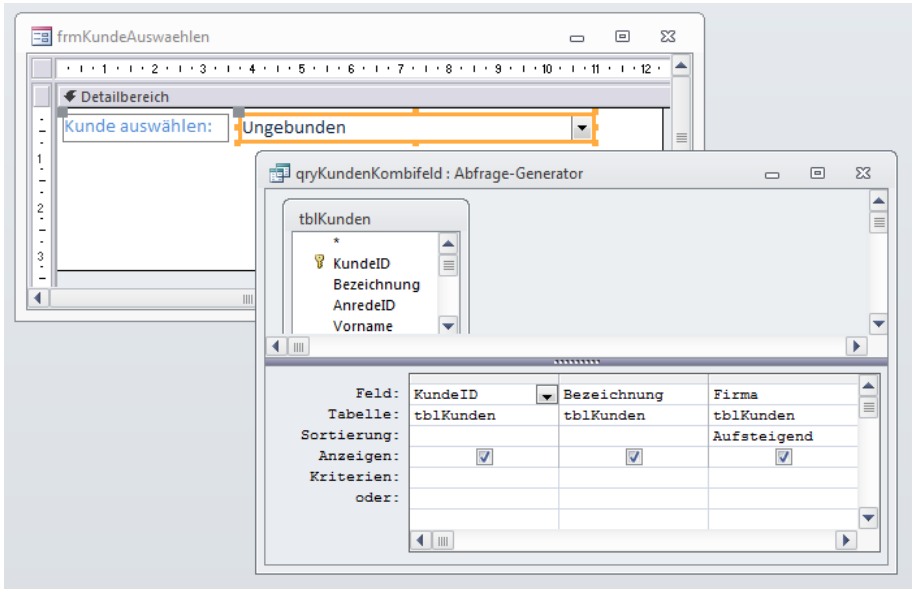


Bild 7: Kombinationsfeld mit Kundenbezeichnungen füllen

Bezeichnungen durchsuchen

Eine weitere Möglichkeit ist es, die Bezeichnung nach bestimmten Werten zu durch-

SQL: SELECT-Abfragen, Teil II

Im ersten Teil dieser Artikelreihe haben Sie erfahren, wie Sie SELECT-Abfrage grundsätzlich aufbauen und wie Sie die gewünschten Felder zur Abfrage hinzufügen. Im zweiten Teil kümmern wir uns um die andere Möglichkeit, die angezeigten Daten einzuschränken: nämlich durch das Filtern der Datensätze mit entsprechenden Bedingungen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1201_SQLBedingungen.mdb**.

Bedingungen in Abfragen

In der Entwurfsansicht von Abfragen stellen Sie Kriterien ganz einfach ein, indem Sie festlegen, für welches Feld das Kriterium gelten soll und den Vergleichswert unten in der Zeile **Kriterium** der entsprechenden Spalte eintragen (siehe Bild 1).

Wenn Sie Kriterien für verschiedene Spalten festlegen möchten, tragen Sie diese einfach in die gewünschten Spalten ein, und wenn Sie mehrere Kriterien durch **AND** oder **OR** verknüpfen möchten, bietet die Entwurfsansicht auch hierfür eine Möglichkeit.

Unter SQL ist das kaum komplizierter, und das beste ist: Wenn Sie sich einmal nicht sicher sind, wie ein SQL-Ausdruck mit einer Bedingung aussehen soll, dann stellen Sie die entsprechende Abfrage einfach in der Entwurfsansicht zusammen und wechseln dann in die SQL-Ansicht – dort finden Sie dann den passenden Ausdruck vor.

Leider gelingt dies nicht mit allen möglichen Bedingungen: Wenn Sie beispielsweise eine verschachtelte und mit **AND** und **OR** versehene Bedingung formulieren möchten, müssen Sie diese auch in der Entwurfsansicht einer Abfrage manuell eintragen.

Das WHERE-Schlüsselwort

Das wichtigste Element beim Zusammenstellen von **SELECT**-Abfragen mit Bedingungen heißt **WHERE**. Es leitet den Teil des SQL-Ausdrucks ein, der die Bedingungen enthält. Die SQL-Ansicht für die Abfrage aus der Abbildung liefert beispielsweise folgenden Ausdruck:

```
SELECT tblArtikel.ArtikelID, tblArtikel.Artikelname
FROM tblArtikel
WHERE (((tblArtikel.Artikelname)="Chai"));
```

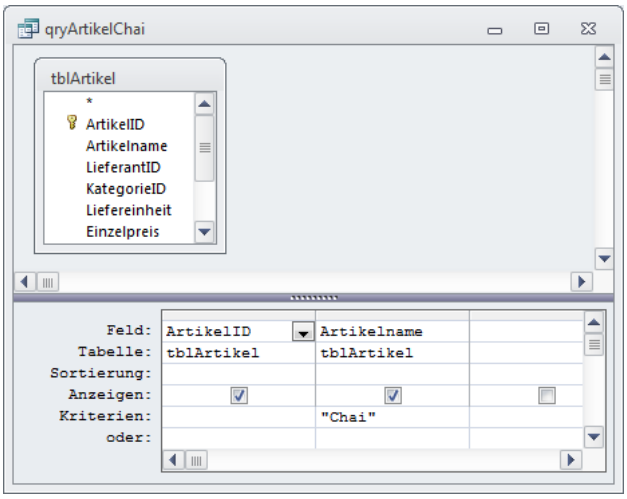


Bild 1: Eingeben von Abfragekriterien in der Entwurfsansicht einer Abfrage

Wie Sie bereits erfahren haben, kann es nie schaden, den Tabellennamen bei der Auflistung der anzuzeigenden Felder mit anzugeben – auf diese Weise verhindern Sie Fehler, wenn beispielsweise ein Feldname in mehreren per Abfrage verknüpften Tabellen vorkommt und nicht eindeutig zugeordnet werden kann. In der **WHERE**-Bedingung ist Access jedoch etwas übervorsichtig. Zwar schaden überzählige Klammern-Paare nicht, jedoch führen diese auch leicht zu Fehlern. Aus dem Ausdruck

```
WHERE (((tblArtikel.Artikelname)="Chai"));
```

können Sie auch leicht den folgenden machen, indem Sie einige Klammern entfernen:

```
WHERE tblArtikel.Artikelname="Chai";
```

Und wenn Sie, was bei einer Abfrage mit nur einer Herkunftstabelle leicht möglich ist, noch die Tabellenbezeichnung weglassen, erhalten Sie schließlich diese Abfrage als eine der einfachsten Varianten einer Auswahlabfrage mit Kriterium:

```
SELECT ArtikelID, Artikelname
FROM tblArtikel
WHERE Artikelname="Chai";
```

Fehlerquelle Zeichenkette

Hier ist gleich ein wichtiger Aspekt und eine häufige Fehlerquelle bei der Zusammenstellung von SQL-Abfragen sichtbar: Der Vergleichsausdruck wird in Anführungszeichen eingefasst. In der Entwurfsansicht einer Abfrage nimmt Access einem die Arbeit ab: Es nimmt Vergleichsausdrücke wie **Chai** für das Feld **Artikelname** klaglos entgegen, wandelt es dann aber in **"Chai"** um. Wenn Sie beispielsweise einen Zahlenwert als Kriterium für das Feld **ArtikelID** eingeben, geschieht dies nicht – die Zahl wird so beibehalten, und auch in der SQL-Ansicht erscheinen keine Anführungszeichen:

```
SELECT ArtikelID, Artikelname
FROM tblArtikel
WHERE ArtikelID=1;
```

Access prüft hier automatisch, welchen Datentyp das Feld hat, für das Sie die Bedingung anlegen, und fügt im Falle des Datentyps **Text** automatisch Anführungszeichen hinzu. Zuvor prüft es jedoch, ob der angegebene Ausdruck eventuell einem Funktionsnamen oder Feldnamen entspricht. Wenn Sie einen fixen Vergleichsausdruck für ein Textfeld festlegen, müssen Sie Anführungszeichen verwenden. Am besten noch einfache Anführungszeichen, dann können Sie einen solchen SQL-Ausdruck später besser in VBA verwenden:

```
SELECT ArtikelID, Artikelname
FROM tblArtikel
WHERE Artikelname='Chai';
```

Vergleichsausdruck zusammenstellen

Der Platz für die **WHERE**-Klausel ist immer direkt hinter der Auflistung der beteiligten Tabellen. Die **WHERE**-Klausel besteht aus einem oder mehreren Ausdrücken der Art **<Vergleichswert 1> <Vergleichsoperator> <Vergleichswert 2>**. Der Ausdruck **<Vergleichswert 1>** enthält meist den Feldwert, **<Vergleichsoperator>** einen Ausdruck wie zum Beispiel **=**, **>**, **>=**, **<**, **<=**, **<>** oder **LIKE** und **<Vergleichswert 2>** den statischen oder dynamischen Vergleichswert wie einen Zahlenwert, eine Zeichenkette oder auch einen Verweis auf ein anderes Feld der Datenherkunft der Abfrage. Unter Access können Sie sogar Verweise auf Inhalte von Steuerelementen in Formularen oder Funktionen angeben – sowohl eingebaute als auch benutzerdefinierte Funktionen sind dabei zulässig. Je nach dem Felddatentyp und dem verwendeten Vergleichsoperator sind verschiedene Vergleichsoperatoren möglich – mehr dazu weiter unten.

Vergleichsausdrücke mit OR oder AND verknüpfen

Wenn Sie gleich mehrere Vergleichsausdrücke angeben, müssen Sie sich entscheiden, ob die Abfrage alle Ergebnisse zurückgibt, für die mindestens einer der Ausdrücke wahr ist oder ob alle Ausdrücke wahr sein müssen – und gelegentlich werden Sie sogar Mischformen verwenden.

Wenn Sie beispielsweise alle Bestellungen ermitteln möchten, die durch die Kunden mit dem Wert **1** oder **2** im Feld **KundeID** getätigt wurden, verwenden Sie etwa folgende Abfrage:

```
SELECT BestellungID, KundeID
FROM tblBestellungen
WHERE KundeID=1 OR KundeID=2;
```

Wenn Sie einen Kunden ermitteln möchten, dessen **Firma** mit **A** und dessen Postleitzahl mit **1** beginnt und die Abfrage alle Felder zurückliefern soll, verwenden Sie etwa folgenden Ausdruck:

```
SELECT *
FROM tblKunden
WHERE Firma Like 'A*' AND PLZ Like '1*';
```

Klammernsetzung

Gelegentlich werden Sie Kombinationen aus mit **AND** und/oder **OR** verknüpften Bedingungen auswerten. Die Abfrage aus Bild 2 etwa soll alle Artikel liefern, deren Artikelname mit dem Buchstaben **C** beginnt und deren Feld **LieferantID** den Wert **1** oder **2** be-

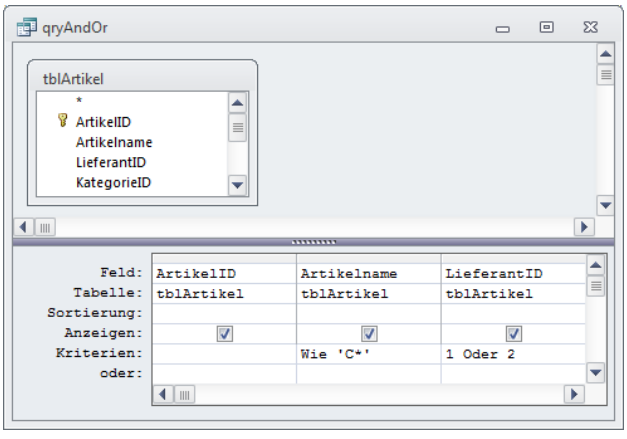


Bild 2: AND und OR in einer einzigen Abfrage

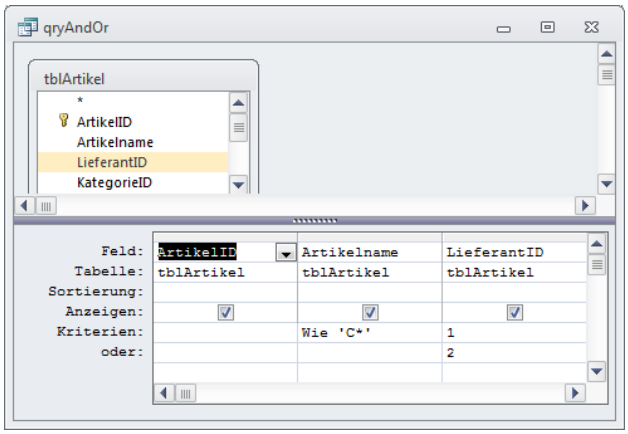


Bild 3: Zwei verknüpfte Tabellen in der Entwurfsansicht einer Abfrage

sitzt. Bei der Notation in der Abbildung setzt Access im SQL-Ausdruck automatisch **LieferantID = 1 OR LieferantID = 2** in Klammern und wertet diesen Ausdruck aus, bevor die Bedingung für den Artikelnamen hinzukommt.

```
SELECT tblArtikel.ArtikelID, tblArtikel.Artikelname,
tblArtikel.LieferantID
FROM tblArtikel
WHERE tblArtikel.Artikelname Like 'C*' AND
(tblArtikel.LieferantID = 1 OR
tblArtikel.LieferantID = 2);
```

Dies ist durchaus bedeutungsvoll. Wenn Sie die Klammer nämlich weglassen, arbeitet Access die Bedingungen einfach von links nach rechts ab:

```
tblArtikel.Artikelname Like 'C*'
AND tblArtikel.LieferantID = 1
OR tblArtikel.LieferantID = 2;
```

Das bedeutet, das Access zuerst alle Einträge ermittelt, deren **Artikelname** mit **C** beginnt und die gleichzeitig den Lieferanten mit der ID **1** haben. Im zweiten Schritt kommen dann alle Artikel hinzu, deren **LieferantID** den Wert **2** hat – unabhängig vom Anfangsbuchstaben. Und das liefert tatsächlich ein ganz anderes Ergebnis als der vorherige Ausdruck. Schließlich sieht auch der Abfrageentwurf ganz anders aus (siehe Bild 3).

Vergleichsoperatoren

Access-SQL stellt die folgenden Vergleichsoperatoren zusammen:

- gleich (**=**), größer als (**>**), kleiner als (**<**), größer gleich (**>=**), kleiner gleich (**<=**) und ungleich (**<>**)
- Nicht (**NOT**)
- Zwischen (**BETWEEN ... AND ...**)
- Mengenvergleich (**IN**)
- Vergleich mit Nullwert (**IS NULL**)
- Wie (**LIKE**, dient zum Vergleich mit Platzhaltern und ist nur für Zeichenketten vorgesehen)

Wenn Sie den Inhalt von Feldern der Abfrage mit Zahlen vergleichen möchten, verwenden Sie die üblichen Operatoren wie größer, kleiner, gleich und deren Kombinationen:

```
SELECT * FROM tblArtikel WHERE Einzelpreis > 10
```

Wenn Sie Artikel ermitteln möchten, deren Preis in einem bestimmten Bereich liegt, haben Sie mehrere Möglichkeiten. Intuitiv scheint die Angabe zweier mit **AND** verknüpfter Kriterien:

```
SELECT * FROM tblArtikel
WHERE Einzelpreis > 10 AND Einzelpreis < 20
```

Sie können für diesen Zweck aber auch den **BETWEEN**-Operator verwenden:

```
SELECT * FROM tblArtikel
WHERE Einzelpreis BETWEEN 10 AND 20
```

Letztere schließt allerdings auch Artikel ein, die 10,- Euro oder 20,- Euro kosten. Das genaue Pendant mit größer/kleiner/gleich müsste also so aussehen:

```
SELECT * FROM tblArtikel
WHERE Einzelpreis >= 10 AND Einzelpreis =< 20
```

Sollten Sie Dezimalzahlen als Vergleichswerte verwenden, geben Sie diese mit dem Punkt als Dezimaltrennzeichen an, außerdem dürfen Sie keine weiteren Zeichen wie etwa Währungssymbole oder Prozentzeichen einfügen:

```
SELECT * FROM tblArtikel WHERE Einzelpreis > 7.5
```

Textvergleiche

Wenn Sie den Inhalt von Textfeldern mit Texten vergleichen möchten, verwenden Sie meist das Gleichzeitszeichen (wenn der exakte Ausdruck gesucht werden soll) oder den **LIKE**-Operator. Der **LIKE**-Ope-

Einfache Suchfunktion für Formulare

Die Suche von Daten in den Tabellen einer Datenbank ist eine Aufgabe, die fast jede Datenbankanwendung erfüllen muss. Die grundlegenden Techniken bestehen darin, Suchbegriffe in ein Formular einzugeben und die daraus resultierende Ergebnismenge in einem geeigneten Steuerelement anzuzeigen – beispielsweise in einem Unterformular in der Datenblattansicht, in einem Listenfeld oder auch einfach in einem gefilterten Formular in der Formularansicht. Die wichtigste Grundlagen ist dabei, die Datenherkunft mit den entsprechenden Kriterien zu spicken. Wie es geht, zeigt dieser Beitrag.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1201_EinfacheSuchfunktion.mdb**.

Artikelabfrage

Für die verschiedenen Beispiele dieses Artikels wird die Artikeltabelle der Beispieldatenbank herhalten. Sie liefert diverse Text- und Zahlenfelder, die sich für das Zusammenstellen der benötigten Abfragen eignen. Der Vollständigkeit halber haben wir die ursprüngliche Tabelle noch um Datumsfeld namens **AngelegtAm** erweitert.

Suchen per Abfrage

Die Grundlagen zum Durchsuchen einer Abfrage nach verschiedenen Kriterien finden Sie beispielsweise in den Artikeln **SQL: SELECT-Abfragen, Teil II**, **Filtern nach Zahlen**, **Filtern von Textfeldern** oder **Datum und Zeit**. Im vorliegenden Artikel erfahren Sie, wie Sie die in diesen Artikeln gewonnenen Kenntnisse dazu nutzen können, dem Benutzer eine Eingabemöglichkeit für Suchbegriffe zu bieten, eine entsprechende Abfrage zu formulieren und das Ergebnis darzustellen.

Einen der möglichen Bausteine haben Sie bereits im Artikel **Einsatz von Parametern** kennengelernt. Sie können einer Abfrage einen Parameter in Form eines in eckigen Klammern eingefassten Ausdrucks als Bedingung hinzufügen, der vor dem Anzeigen des Abfrageergebnisses abgefragt und ausgewertet wird.

In Grundzügen bauen wir auf dieser Technik auf, Sie werden jedoch auch anderen Techniken kennenlernen, um das gewünschte Ergebnis zu erhalten.

Aufbau der Formulare

In den Beispielen soll der Benutzer den Namen eines Artikels oder einen Teil des Namens in ein Textfeld eingeben. Nach einem Mausklick auf die Suchen-Schaltfläche soll ein Unterformular in der Datenblattansicht die gefundenen Datensätze als Suchergebnis anzeigen.

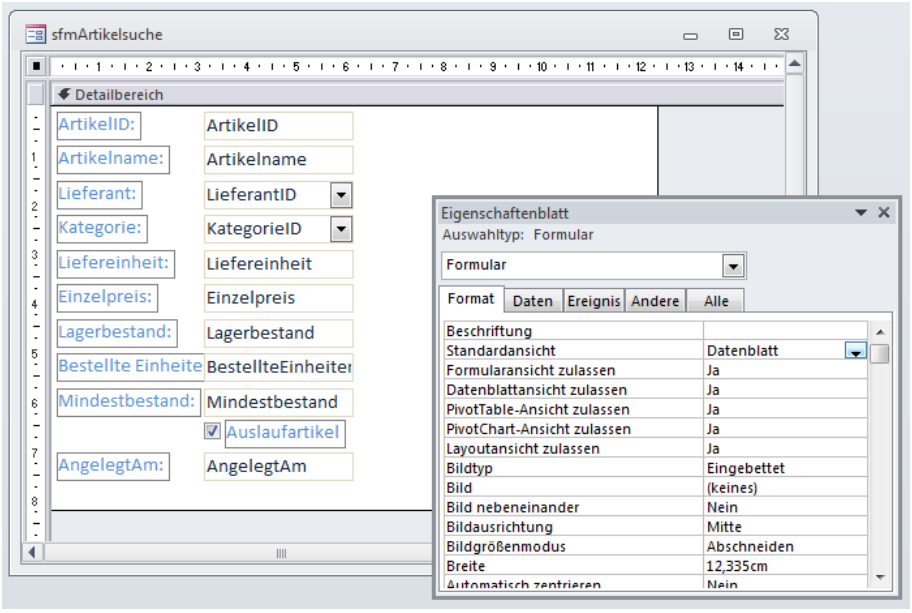


Bild 1: Entwurfsansicht des Unterformulars **sfmArtikelsuche**

Wir erstellen zunächst das Formular an, das später als Unterformular eingesetzt wird. Legen Sie dazu ein neues Formular an und speichern Sie es unter dem Namen **sfmArtikelsuche** (**sfm** steht für **Subform**). Wählen Sie für die Eigenschaft **Datenherkunft** zunächst die Tabelle **tblArtikel** aus.

Nach der Auswahl stehen in der Feldliste alle Felder der gewählten Datenherkunft bereit. Ziehen Sie alle Felder in den Detailbereich der Entwurfsansicht des Formulars. Stellen Sie dann die Eigen-

schaft **Standardansicht** auf den Wert **Datenblatt** ein (siehe Bild 1).

Speichern Sie nun das Unterformular und schließen Sie es. Anschließend erstellen Sie ein weiteres neues Formular und speichern es unter dem Namen **frmArtikelsuche**. Es handelt sich dabei um das Hauptformular, das gleich das Unterformular aufnehmen wird. Zuvor legen Sie jedoch noch die benötigten Steuerelemente an. Das erste ist ein Textfeld namens **txtSuche**, das zweite eine Schaltfläche namens **cmdSuche** (siehe Bild 2).

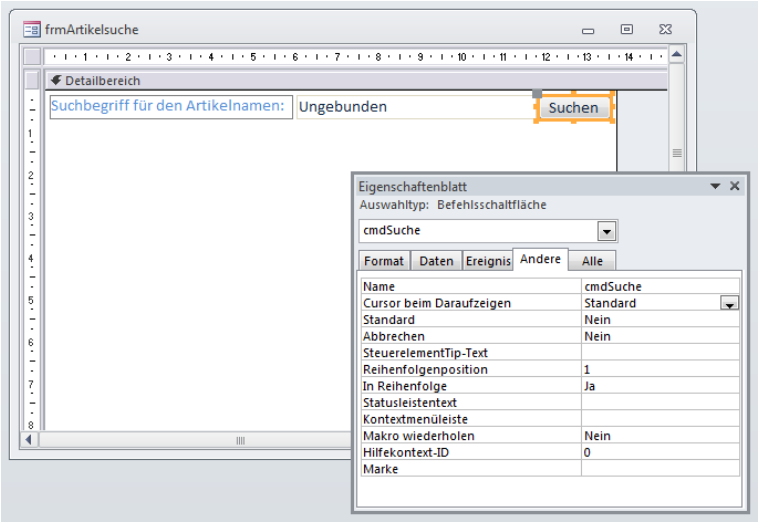


Bild 2: Hauptformular der Artikelsuche

Nun fügen Sie das Unterformular **sfmArtikelsuche** ein, indem Sie es mit der Maus aus dem Datenbankfenster beziehungsweise dem Navigationsbereich in den Detailbereich des Entwurfs des Formulars **frmArtikelsuche** ziehen.

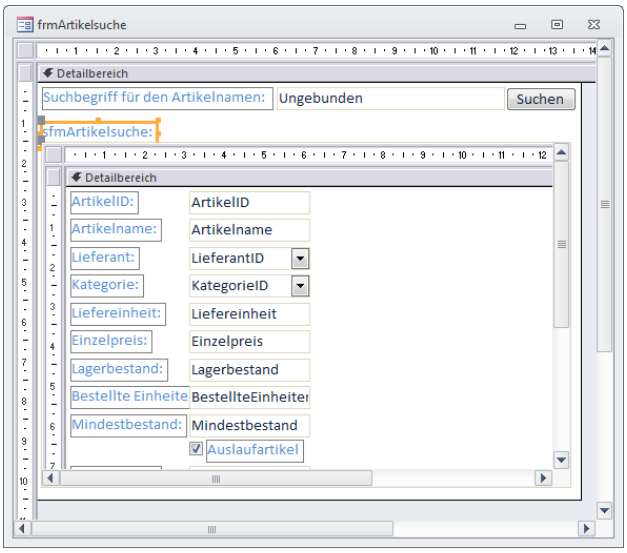


Bild 3: Haupt- und Unterformular in der Entwurfsansicht

Das Formular mit dem Unterformular sieht in der Entwurfsansicht nun wie in Bild 3 aus. Sie können nun noch das Bezeichnungsfeld des Unterformulars mit der Beschriftung **Suchergebnis** ausstatten und gegebenenfalls Formular und Unterformular noch etwas breiter gestalten, damit der Benutzer mehr Daten auf einen Blick sieht und weniger scrollen muss. Stellen Sie außerdem die Eigenschaften **Bildlaufleisten**, **Navigationsschaltflächen** und **Datensatzmarkierer** auf **Nein** ein – diese Elemente brauchen Sie für das Hauptformular nicht.

Nach einem Wechsel in die Formularansicht können Sie außerdem noch die Schriftgröße im Unterformular anpassen, indem Sie auf einen der Spaltenköpfe klicken und dann in der Symbolleiste oder im Ribbon den gewünschten Wert für die Schriftgröße eingeben. Das Formular ist nun fast fertig und sieht wie in Bild 4 aus.

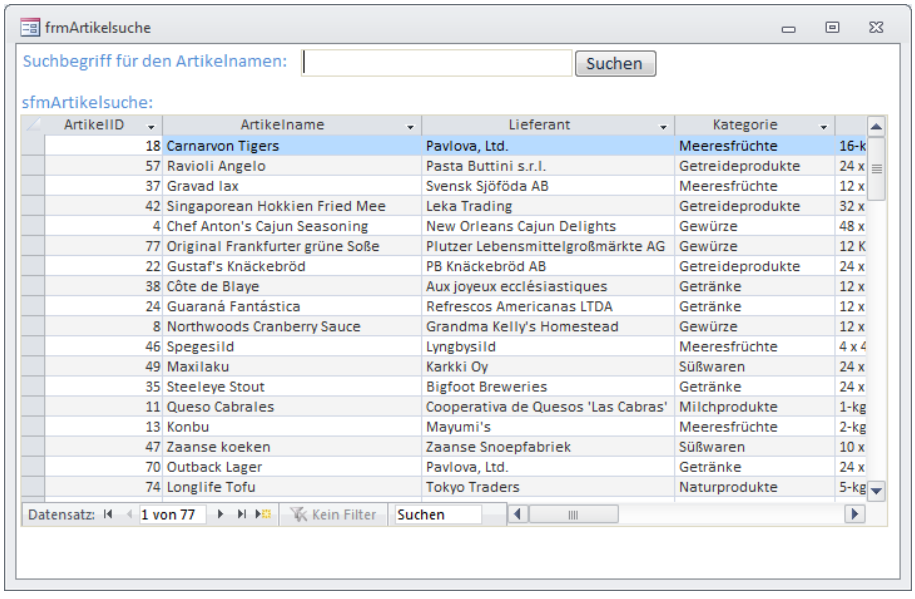


Bild 4: Ohne weitere Handgriffe zeigt das Unterformular einfach alle Datensätze an.

Suchen per Filter

Die einfachste Möglichkeit, die Datensätze des Unterformulars durch die Eingabe eines Suchbegriffs einzu-

schränken, ist der Einsatz der Eigenschaft **Filter** des Unterformulars.

Dies funktioniert folgendermaßen: Der Benutzer gibt einen Suchbegriff in das Textfeld **txtSuche** ein und betätigt dann die Schaltfläche mit der Beschriftung **Suchen**. Dies löst eine Ereignisprozedur an, die Sie wie folgt anlegen:

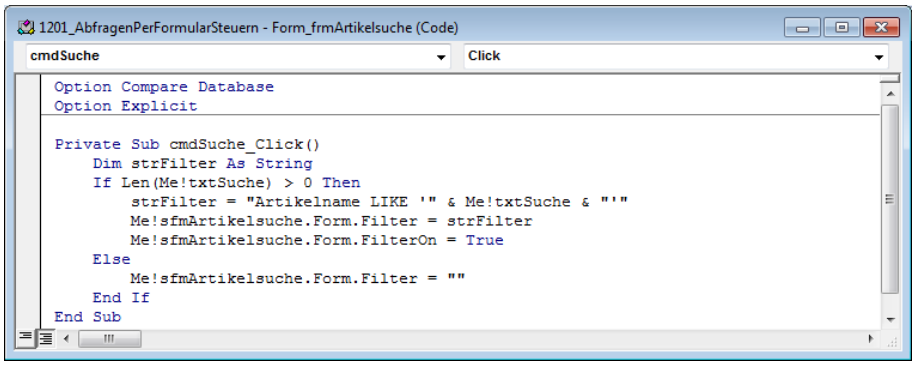


Bild 5: Diese Prozedur führt den Suchvorgang durch.

- Öffnen Sie das Formular **frmArtikelsuche** in der Entwurfsansicht.
- Klicken Sie auf die Schaltfläche und aktivieren Sie mit der Taste **F4** das Eigenschaftsfenster, sofern noch nicht sichtbar.
- Wählen Sie für die Eigenschaft **Beim Klicken** auf der Registerseite **Ereignis** den Eintrag **[Ereignisprozedur]** für die Eigenschaft **Beim Klicken** aus und klicken Sie anschließend auf die Schaltfläche rechts mit den drei Punkten (...).
- Im VBA-Editor erscheint nun die leere Prozedur, die Sie wie folgt füllen (siehe auch Bild 5):

```
Private Sub cmdSuche_Click()
    Dim strFilter As String
    If Len(Me!txtSuche) > 0 Then
        strFilter = "Artikelname LIKE '" _
            & Me!txtSuche & "'"
        Me!sfmArtikelsuche.Form.Filter = strFilter
        Me!sfmArtikelsuche.Form.FilterOn = True
    Else
        Me!sfmArtikelsuche.Form.Filter = ""
    End If
End Sub
```

Wenn Sie nun in die Formularansicht wechseln, können Sie die Prozedur gleich ausprobieren. Geben Sie einen Suchbegriff in das Suchfeld ein, zum Beispiel C*. Klicken Sie dann auf die Schaltfläche **Suchen**, löst dies die soeben angelegte Prozedur aus.

Die **If**-Bedingung prüft zunächst, ob die Länge der Zeichenkette im Textfeld **txtSuche** länger als 0 Zeichen ist, also ob dort überhaupt ein Suchbegriff eingetragen wurde. Falls ja, wird ein Filterkriterium für die Eigenschaft Filter zusammengesetzt, das aus der Angabe des zu durchsuchenden Feldes, dem Vergleichsoperator **LIKE** und einem Vergleichsausdruck besteht, der sich aus zwei Hochkommata und dem Inhalt des Textfeldes zusammensetzt.

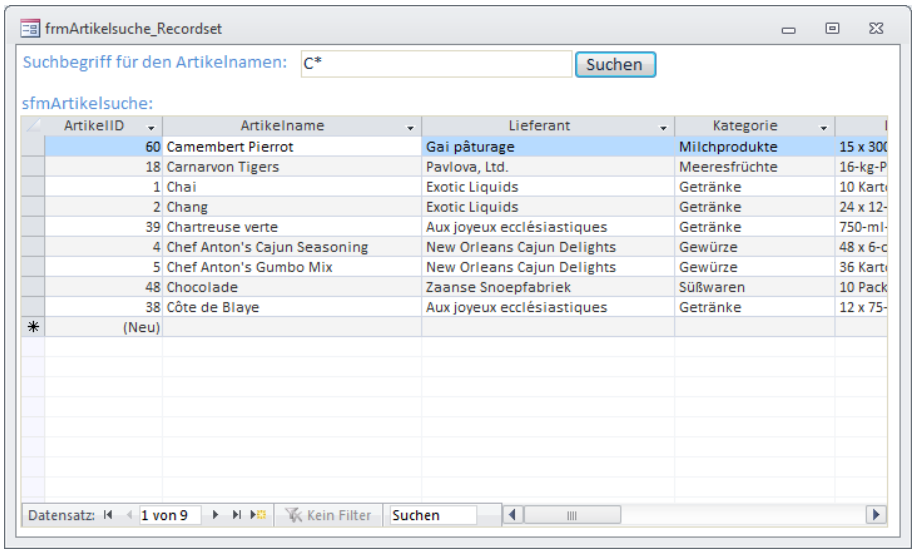


Bild 6: Das Suchformular in Aktion

Dieser Ausdruck wird anschließend der Eigenschaft Filter des im Unterformularsteuerelements enthaltenen Formulars zugewiesen. Dies allein bewirkt noch nichts, erst muss noch der Wert der Eigenschaft **FilterOn** auf den Wert **True** eingestellt werden. Das Ergebnis sieht wie in Bild 6 aus.

Andersherum soll das Auslösen der Suche bei nicht vorhandenem Suchbegriff alle Einträge der Tabelle **tblArtikel** anzeigen. Dies

geschieht, wenn die Bedingung der **If**-Bedingung nicht erfüllt, die Länge der Zeichenkette im Textfeld **txtSuche** also **0** ist. In diesem Fall wird schlicht der Ausdruck für die Eigenschaft **Filter** auf eine leere Zeichenkette eingestellt (**""**).

Nun ist dies alles nur halb so ergonomisch, wenn der Benutzer jedes Mal auf die Schaltfläche drücken muss, anstatt einfach nur die Eingabetaste zum Absenden des Suchbegriffs zu betätigen.

Damit diese Schaltfläche automatisch beim Betätigen der Eingabetaste ausgelöst wird, stellen Sie die Eigenschaft **Standard** für diese Schaltfläche auf den Wert **Ja** ein (siehe Bild 7).

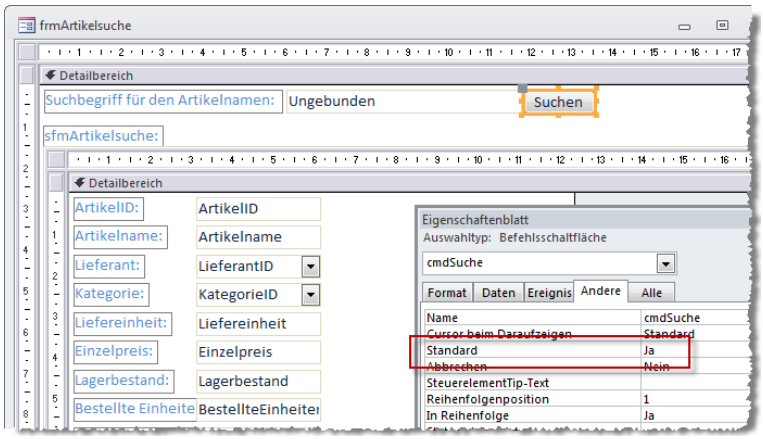


Bild 7: Einstellen der Schaltfläche **cmdSuche** als Standardschaltfläche des Formulars

Suche per Recordsource

Das Zuweisen der Tabelle tblArtikel zur Eigenschaft Datenherkunft können Sie auch per VBA-Code erledigen. Der Eigenschaft **Datenherkunft** lassen sich außerdem nicht nur Tabellennamen wie **tblArtikel**, sondern auch Abfragenamen oder gar SQL-Ausdrücke wie **SELECT * FROM tblArtikel** zuweisen.

Dies können Sie sich auch für die Programmierung einer Suchfunktion zunutze machen. In diesem Fall stellen Sie in der Ereignisprozedur, die durch die Schaltfläche **cmdSuchen** ausgelöst wird, einen SQL-Ausdruck zusammen, der anschließend der Eigenschaft **Recordsource** des im Unterformularsteuerelement **sfmArtikelsuche** angezeigten Formulars zugewiesen wird:

```
Private Sub cmdSuche_Click()
    Dim strSQL As String
    If Len(Me!txtSuche) > 0 Then
        strSQL = "SELECT * FROM tblArtikel WHERE " _
            & "Artikelname LIKE '" & Me!txtSuche & "'"
    Else
        strSQL = "SELECT * FROM tblArtikel"
    End If
    Me!sfmArtikelsuche.Form.RecordSource = strSQL
End Sub
```

Sollte hier kein Suchbegriff vorliegen, wird die Abfrage **SELECT * FROM tblArtikel** als Datenherkunft übergeben. Diese Version der Suchfunktion finden Sie in der Beispieldatenbank im Formular **frmArtikelsuche_Recordset**.

Suche mit Platzhalter

In der aktuellen Form kann der Benutzer die Platzhalter selbst eingeben, sie werden dann automatisch in die Suche mit einbezogen. Wenn Sie etwa den Ausruck **A*** verwenden, lautet der Filter des ersten Beispiels

```
Artikelname LIKE 'A*'
```

und die SELECT-Anweisung des zweiten Beispiels sieht so aus:

```
SELECT * FROM tblArtikel WHERE Artikelname LIKE 'A*'
```

Sie können natürlich auch gleich entsprechende Platzhalter in die Prozeduren zum Zusammenstellen der Filter beziehungsweise der **SELECT**-Abfrage integrieren.

Beim Zusammenstellen des Filters für das erste Beispiele verwenden Sie dann etwa den folgenden Ausdruck:

```
strFilter = "Artikelname LIKE '" & Me!txtSuche & "*'"
```

Entscheidend ist das Sternchen als Platzhalter am Ende des Ausdrucks. Wenn der Filter alle Artikelnamen erfassen soll, welche die angegebene Zeichenkette enthalten, können Sie auch vorn und hinten einen Platzhalter einfügen:

```
strFilter = "Artikelname LIKE '*" & Me!txtSuche & "*'"
```

Beim Zusammenstellen der **SELECT**-Anweisung im zweiten Beispiel würde das so aussehen – hier erst der Ausdruck für alle Artikelnamen, die mit dem angegebenen Suchbegriff beginnen:

```
strSQL = "SELECT * FROM tblArtikel WHERE " _
    & "Artikelname LIKE '" & Me!txtSuche & "*'"
```

Sollen alle Artikel angezeigt werden, deren Artikelname den Suchbegriff enthält, fügen Sie noch ein Sternchen hinzu:

```
strSQL = "SELECT * FROM tblArtikel WHERE " _
    & "Artikelname LIKE '*" & Me!txtSuche & "*'"
```

Suche nach Zahlenwerten

Nachdem wir eine Suche nach einer Zeichenkette implementiert haben, fügen wir eine Suche nach Zahlenwerten hinzu – genau genommen die Suche nach Artikeln, deren Einzelpreis sich in bestimmten Grenzen befindet. Für den Start prüfen wir zunächst den genauen Preis des Artikels, später soll dann ein Preisbereich angegeben werden können. In der Beispieldatenbank haben wir das Formular **frmArtikelsuche** zum Erweitern der Suchfunktion in ein neues Formular namens **frmArtikelsucheMitPreis** kopiert. Diesem fügen Sie nun zunächst zwei neue Steuerelemente hinzu – ein Textfeld namens **txtSucheEinzelpreis** und eine Schaltfläche namens **cmdSucheEinzelpreis** (siehe Bild 8). Für die Schaltfläche hinterlegen Sie die folgende Prozedur, die durch das Ereignis **Beim Klicken** ausgelöst wird:

```
Private Sub cmdSucheEinzelpreis_Click()
    Dim strFilter As String
    If Len(Me!txtSucheEinzelpreis) > 0 Then
        strFilter = "Einzelpreis = " _
            & Me!txtSucheEinzelpreis
        Me!sfmArtikelsuche.Form.Filter = strFilter
        Me!sfmArtikelsuche.Form.FilterOn = True
    Else
        Me!sfmArtikelsuche.Form.Filter = ""
    End If
End Sub
```

Die Prozedur prüft wiederum, ob das Textfeld **txtSucheEinzelpreis** überhaupt einen Wert enthält. Anschließend setzt sie einen Filter zusammen, der das Feld **Einzelpreis** der Tabelle **tblArtikel** mit dem im Textfeld **txtSucheEinzelpreis** vergleicht. Dies gelingt ganz hervorragend, bis ein Benutzer einen Wert eingibt, den Access nicht als Zahlenwert interpretieren kann – zum Beispiel eine Zahl mit Währung wie **10,- EUR**. Dies löst dann den Fehler aus Bild 9 aus.

Wie können Sie dies ändern? Kein Problem: Sie müssen nur vor der Auswertung prüfen, ob das Feld einen numerischen Wert enthält.Dazu stellt VBA die Funktion **IsNumeric** zur Verfügung. Diese integrieren Sie einfach in die vorhandene Prozedur (vollständige Version siehe Formularmodul):

```
If IsNumeric(Me!txtSucheEinzelpreis) Then
    '... Filter zusammenstellen und anwenden
Else
    MsgBox "Bitte geben Sie eine Zahl für " _
        & "das Suchfeld 'Einzelpreis' ein.", _
        vbExclamation, "Falsche Eingabe"
    Me!txtSucheEinzelpreis.SetFocus
    Exit Sub
End If
```

Wenn das Feld einen numerischen Wert enthält, filtert die Prozedur das Unterformular wie gewünscht. Falls nicht, erscheint zuerst ein Meldungsfenster, dann wird der Fokus auf das Textfeld zur Eingabe des Kriteriums verschoben und die Prozedur verlassen.

Auch die Fehlermeldung müssen wir noch beleuchten. Diese lautete:

Bild 8: Einstellen der Schaltfläche **cmdSuche** als Standardschaltfläche des Formulars

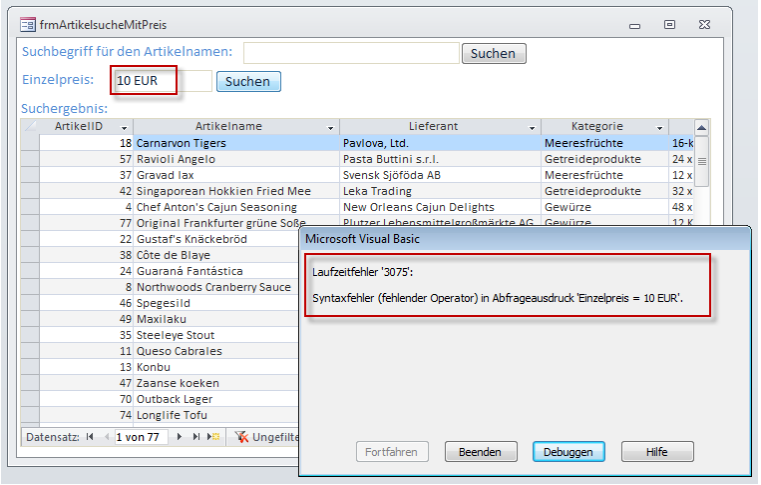


Bild 9: Fehler bei Eingabe von Text in ein Zahlenfeld

Arrays

Arrays – auch Datenfelder genannt – bieten die Möglichkeit, mehrere gleichartige Daten in einer Art Liste zu speichern und über einen Index darauf zuzugreifen. Array können ein- oder mehrdimensional sein. Dieser Artikel zeigt, wie Sie Arrays deklarieren, diese mit Daten füllen und die gewünschten Informationen wieder abrufen. Außerdem lernen Sie eine Reihe VBA-Anweisungen kennen, mit denen der Umgang mit Arrays noch viel leichter wird.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1201_Arrays.mdb**.

Array-Datentyp

Ein Array kann beliebige Datentypen annehmen. Die einzige Einschränkung ist: Jedes Element des Array muss den gleichen Datentyp besitzen. Dies ist jedoch keine große Einschränkung, denn die meisten Anwendungen von Array basieren ohnehin auf gleichartigen Daten.

Einfache Beispiele

Im einfachsten Fall ist ein Array zweidimensional. Das bedeutet, dass Sie für jeden Indexwert des Array entsprechend seiner Größe einen Wert im Array speichern können.

Wofür benötigt man solche Konstrukte? Nun, Sie könnten zum Beispiel zehn gleichartige Werte benötigen, die Sie in entsprechenden Variablen speichern - etwa so:

```
Dim str1 As String
Dim str2 As String
Dim str3 As String
...
str1 = "Müller"
str2 = "Meier"
str3 = "Schmitz"
```

Mit diesen Variablen können Sie nun verschiedene Dinge im Code anstellen. Dies hat nur zwei Nachteile: Erstens müssen Sie jeden Zugriff auf einen dieser Werte separat durchführen, auch wenn Sie mal eben die Werte aller Variablen ausgeben möchten. Mal eben die Daten aus einer Tabelle in die Variablen schreiben? Macht zehn Anweisungen. Die Daten im Debug-Fenster ausgeben? Wieder zehn Anweisungen. Und der zweite Nachteil: Wenn ein elfter Wert hinzu kommt, müssen Sie den kompletten Code anpassen.

Für solche Fälle gibt es Array. Für ein Array benötigen Sie nur ein einzige Variable, die beliebig viele Werte aufnimmt. Sie können gleich bei der Deklaration festlegen, wieviele Werte dies sind oder die Obergrenze später anpassen.

Schauen wir uns an, wie dies funktioniert. Wenn Sie beispielsweise eine Variable mit vier Werten verwenden möchten, deklarieren Sie diese wie folgt:

```
Dim strZahlen(3)
```

Wieso lautet der Wert in Klammern **3**, wenn wir doch drei Werte benötigen? Ganz einfach: Der hier angegebene Wert gibt den Wert des größten Index-Eintrags an. Normalerweise beginnt die Zählung des Index bei Arrays bei **0**, also können Sie die vier Indexwerte **0**, **1**, **2** und **3** verwenden.

In der folgenden Prozedur versuchen wir, mehr als diese vier Einträge zu füllen:

```
Public Sub EinfachesArray()
    Dim strZahlen(3) As String
    strZahlen(0) = "Null"
    strZahlen(1) = "Eins"
    strZahlen(2) = "Zwei"
    strZahlen(3) = "Drei"
    strZahlen(4) = "Vier"
End Sub
```

Dies führt zu dem Fehler aus Bild 1, und zwar in der Zeile, die versucht, einen Wert für den Index **4** zu vergeben.

Die Tatsache, dass der erste Indexwert **0** lautet, ist nicht unumstößlich. Wenn Sie im Kopf des betroffenen VBA-Moduls die folgende Zeile einfügen, beginnt die Zählung bei **1**:

```
Option Base 1
```

Der Fehler bei der obigen Prozedur tritt nun nicht erst in der vierten, sondern direkt in der ersten Zeile auf: den Index mit dem Wert **0** gibt es nämlich nun

nicht mehr. Normalerweise benötigen Sie aber keine Index-Änderung – es sei denn, Sie möchten Daten zwischen einem Array und einem anderen Objekt hin- und hertransferieren, dessen Index erst bei 1 beginnt. Dann können Sie besser den kleinsten Index mit **Option Base** festlegen als überall 1 abzuziehen oder hinzuzuaddieren, um mit gleichen Indexwerten arbeiten zu können.

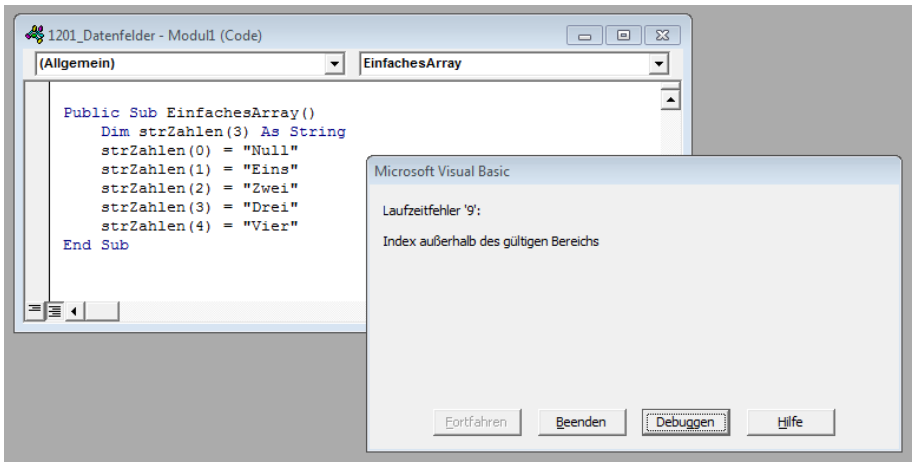


Bild 1: Fehler beim Versuch, einen nicht vorhandenen Eintrag im Array zu referenzieren

Indexbereich

Sie können sich auch komplett von den vom System oder von **Option Base** vorgegebenen Index-Grenzen freimachen. Dazu geben Sie einfach eigene Indexgrenzen an – die folgende Routine läuft dann ohne Fehler durch:

```
Public Sub EinfachesArrayMitBereich()
    Dim strZahlen(0 To 4) As String
    strZahlen(0) = "Null"
    strZahlen(1) = "Eins"
    strZahlen(2) = "Zwei"
    strZahlen(3) = "Drei"
    strZahlen(4) = "Vier"
End Sub
```

Eigentlich ist die Angabe des Indexbereichs ohnehin die sinnvollste Art, ein Array zu deklarieren – vorausgesetzt, Sie kennen die Dimensionen des Arrays überhaupt bereits beim Deklarieren.

Ein- und mehrdimensionale Array

Die bisher vorgestellten Beispiele basierten auf eindimensionalen Array. Ein Array kann aber durchaus mehrere Dimensionen annehmen.

Wenn Sie beispielsweise ein Array mit zwei mal zwei Elementen deklarieren und mit entsprechenden Werten füllen möchten, verwenden Sie die folgende Routine:

```
Public Sub ZweiMalZwei()
    Dim intZahlen(1 To 2, 1 To 2) As Integer
    intZahlen(1, 1) = 1
    intZahlen(1, 2) = 2
    intZahlen(2, 1) = 3
    intZahlen(2, 2) = 4
End Sub
```

Arrays durchlaufen

Wenn Sie die Indexgrenzen eines Array kennen, können Sie die im Array gespeicherten Werte problemlos mithilfe einer Schleife durchlaufen.

Dazu verwenden Sie einfach eine **For...Next**-Schleife mit einer Laufvariable, die Sie später als Index für den Zugriff auf die im Array gespeicherten Werte verwenden:

```
Public Sub EinfachesArrayMitBereich()
    Dim strZahlen(0 To 4) As String
    Dim i As Integer
    strZahlen(0) = "Null"
    ... weitere Zuweisungen
    For i = 0 To 4
        Debug.Print strZahlen(i)
    Next i
End Sub
```

Indexgrenzen ermitteln

Auch wenn Sie die Indexgrenzen nicht kennen, können Sie ein Array problemlos durchlaufen. Die Grenzen ermitteln Sie dabei mit den beiden Funktionen **UBound** und **LBound**, denen Sie einfach den Namen der Array-Variablen übergeben:

```
Public Sub IndexgrenzenErmitteln()
    Dim strZahlen(3) As String
    Dim i As Integer
    strZahlen(0) = "Null"
    ... weitere Zuweisungen
```

```
    Debug.Print "Untere Grenze: " & LBound(strZahlen)
    Debug.Print "Obere Grenze: " & UBound(strZahlen)
    For i = LBound(strZahlen) To UBound(strZahlen)
        Debug.Print strZahlen(i)
    Next i
End Sub
```

Indexgrenzen zur Laufzeit einstellen

Wie bereits erwähnt, brauchen Sie die Indexgrenzen nicht direkt bei der Deklaration festzulegen – in manchen Fällen ist dies sogar noch nicht einmal möglich. Stattdessen deklarieren Sie einfach eine Variable mit einem leeren Klammernpaar.

Später erweitern Sie die Größe des Arrays beispielsweise innerhalb einer Schleife nach Wunsch. Dabei verwenden Sie die Anweisung **ReDim** und geben den Variablennamen des Array samt gewünschter Größe an.

Das Schlüsselwort **Preserve** sorgt dafür, dass bereits vorhandene Daten nicht gelöscht werden:

```
Public Sub DynamischesArray()
    Dim strMonate()
    Dim i As Integer
    For i = 1 To 12
        ReDim Preserve strMonate(1 To i)
        strMonate(i) = MonthName(i)
    Next i
    For i = 1 To 12
        Debug.Print strMonate(i)
    Next i
End Sub
```

Würden Sie das Schlüsselwort **Preserve** in diesem Fall weglassen, enthielte das Array bei der Ausgabe nur einen einzigen Wert, und zwar für das zwölfte Element.

Ein Array mit den Daten einer Tabelle füllen

Wenn Sie beispielsweise ein String-Array mit den Namen der Artikel der Tabelle **tblArtikel** füllen möchten, verwenden Sie eine Routine wie die Folgende:

```
Public Sub ArtikelArray()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim strArtikel()
    Dim i As Integer
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT Artikelname
                  FROM tblArtikel", dbOpenDynaset)
    Do While Not rst.EOF
        i = i + 1
        ReDim Preserve strArtikel(1 To i)
        strArtikel(i) = rst!Artikelname
        rst.MoveNext
    Loop
    For i = LBound(strArtikel) To UBound(strArtikel)
        Debug.Print strArtikel(i)
    Next i
    Set db = Nothing
End Sub
```

Die Routine erstellt zunächst ein Recordset auf Basis der Tabelle **tblArtikel**. Dieses durchläuft sie in einer **Do While**-Schleife und erhöht dabei die Variable **i** für jeden Datensatz um **1**.

i ist gleichzeitig die neue Obergrenze des Array und laufender Index zum Zuweisen des aktuellen Artikelnamens in das Array. Anschließend wird das Array durchlaufen, um zu prüfen, ob alle Artikelnamen enthalten sind.

Auslesen und Zuweisen

Natürlich greifen Sie nicht nur innerhalb von Schleifen auf die Werte von Array zu. Sie können auch eines der Elemente herauspicken, um dieses auszulesen oder mit einem neuen Wert zu füllen.

Einen Wert geben Sie einfach durch Angabe des Arraynamens und des Indexwertes aus:

```
Debug.Print strArtikelname(10)
```

Andererseits füllen Sie einen beliebigen Wert durch eine einfache Zuweisung:

```
strArtikelname(20) = "Neuer Artikel"
```

Array als Variant

Sie können den Inhalt einer Array-Variablen auch einer Variant-Variablen zuweisen und danach genauso damit arbeiten, als ob Sie ein Array verwenden.

In manchen Fällen ist dies sogar unabdingbar – beispielsweise, wenn sie den Inhalt eines Array als Parameter oder Rückgabewert einer Funktion übergeben möchten. Sollten Sie nämlich versuchen, das

Kommunikationsdaten im Griff

Heutzutage ist es längst nicht damit getan, je ein Feld für Telefonnummer und Telefax in einer Kundentabelle zu reservieren. Jeder Kunde hat zusätzlich noch mindestens eine Handynummer, ist unter der privaten Nummer zu erreichen und bietet eine oder mehrere E-Mail-Adressen an. Wer solche Daten flexibel in seiner Datenbank speichern möchte, braucht zwei Zusatztabellen und einige Formularanpassungen. Wie die Lösung aussieht, erfahren Sie in diesem Artikel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1202_Kommunikationsdaten.mdb**.

Kommunikationsdaten einfach

In vielen Fällen dürfte es ausreichen, in der Kundentabelle jeweils ein Feld für Telefon-, Telefax- und Mobilnummer sowie E-Mail zu reservieren und diese Felder im Formular zum Bearbeiten der Kundendaten bereitzustellen. Im Artikel **Kunden per Bezeichnung verwalten** haben wir dies wie in Bild 1 erledigt.

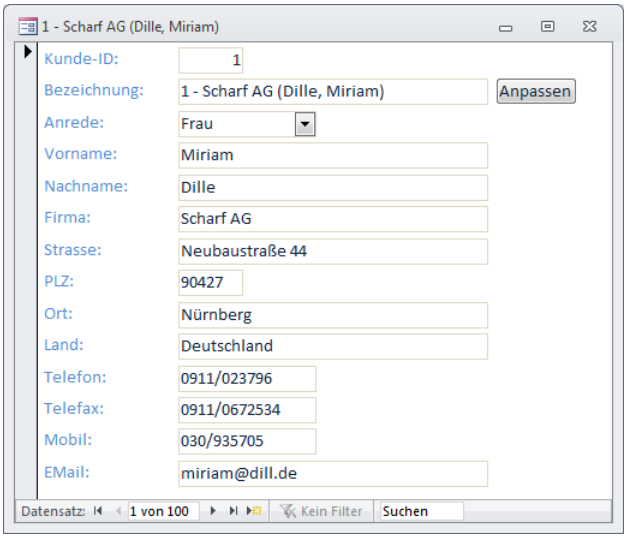


Bild 1: Einfache Variante: Die Kommunikationsdaten werden einfach in der Kundentabelle gespeichert.

Der vorliegende Artikel stellt eine alternative Methode vor, Kommunikationsdaten wie Telefonnummern, Faxnummern, Handynummern oder E-Mail-Adressen zu speichern. Dabei sollen zu jedem Kunden nur die Daten gespeichert werden, die tatsächlich anfallen, es werden keine Felder für die verschiedenen Kommunikationsarten reserviert, sondern über weitere Tabellen dynamisch hinzugefügt.

Dies wird durch zwei zusätzliche Tabellen zur Tabelle **tblKunden** realisiert. Die erste heißt **tblKommunikationsarten** und speichert eigentlich nur Lookup-Daten, nämlich die Bezeichnungen der verschiedenen Kommunikationsarten – also **Telefon dienstlich**, **Telefon privat** und so weiter. Sie besteht nur aus zwei Feldern, deren Aufbau Sie Bild 2 entnehmen können.

Gefüllt sieht die Tabelle **tblKommunikationsarten** wie in Bild 3 aus. Sie können die enthaltenen Daten beliebig anpassen oder ergänzen.

Bild 2: Die Tabelle **tblKommunikationsarten** speichert die Bezeichnungen der Kommunikationsarten.

Mit der Tabelle für die Kommunikationsarten allein können wir jedoch noch nicht viel anfangen. Wir benötigen eine weitere Tabelle, mit der wir die Kommunikationsarten und die Kunden in Beziehung bringen und die entscheidenden Daten, nämlich Telefonnummern, E-Mail-Adressen und so weiter verwalten. Diese Tabelle stellt eine m:n-Beziehung zwischen den Tabellen **tblKunden** und **tblKommunikationsarten** her und heißt **tblKommunikationsdetails**. Den Entwurf dieser Tabelle finden Sie in Bild 4. Neben dem Primärschlüsselfeld **KommunikationsdetailID** enthält die Tabelle zunächst zwei Fremdschlüssel-

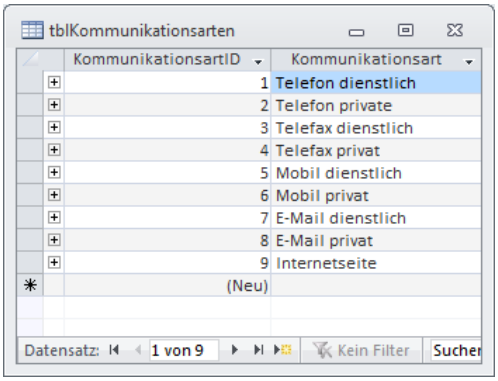
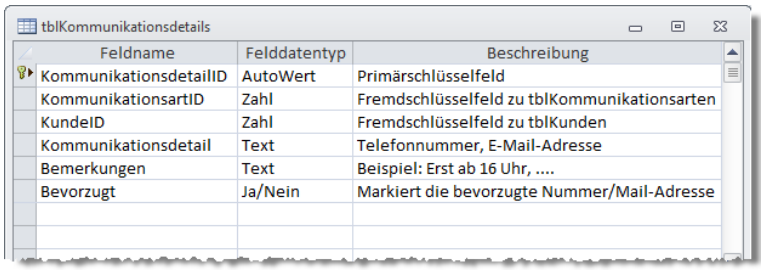


Bild 3: Die Tabelle **tblKommunikationsdaten** mit einigen Beispieldatensätzen

felder namens **KommunikationsartID** und **KundeID**, mit denen die ein Kunde mit einem Eintrag der Tabelle **tblKommunikationsarten** zusammengeführt wird.



Feldname	Felddatentyp	Beschreibung
KommunikationsdetailID	AutoWert	Primärschlüsselfeld
KommunikationsartID	Zahl	Fremdschlüsselfeld zu tblKommunikationsarten
KundeID	Zahl	Fremdschlüsselfeld zu tblKunden
Kommunikationsdetail	Text	Telefonnummer, E-Mail-Adresse
Bemerkungen	Text	Beispiel: Erst ab 16 Uhr, ...
Bevorzugt	Ja/Nein	Markiert die bevorzugte Nummer/Mail-Adresse

Bild 4: Die Tabelle **tblKommunikationsdetails** stellt die Beziehung zwischen Kunde und Kommunikationsart her und nimmt die eigentlich wichtigen Daten auf.

Um diese Fremdschlüsselfelder anzulegen, tragen Sie zunächst die Feldnamen ein und wählen dann für den Datentyp jeweils den Wert **Nachschlagefeld** aus. Dies startet den Nachschlagefeld-Assistenten. Für das Feld **KommunikationsartID** wählen Sie damit die Tabelle **tblKommunikationsarten** als Datenherkunft aus und fügen die beiden Felder **KommunikationsartID** und **Kommunikationsart** hinzu. Die Sortierung legen Sie nicht explizit fest, daher werden die Einträge des Nachschlagefeldes nach dem Wert des Primärschlüsselfeldes sortiert. Dies ist in Ordnung, da die Einträge bereits in der gewünschten Reihenfolge hinzugefügt wurden

Für das Feld **KundeID** wählen Sie im Nachschlagefeld-Assistent die Tabelle **tblKunden** aus und fügen die beiden Felder **KundeID** und **Bezeichnung** hinzu (dies ist übrigens ein guter Einsatzzweck für das Feld **Bezeichnung**, über das Sie im Artikel **Kunden per Bezeichnung verwalten** mehr erfahren). Im Gegensatz zum Nachschlagefeld für die Kommunikationsarten sollten Sie für dieses Nachschlagefeld eine Sortierung festlegen, und zwar eine aufsteigende Sortierung nach dem Feld **Bezeichnung**.

Für jede Kombinationen aus **KommunikationsartID** und **KundeID** nimmt die Tabelle im Feld **Kommunikationsdetail** die Telefonnummer, die E-Mail-Adresse et cetera auf. Ein weiteres Feld namens Bemerkungen erfasst Informationen etwa über die Erreichbarkeit der verschiedenen Telefonnummern. Wenn jemand also beispielsweise immer nur vormittags im Büro erreichbar ist und nachmittags nur per Handy, würde man dies hier eintragen. Schließlich gibt es noch ein **Ja/Nein**-Feld namens **Bevorzugt**, mit dem Sie eine der Kommunikationsarten für den Kunden festlegen können, die primär genutzt werden soll.

Eine Übersicht der Tabellen und ihrer Beziehungen finden Sie im Datenmodell aus Bild 5. Dort legen Sie, sofern Sie nicht mit Access 2010 arbeiten und dies bereits im Nachschlagefeld-Assistent erledigt haben, noch referenzielle Integrität und Löschweitergabe für die beiden neuen Beziehungen fest.

Kommunikationsarten im Formular

Nun folgt der interessante Teil der Aufgabe: Die Kommunikationsinformationen sollen per Formular verwaltet werden. Wie es bei den Daten aus Tabellen, die per m:n-Beziehung verknüpft sind, üblich ist, kommt hier als erstes ein Unterformular in der Datenblattansicht in Frage – zumindest, wenn Sie die Daten auch noch direkt im Formular bearbeiten wollen.

Wir möchten dem bestehenden Formular **frmKunden** also ein Unterformular hinzufügen, dass alle bestehenden Kommunikationsdetails zum jeweils angezeigten Kunden bereitstellt und das Bearbeiten, Löschen und Hinzufügen erlaubt.

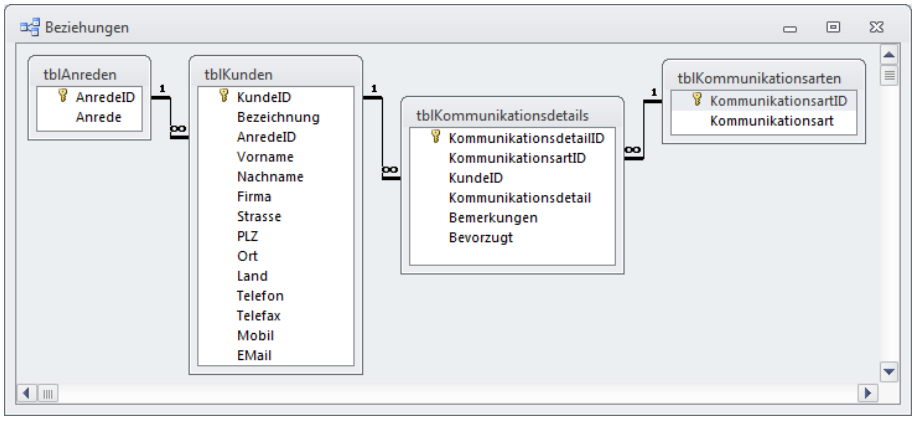


Bild 5: Das Datenmodell verdeutlicht die Zusammenhänge zwischen den Tabellen zur Verwaltung der Kommunikationsdaten.

Normalerweise würde man tippen, dass das Unterformular die Daten der beiden Tabellen **tblKommunikationsdetails** und **tblKommunikationsarten** anzeigen soll. Grundsätzlich stimmt das, allerdings müssen wir die Tabelle **tblKommunikationsarten** nicht explizit in die Da-

Bestellungen in Tabellen verwalten

Das Verwalten von Bestellungen in einer Access-Datenbank ist für Einsteiger eine echte Herausforderung. Wer Kunden, Bestellungen, Bestellpositionen und Artikel sauber pflegen möchte, braucht ein sauber normalisiertes Datenmodell – und eine Menge Hirnschmalz, damit alle Felder in den richtigen Tabellen landen. Dieser Artikel zeigt, wie Sie das Datenmodell aufbauen und welche Varianten es gibt.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1202_BestellungenInTabellen.mdb**.

Einfacher Ansatz

Dieser Artikel soll zunächst einmal den grundsätzlichen Umgang mit Bestellungen und den dabei anfallenden Daten erläutern. Es gibt unendlich viele Varianten, auf die wir hier (noch) nicht eingehen wollen – erstmal steht das grundsätzliche Verständnis der vorgestellten Konstellation auf dem Plan.

Artikel verwalten

Die erste Tabelle des in diesem Artikel beschriebenen Datenmodells soll die Artikeldaten speichern. Ob es sich bei diesen Artikeln um Produkte oder Dienstleistungen handelt, spielt keine Rolle – Tatsache ist, dass Sie diese möglichst vielen Kunden über eine oder mehrere Bestellungen zuordnen möchten.

Welche Informationen speichert man also zu einem Artikel? Das Sie ohne einen Primärschlüssel nicht auskommen, ist klar: Er ist das eindeutige Merkmal einer jeden Tabelle, über das Sie die Datensätze dieser Tabelle anderen Tabellen über ein Fremdschlüsselfeld zuordnen können. Dieses Feld soll in unserem Fall **ArtikelID** heißen und als Autowertfeld ausgelegt werden. Dieses Feld legen Sie gleich nach dem Erstellen einer neuen Tabelle namens **tblArtikel** an.

Das zweite wichtige Merkmal eines Artikels ist der Artikelname. Ein Feld namens **Artikelname** mit dem Datentyp **Text** und der Feldlänge **255** speichert die Bezeichnungen der Artikel.

Jeder Artikel hat natürlich einen Preis. Diesen legen wir in einem Währungsfeld namens **Einzelpreis** fest. Schließlich kann es verschiedene Mehrwertsteuersätze geben, die in einem weiteren Feld namens **Mehrwertsteuersatz** landen sollen. Dieses Feld erhält ebenfalls den Datentyp **Währung** – dadurch erhalten Sie die notwendige Genauigkeit für spätere Berechnungen der Brutto- und Netto-Rechnungsbeträge (siehe Bild 1).

Der Mehrwertsteuersatz soll allerdings nicht mit dem Währungszeichen versehen werden, sondern als Prozentzahl angezeigt werden. Dazu stellen Sie die Eigenschaft **Format** des Feldes in der Entwurfsansicht der Tabelle auf den Wert **Prozentzahl** ein.

Wenn Sie tatsächlich nur Bestellungen verwalten möchten, was hier der Fall ist, reichen diese Felder bereits aus. Varianten, die weitere Informationen erfordern, besprechen wir in weiteren Artikeln. Dort geht es etwa um Artikelbestände, Nachbestellungen, Artikelkategorien und Lieferanten.

Kunden verwalten

Zu einer Bestellung gehört selbstverständlich ein Kunde. Diesen wollen wir auf die einfachste Art und Weise anlegen, und zwar mit einer einzigen Anschrift statt mit mehreren Varianten wie Liefer- oder Rechnungsanschrift. Diese Tabelle heißt **tblKunde** und enthält die folgenden Felder:

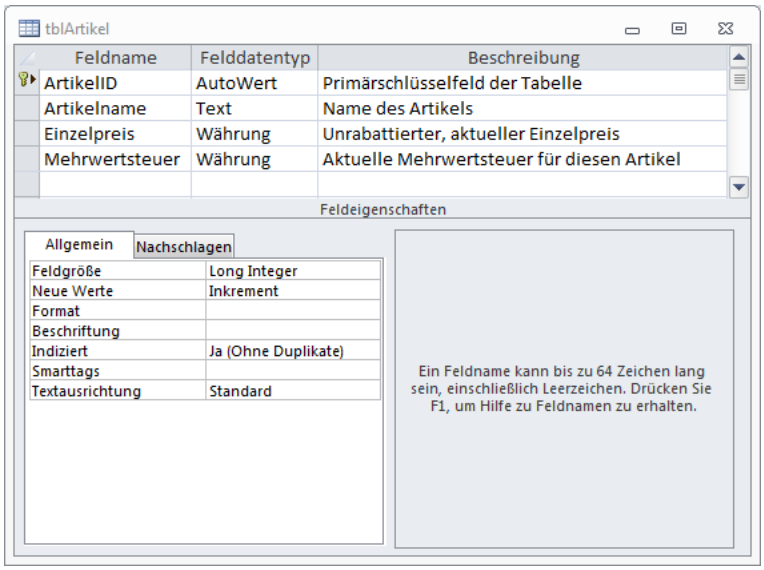


Bild 1: Entwurf der Tabelle **tblArtikel**

- **KundeID**: Primärschlüsselfeld der Tabelle
- **Firma**: Firma des Kunden
- **AnredeID**: Fremdschlüsselfeld zur Auswahl eines der Datensätze der Tabelle **tblAnreden**
- **Vorname**: Vorname des Kunden
- **Nachname**: Nachname des Kunden
- **Bezeichnung**: Eindeutige Bezeichnung, die je nach den vorhandenen Daten aus Firmenname, Vorname, Nachname, Anrede oder Kundennummer zusammengesetzt werden kann.
- **Strasse**: Straße des Kunden
- **PLZ**: PLZ des Kunden
- **Ort**: Ort des Kunden
- **Land**: Land des Kunden
- **EMail**: E-Mail-Adresse des Kunden
- **Telefon**: Telefonnummer des Kunden
- **Telefax**: Telefaxnummer des Kunden
- **Mobil**: Handynummer des Kunden

Um das Datenmodell bezüglich der Kunden zu komplettieren, hängen wir noch die Tabelle **tblAnreden** an, welche die Daten für das Feld **AnredeID** liefert. Zusammen sehen diese beiden Tabellen dann wie in Bild 2 aus.

Eine Erweiterung dieser Kundentabelle könnte aus Feldern für eine zweite Adresse bestehen, die dann entweder als abweichende Liefer- oder Rechnungsadresse dient. Der Einfachheit halber gehen wie in diesem Artikel jedoch davon aus, dass die Kunden nur eine Anschrift als Liefer- und Rechnungsadresse verwenden.

Bestellungen verwalten

Welche Daten müssen Sie überhaupt speichern, um eine Bestellung aufzunehmen? Gibt es überhaupt je Bestellung einen Datensatz oder müssen wir sogar mehrere Datensätze pro Bestellung anlegen, um auch die einzelnen Artikel pro Bestellung zu erfassen? Wir müssen beides berücksichtigen: Natürlich gibt es pro Bestellung Daten, die nur einmal erfasst werden müssen, da diese für alle Artikel der Bestellung gleich sind – zum Beispiel der Kunde oder das Bestelldatum. Zu einer Bestellung gehören aber oft auch mehr als ein Artikel – auf diesen Punkt kommen wir später zu sprechen. Als erstes legen wir eine Tabelle an, mit der Sie die Basisdaten einer jeden Bestellung erfassen.

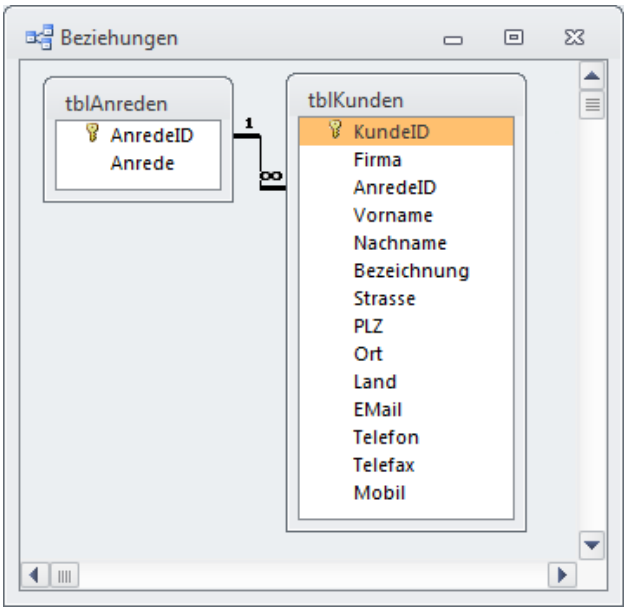


Bild 2: Die beiden Tabellen **tblKunden** und **tblAnreden** enthalten die Kundendaten.

Als Erstes benötigen Sie ein Primärschlüsselfeld, dass jede Bestellung eindeutig identifiziert. Danach legen Sie fest, welcher Kunde die Bestellung aufgibt. Dies erledigen Sie, indem Sie ein Fremdschlüsselfeld namens **KundeID** einrichten und dieses mit den Datensätzen der Tabelle **tblKunden** verknüpfen.

Wann wurde die Bestellung aufgegeben? Diese Frage beantworten wir in einem Datumsfeld namens **Bestelldatum**. Mehr benötigen wir zunächst nicht: Wir gehen zunächst davon aus, dass Sie die Bestellungen allein erfassen beziehungsweise dass die Anwendung nicht von verschiedenen Mitarbeitern verwendet wird. Daher pflegen wir keine Mitarbeiter-Tabelle in das Datenmodell ein.

Eine Erweiterung der Tabelle **tblBestellungen** wären Felder, mit denen Sie die für die Bestellung zu verwendende Adresse des Kunden speichern. Immerhin kann sich die Adresse des Kunden im Laufe der Zeit ändern. Wenn Sie die Lieferadresse zusätzlich in der Bestellungen-Tabelle speichern, können Sie später

Bestellungen in Formularen verwalten

Wer einmal einen Bestellvorgang auf Tabellenebene abgebildet und die entsprechenden Formulare und Berichte erstellt und verstanden hat, kann guten Gewissens behaupten, dass er kein Einsteiger mehr ist. Das Datenmodell erläutern wir im Artikel Bestellungen in Tabellen verwalten, die notwendigen Formulare stellt der vorliegende Artikel vor und um die Berichte kümmern wir uns im Artikel Bestellungen per Bericht ausgeben – es sind also nur noch wenige Schritte bis zum nächsten Level!

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1202_BestellungenInFormularen.mdb**.

Bestellungen per Formular

Die folgenden Ausführungen beziehen sich auf das Datenmodell, dass wir im Artikel **Bestellungen in Tabellen verwalten** erstellt haben. Dieses enthält im wesentlichen eine Tabelle namens **tblBestellungen**, die grundlegende Bestellinformationen aufnimmt, eine Tabelle mit Artikeln (**tblArtikel**) sowie eine Tabelle namens **tblBestellpositionen**, mit der Sie die einzelnen Artikel einer Bestellung zuordnen können.

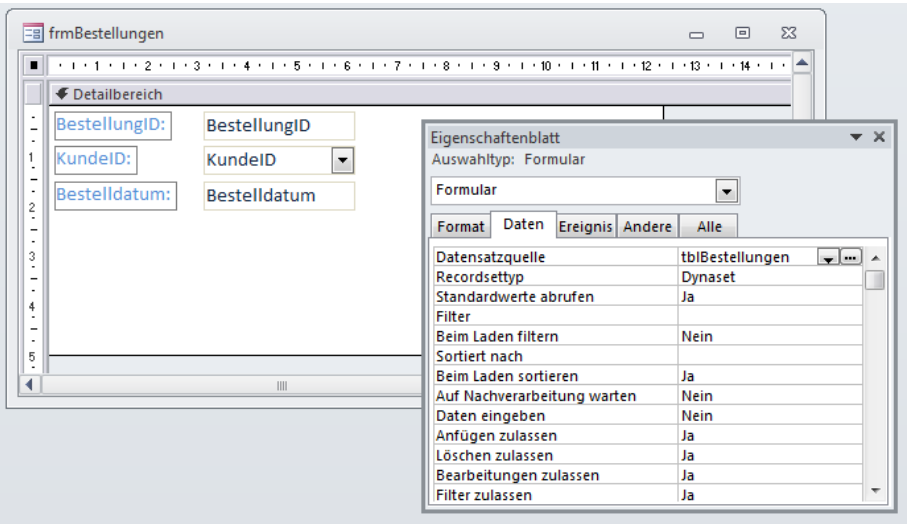


Bild 1: Hauptformular zur Eingabe der Bestelldaten

Für die Abbildung einer m:n-Beziehung bieten sich mehrere Konstellationen an, aber nur die Variante mit Haupt- und Unterformular ist sinnvoll. Dabei zeigt das Hauptformular die Daten der Tabelle **tblBestellungen** an.

Das Unterformular erfasst die bei der Bestellannahme anfallenden Bestellpositionen. Sie benötigen eigentlich noch weitere Formulare, etwa zur Verwaltung der Kunden oder der Artikel, diese beschreiben wir jedoch in weiteren Artikeln. Der vorliegende Artikel konzentriert sich auf die Eingabe der Bestelldaten.

Hauptformular

Das Hauptformular soll jeweils einen Datensatz der Tabelle **tblBestellungen** erfassen und auch das Unterformular zur Eingabe der Bestellpositionen aufnehmen. Legen Sie daher zunächst ein Formular namens **frmBestellungen** an und weisen Sie diesem die Tabelle **tblBestellungen** als Datenherkunft zu.

Ziehen Sie die drei Felder **BestellungID**, **KundeID** und **Bestelldatum** in den Detailbereich des Formulars (siehe Bild 1).

Ausrichten und anordnen können Sie die Steuerelemente später, wenn keine Änderungen mehr vorgenommen werden sollen.

Wenn Sie das Feld **KundeID** der Tabelle **tblBestellungen** wie im Artikel **Bestellungen in Tabellen verwalten** beschrieben als Nachschlagefeld ausgeführt haben, legt Access beim Hinzufügen des Feldes zum Formular automatisch ein Kombinationsfeld an, dass

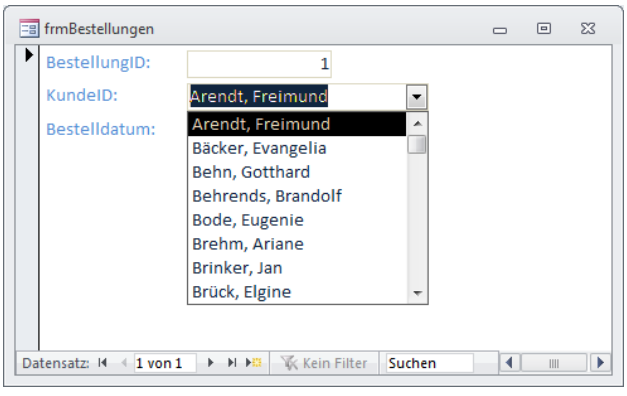


Bild 2: Auswahl des Kunden per Kombinationsfeld

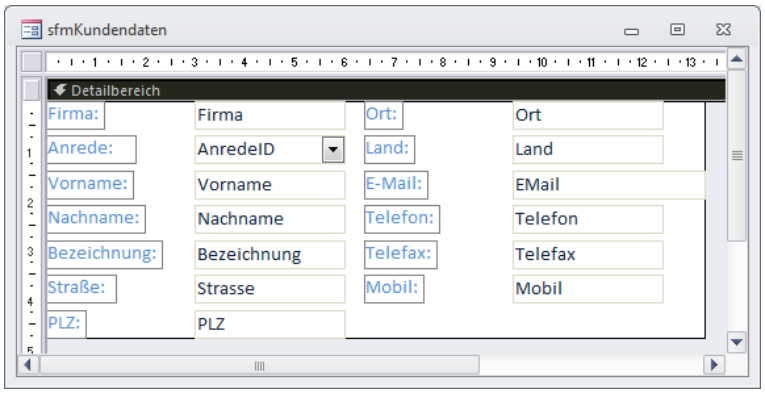


Bild 3: Unterformular zur Anzeige der Kundendaten

die gleichen Eigenschaften wie das entsprechende Nachschlagefeld aufweist (siehe Bild 2).

Kundendaten anzeigen

Gegebenenfalls möchten Sie nach der Auswahl des Kunden für eine Bestellung dessen aktuelle Adressdaten einsehen. Für diesen Fall verwenden wir ein Unterformular, dass alle notwendigen Daten anzeigt.

Das Formular heißt **sfmKundendaten** und enthält die Tabelle **tblKunden** als Datenherkunft. Ziehen Sie alle Felder dieser Tabelle außer **KundeID** in den Detailbereich des Formulars und ordnen Sie diese etwa wie in Bild 3 an – wie gesagt, optische Anpassungen nehmen wir später vor.

Dann schließen Sie das Unterformular **sfmKundendaten**, öffnen das Formular **frmBestellungen** in der Entwurfsansicht und ziehen das Unterformular **sfmKundendaten** aus dem Datenbankfenster beziehungsweise Navigationsbereich in den Detailbereich des Hauptformulars. Platzieren es etwa so wie in Bild 4.

Wenn Sie nun das Unterformular-Steuerelement markieren, zeigt das Eigenschaftsfenster auf der Registerseite **Daten** wahrscheinlich für die beiden Eigenschaften **Verknüpfen von** und **Verknüpfen nach** den Wert **KundeID** an. Sollte dies nicht der Fall sein,

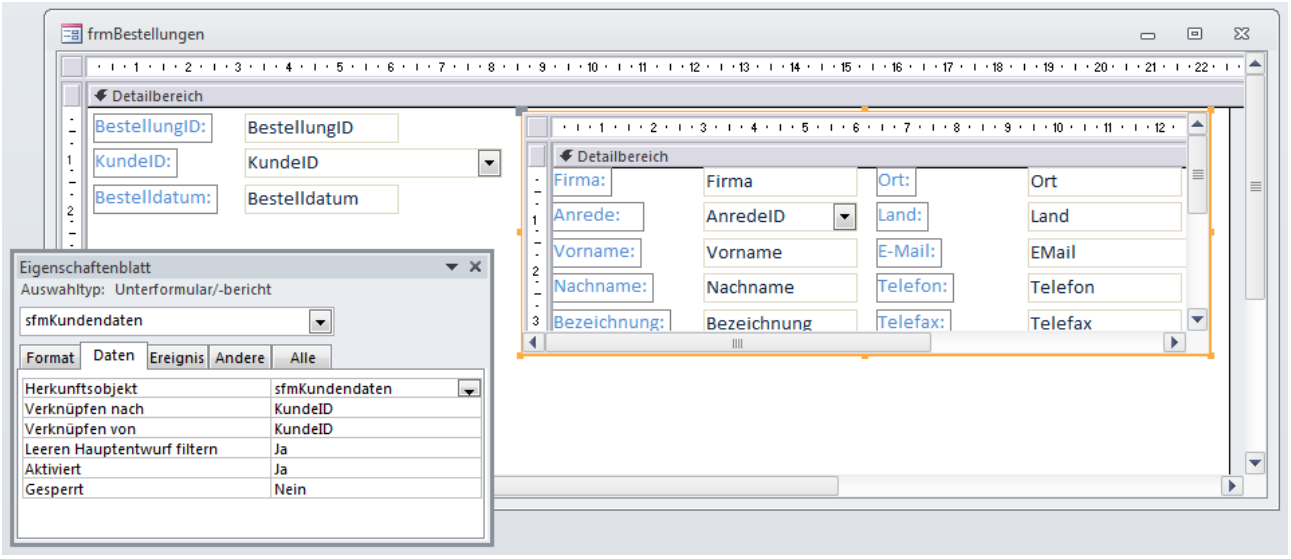


Bild 4: Herstellen der Verknüpfung zwischen Haupt- und Unterformular

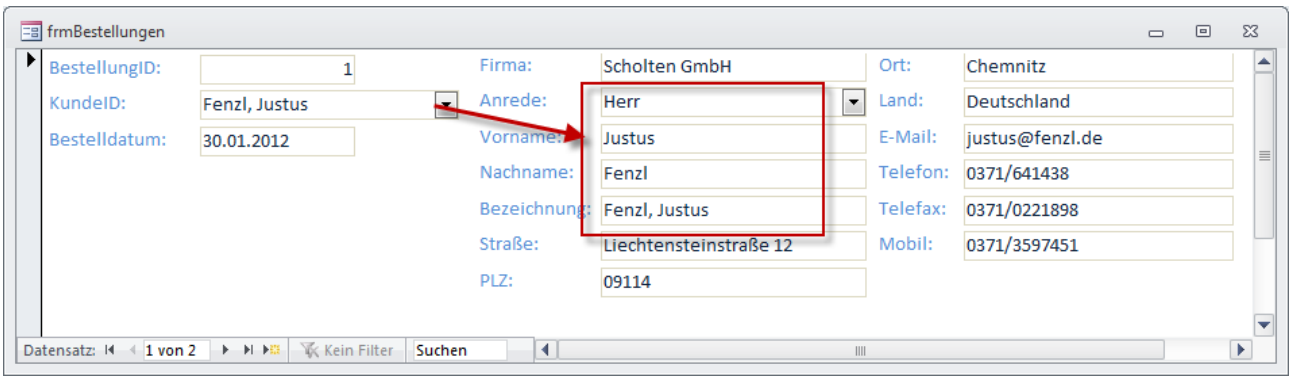


Bild 5: Synchronisieren des ausgewählten Kunden mit dem Unterformular **sfmKundendaten**

holen Sie diese Einstellung nach.

Damit sorgen Sie dafür, dass nach dem Auswählen eines Kunden mit dem Kombinationsfeld der entsprechende Kundendatensatz im Unterformular **sfmKundendaten** angezeigt wird (siehe Bild 5).

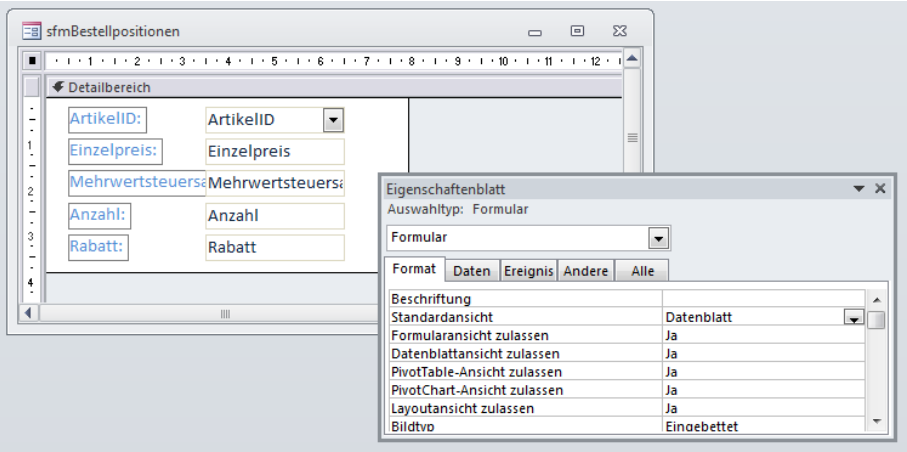


Bild 6: Unterformular zur Eingabe von Bestellpositionen

Unterformular zur Anzeige der Bestellungen

Die Bestellungen werden ebenfalls in einem Unterformular angezeigt. Allerdings liegt hier keine 1:n-Beziehung wie bei den Tabellen **tblBestellungen** und **tblKunden** vor, sondern scheinbar eine m:n-Beziehung.

Warum scheinbar? Nun: Das Unterformular zur Anzeige der Bestellpositionen enthält in der Tat nur Daten der Verknüpfungstabelle **tblBestellpositionen**. Die Artikel, die über die Verknüpfungstabelle mit dem Datensatz der Tabelle **tblBestellungen** des Hauptformulars verknüpft werden, erscheinen nur in Form des im Nachschlagefeld **ArtikelID** der Tabelle **tblBestellpositionen** angezeigten Artikelnamens.

Die Datenherkunft des Unterformulars **sfmBestellpositionen** beschränkt sich daher auch die Tabelle **tblBestellpositionen** – legen Sie diese Tabelle also als Datenherkunft eines neuen Formulars namens **sfmBestellpositionen** fest.

Ziehen Sie die Felder **ArtikelID**, **Einzelpreis**, **Mehrwertsteuersatz**, **Anzahl** und **Rabatt** in den Entwurf des Formulars **sfmBestellpositionen** (siehe Bild 6).

Stellen Sie außerdem die Eigenschaft **Standardansicht** für dieses Formular auf den Wert **Datenblatt** ein. Diese Ansicht zeigt alle Steuerelemente an, die sich im Detailbereich des Formulars befinden, daher dürfen Sie Felder wie **BestellpositionID** oder **BestellungID** gar nicht erst hinzu. Es reicht, wenn diese in der Datenherkunft enthalten sind, um etwa den Inhalt des Unterformulars über das Fremdschlüsselfeld **BestellungID** des Unterformulars mit dem Hauptformular zu verknüpfen.

Beachten Sie, dass auch hier das Feld **ArtikelID** als Kombinationsfeld ausgeführt wird. Auf diese Weise können Sie später leicht den gewünschten Artikel zu einer Bestellung hinzufügen.

In der Datenblattansicht verwendet Access die Bezeichnungsfelder als Spaltenüberschriften. Sie sollten diese daher vorher entsprechend anpassen – so können Sie **Mehrwertsteuersatz** in **MwSt.-Satz** ändern oder **ArtikelID** in **Artikel**.

Nun schließen Sie das Unterformular **sfmBestellpositionen**, öffnen das Formular **frmBestellungen** in der Entwurfsansicht und ziehen das Unterformular aus dem Datenbankfenster beziehungsweise dem Navigationsbereich von Access in den Detailbereich des Hauptformulars.

Markieren Sie das Unterformular-Steuerelement, um zu prüfen, ob das Eigenschaftsfenster für die beiden Eigenschaften **Verknüpfen von** und **Verknüpfen nach** jeweils den Wert **BestellungID** anzeigt (siehe Bild 7). Nur dann zeigt das Unterformular genau die Bestellpositionen zum im Hauptformular enthaltenen Bestellvorgang an. Sollten diese Eigenschaften nicht den Wert **BestellungID** enthalten, stellen Sie diese nun entsprechend ein.

Anschließend können Sie bereits in die Formularansicht des Formulars **frmBestellungen** wechseln und die erste Bestellung anlegen (siehe Bild 8). Dazu wählen Sie im Hauptformular zunächst den Kunden aus und tragen das Bestelldatum ein. Danach wählen Sie mit dem Kombinationsfeld des Unterformulars **sfmBestellpositionen** die gewünschten Artikel für die Bestellung aus. Leider fehlt hier noch etwas: Die Felder **Einzelpreis** und **Mehrwertsteuersatz** werden noch nicht automatisch mit den Daten des ausgewählten Artikels gefüllt. Das Feld **Anzahl** könnten

Rechnen in Formularen

Es gibt verschiedene Gelegenheiten, in Formularen die Ergebnisse von Rechenoperationen zu präsentieren. Dabei lassen sich viele Berechnungen in Abfragen auslagen, die als Datenherkunft der Formulare dienen, manchmal gelingt dies jedoch nicht. Dieser Artikel zeigt, wie Sie einfache Berechnungen bis hin zu formularübergreifenden Berechnungen durchführen und was Sie dabei beachten müssen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1202_RechnenInFormularen.mdb**.

Einfache Formeln in Textfeldern

Beginnen wir mit einem scheinbar einfachen Beispiel. Im Formular aus Bild 1 finden Sie zwei Textfelder namens **txtSummand1** und **txtSummand2**, die zur Eingabe zweier Summanden dienen. Das Textfeld **txtSumme** soll die Summe der beiden Werte liefern. Damit dies geschieht, weisen Sie der Eigenschaft Steuerelementinhalt den folgenden Ausdruck zu:

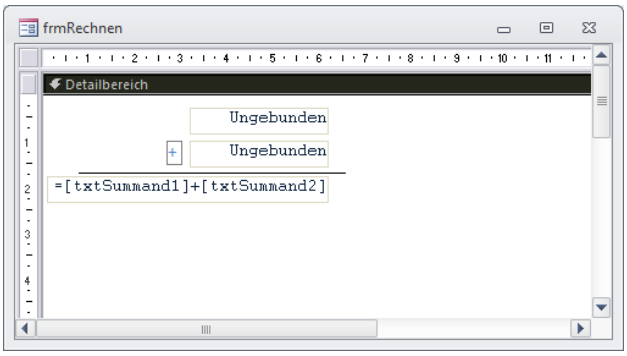


Bild 1: Summieren zweier Textfelder

```
=[txtSummand1]+[txtSummand2]
```

Ein Wechsel in die Formularansicht und die Eingabe zweier Zahlen zeigt, dass dies so nicht funktioniert: Der Plus-Operator addiert die Zahlen nicht, sondern fasst diese als Zeichenfolgen auf und führt die beiden Textketten zusammen (siehe Bild 2). Access geht davon aus, dass ein ungebundenes Textfeld eine Zeichenkette enthält und fügt diese durch den Plus-Operator zusammen. Wenn Sie sicherstellen wollen, dass Access mit Zahlen arbeitet, können Sie den Summanden beispielsweise die **Val**-Funktion voranstellen (die dann mit **Wert** übersetzt wird). Der Ausdruck sieht dann etwa so aus:

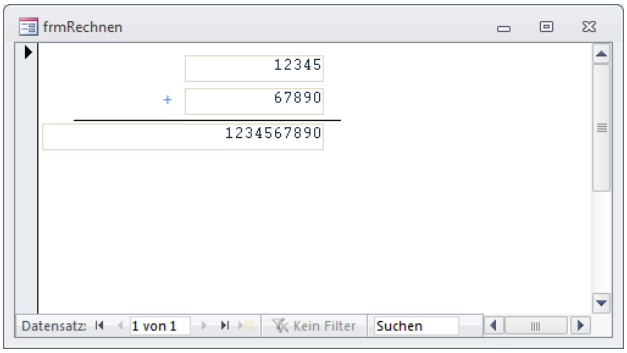


Bild 2: Zeichenverkettung statt Addition

```
=Wert([txtSummand1])+Wert([txtSummand2])
```

Allerdings streikt Access dann, wenn die Textfelder beim Öffnen des Formulars keinen Wert enthalten und liefert als Ergebnis einen Ausdruck wie **#Typ!**.

Fehler bei leeren Feldern

In diesem und in anderen Fällen kann die **Wert**-Funktion die enthaltenen **Null**-Werte nicht verarbeiten. Verwenden Sie dann die **Nz**-Funktion, um für gegebenenfalls leere Textfelder den Wert **0** zu liefern:

```
=Wert(Nz([txtSummand1];0))+Wert(Nz([txtSummand2];0))
```

Probleme gibt es hier immer noch, wenn Sie Zahlen mit Nachkommastellen angeben. Die **Val**-Funktion erkennt nur den Punkt als Dezimaltrennzeichen. Sollten Sie alternativ das Komma als Dezimaltrennzeichen verwenden wollen, was in der deutschen Systemeinstellung eingestellt sein dürfte, können Sie alternativ die Funktion **CDbl** verwenden, um den vermeintlichen Text in eine Zahl umzuwandeln. Der Punkt als Trennzeichen wird dann allerdings komplett ignoriert – aus **12.5** wird dann **125**.

Rechnen mit gebundenen Textfeldern

Viel einfacher wird dies, wenn die Textfelder an Zahlenfelder einer Tabelle gebunden sind. Dies gilt vor allem, weil Access prüft, welchen Datentyp diese Felder haben und automatisch eine Addition statt einer Zeichenverkettung durchführt. Im Beispiel verwenden wir die Tabelle **tblZahlen**, das zwei Felder namens **Zahl1** und **Zahl2** enthält. Beide Felder sind als Zahlenfelder mit dem Untertyp **Single** ausgestattet, um Nachkommastellen zu verarbeiten. Hier brauchen Sie keine Konvertierung mehr durchzuführen. Allerdings sollten Sie die

Nz-Funktion einsetzen, um leere Felder als **0** zu verarbeiten. Da es sich um gebundene Felder mit dem Datentyp **Zahl** handelt, brauchen Sie den zweiten Parameter der **Nz**-Funktion nicht anzugeben – dieser verwendet in diesem Fall gleich den Wert **0** (siehe Bild 3).

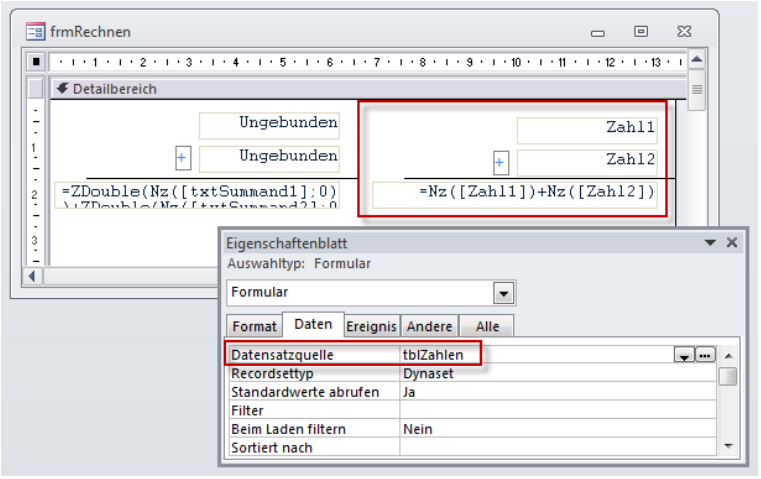


Bild 3: Beispiel für die Addition gebundener Felder

Berechnung in Abfrage auslagern

Sie können die Berechnung auch in eine Abfrage auslagern. Diese wird dann später als Datenherkunft des Formulars angegeben und liefert das Feld mit dem Berechnungsergebnis gleich mit. Eine Abfrage auf Basis der Tabelle **tblZahlen**, welche die Werte der Felder **Zahl1** und **Zahl2** addiert, sieht wie in Bild 4 aus.

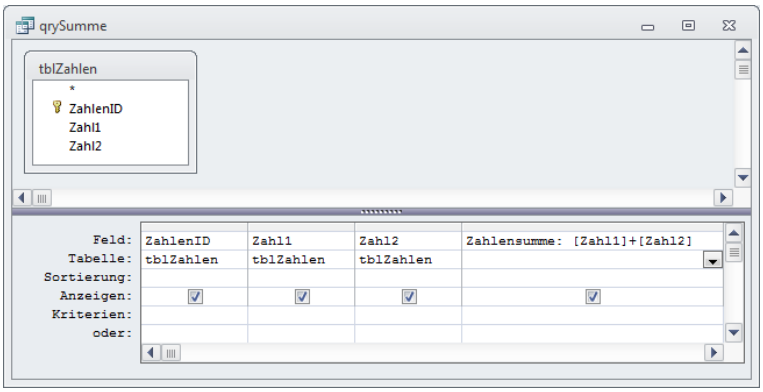


Bild 4: Addition per Abfrage

Im Beispielformular **frmZahlen** haben wir die bereits darüber verwendeten Felder **Zahl1** und **Zahl2** erneut untereinander angelegt und das Berechnungsergebnis gleich darunter platziert. Was auch bereits in der Datenblattansicht der Abfrage hätte auffallen können: Die Genauigkeit bei Berechnung in der Abfrage entspricht nicht der bei der Berechnung direkt im Formular. Bei der Berechnung im Formular verwendet Access auch für das Ergebnis den Single-Datentyp, für die Berechnung in der Abfrage anscheinend den Double-Datentyp. Warum sich dies derart auf das Ergebnis auswirkt, ist uns nicht bekannt. Das einfachste Gegenmittel zur Sicherung hoher Genauigkeit auch bei Zahlen mit Nachkommastellen ist es, den Datentyp **Währung** zu verwenden. Wenn Sie das Format eines solchen Feldes auf **Allgemeine Zahl** einstellen, wird das Währungssymbol nicht angezeigt und Sie können mit Festkommazahlen statt mit Gleitkommazahlen arbeiten. Auch hierfür haben wir ein Beispiel zur Tabelle **tblZahlen**, zur Abfrage **qrySumme** und zum Formular **frmRechnen** hinzugefügt.

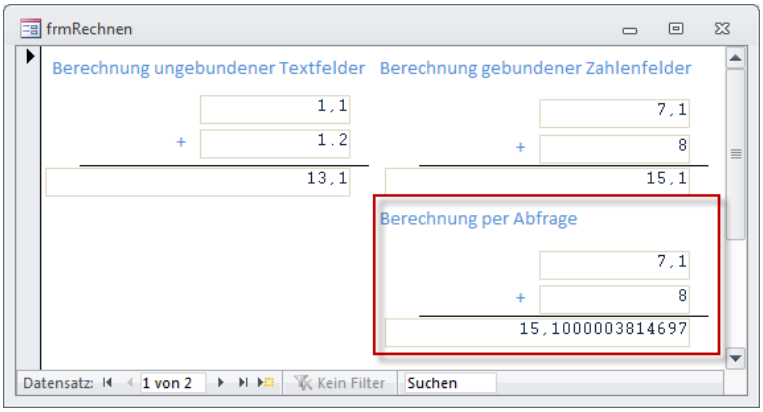


Bild 5: Addition per Abfrage im Formular

Berechnungen in Formularen

Gegebenenfalls möchten Sie dem Benutzer nicht nur, wie in obigem Beispiel, die Möglichkeit zum Summieren zweier Zahlen geben, sondern diesem auch die Eingabe individueller Berechnungsformeln erlauben. Hier hilft die **Eval**-Funktion weiter. Sie nimmt einen Ausdruck wie beispielsweise eine Rechenoperation wie **1+4** entgegen und liefert das Ergebnis zurück (siehe Bild 5). Um dies zu testen, legen Sie in einem Formular zwei Textfelder namens **txtAusdruck** und **txtErgebnis** an. Der Ausdruck soll berechnet werden, wenn sich die Einfügemarke im Textfeld **txtAusdruck** befindet und der Benutzer die Eingabetaste betätigt. Dazu legen Sie eine Ereignisprozedur an, die durch das Ereignis **Bei Taste ab** des Textfeldes **txtAusdruck** ausgelöst wird. Diese Prozedur sieht

Rechnen in Berichten

Berichte bieten die gleichen Möglichkeit zur Berechnung von Werten wie Formulare – und noch einige mehr. Im Gegensatz zu Formularen bieten Berichte die Möglichkeit, eine laufende Summe zu bilden oder Berechnungen über komplette Gruppierungen durchzuführen. In diesem Artikel lernen Sie die Grundlagen zum Rechnen in Berichten.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1203_RechnenInBerichten.mdb**.

Datensatzweise berechnen

Wie in Formularen können Sie auch innerhalb eines Datensatzes eines Berichts Berechnungen durchführen. Dies gelingt entweder direkt in der zugrunde liegenden Abfrage oder aber durch Angabe der Berechnungsformeln in den Steuerelementen zur Anzeige des Ergebnisses.

Als Grundlage für die Beispiele dieses und der folgenden Abschnitte dient ein Bericht, der die Tabelle **tblArtikel** beziehungsweise eine darauf aufbauende Abfrage als Datenherkunft verwendet (siehe Bild 1).

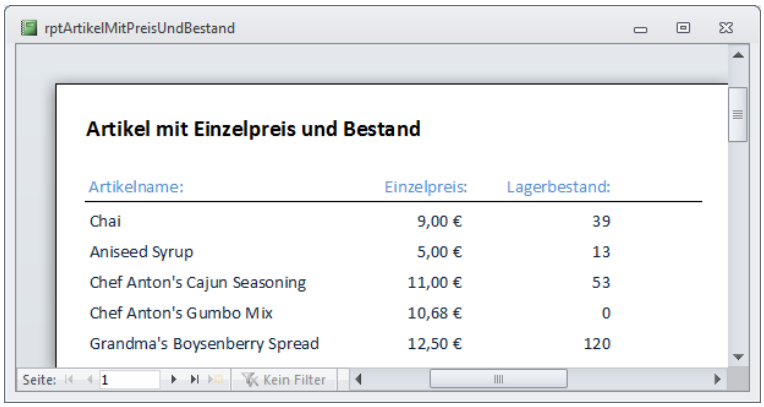


Bild 1: Grundlage für die Beispiele dieses Artikels: ein Bericht mit Artikeln, Einzelpreisen und Lagerbeständen.

Berechnung im Textfeld

Im ersten Beispiel soll der Bericht den Wert der gelagerten Artikel pro Artikel ausgeben (siehe Bild 3). Dazu fügen Sie dem Bericht im Detailbereich ein Steuerelement etwa namens **txtWertLagerbestand** hinzu, das den folgenden Ausdruck für die Eigenschaft **Steuerelementinhalt** verwendet:

```
=Einzelpreis*Lagerbestand
```

Die Entwurfsansicht des Berichts sieht nun wie in Bild 2 aus. Gegebenenfalls müssen Sie noch die Eigenschaft **Format** des Steuerelements **txtWertLagerbestand** auf **Währung** einstellen.

Bild 3: Eine weitere Zeile zeigt das Produkt aus Einzelpreis und Lagerbestand an.

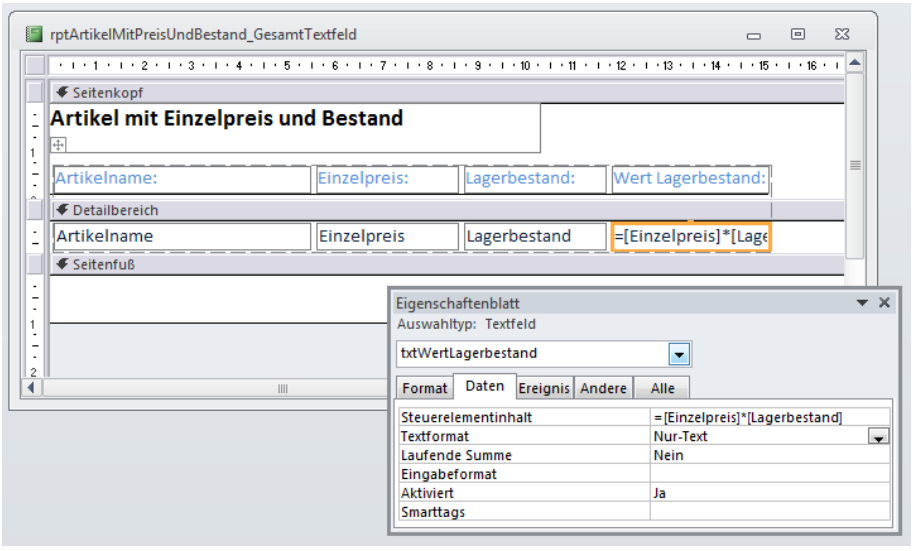


Bild 2: Hinzufügen eines Feldes zur Berechnung des Wertes der Artikel im Lager

Berechnung per Abfrage

Das gleiche Ergebnis erhalten Sie, wenn Sie eine Abfrage namens **qryWertLagerbestand** erstellen (siehe Bild 4). Das berechnete Feld **WertLagerbestand** enthält die folgende Berechnungsformel:

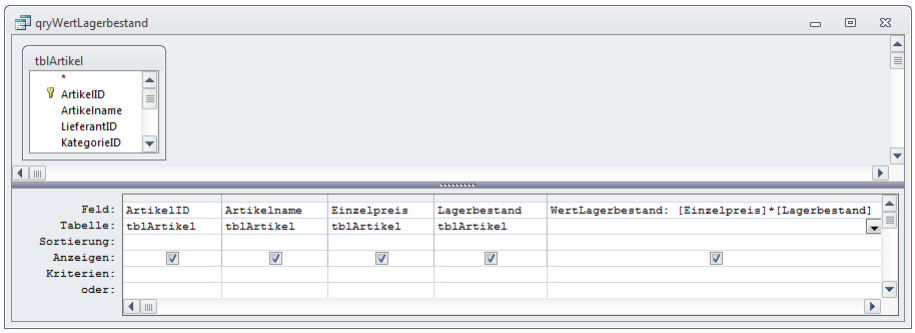


Bild 4: Abfrage zur Berechnung des Werts des Lagerbestands je Artikel

```
WertLagerbestand:
[Einzelpreis]*[Lagerbestand]
```

Der Bericht **rptArtikelMitPreisUndBestand_GesamtAbfrage** verwendet diese Abfrage als Datenherkunft. Er ist genauso aufgebaut wie der Bericht aus dem vorherigen Beispiel, verwendet aber statt eines Textfeldes mit einer Berechnungsformel gleich das in der Abfrage berechnete Feld (siehe Bild 5).

Bild 5: Bericht mit Berechnungsergebnis aus einer Abfrage

Gesamtsumme für einen Bericht

Als nächstes soll der Bericht auf der letzten Seite die Gesamtsummen der gelagerten Artikel und des Warenwerts ausgeben. Dazu bietet sich der Berichtsfuß-Bereich an, den Sie mit einem Rechtsklick auf einen der Bereichsköpfe und anschließende Auswahl des Kontextmenüeintrags **Seitenkopf/-fuß** in der Entwurfsansicht des Berichts hinzufügen.

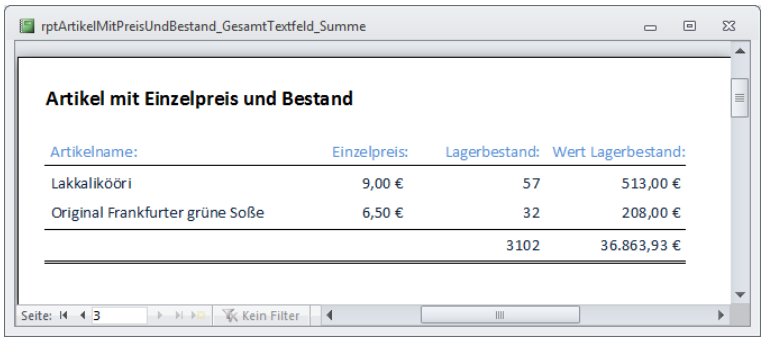


Bild 6: Summe der gelagerten Artikel und ihrer Preise

Den Berichtskopfbereich lassen Sie verschwinden, indem Sie seine Höhe auf **0cm** einstellen (entweder durch Setzen der Eigenschaft oder durch Ziehen mit der Maus). Gleiches erledigen Sie für den Seitenfußbereich des Berichts.

Das Ergebnis soll wie in Bild 6 aussehen. Um dieses zu erreichen, führen Sie zwei Schritte durch. Als Erstes legen Sie direkt unter dem Feld **Lagerbestand** im Bereich **Berichtsfuß** ein neues Steuerelement namens **txtLagerbestandSumme** an.

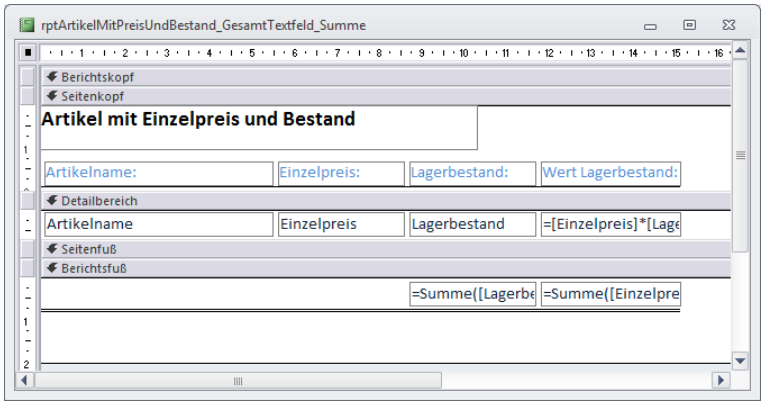


Bild 7: Entwurf des Berichts mit Summen im Berichtsfuß

Wenn Sie das entsprechende Steuerelement aus dem Detailbereich kopieren und im Berichtsfuß wieder einsetzen, hat dieses gleich die richtige Höhe und Breite, außerdem ist die Eigenschaft **Steuerelementinhalt** bereits mit dem Wert **Lagerbestand** ge-

Gruppierungen in Berichten

Berichte bieten zwei Möglichkeiten, Daten hierarchisch zu strukturieren: per Unterbericht oder durch sogenannte Gruppierungen. Letztere ermöglicht es beispielsweise, alle Bestellungen eines Kunden anzuzeigen und zu jeder Bestellung auch noch die Bestelldetails mit den Artikeln anzuzeigen oder alle Artikelkategorien mit den dazugehörigen Artikel zu präsentieren. Gruppierungen erlauben dabei eine ganze Reihe von Einstellungen, die dieser Artikel vorstellt.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1203_GruppierungenIn-Berichten.mdb**.

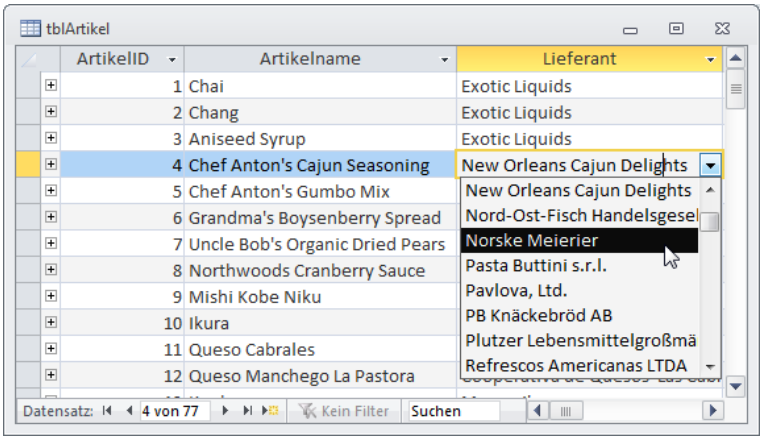


Bild 1: Diese Daten reichen, um Artikel nach Lieferant gruppiert darzustellen.

Datenherkunft für Gruppierungen

Voraussetzung für die gruppierte Ansicht von Daten in einem Access-Bericht ist eine entsprechend vorbereitete Datenherkunft. Dabei kann es sich um eine Tabelle oder auch um eine Abfrage handeln – je nachdem, ob die Tabelle bereits alle für eine anschauliche Gruppierung nötigen Daten enthält. Wenn Sie beispielsweise alle Artikel der Tabelle **tblArtikel** der Beispieldatenbank nach Lieferanten gruppiert anzeigen möchten, sollte die Tabelle auch alle anzuzeigenden Daten liefern. Im Falle der Artikel aus Bild 1 funktioniert das: Die Tabelle selbst enthält zwar nur ein Fremdschlüsselfeld zum Festlegen eines Datensatzes der verknüpften Tabelle **tblLieferanten**, aber Access erkennt in Berichten automatisch auch die im Nachschlagefeld angezeigten Daten. Und dabei wird noch nicht einmal ein Kombinationsfeld angezeigt (das in einem Bericht ja ohnehin nicht funktioniert), sondern ein Textfeld.

Wenn die Daten für den Aufbau der Gruppierung im Bericht hingegen aus mehreren Tabellen stammen, müssen Sie eine geeignete Abfrage erstellen. Im zweiten Beispiel weiter unten werden wir in einem Bericht alle Kunden samt Bestellungen und Bestellpositionen anzeigen. Dazu benötigen wir dann bereits eine Abfrage, die auf den Tabellen **tblKunden**, **tblBestellungen** und **tblBestelldetails** basiert (eigentlich wäre auch noch **tblArtikel** gefragt, aber den Artikelnamen beziehen wir aus dem entsprechenden Nachschlagefeld der Tabelle **tblBestelldetails**).

Aufbau einer Gruppierung

Eine Gruppierung in einem Access-Bericht ist immer dann sinnvoll, wenn Sie Daten aus hierarchisch aufgebauten Tabellen strukturiert anzeigen möchten. Ein einfacher Fall ist das oben beschriebene Beispiel der Lieferanten und Artikel. Der Bericht soll jeweils in einem Kopfbereich alle Lieferanten anzeigen und unterhalb des Kopfbereichs, auch Gruppenkopf genannt, alle Artikel liefern, die der jeweilige Lieferant anbietet.

Um einen solchen Bericht aufzubauen, fügen Sie einem neuen, leeren Bericht in der Entwurfsansicht zunächst die Tabelle **tblArtikel** als Wert für die Eigenschaft **Datenherkunft** hinzu.

Der Bericht zeigt nicht automatisch einen entsprechenden Gruppenkopf an, da er ja noch gar nicht weiß, dass er gruppierte Daten anzeigen soll. Dies teilen wir ihm aber gleich mit, und zwar im Bereich **Gruppieren, Sortieren und Summe** unterhalb des Berichts (Access 2007 und jünger) beziehungsweise im Dialog **Sortieren und gruppieren** (Access 2003 und älter). Diesen Bereich öffnen Sie in allen aktuelleren Access-Versionen über den Kontextmenü-Eintrag **Sortieren und gruppieren** eines der Köpfe der Berichtsbereiche (siehe Bild 2).

Gruppieren unter Access 2003 und älter

Unter Access 2003 und älter finden Sie den kleinen, übersichtlichen Dialog aus Bild 3 vor, wenn Sie eine

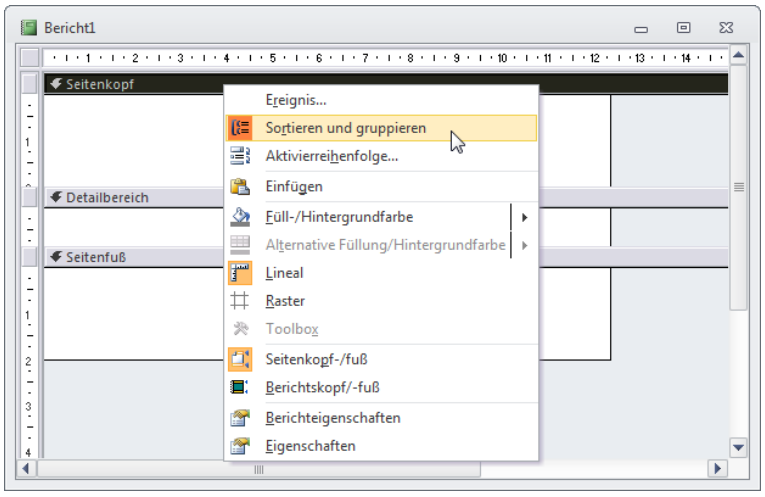


Bild 2: Diese Daten reichen, um Artikel nach Lieferant gruppiert darzustellen.

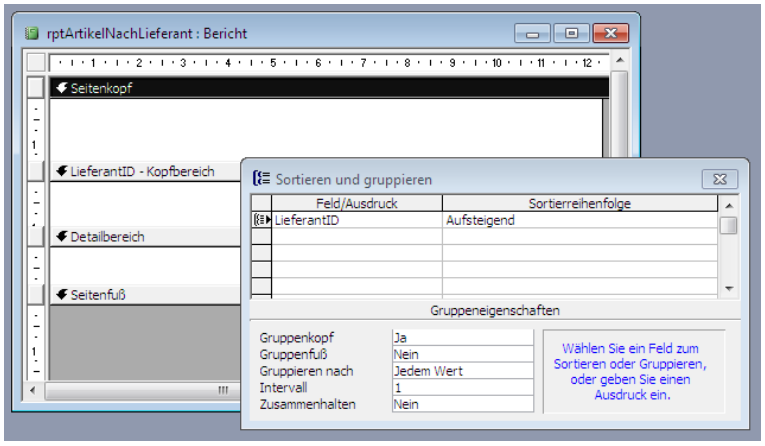


Bild 3: Der Dialog zum Anlegen und Anpassen von Gruppierungen unter Access 2003 und älter

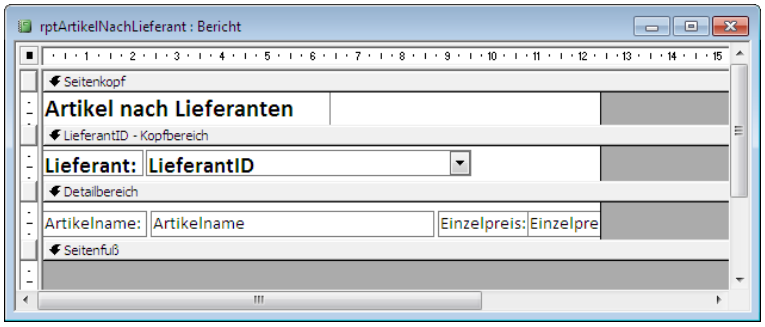


Bild 4: Entwurf eines Berichts mit einem einfachen Gruppenkopfbereich

Gruppierung anlegen möchten. Dieser Dialog zeigt im oberen Teil eine Liste aller vorhandenen Sortierungen und Gruppierungen an und im unteren Bereich die Eigenschaft des jeweils oben markierten Eintrags.

Wenn Sie wissen, nach welchem Feld Sie die Gruppierung aufbauen möchten, wählen Sie dieses Feld in der Spalte **Feld/Ausdruck** aus. Eine Gruppierung ist immer auch eine Sortierung, also legen Sie in der rechten Spalte fest, ob die Sortierung aufsteigend oder absteigend erfolgen soll.

Bislang haben Sie nur eine Sortierung angelegt, denn es fehlt das entscheidende Element der Gruppierung: der Gruppenkopf und/oder der Gruppenfuß. Damit den beiden Eigenschaften **Gruppenkopf** und **Gruppenfuß** legen Sie fest, welche zusätzlichen Bereiche je Gruppe angezeigt werden sollen. Wenn Sie dies nicht tun, werden die Datensätze lediglich nach dem angegebenen Feld sortiert.

Stellen Sie hingegen den Wert **Gruppenkopf** oder **Gruppenkopf** auf **Ja** ein, wird im gleichen Moment ein entsprechender Bereich im Berichtsentwurf eingeblendet.

Diesen sollten Sie tunlichst nutzen, um dort Daten bezüglich des Gruppierungsfeldes anzugeben – beispielsweise also das Feld **LieferantID**. Bild 4 zeigt den Entwurf eines Bericht mit einer Gruppierung nach dem Lieferanten. Das bedeutet, dass für jeden Lieferanten ein neuer Gruppenkopf angezeigt wird, der den Namen des Lieferanten enthält. Nach dem Gruppenkopf folgen die Artikel, deren Lieferant dem im Gruppenkopf angezeigten Lieferant entsprechen – hier mit Artikelname und Einzelpreis. Lieferanten, die für keinen Artikel als Lieferant angegeben wurden, erscheinen in keinem Gruppenkopf.

Schauen Sie sich in Bild 5 an, wie der Bericht in der Seitenansicht aussieht. Wenn Sie nun noch dafür sorgen möchten, dass die Artikel einer jeden Gruppierung in alphabetischer Reihenfolge erscheinen, fügen Sie im Dialog **Sortieren und gruppieren** in einer weiteren Zeile das Feld **Artikelname** hinzu – ohne einen Gruppenkopf oder -fuß zu definieren (siehe Bild 6). Beachten Sie die Reihenfolge: Wenn Sie die Sortierung oberhalb der Gruppierung anlegen (die Reihenfolge können Sie per Drag and Drop einstellen), erhalten Sie in der Seitenansicht das Ergebnis aus Bild 7. Dort werden tatsächlich zunächst zunächst die Artikel nach dem Alphabet sortiert (und gruppiert – wenn auch ohne Gruppenkopf) und erst dann die Gruppen-

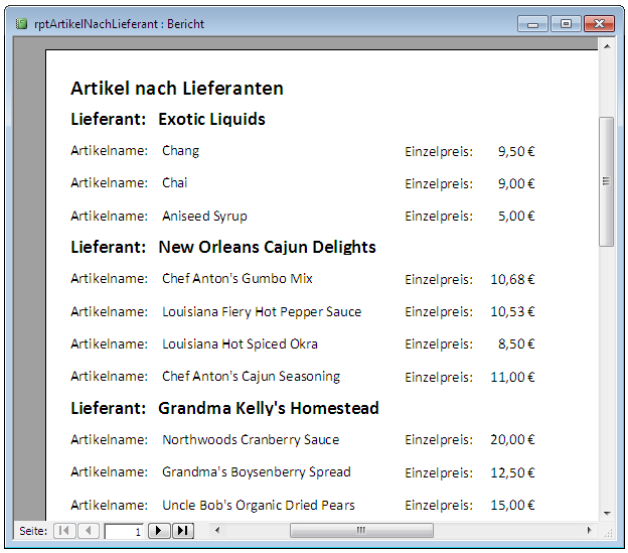


Bild 5: Einfacher Kopfbereich mit Lieferanten in der Seitenansicht

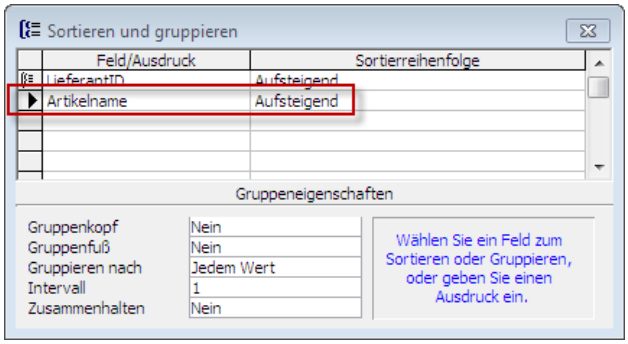


Bild 6: Festlegen einer Sortierung innerhalb der Elemente einer Gruppierung

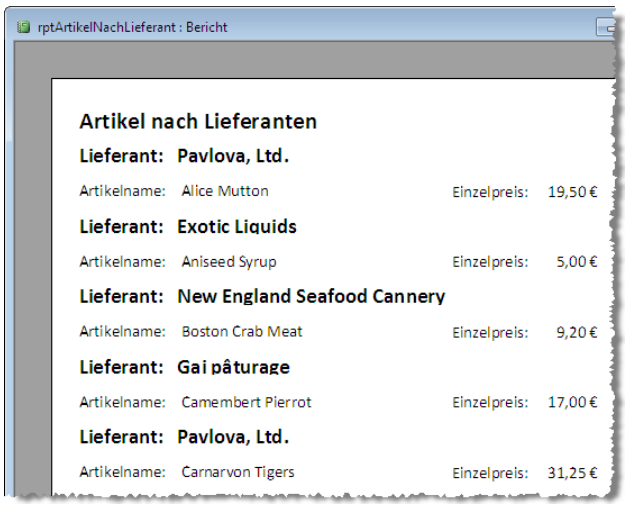


Bild 7: Diese Gruppierung mit Kopfbereich ist sinnlos.

köpfe für die Gruppierung nach dem Feld **LieferantID** hinzugefügt. Das macht natürlich keinen Sinn – den Lieferant könnten Sie dann auch gleich im Detailbereich einfügen.

Gruppierungseigenschaften

Der Dialog **Sortieren und gruppieren** bietet neben **Gruppenkopf** und **Gruppenfuß** noch weitere Eigenschaften. Die Eigenschaft **Gruppieren nach** beispielsweise erlaubt als Einstellung **Jedem Wert** oder **Intervall**. Wenn Sie einfach nach diskreten Datensätzen wie im vorherigen Beispiel nach Lieferanten gruppieren, verwenden Sie die Einstellung **Jedem Wert**. Wenn Sie hingegen ganze Bereiche von Werten in einer Gruppierung zusammenfassen möchten, gelingt dies mit der Einstellung **Intervall**. In diesem Fall legen Sie mit der Eigenschaft **Intervall** das eigentliche Intervall fest.

Nach Anfangsbuchstabe gruppieren

Wenn Sie etwa die Tabelle **tblKunden** als Datenherkunft verwenden, können Sie die Kunden nach dem Anfangsbuchstaben gruppieren lassen. Dazu ziehen Sie das Feld **Firma** in den Dialog **Sortieren und gruppieren**, legen einen Gruppenkopf an und stellen die Eigenschaft **Gruppieren nach** auf den Wert **Anfangszeichen** ein (wird nur bei Textfeldern angeboten). Damit die Kunden innerhalb der Gruppierung auch noch alphabetisch sortiert angezeigt werden, fügen Sie das Feld **Firma** ein weiteres Mal zum Dialog **Sortieren und Gruppieren** hinzu – diesmal als reines Sortierkriterium. Im Gruppenkopf legen Sie ein neues Textfeld namens **txtAnfangsbuchstabe** an und stellen für seine Eigenschaft **Steuerelementinhalt** den Wert **=Links("Firma";1)** ein, was den jeweiligen Anfangsbuchstaben der Gruppe ausgibt (siehe Bild 8). Das Ergebnis zeigt Bild 9.

Alternative Intervalle einstellen

Wenn Sie im vorherigen Beispiel ein anderes Intervall als **1** einstellen, fasst die Gruppierung nicht etwa jeweils zwei Gruppen von Kundennamen zusammen (also etwa **A-B**, **C-D** und so weiter), sondern wählt die ersten beiden Anfangsbuchstaben als Kriterium für die Zusammenstellung der Gruppen zusammen! Auch das kann hilfreich sein, beispielsweise wenn Sie sehr viele Namen in einem Bericht anzeigen möchten. Sie sollten dann allerdings auch den Gruppenkopf so anpassen, dass dieser die ersten beiden Buchstaben anzeigt – die Eigenschaft **Steuerelementinhalt** des Steuerelements **txtAnfangsbuchstabe** würden Sie dann also mit dem folgenden Ausdruck füllen:

```
=Links("Firma";1)
```

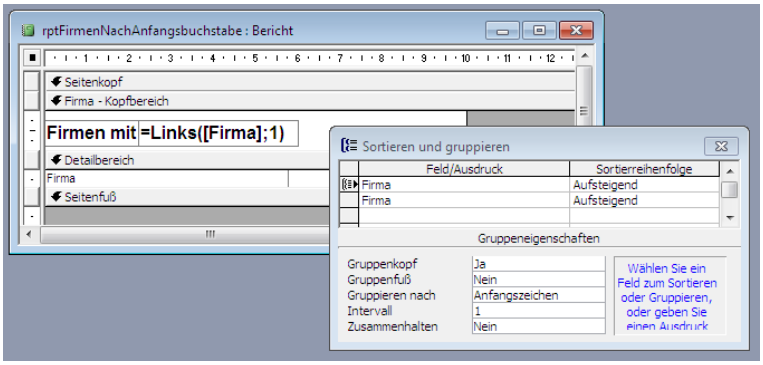


Bild 8: Einstellungen für die Ausgabe von Kundennamen nach dem Anfangsbuchstaben

Stellen Sie also das Gruppierungskriterium auf den folgenden Ausdruck ein:

```
=Asc(Links([Firma];1))-1
```

Damit wird der Zahlenwert des ersten Buchstaben ermittelt. Das Subtrahieren von **1** am Ende der Zeile gleicht einen scheinbaren Bug von Access aus, durch den die Gruppierung je nach dem Gruppierungskriterium immer etwas beschnitten wird.

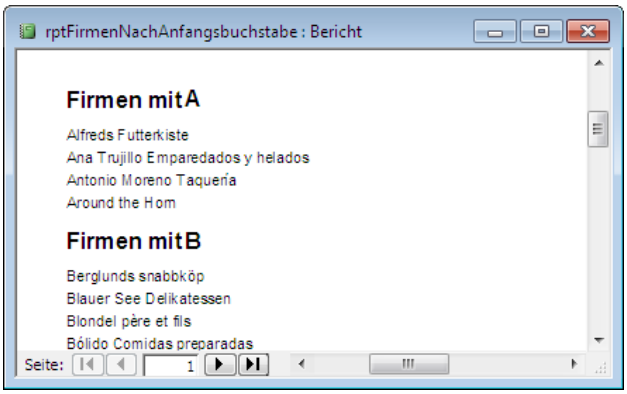


Bild 9: Ausgabe der Kundennamen nach dem Anfangsbuchstaben

Außerdem erhält die Eigenschaft **Steuerelementinhalt** des Textfeldes im Gruppenkopf diesen Ausdruck:

```
=Links([Firma];1) & "-" & Zchn(Asc(Links([Firma];1))+1)
```

Das sieht recht kompliziert aus, ist es aber nicht. Der linke Teil liefert den ersten Buchstaben der Gruppe (zum Beispiel **A**), der rechte den zweiten (also etwa **B**). Der rechte Ausdruck ermittelt von innen nach außen zunächst den ersten Buchstaben der aktuellen Firma. Die **Asc**-Funktion ermittelt den ASCII-Code (bei **A** etwa **65**). Zu diesem Wert wird **1** addiert, was **66** ergibt. Mit der **Chr**-Funktion erhalten wir den dieser Zahl entsprechenden Buchstaben, also **B**. Richtig rund läuft dies nur, wenn es zu jedem Buchstaben eine Firma gibt (siehe Bild 10).

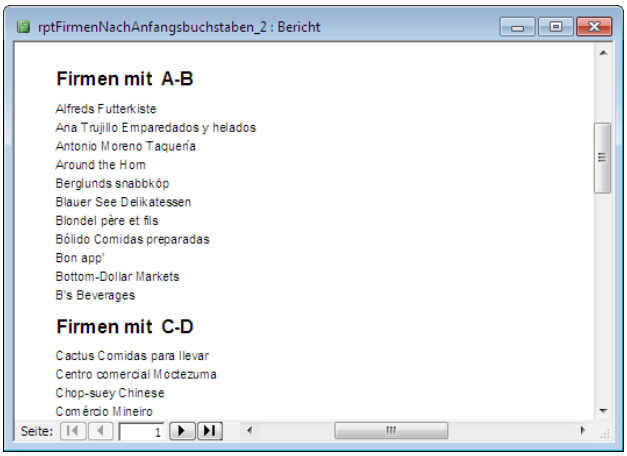


Bild 10: Kundennamen in Gruppen von zwei Anfangsbuchstaben

Intervallgruppen

Wenn Sie tatsächlich mehrere Gruppen zu einer zusammenfassen möchten, also beispielsweise jeweils zwei Anfangsbuchstaben, dann gelingt dies nur mit einem Zahlenwert als Gruppierungskriterium. Das ist aber kein Problem: Schließlich lassen sich Buchstaben mit der **Asc**-Funktion in die entsprechenden ASCII-Werte umwandeln (**Asc("A")** liefert beispielsweise **65**).

Sonderfall Datum

Wenn Sie beispielsweise mit einem Bericht alle Kunden ausgeben möchten, die in einem bestimmten Monat eine Bestellung aufgegeben haben, gelingt dies mit einer Gruppierung ganz einfach. Sie müssen dazu einfach nur eine entsprechende Datenherkunft zusammenstellen, in diesem Fall eine Abfrage auf Basis der beiden Tabellen **tblKunden** und **tblBestellungen** (siehe Bild 11). Diese Abfrage fügen Sie dem Bericht als **Datenherkunft** hinzu. Anschließend legen Sie zunächst die Gruppierung nach dem Feld **Bestellung** an. Öffnen Sie nun das Auswahlfeld für die Eigenschaft **Gruppieren nach**, erscheinen ganz andere Werte als noch im vorherigen Beispiel: Sie können hier nach **Jahr**, **Monat**, **Quartal**, **Tag**, **Stunde**, **Minute** oder **Sekunde** gruppieren. Wir möchten alle Besteller eines Monats ermitteln, also wählen wir den Wert **Monat**. Danach blenden Sie durch Einstellen der Eigenschaft **Gruppenkopf** auf den Wert **Ja** den Gruppenkopf ein (siehe Bild 12). Für die Anzeige benötigen wir jedoch einen speziellen Ausdruck: Es soll ja nicht ein spezielles Datum angezeigt werden, sondern der Monat, der alle in diesem Monat fallenden

Bericht als PDF speichern

Berichte sind eine komfortable und einfache Art, Daten aus den Tabellen einer Access-Datenbank für die Ausgabe auf Papier vorzubereiten. In der heutigen Zeit tendieren wir allerdings dazu, nicht mehr sämtliche Kommunikation auszudrucken, sondern verschicken das eine oder andere Dokument gleich per E-Mail – schneller geht es kaum. Ein Bericht lässt sich allerdings nicht so einfach per E-Mail verschicken: vorher müssen Sie ihn noch in einem Format auf der Festplatte speichern, das möglichst von allen Empfängern gelesen werden kann.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1203_BerichtZuPDF.mdb**. Außerdem benötigen Sie die Dateien **dynapdf.dll** und **StrStorage.dll**.

Berichte speichern

Berichte sind wie Abfragen prinzipiell Anweisungen, wie Daten zu selektieren und auszugeben sind. Sie legen fest, welche Daten der Bericht anzeigen soll und bestimmen den Aufbau des Berichts und somit die Verteilung der Daten auf dem Bericht. Dies alles legen Sie im Entwurf von Berichten und Abfragen fest. Wenn Sie Berichte oder Abfragen öffnen, bilden diese den aktuellen Zustand der Daten entsprechend den im Berichts- oder Abfrageentwurf festgelegten Vorgaben ab.

Wenn Sie den Bericht schließen, einen der angezeigten Datensätze ändern oder löschen und den Bericht wieder öffnen, zeigt dieser die aktualisierte Fassung der Daten an. Damit gerät beispielsweise die Darstellung von Rechnungen per Bericht zu einem gefährlichen Spiel: Wenn Sie den Bericht einmal ausdrucken, sich dann bestimmte Werte wie biespielsweise die Rechnungsadresse ändern und die Rechnung erneut ausgedruckt werden soll, weil der Kunde seine Kopie verloren hat, erhält dieser nicht mehr die gleiche Rechnung wie zuvor.

Sie könnten dem zwar vorbeugen, indem Sie mit einem Rechnungsdatensatz sämtliche Kunden- und Rechnungsdaten speichern, aber es geht auch einfacher: Warum sichern Sie die Rechnung (neben der Kopie, die in Ihrem Rechnungsordner landet) nicht einfach als eigene Datei? Damit haben Sie nicht nur die Möglichkeit, jederzeit auf die Rechnung im Originalzustand zuzugreifen, sondern können diese auch bequem per E-Mail versenden. Wenn der Bericht einmal etwa als .pdf-Datei vorliegt, können Sie beispielsweise bei Rechnungen ein Feld in der entsprechenden Tabelle hinterlegen, um den Pfad und den Dateinamen zur Rechnungsdatei zu speichern und so später leicht von der Access-Anwendung aus auf die gespeicherte Rechnung zugreifen.

Versionsabhängige PDF-Erstellung

Je nach Access-Version gibt es unterschiedliche Ansätze, um aus einem Bericht eine PDF-Datei zu erstellen. Bis Access 2003 haben viele Access-Entwickler ein Tool von Stephen Lebans verwendet, das kostenlos ist und lediglich das Hinzufügen zweier DLLs im Anwendungsverzeichnis sowie einige Zeilen VBA-Code erfordert (www.lebans.com). Ab Access 2007 ist das Erstellen von PDFs mit Bordmitteln möglich (gegebenenfalls erst nach Installation von **http://www.microsoft.com/downloads/de-de/details.aspx?displaylang=de&FamilyID=f1fc413c-6d89-4f15-991b-63b07ba5f2e5**). In den folgenden Abschnitten lernen Sie beide Varianten kennen.

PDF-Export mit Access 2003 und älter

Unter Access 2003 und älter gibt es keine eingebaute Möglichkeit, einen Bericht in ein PDF-Dokument zu konvertieren. Glücklicherweise hat sich ein Access-Entwickler gefunden, der eine solche Funktion kostenlos zur Verfügung stellte: Stephen Lebans. Seine Funktion basiert auf dem Einsatz zweier DLL-Dateien, die sich im gleichen Verzeichnis wie die Datenbank (oder auch im Standard-Verzeichnis für Systemdateien) befinden sollten, und einem Standardmodul mit einigen wenigen Funktionen.

Die DLLs heißen **StrStorage.dll** und **dynapdf.dll** und kommen in verschiedenen Versionen vor. In unseren Tests zeigt sich, dass die aktuellste auf www.lebans.com verfügbare Version nicht mit Windows 7 zusammenarbeitet (Access stürzte komplett ab).

Mit einer älteren Version, die Sie auch im Download zu diesem Artikel finden, gelingt der Export nach PDF jedoch ohne Probleme. Das Modul **mdlReportToPDF** wurde von uns geringfügig modifiziert und unterscheide sich daher von der Version, die Sie in der Beispieldatenbank von Stephen Lebans finden.

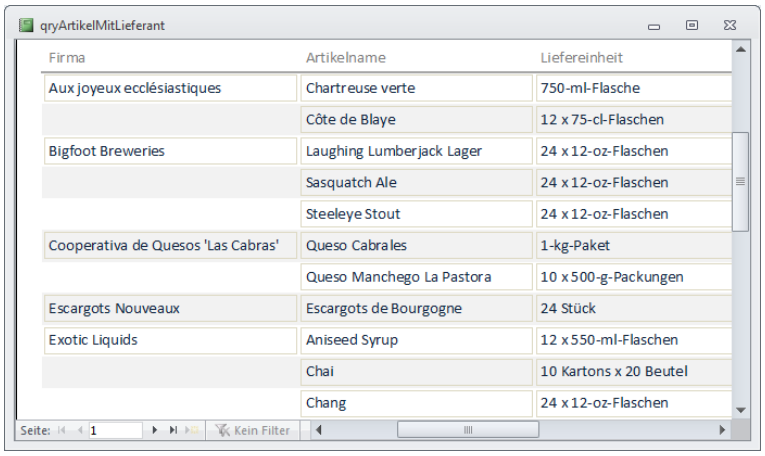


Bild 3: Artikelliste mit einfacher Anzeige des jeweiligen Lieferanten

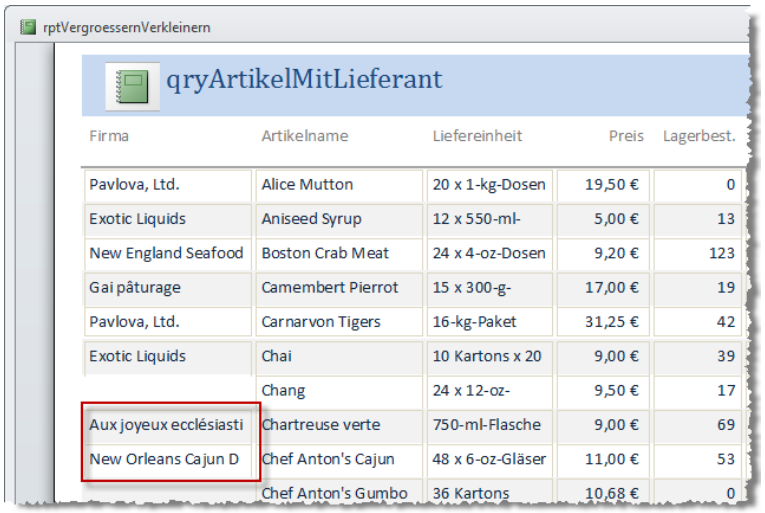


Bild 4: Zu lange Texte werden abgeschnitten.

Dies funktioniert nur wirklich gut, wenn die Datensätze wie in diesem Beispiel zunächst nach der Spalte sortiert sind, die ausblendet werden soll – anderenfalls werden Duplikate auch nur dort ausblendet, wo diese mehrfach hintereinander erscheinen.

Vergrößerbar

Wenn Sie einen Bericht wie den aus dem vorherigen Beispiel statt im Querformat im Hochformat anzeigen möchten, müssen die Spalten so schmal wie möglich angelegt werden, damit alle Felder erscheinen. Hier und da wird es dabei eng: Wie in Bild 4 zu sehen, werden zu lange Texte einfach abgeschnitten. Dies ist jedoch nur ein Problem, wenn die Texte aus einem einzigen Wort bestehen und nicht automatisch getrennt werden.

Anderenfalls können Sie Access dazu bringen, dass es einfach die Höhe der Zeilen ein wenig vergrößert. Dazu benötigen Sie zwei Einstellungen, die Sie allerdings in einem Schlag vornehmen: Stellen Sie einfach den Wert der Eigenschaft **Vergrößern** des betroffenen Steuerelements, hier also nacheinander für **Firma** und **Artikelname**, auf den Wert **Ja** ein (siehe Bild 5).

Dadurch sorgen Sie automatisch dafür, dass auch die Eigenschaft **Vergrößerbar** des Bereichtsbereiches, in dem sich das Steuerelement befindet (hier der Detailbereich), auf den Wert **Ja** eingestellt wird.

Das Ergebnis können Sie in Bild 6 betrachten: Die Werte, die nicht komplett in eine Zeile passen, werden einfach in der zweiten Zeile des vergrößerten Textfeldes fortgesetzt.

Verkleinerbar

Wenn Sie Adressblöcke in einem Bericht auf Basis mehrerer gebundener Felder erstellen, kann es vorkommen, dass aufgrund fehlender Feldwerte zu unschönen Leerräumen kommt – zum Beispiel wie in Bild 7, wo mal eine Straße, mal der Ansprechpartner fehlt. Dieses Verhalten können Sie mit der richtigen Einstellung der

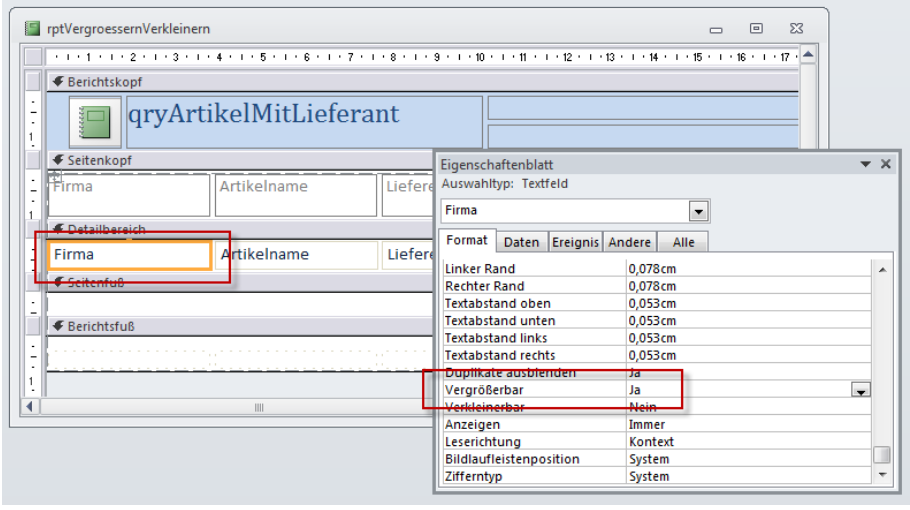


Bild 5: Zu lange Texte werden abgeschnitten.

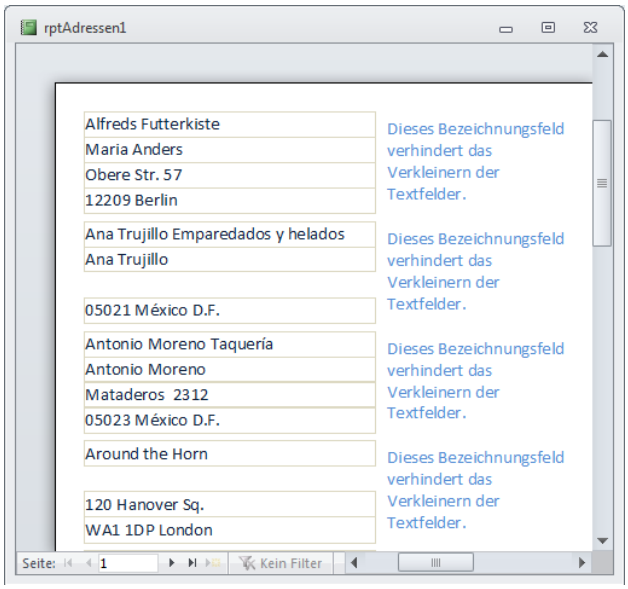


Bild 10: Liegen Steuerelemente auf gleicher Höhe wie zu verkleinernde Textfelder, werden die Textfelder nicht verkleinert.

Berichte zusammenfassen

Früher oder später möchten Sie vielleicht einmal mehrere Berichte zusammenfassen. Das kann beispielsweise interessant sein, wenn Sie zwei Berichte immer gemeinsam drucken.

Das ist kein Problem: Erstellen Sie einfach einen neuen Bericht und fügen Sie die beiden zusammenzuführenden Berichte in den Detailbereich des neuen Berichts ein (siehe Bild 11).

Nun stellt sich noch die Frage, ob die Berichte unmittelbar untereinander ausgegeben werden sollen oder ob der zweite Bericht auf einer neuen Seite beginnen soll. Im ersten Fall brauchen Sie nichts weiter zu erledigen, im zweiten Fall ist nur ein weiteres Steuerelement nötig.

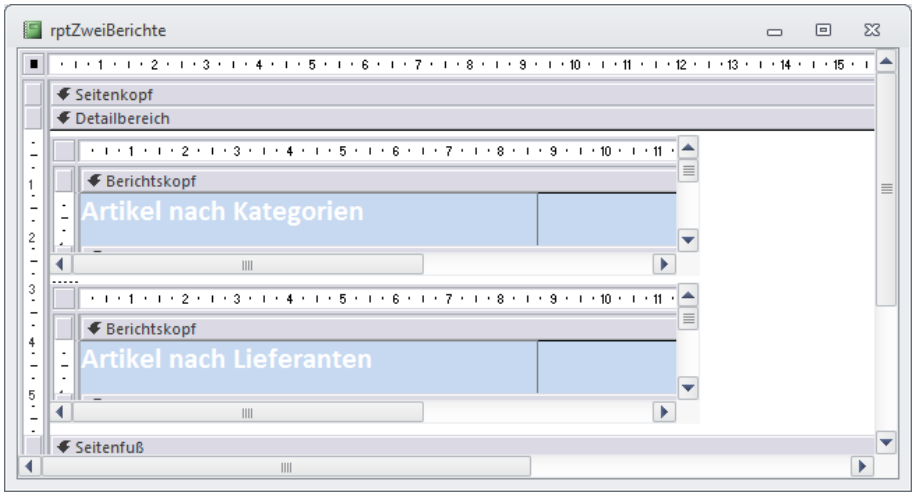


Bild 11: Zwei Berichte können Sie auf einmal ausgeben oder drucken, indem Sie beide Berichte in einen neuen Bericht einfügen.

Dabei handelt es sich um das Steuerelement Seitenumbruch. Platzieren Sie dieses einfach zwischen den beiden Berichten, damit der zweite Bericht auf einer neuen Seite beginnt. Damit haben Sie die einfachste Variante eines Unterberichts erstellt! Das Ergebnis erkennen Sie in Bild 12.

Beachten müssen Sie eigentlich nur, dass das Unterbericht-Steuerelement mindestens die Breite der Unterberichte selbst aufweist.

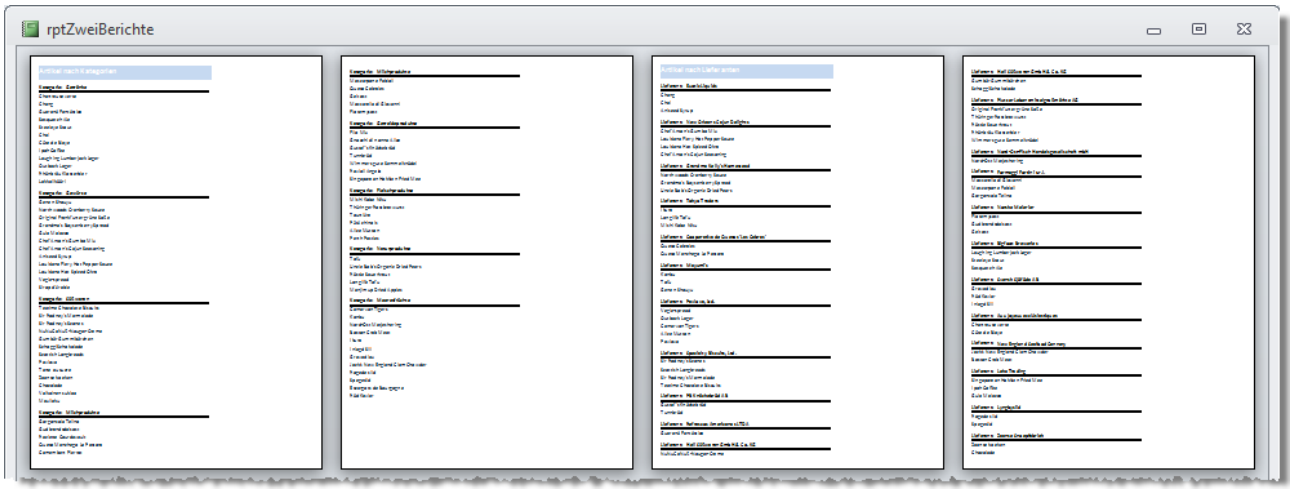


Bild 12: Zwei Berichte in einer Vorschau

Daten für die Ausgabe formatieren

Die in Ihrer Datenbank gespeicherten und die tatsächlich in Formularen und Berichten angezeigten Daten unterscheiden sich gelegentlich bezüglich ihres Formats. Dies tritt vor allem bei Datumsangaben in Erscheinung, aber auch bei den übrigen Zahlendatentypen. Dieser Artikel zeigt, wie Sie in Formularen und Berichten für das richtige Erscheinungsbild sorgen und stellt die dafür nötigen Funktionen vor.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1204_Format.mdb**.

Schein und Sein

Datumswerte sind ein anschauliches Beispiel für den Unterschied zwischen den angezeigten und den tatsächlich gespeicherten Daten. Wenn ein Formularfeld Ihnen den Ausdruck **1.1.2012** präsentiert, wird dieser intern als Zahlwert gespeichert – in diesem Fall **40.909**. Dies ist die Anzahl der Tage, die seit dem **30.12.1899** verstrichen sind (dieses Datum entspricht dem Zahlenwert **0**). Zeitangaben wie **12:00:00** hingegen werden als Bruchteile eines Tages gespeichert. Ein Tag entspricht dabei dem Wert **1**, also entspricht **12:00:00** dem Wert **0,5**.

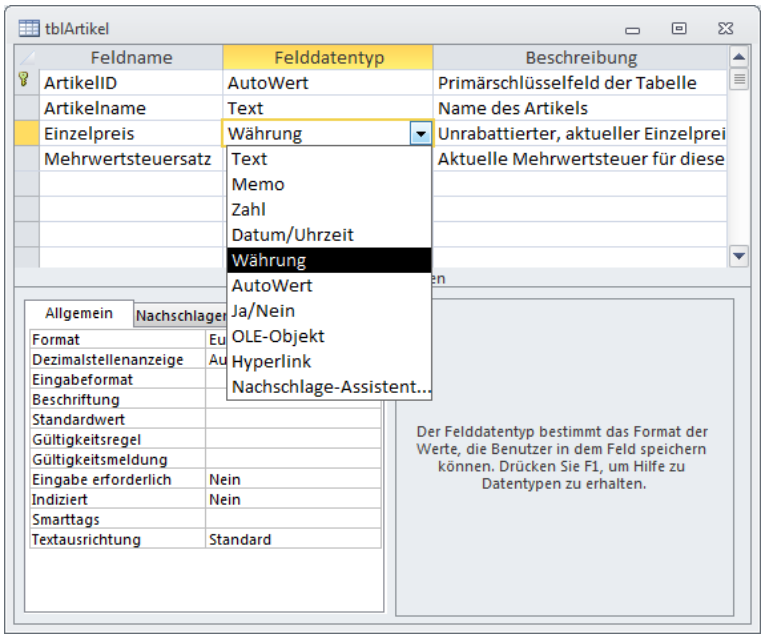


Bild 1: Die Auswahl des Felddatentyps stellt Feldeigenschaften wie Datentyp, Feldgröße und Format ein.

Genau wie Teile des Datums durch Worte ersetzt werden können (Wochentage, Monatsnamen) oder die Uhrzeit durch das Literal **Uhr** ergänzt wird, versieht Access Währungsangaben mit entsprechenden Einheiten wie **EUR/€** oder **$**.

Für die Ausgabe in Formularen oder Berichten legen Sie selbst das Erscheinungsbild mit entsprechenden Eigenschaften oder Funktionen genau fest. Manche Felddatentypen implizieren solche Angaben. So führt zum Beispiel die Festlegung des Felddatentyps **Datum/Uhrzeit** automatisch dazu, dass die gespeicherten **Long**/**Double**-Werte in einem gut lesbaren Format ausgegeben werden, und der Felddatentyp **Währung** legt nicht nur fest, dass der Datentyp **Decimal** verwendet wird, sondern auch, wieviele Nachkommastellen angezeigt werden und dass ein Währungssymbol angehängt werden soll.

An dieser Stelle wird auch der Unterschied zwischen Felddatentyp und Datentyp augenscheinlich: Während der Datentyp lediglich den Wertebereich für das Feld festlegt, stellt der Felddatentyp gleich noch weitere Eigenschaften ein. Dazu gehört beispielsweise das Format, in dem die Daten standardmäßig angezeigt werden oder die Anzahl der anzuzeigenden Nachkommastellen (siehe Bild 1).

Unabhängig davon, welche Einstellung der Felddatentyp liefert, können Sie für die Darstellung in Formularen und Berichten beliebige eigene Formatierungen vornehmen. Dafür gibt es einige Spezialfunktionen etwa zum Aufbereiten von Währungs- oder Datumsangaben, aber auch einen Alleskönner: Die Funktion **Format** biegt Ihre Daten sowohl in der Benutzeroberfläche als auch unter VBA zurecht.

Die Format-Funktion

Diese Funktion erwartet vier Parameter, von denen nur der erste ein Pflichtparameter ist:

```
Format(<Expression>, [<Format>], <FirstDayOfWeek>,
<FirstWeekOfYear>)
```

Der Parameter **Expression** enthält den zu formatierenden Wert. Die übrigen drei Parameter sehen so aus:

- **Format** (Datentyp **String**): Erwartet einen Ausdruck mit dem Zielformat.
- **FirstDayOfWeek**: Erwartet eine der Konstanten der Enumeration **vbDayOfWeek**.
- **FirstWeekOfYear**: Erwartet eine Konstante der Enumeration **vbFirstDayOfWeek**.

Wenn die **Format**-Funktion nun den mit dem Parameter **Ausdruck** angegebenen Wert entsprechend den mit dem Parameter **Format** angegebenen Format ausgibt, warum ist der Parameter **Format** dann ein optionaler Parameter? Die Lösung ist einfach: Die Funktion formatiert den übergebenen Wert auch dann, wenn der Parameter **Format** keinen Wert enthält. So liefert **Format(1.1)** beispielsweise den Wert **1,1** – das Dezimalzeichen wird also in das in den Systemeinstellungen festgelegte Zeichen umgewandelt. Der Parameter **Format** ist der wichtigste Parameter dieser Funktion, daher werden wir diesen in den folgenden Abschnitten genau erläutern. Die beiden übrigen Parameter sind nur für die Formatierung von Datumsangaben wichtig – mehr.dazu in **Die Format-Funktion**.

Einsatzzwecke

Nehmen wir an, Sie möchten ein Datum mit ausgeschriebenem Monat ausgeben. Dann würden Sie die Format-Funktion wie folgt definieren (dies nur als Vorgriff und als Beispiel für das Aufzeigen der Einsatzzwecke):

```
Format("1.1.2012", "d. mmmm yyyy")
```

Das Ergebnis sieht so aus:

```
1. Januar 2012
```

Wo können Sie diese Funktion nun überall einsetzen? Am einfachsten gelingt dies im Direktfenster des VBA-Editors (zu öffnen mit der Tastenkombination **Strg + G**). Dort geben Sie die Funktion einfach mir führendem **Debug.Print** ein und erhalten direkt das Ergebnis. Dieser Ort eignet sich hervorragend, um die Funktionsweise der **Format**-Funktion zu testen (siehe Bild 2).

In der Praxis können Sie bereits im Tabellenentwurf ein individuelles Format festlegen. Dazu benötigen Sie noch nicht einmal die **Format**-Funktion. Access stellt nämlich für solche Felddatentypen, bei denen

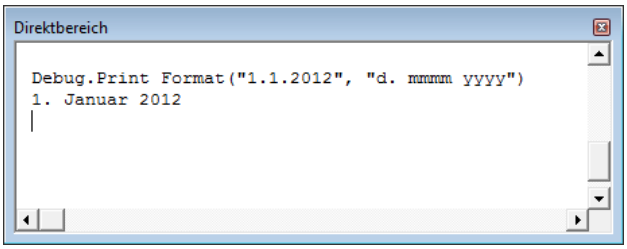


Bild 2: Testen der Format-Funktion im Direktfenster

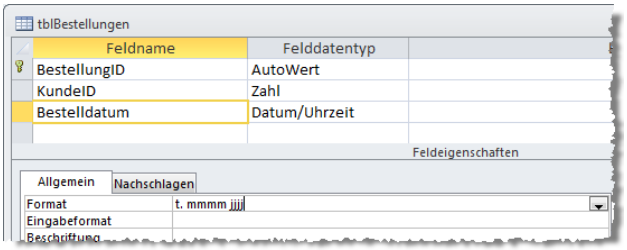


Bild 3: Die **Format**-Eigenschaft im Tabellenentwurf ...

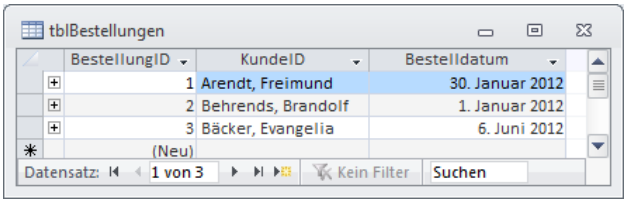


Bild 4: ... und in der Datenblattansicht.

es sinnvoll ist, bereits eine Eigenschaft namens **Format** zur Verfügung. Dort müssen Sie nur die Formatierungsanweisung eintragen, denn der zu formatierende Wert wird ja durch das entsprechende Feld der Tabelle bereitgestellt. Schauen wir uns das Feld **Bestelldatum** der Tabelle **tblBestellungen** an. Dort finden Sie die Eigenschaft **Format** im Bereich **Allgemein** vor (siehe Bild 3). Wenn Sie die unter VBA verwendete Variante dort eintragen, also **d. mmmm yyyy**, wird der Ausdruck in der deutschsprachigen Access-Version gleich in diesen Ausdruck umgewandelt:

```
\d. mmmm" yyyy"
```

Der Grund ist, dass ein deutschsprachiges Access in allen Ausdrücken für Eigenschaften von Tabellen, Abfragen, Formularen, Berichten und Makros die deutsche Variante für Funktionen, Schlüsselwörter und Parameter erwartet. Es versucht hier, die deutsche Version zu ermitteln und scheitert kläglich. Diese muss nämlich so lauten:

```
t. mmmm jjjj
```

Damit erhalten Sie dann auch die Datumsausgabe wie in Bild 4. Auf die gleiche Weise setzen Sie die Format-Werte auch in den **Format**-Eigenschaften von Feldern

Die Format-Funktion

Access bietet verschiedene Stellen und Möglichkeiten, Daten für die Ausgabe in die gewünschte Form zu bringen. Eine wichtige Variante ist die Format-Funktion: Sie können diese überall einsetzen, wo Daten aus Tabellen angezeigt oder per VBA-Code zugewiesen werden. Dieser Artikel stellt die Format-Funktion im Detail vor.

Parameter der Format-Funktion

Die Format-Funktion verwendet vier Parameter. Der erste enthält den zu formatierenden Ausdruck, wobei dieser als Zahl oder in Anführungszeichen als Zeichenkette angegeben werden kann, aber auch als Variable oder VBA-Funktion beziehungsweise als Referenz auf ein Steuerelement oder ein Feld einer Tabelle oder Abfrage.

Es funktionieren also beispielsweise folgende Varianten:

```
Debug.Print Format(1234567890, ...)

Debug.Print Format("1234567890", ...)

Dim strAusdruck As String
strAusdruck = "1234567890"
Debug.Print Format(strAusdruck, ...)

Debug.Print Format(Me!txtAusdruck, ...)

Debug.Print Format(rst!Ausdruck)

Debug.Print Format(DLookup("Ausdruck", 7
                           "tblAusdruecke"), ...)
```

Der zweite Parameter erwartet eine Anweisung, wie der Ausdruck formatiert werden soll.

Dabei kann es sich um ein benanntes Format handeln, wie es auch etwa für die **Format**-Eigenschaft von Tabellen- und Abfragefeldern oder Formular- oder Berichtssteuerelementen ausgewählt werden kann (mehr dazu im Artikel **Daten für die Ausgabe formatieren**).

Der vorliegende Artikel konzentriert sich jedoch auf die Verwendung benutzerdefinierter Formatierungen, bei denen Sie den Ausdruck mithilfe geeigneter Platzhalter und Literale formatieren.

Der dritte und vierte Parameter wird nur berücksichtigt, wenn ein Datumsausdruck formatiert werden soll – dazu später mehr.

Benutzerdefinierte Zahlenformate

Ein häufiger Anwendungsfall für die **Format**-Funktion ist die Formatierung von Zahlen. Das Zahlenformat wird dabei aus verschiedenen Zeichen zusammengesetzt, die entweder als Platzhalter oder als Literal verwendet werden. Dabei handelt es sich um die folgenden Zeichen:

- **0**: Das Zeichen **0** ist der Platzhalter für eine Ziffer oder **0**. Wenn die entsprechende Stelle keine Ziffer enthält, wird statt der Ziffer das Zeichen **0** angezeigt. Damit lassen sich beispielsweise führende **0**-Zeichen anzeigen:

  ```
  Debug.Print Format(1234, "000000")
  01234
  ```

  Das funktioniert auch für Nachkommastellen – allerdings werden die **0**-Zeichen hinten angehängt:

  ```
  Debug.Print Format(123.456, "0000.0000")
  0123,4560
  ```

  Im Nachkomma-Bereich wird das Zeichen 0 vorrangig zur Begrenzung der angezeigten Stellen eingesetzt. Sind mehr Stellen vorhanden als 0-Zeichen im Zahlenformat, werden diese abgeschnitten und auf die letzte angezeigte Stelle gerundet:

  ```
  Debug.Print Format(0.55, "0.0")
  0,6
  ```

- Raute (**#**): Das Raute-Zeichen verhält sich grundsätzlich wie das **0**-Zeichen. Allerdings wird, wenn an der Stelle des Raute-Zeichens keine Zahl vorliegt, nichts angezeigt:

  ```
  Debug.Print Format(1.23, "###.###")
  1,23
  ```

  Die Ziffer **0** wird als führendes oder folgendes Zeichen abgeschnitten. Extrembeispiel (das Ergebnis ist ein Komma, kein Fliegendreck):

```
Debug.Print Format(0.000, "###.###")
.
```

Wenn der Ausdruck zumindest eine führende und eine folgende **0** enthalten soll, fügen Sie statt des Raute-Zeichens das **0**-Zeichen ein.

Mit dem Raute-Zeichen können Sie auch Nachkommastellen begrenzen und runden:

```
Debug.Print Format(1.55, "#.#")
1,6
```

- Punkt (**.**): Der Punkt ist der Platzhalter für ein Dezimalzeichen. Warum benötigt man dafür einen Platzhalter? Nun: Je nach dem in der Systemsteuerung festgelegten Gebietsschema verwendet Access unterschiedliche Trennzeichen wie das Komma oder den Punkt. Beispiele haben Sie bereits oben gesehen.

- Prozentzeichen (**%**): Das Prozentzeichen hat mehrere Funktionen. Zunächst wird es dem Ausdruck hinzugefügt. Außerdem sorgt es dafür, dass der angegebene Zahlenwert mit **100** multipliziert wird:

```
Debug.Print Format(0.5, "0.0 %")
50,0 %
```

- Komma (**,**): Das Komma wird als Tausendertrennzeichen interpretiert:

```
Debug.Print Format(1234, "#,##0")
1.234
```

Damit das Trennzeichen erscheint, muss die Zahl im vorherigen Beispiel mindestens vier Stellen haben.

Sonst entfällt das Tausendertrennzeichen einfach:

```
Debug.Print Format(123, "#,###")
123E
```

Ein Tausendertrennzeichen direkt links vom Dezimaltrennzeichen oder ganz rechts, wenn kein Dezimaltrennzeichen vorliegt, teilt die angegebene Zahl durch **1.000** und rundet entsprechend der angegebenen Nachkommastellen:

```
Debug.Print Format(1000, "0,")
1
```

Zwei Tausendertrennzeichen teilen entsprechend durch **1.000.000** – hier mit Nachkommastellen und Rundung auf die drittletzte Stelle:

```
Debug.Print Format(1234567, "0,,.###")
1,235
```

- Wissenschaftliches Format (**E+**, **E-**, **e+**, **e-**): Hiermit können Sie Zahlen im wissenschaftlichen Format anzeigen. Beispiele:

```
Debug.Print Format(123456, "#0.0000E+0")
12,3456E+4

Debug.Print Format(0.00123456, "#0.0000E+0")
12,3456E-4
```

Zahlen mit Einheit anzeigen

Wenn Sie Zahlen mit Einheit anzeigen möchten, also beispielsweise **0°C** oder **27,50 €**, fügen Sie die entsprechenden Literale in der Regel einfach zum Zahlenformat hinzu:

```
Debug.Print Format(0, "0°C")
0°C
```

Vom Wert abhängige Formatierung

Bei der Formatierung von Zahlenwerten können Sie für positive und negative Zahlen sowie für 0 und NULL verschiedene Formatierungen festlegen.

Dazu legen Sie die Formate in dieser Reihenfolge durch Semikola statt eines einzelnen Formatausdrucks fest.

Die folgende Funktion gibt beispielsweise für **1** den Ausdruck **Positiv: 1**, für **-1** den Ausdruck **Negativ: -1** und für **0** den Ausdruck **Neutral: 0** aus:

```
Public Function Multiformat(Optional lngZahl As Long)
    Debug.Print Format(lngZahl, _
    """Positiv: ""0;""Negativ: ""0;""Neutral: ""0")
End Function
```

Datum formatieren

Für Datumsangaben gibt es eine Reihe spezieller Platzhalter, mit denen Sie das Datum beliebig formatieren können. Speziell bei den Platzhaltern für Tage, Monate, Jahre, Stunden, Minuten und Sekunden müssen Sie darauf achten, unter VBA die englische

Farbige und vom Wert abhängige Formatierung

Datenblätter kommen unter Access recht farblos daher. Sie können zwar global für alle Spalten und Zeilen eine andere Schrift- oder Hintergrundfarbe einstellen, aber nicht für eine einzelne Spalte. Genau dies ermöglicht jedoch die Format-Eigenschaft – und nicht nur das: Sie können damit sogar positive und negative Zahlenwerte gesondert hervorheben oder auch Nullwerte markieren. Und damit lassen sich richtig praktische Dinge basteln, wie etwa die Anzeige eines Platzhalters wie <Bitte Vorname angeben> für Pflichtfelder eines Formulars.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1204_FarbigeFormatierung.mdb**.

Farbige Spalten

Sollten Sie einmal den Wunsch verspüren, ein Datenblatt ähnlich farbig wie in Bild 1 zu gestalten, brauchen Sie gar nicht viel zu erledigen. Sie müssen nur die **Format**-Eigenschaft des jeweiligen Feldes im Tabellenentwurf entsprechend zu ändern (siehe Bild 2).

Bild 1: Datenblatt mit verschiedenen Farben

Für die blauen Artikelnamen reicht beispielsweise der folgende Ausdruck:

```
@[Blau]
```

Die grünen Artikelpreise erhalten Sie mit diesem Ausdruck:

```
0,00[Grün] €
```

Der rote Mehrwertsteuersatz schließlich wird so formatiert:

```
0[Rot] %
```

Sie erkennen an diesen Beispielen bereits, dass Sie nur eine in eckigen Klammern eingefasste Bezeichnung der gewünschten Farbe eingeben müssen. Dabei gibt es allerdings nur wenig Auswahl, nämlich **Schwarz**, **Rot**, **Grün**, **Gelb**, **Blau**, **Magenta**, **Zyan** und **Weiß**.

Ein wichtiger Faktor ist, dass Sie unbedingt die entsprechenden Platzhalter für die tatsächlichen Werte eintragen. Für ein Tabellenfeld mit dem Datentyp **Text** reicht dabei die Angabe des **@**-Zeichens aus. Zahlenformate hingegen müssen explizit angegeben werden, also zum Beispiel mit **0,00**. Weitere Informationen zu den möglichen Platzhaltern finden Sie im Artikel **Die Format-Funktion**.

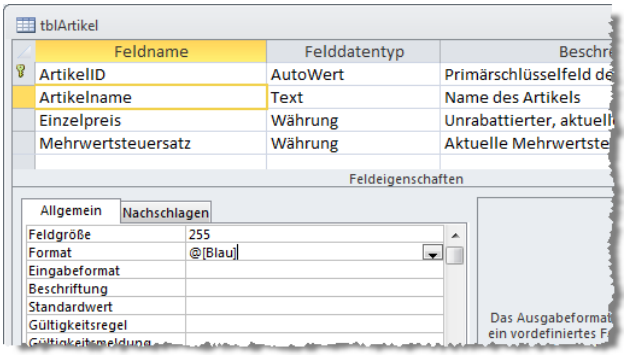


Bild 2: Format für eine farbige Spalte

Es gibt noch eine weitere Besonderheit: Die Farbzuweisung funktioniert nicht mit benannten Formaten zusammen. Wenn Sie also etwa **Datum, lang** verwenden, um das Datum mit Tag und ausgeschriebenem Monat auszugeben, wird die Angabe einer Farbe ignoriert. In diesem Fall müssen Sie den Ausdruck selbst zusammenstellen, also beispielsweise so:

```
tttt", "t. mmmm jjjj[Magenta]
```

Übrigens ist es egal, an welcher Stelle Sie die Farbe eintragen – Sie können diese sogar mitten im Ausdruck platzieren:

```
tttt", "t. mm[Magenta]mm jjjj
```

Die Farbangabe können Sie nur in der **Format**-Eigenschaft von Feldern in Tabellen und Abfragen oder

Listenfeldauswahl speichern und wiederherstellen

Wie Sie die durch einen Benutzer ausgewählten Einträgen im Listenfeld auslesen, haben Sie bereits im Artikel "Mehrfachauswahl in Listenfeldern auslesen" erfahren. In diesem Folgeartikel geht es darum, eine solche Auswahl auf Basis der in einer Tabelle gespeicherten Daten wieder herzustellen. Dazu müssen wir diese Auswahl jedoch erst einmal speichern – und später kümmern wir uns um die Reproduktion

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1204_Mehrfachauswahl.mdb**.

Listenfeldauswahl ganz einfach

Normalerweise können Sie Listenfelder, genau wie Kombinationsfelder, mit einer Datensatzherkunft versehen (also etwa mit einer Tabelle oder Abfrage, welche die Daten liefert) und das Steuerelement über die Eigenschaft Steuerelementinhalt an ein Feld einer weiteren Tabelle binden.

Ein Beispiel ist die Auswahl einer Anrede für einen Kunden. Normalerweise verwenden Sie hier ein Kombinationsfeld, aber Sie können dies – ausreichend Platz vorausgesetzt – auch mit einem Listenfeld erledigen. Der Vorteil ist, dass man alle zur Auswahl stehenden Einträge direkt sehen kann – zumindest, wenn es nicht mehr Einträge gibt, als das Listenfeld im sichtbaren Bereich anzeigen kann (siehe Bild 1). Das Listenfeld ist mit der Tabelle **tblAnreden** als Datensatzherkunft ausgestattet und es ist an das Feld **AnredeID** der im Formular angezeigten Tabelle **tblKunden** gebunden. Damit das Listenfeld nur die Anrede, nicht aber die Werte des Feldes **AnredeID** anzeigt, stellen Sie die Eigenschaften **Spaltenanzahl** und **Spaltenbreite** auf die Werte **2** und **0cm** ein. So wird das erste Feld mit der **AnredeID** ausgeblendet. Dieses muss aber dennoch als gebundene Spalte vorliegen: Access sucht nämlich in der Tabelle **tblAnreden** nach dem Datensatz, dessen als gebundene Spalte angegebenes Feld den gleichen Wert enthält wie das in der Eigenschaft **Steuerelementinhalt** angegebene Feld, also das Feld **AnredeID** der Tabelle **tblKunden**.

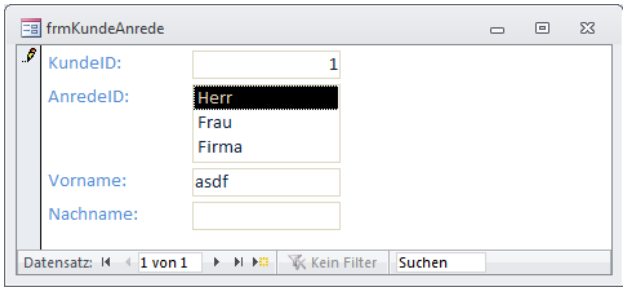


Bild 1: Lookup-Daten per Listenfeld auswählen

Damit haben Sie gleich zwei Fliegen mit einer Klappe geschlagen: Erstens wird die Auswahl so automatisch gespeichert und zweitens wird sie beim Anzeigen eines Datensatzes auch gleich wieder hergestellt.

Mehrfachauswahl speichern

Schauen wir uns nun an, wie sich dies für die Mehrfachauswahl realisieren lässt. Wenn Sie dies mit der Mehrfachauswahl durchführen, ist der Aufwand schon erheblich höher.

Als Beispiel verwenden wir das Zuweisen von Publikationen zu Kunden, als eine Art Verteilerliste. Dazu benötigen wir drei Tabellen: erstens eine Tabelle **tblKunden** zum Speichern der Kundendaten, zweitens die Tabelle **tblPublikationen** und noch eine dritte Tabelle, mit der wir die Publikationen den Kunden zuweisen. Damit realisieren wir eine m:n-Beziehung zwischen den Tabellen **tblKunden** und **tblPublikationen**. Dies bedeutet, dass wir über die dritte Tabelle, die wir schlicht **tblVerteiler** nennen, jede Publikation jedem Kunden zuordnen können. Das Datenmodell sieht wie in Bild 2 aus. Dabei enthält die Tabelle **tblVerteiler** zwei Fremdschlüsselfelder, mit denen Sie je Datensatz einen Eintrag der Tabelle **tblKunden** und einen Eintrag

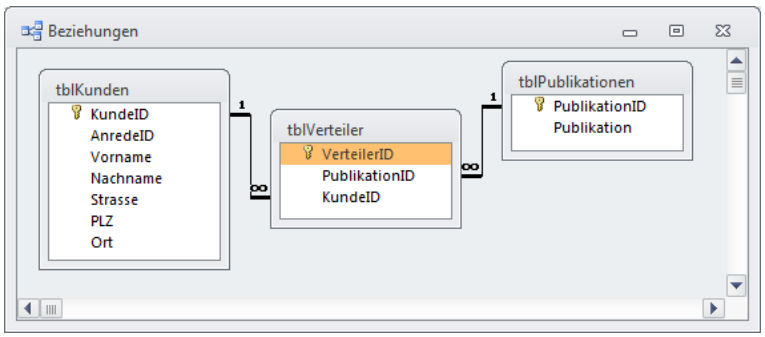


Bild 2: Datenmodell für den Publikationsverteiler

der Tabelle **tblPublikationen** auswählen können. Damit kein Kunde eine Publikation doppelt erhält, haben wir außerdem einen eindeutigen Index für die beiden Felder **PublikationID** und **KundeID** festgelegt (siehe Bild 3).

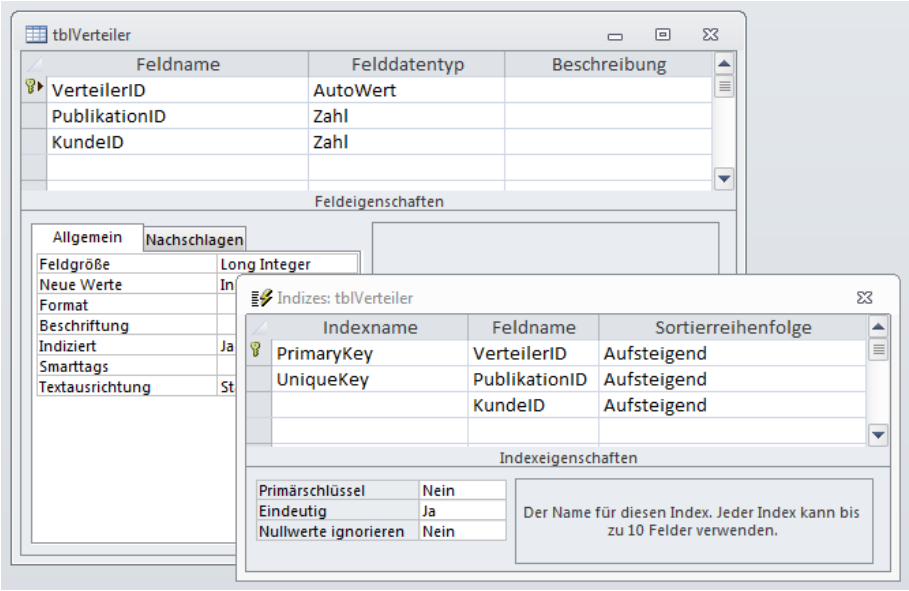


Bild 3: Eindeutiger Index für die Kombination aus Kunde und Publikation

Formular zur Auswahl von Publikationen

Mit diesen Tabellen ausgestattet können wir uns an den Entwurf des Formulars zur Auswahl der Publikationen je Benutzer begeben. Das Formular soll **frmVerteiler** heißen und verwendet die Tabelle **tblKunden** als Datenherkunft. Ziehen Sie die notwendigsten Felder der Tabelle **tblKunden** in den Detailbereich und fügen Sie dem Formular ein Listenfeld namens **lstPublikationen** hinzu. Das sind vorerst alle notwendigen Steuerelemente, das Formular sollte nun so aussehen wie in Bild 4.

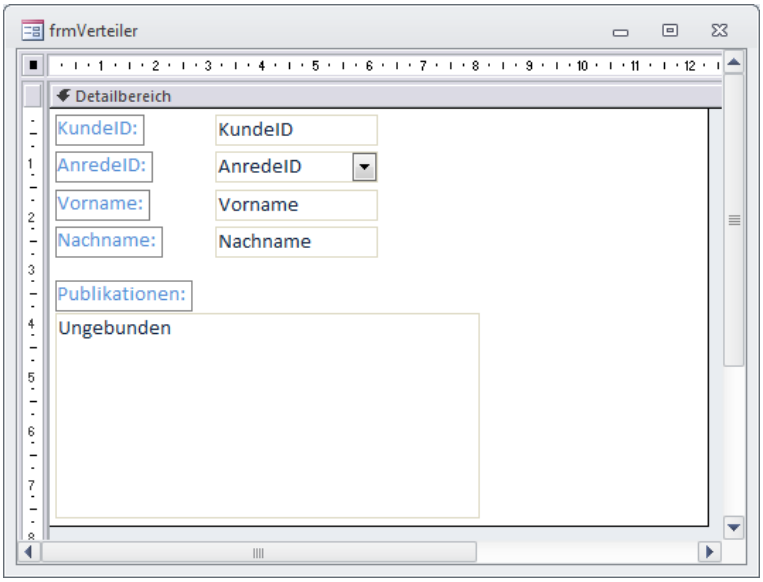


Bild 4: Entwurf des Formulars zum Festlegen der Verteilerlisten

Nun kümmern wir uns um die im Listenfeld angezeigten Daten. Eigentlich soll das Listenfeld nur die Liste aller Publikationen anzeigen. Aber wir benötigen doch noch Informationen darüber, ob eine Publikation über die Tabelle **tblVerteiler** einem Kunden zugewiesen wurde? Das stimmt prinzipiell. Allerdings benötigen wir diese Information nur zum Markieren der zugeteilten Publikationen. Es werden jedoch immer alle Publikationen angezeigt – wir brauchen das Listenfeld **lstPublikationen** also einfach nur mit der Tabelle **tblPublikationen** als Datensatzherkunft zu versehen. Markieren Sie also das Listenfeld im Formularentwurf und stellen Sie die Eigenschaft **Datensatzherkunft** entsprechend ein. Außerdem soll nur die Publikation, nicht aber die **PublikationID** erscheinen. Dies erreichen wir durch Setzen der Eigenschaften **Spaltenanzahl** und **Spaltenbreiten** auf die Werte **2** und **0cm**. Stellen Sie außerdem die Eigenschaft Mehrfachauswahl auf **Einfach** ein (oder **Erweitert** – je nach Anforderung). Das Formular bietet bereits jetzt die Möglichkeit, eine oder mehrere Publikationen je Kunde auszuwählen – allerdings

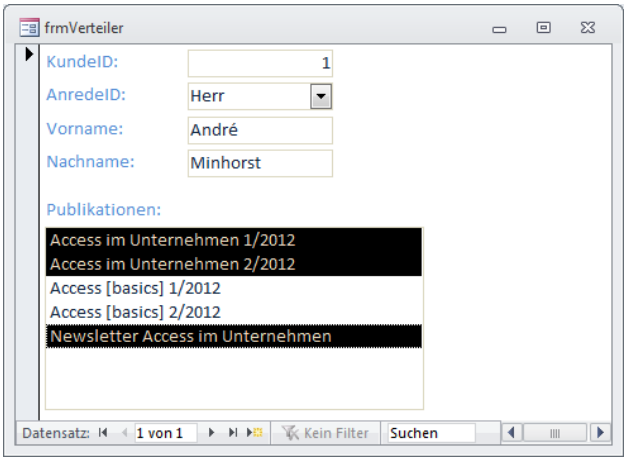


Bild 5: Das Formular zum Festlegen von Verteilerlisten in der Formularansicht

Ereignisse in Formularen, Teil 1: Reihenfolge

Wenn Sie Formulare automatisieren möchten, um beispielsweise beim Öffnen, beim Schließen, beim Wechseln oder Speichern eines Datensatzes selbst programmierte Aktionen durchzuführen, sollten Sie die wichtigsten Ereignisse von Formularen und deren Abfolge kennen. Dieser Artikel zeigt, wie Sie die Reihenfolge der Ereignisse beim Öffnen und Schließen von einfachen Formularen, Formularen mit Unterformularen und gebundenen Formularen ermitteln.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1204_EreignisseInFormularen.mdb**.

Ereignisse anlegen

In einigen Artikeln haben Sie bereits erfahren, wie Sie schnell ein Ereignis für ein Formular oder ein Steuerelement anlegen. Dabei öffnen Sie das Formular in der Entwurfsansicht, zeigen die Seite Ereignisse des Eigenschaftsfensters an, wählen für die gewünschte Ereigniseigenschaft den Wert **[Ereignisprozedur]** aus und klicken dann auf die Schaltfläche mit den drei Punkten (siehe Bild 1). Access erstellt den Rahmen der Ereignisprozedur dann automatisch und zeigt diesen im VBA-Fenster an.

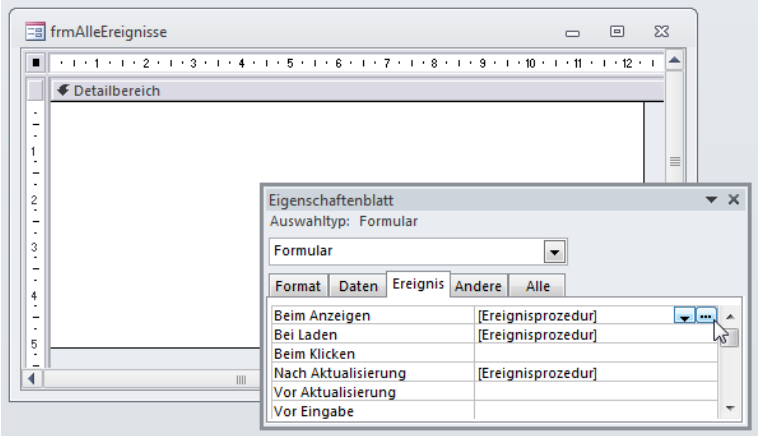


Bild 1: Anlegen von Ereignissen von der Benutzeroberfläche aus

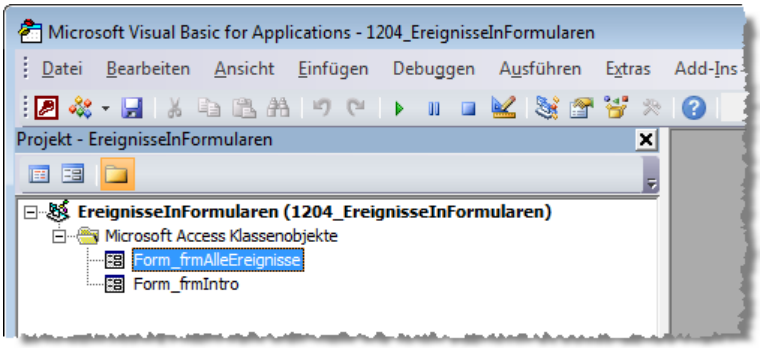


Bild 2: Das Klassenmodul im Projekt-Explorer

Es gibt jedoch noch eine einfachere Methode. Dazu müssen Sie zumindest das Klassenmodul für das entsprechende Formular angelegt haben, was Sie entweder durch Erstellen einer Ereignisprozedur auf klassischem Wege durchführen – oder Sie stellen einfach die Eigenschaft **Enthält Modul** im Eigenschaftsfenster auf **Ja** ein. Access legt dann ein leeres Klassenmodul für das Formular an.

Wenn Sie nun beispielsweise mit der Tastenkombination **Alt + F11** zum VBA-Editor wechseln und den Projekt-Explorer mit **Strg + R** aktivieren, finden Sie dort einen entsprechenden Eintrag vor (siehe Bild 2). Das Klassenmodul öffnen Sie mit einem Doppelklick auf den Eintrag im Projekt-Explorer.

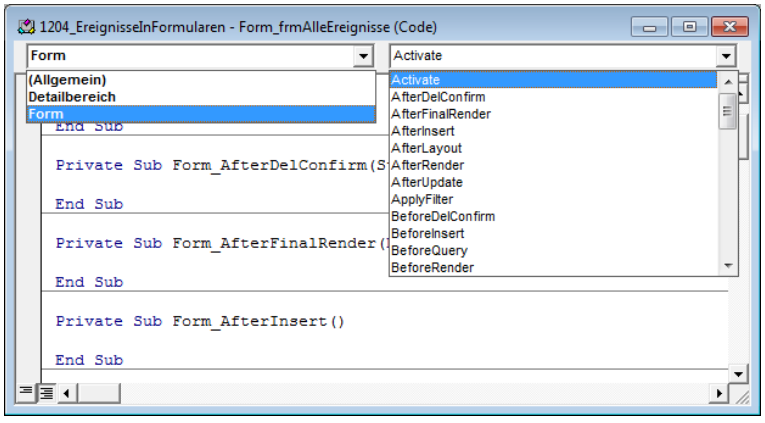


Bild 3: Anlegen von Ereignisprozeduren per VBA-Editor

Wenn Sie nun im Klassenmodul eine Ereignisprozedur für ein Ereignis anlegen möchten, das beispielsweise beim Öffnen des Formulars ausgelöst wird, wählen Sie zunächst den Eintrag **Form** aus dem linken Kombinationsfeldes des Codefensters aus und dann den Namen des gewünschten Ereignisses aus dem rechten Kombinationsfeld (siehe Bild 3). Dies legt nicht nur die Kopf- und die Fußzeile der

Prozedur an, sondern fügt auch den Wert **[Ereignisprozedur]** in die entsprechende Eigenschaft des Formulars ein. Auf diese Weise können Sie schnell mehrere Ereignisprozedur erstellen. Im Programmiereralltag kommt dies wahrscheinlich nicht allzu oft vor. Wenn Sie aber wie in diesem Artikel den Umgang mit Ereignisprozeduren kennenlernen wollen, kann es helfen, schnell mal ein paar Prozeduren anzulegen.

Übrigens funktioniert diese Vorgehensweise zum Anlegen von Prozeduren nicht immer: Wenn Sie etwa ein Formular als modalen Dialog öffnen (also mit **DoCmd.OpenForm "Formularname", WindowMode:=acDialog**), können Sie wegen des Öffnungsmodus zunächst gar keine Änderungen am Code vornehmen. Klicken Sie dann auf die **Zurücksetzen**-Schaltfläche (die mit dem Stop-Symbol), können Sie den Code bearbeiten. Das Hinzufügen einer Ereignisprozedur über die beiden Kombinationsfeld oben im Codefenster legt dann allerdings nicht automatisch den Wert **[Ereignisprozedur]** für die entsprechende Ereigniseigenschaft an.

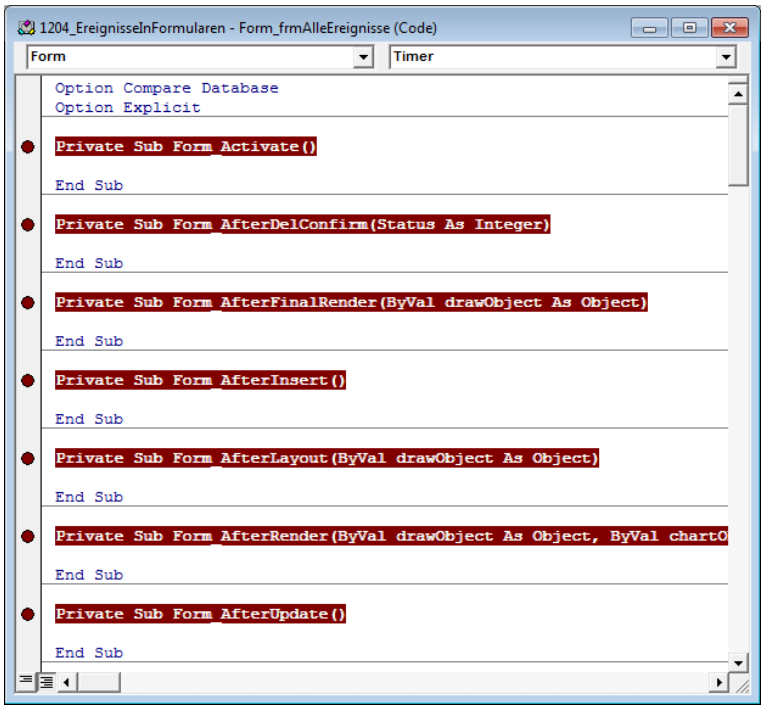


Bild 4: Haltepunkte zu Ereignisprozeduren hinzufügen

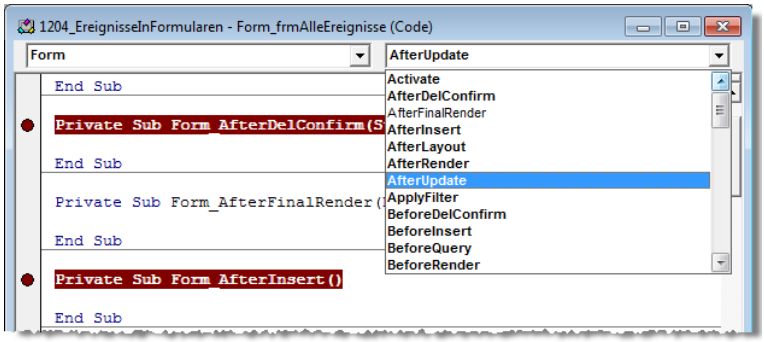


Bild 5: Implementierte Ereignisse werden in der Liste gekennzeichnet.

Ereignisabfolge

Bevor wir uns die einzelnen Ereignisse ansehen, lernen Sie eine wichtige Technik kennen, um die Abfolge von Ereignissen zu ermitteln und um zu prüfen, ob ein Ereignis überhaupt in einem bestimmten Zusammenhang ausgelöst wird. Dazu könnten Sie zunächst einige Ereignisprozeduren zum Formular hinzufügen, die den Eindruck erwecken, dass sie beim schlichten Öffnen und Schließen des Formulars ausgelöst werden.

Aber wissen Sie was: Wir gehen auf Nummer sicher und legen gleich Ereignisprozeduren für alle Ereignisse an, um genau zu ermitteln, wann welche Ereignisprozedur ausgelöst wird. Öffnen Sie also das Klassenmodul des betroffenen Formulars und wählen Sie nacheinander alle Einträge des rechten Kombinationsfeldes im Codefenster aus – solange, bis Sie meinen, Sie hätten für jeden Eintrag eine Ereignisprozedur zum Modul hinzugefügt. Um zu prüfen, ob Sie alle erwischt haben, gibt es eine einfachere Methode als den manuellen Abgleich: Setzen Sie einfach für jeden Prozedurkopf einen Haltepunkt. Dies erreichen Sie zum Beispiel auf die folgenden beiden Arten:

- Klicken Sie auf Höhe des Prozedurkopfes auf den grauen Bereich am linken Rand des Codefensters.
- Oder setzen Sie einfach die Einfügemarke auf den Prozedurkopf und betätigen Sie die Schaltfläche **F9**.

Das Ergebnis sieht anschließend etwa wie in Bild 4 aus. Wie aber sollen wir nun besser als zuvor prüfen, ob wir für jedes Ereignis eine Ereignisprozedur angelegt haben? Ganz einfach: Der VBA-Editor hat im Kombinationsfeld der Ereignisse nun alle Einträge markiert, denen Sie einen Haltepunkt hinzugefügt haben (siehe Bild 5).

Null-Werte und wie man damit umgeht

Null ist nicht 0 ist nicht "". Das sind die grundlegenden Weisheiten, die man als Access-Entwickler verinnerlichen muss. Was ist aber dann Null? Und wie erkenne ich die Unterschiede etwa zu 0 oder einer leeren Zeichenfolge? In diesem Artikel erfahren Sie mehr zu diesem Thema und lernen den Umgang mit Null-Werten samt der dazu notwendigen Funktionen wie IsNull oder Nz kennen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1205_NullWerte.mdb**.

Null, 0 und die leere Zeichenfolge

Wenn Sie auf die Daten einer Tabelle zugreifen – beispielsweise über eine Domänenfunktion wie **DLookup** – erhalten Sie verschiedenste Werte. Im Optimalfall finden Sie den gesuchten Wert wie etwa den Vornamen eines Kunden vor. Vielleicht wurde aber gar kein Vorname in den entsprechenden Datensatz eingetragen – dann ist das Feld leer und **DLookup** liefert den Wert **Null** zurück (Wir gehen an dieser Stelle davon aus, dass es einen Datensatz mit dem Wert **1** im Feld **KundeID** gibt – anderenfalls würde **DLookup** auch den Wert **Null** zurückliefern, aber diesmal, weil überhaupt kein passender Datensatz gefunden wurde).

Das können Sie prüfen, indem Sie im Direktbereich des VBA-Editors (zu öffnen mit **Strg + G**) die folgenden Anweisung absetzen:

```
? DLookup("Vorname", "tblKunden", "KundeID = 1")
Null
```

Wenn Sie das Ergebnis dieses Aufrufs von **DLookup** nun innerhalb einer kleinen Prozedur in eine Variable des Datentyps **String** eintragen möchten, führt dies zu einem Fehler (siehe Bild 1): **Unzulässige Verwendung von Null**. **Null** ist also zumindest schon einmal nichts, was man einer **String**-Variablen als Inhalt zuweisen kann.

Was tun? Wahrscheinlich möchten Sie der Variablen an dieser Stelle schlicht und einfach eine leere Zeichenkette zuweisen, also den Ausdruck **""**.

Die IsNull-Funktion

Um dies zu erledigen, können Sie vorab prüfen, ob das Feld den Wert **Null** enthält und in diesem Fall die leere Zeichenkette übergeben.

Dieses Unterfangen hat bereits vielen Access-Entwicklern Kopfzerbrechen bereitet. Die folgende **If... Then**-Bedingung prüft, ob der Vorname des Kunden mit dem Wert **1** im Feld **KundeID** vorhanden ist:

```
If DLookup("Vorname", "tblKunden", "KundeID = 1") _
        = Null Then
    MsgBox "Vorname nicht vorhanden"
Else
    MsgBox "Vorname vorhanden"
End If
```

Wieder erwarten liefert der Code ein Meldungsfenster, wonach der Kunde vorhanden ist. Was läuft hier schief? Nun: Der Vergleich mit dem Wert **Null** liefert immer den Ausdruck **Null**. Das bestätigt auch die folgende Anweisung im Direktfenster:

```
? IsNull(Dlookup("Vorname", "tblKunden", _
    "KundeID = 1") = NULL)
Wahr
```

Die Prüfung eines Ausdrucks auf den Wert **Null** führen Sie mit der Funktion **IsNull** durch. Diese erwartet

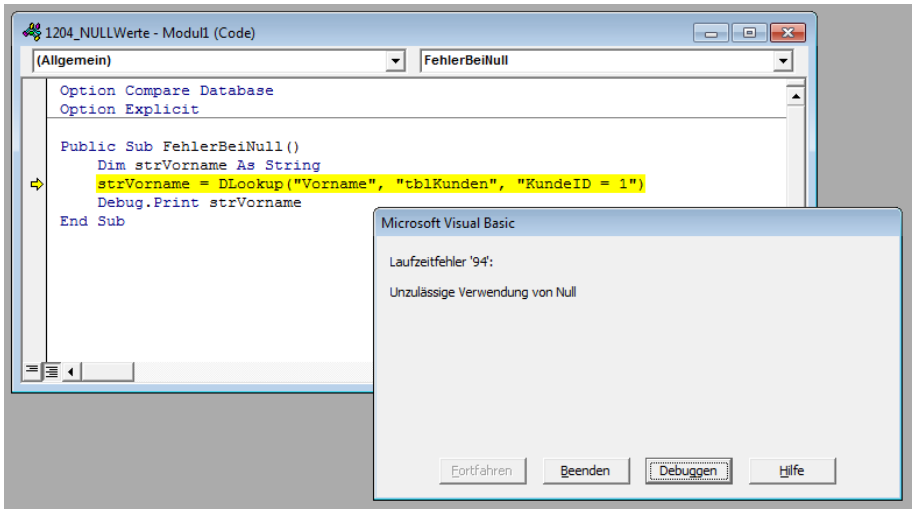


Bild 1: Fehler beim Versuch, einer String-Variablen einen NULL-Wert zuzuweisen

Gruppierungen in Abfragen

Abfragen lassen sich längst nicht nur dazu benutzen, die gewünschten Felder oder Datensätze einer oder mehrerer Tabellen darzustellen. Sie können Daten auch nach bestimmten Kriterien zu Gruppen zusammenfassen und für diese Gruppen verschiedene Informationen abfragen – beispielsweise die Summe bestimmter Zahlenwerte, den größten oder kleinsten Wert und vieles mehr. In diesem Artikel erfahren Sie alles über die Gruppierungsfunktionen in Abfragen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1205_GruppierungenIn-Abfragen.mdb**.

Abfragen gruppieren

Um eine Abfrage mit einer Gruppierung zu erstellen, legen Sie zunächst eine gewöhnliche Abfrage an. Fügen Sie die Tabellen hinzu, deren Daten Sie gruppiernen möchten und schließen Sie den Dialog **Tabelle anzeigen**. Nun können Sie die gewünschten Felder zum Entwurfsraster des Abfrageentwurfs hinzufügen.

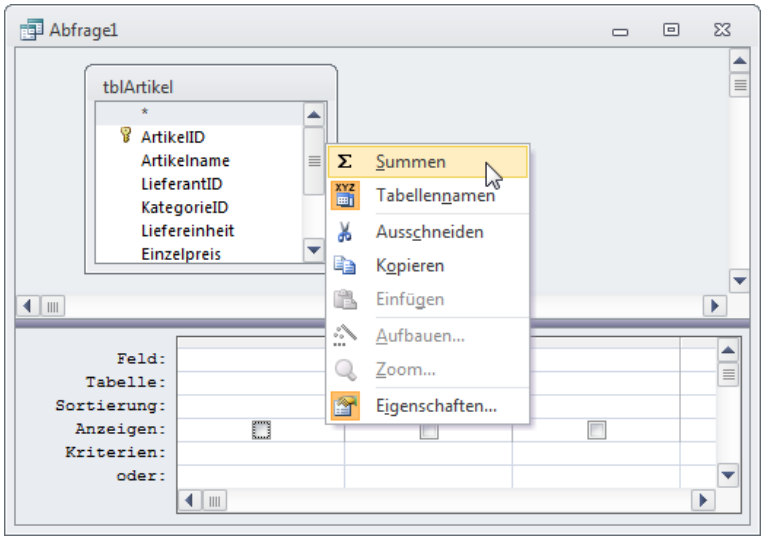


Bild 1: Einblenden der Zeile zum Festlegen der Gruppierungseigenschaften

Wie aber gruppieren Sie die Daten? Im aktuellen Zustand des Abfrageentwurfs gelingt dies gar nicht. Sie müssen dazu zunächst eine weitere Zeile im Entwurfsraster einblenden, was versionsübergreifend am einfachsten über das Kontextmenü des Entwurfsrasters gelingt. Dort finden Sie in älteren Access-Versionen den Eintrag **Funktionen** vor, in neuere Versionen heißt der Befehl **Summen**. In beiden Fällen wird dieser jedoch durch ein Summenzeichen gekennzeichnet (siehe Bild 1).

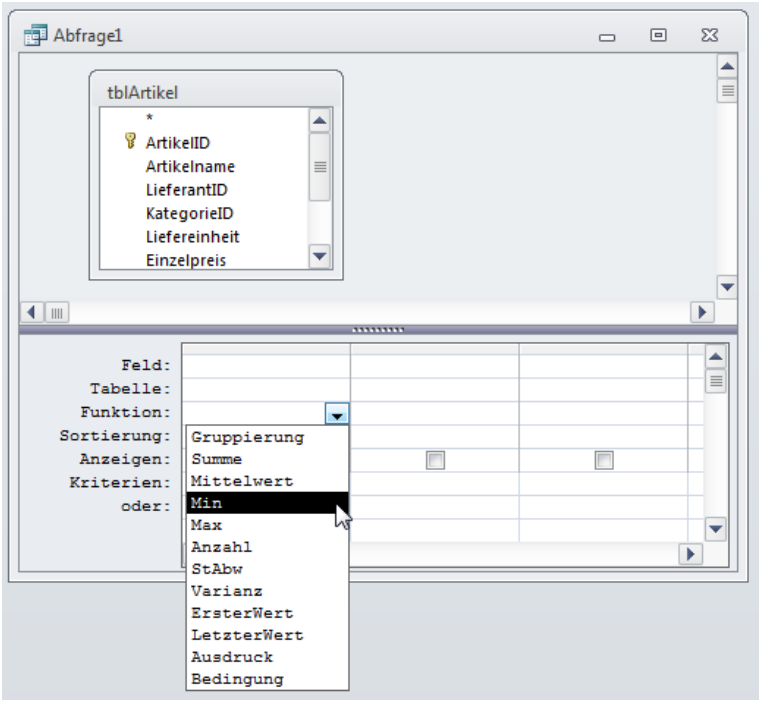


Bild 2: Auswählen einer Gruppierungsfunktion

Gleich danach blendet Access eine neue Zeile im Entwurfsraster ein, mit dem sich die Einträge aus Bild 2 auswählen lassen.

Artikel pro Kategorie zählen

Im ersten Beispiel wollen wir herausfinden, wieviele Artikel jede Kategorie in der Tabelle **tblArtikel** stecken. Dazu fügen wir das Feld **KategorieID** zwei Mal zum Entwurfsraster der Abfrage hinzu. Warum gleich zwei Mal? Das erste Exemplare soll die Namen der Kategorien anzeigen, das zweite die Anzahl der Artikel je Kategorie.

Damit die erste Spalte den Namen der Kategorie ausgibt und die zweite die Anzahl der Artikel je Kategorie, wählen Sie für die erste Spalte in der Zeile **Gruppierung** den Eintrag **Gruppierung** und für die zweite den Eintrag **Anzahl** aus (siehe Bild 3).

Das Ergebnis erhalten Sie bereits, wenn Sie nun in die Datenblattansicht wechseln. Es sieht wie in Bild 4 aus – genau wie gewünscht! Hier gibt es eigentlich nur einen Verbesserungsvorschlag: Die Spaltenüberschrift **AnzahlVonKategorieID** haben wir erstens nirgends festgelegt und wirkt zweitens nicht besonders professionell. Access legt für Gruppierungsspalten, die mit einer Aggregatfunktion versehen wurden, automatisch eine aus der Aggregatfunktion und dem Feldnamen bestehende Spaltenüberschrift fest.

Dieses kleine Manko ist jedoch schnell behoben: Stellen Sie dem Feldnamen im Entwurfsraster einfach die gewünschte Spaltenüberschrift voran, schon sieht das Datenblatt perfekt aus (siehe Bild 5).

Gruppierung nach mehr als einem Feld

Dies war eine der einfachsten Gruppierungen, die Sie mit einer Access-Abfrage realisieren können. Um die

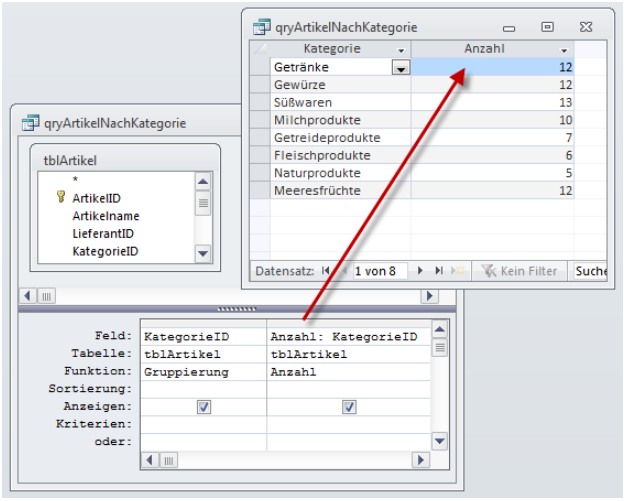


Bild 5: Finetuning der Spaltenüberschrift einer Aggregatspalte

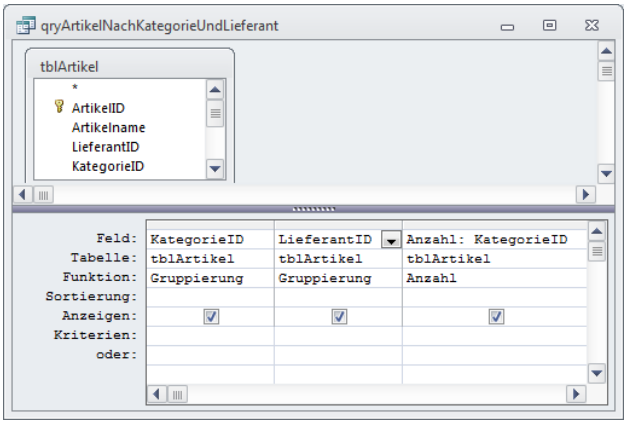


Bild 6: Entwurf einer Abfrage mit zwei Gruppierungskriterien

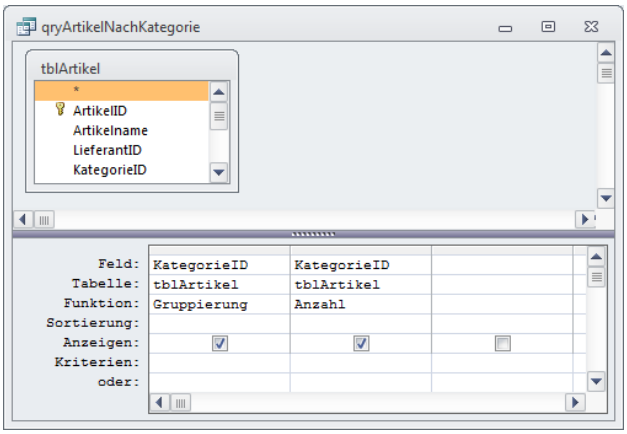


Bild 3: Hinzufügen zweier Felder zum Ausgaben des Kategorienamens und der Anzahl der Artikel je Kategorie

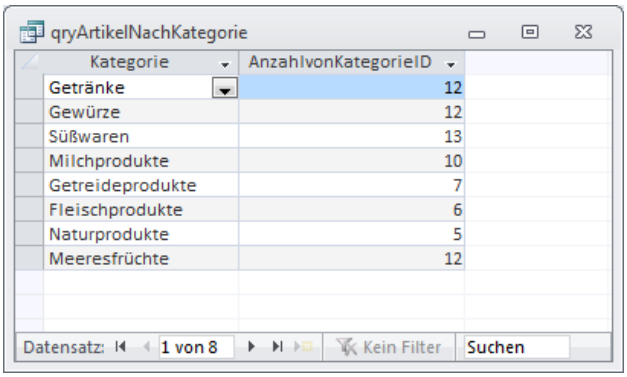


Bild 4: Ausgabe der Artikel je Kategorie

Funktion des für das erste Feld gewählten Eintrags **Gruppierungen** richtig zu verstehen, verwenden wir im zweiten Beispiel ein zusätzliches Feld als Gruppierung – nämlich **LieferantID**.

Dieses Feld fügen Sie zwischen den beiden bereits vorhandenen Feldern ein. In der Zeile **Funktionen** legt Access automatisch den Eintrag **Gruppierung** fest – was in diesem Fall ja auch erwünscht ist. Der Entwurf sieht nun wie in Bild 6 aus.

Ein Wechsel in die Datenblattansicht zeigt, dass die Abfrage nun nicht mehr nur nach den Kategorien, sondern nach jeder Kombination aus Kategorie und Lieferant gruppiert und für jede Gruppe die Anzahl der Artikel liefert (siehe Bild 7).

Eine gruppierte Abfrage kombiniert also alle möglichen Kombinationen der Felder, die in der Zeile **Funktion** den Wert **Gruppierung** enthalten, und liefert weitere Daten entsprechend der dafür vorgesehenen Aggregatfunktionen.

Sie müssen übrigens für jede Spalte der Abfrage eine Funktion auswählen.

Kreuztabellenabfragen

Herkömmliche Auswahlabfragen liefern zeilenweise Daten, wobei jede Zeile gegebenenfalls noch die aggregierten Werte für eine Gruppe von Datensätzen enthält – dies erreichen Sie durch die Gruppierung der Abfragedaten. Wenn die Daten nach mehreren Feldern gruppiert werden sollen, also beispielsweise nach Kunde und Artikel, wird dies in einer Auswahlabfrage schnell unübersichtlich. Den nötigen Überblick erhalten Sie, wenn Sie solche Daten in einer Kreuztabelle anzeigen. Dieser Artikel zeigt, wie Sie Kreuztabellen erstellen und was Sie dabei beachten müssen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1205_Kreuztabellenabfragen.mdb**.

Vorteile von Kreuztabellenabfragen

Wenn Sie Daten nach mehr als einem Feld gruppieren, also beispielsweise den Umsatz nach Kunden und Artikelkategorie ermitteln möchten, können Sie dies wie in Bild 1 erledigen. Optimal ist diese Ansicht jedoch nicht. Viel besser wäre es, wenn der Kunde als Zeilen- und die Kategorie als Spaltenkopf angezeigt würde und die Kreuzungspunkte von Zeilen und Spalten die Umsätze je Kunde und Kategorie enthielten. Das Ergebnis sieht dann wie in Bild 2 aus und wurde durch den Einsatz einer Kreuztabellenabfrage erreicht. Der Vorteil ist, dass Sie die Werte direkt vergleichen können: Sie sehen, wie sich die Umsätze eines Kunden auf die Kategorien aufteilen, können aber auch die Umsätze verschiedener Kunden für eine bestimmte Kategorie vergleichen.

qryUmsatzJeKategorieUndKunde

KundeID	Firma	Kategoriename	Umsatz
1	Alfreds Futterkiste	Getränke	276,75 €
1	Alfreds Futterkiste	Gewürze	669,40 €
1	Alfreds Futterkiste	Meeresfrüchte	260,75 €
1	Alfreds Futterkiste	Milchprodukte	627,50 €
1	Alfreds Futterkiste	Naturprodukte	302,10 €
2	Ana Trujillo Emparedados y helados	Getränke	30,00 €
2	Ana Trujillo Emparedados y helados	Getreideprodukte	35,00 €
2	Ana Trujillo Emparedados y helados	Meeresfrüchte	30,00 €
2	Ana Trujillo Emparedados y helados	Milchprodukte	539,40 €
2	Ana Trujillo Emparedados y helados	Naturprodukte	34,88 €
2	Ana Trujillo Emparedados y helados	Süßwaren	32,20 €

Datensatz: 1 von 598 | Kein Filter | Suchen

Bild 1: Gruppierung nach Kunde und Kategorie

qryUmsatzJeKategorieUndKunde_Kreuztabelle

KundeID	Firma	Fleischprodukte	Getränke
1	Alfreds Futterkiste		276,75
2	Ana Trujillo Emparedados y helados		30
3	Antonio Moreno Taquería	761	879,5
4	Around the Horn	547,2	613,5
5	Berglunds snabbköp	1129,25	4149,34
6	Blauer See Delikatessen	74,5	171
7	Blondel père et fils	2488,3	1987,96
8	Bólido Comidas preparadas	1513,43	155
9	Bon app'	411,45	1137,95
10	Bottom-Dollar Markets	846	881,38
11	B's Beverages		922,5
12	Cactus Comidas para llevar		545,5
13	Centro comercial Moctezuma		
14	Chop-suey Chinese	171,36	342,75

Datensatz: 1 von 89 | Kein Filter | Suchen

Bild 2: Gruppierung nach Kunde und Kategorie per Kreuztabelle

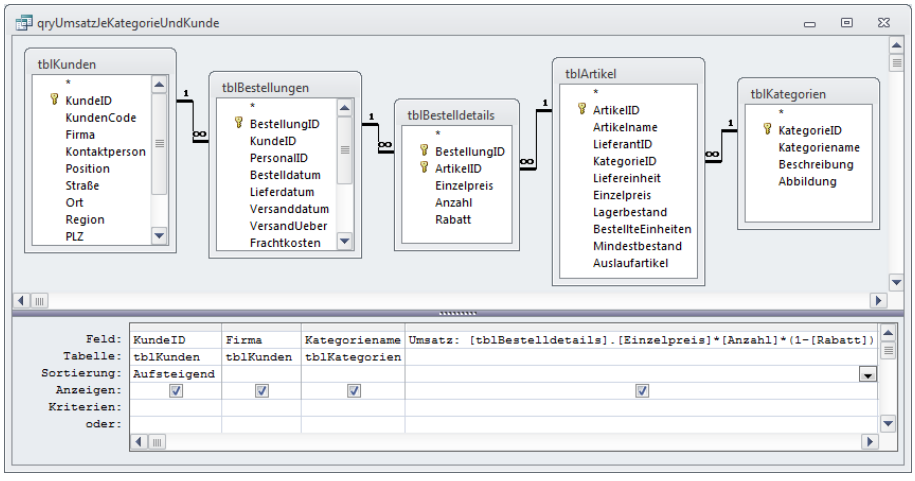


Bild 3: Abfrage als Vorbereitung auf die Kreuztabelle

Kreuztabellenabfrage erstellen

Bevor wir uns um die Erstellung einer Kreuztabellenabfrage mit dem Assistenten kümmern, schaffen wir die notwendigen Voraussetzungen.

Das bedeutet, dass Sie eine Abfrage erstellen, die alle Felder enthält, die in der Kreuztabelle angezeigt werden sollen. In unserem Fall betrifft das gleich eine ganze

Reihe von Tabellen, aus denen folgende Felder zum Abfrageentwurf hinzugefügt werden sollen (siehe Bild 3):

- Tabelle **tblKunden**: Felder **KundeID**, **Firma** als Zeilenüberschrift
- Tabelle **tblBestelldetails**: Berechnetes Feld mit dem Ausdruck **Umsatz: [tblBestelldetails].[Einzelpreis]*[Anzahl]*(1-[Rabatt])** als berechnetes Feld
- Tabelle **tblKategorien**: Feld **Kategoriename** als Spaltenüberschrift

Die übrigen beiden Tabellen **tblBestellungen** und **tblArtikel** benötigen Sie ebenfalls, damit Access die Beziehung zwischen den Tabellen mit den in der Abfrage enthaltenen Feldern erkennen kann.

Abfrage-Assistent für Kreuztabellenabfragen

Den Assistenten zur Erstellung von Kreuztabellenabfragen starten Sie je nach Access-Version auf unterschiedliche Weise. In neueren Access-Versionen (ab Access 2007) finden Sie den Eintrag im Ribbon unter **Erstellen|Abfrageassistent**, in älteren Access-Versionen klicken Sie doppelt auf den Eintrag **Erstellt eine Abfrage unter Verwendung des Assistenten** im Bereich **Abfragen** des Datenbankfensters. Der Assistent sieht in allen Versionen etwa gleich aus. Im ersten Schritt wählen Sie zunächst die Tabelle oder Abfrage aus, die als Datenherkunft für die Kreuztabellenabfrage dient (sieBild 4).

Aus den Feldern dieser Datenherkunft wählen Sie nun die Felder für die Zeilenüberschriften, die Spaltenüberschriften und für die gruppierten, berechneten Werte aus. Der Dialog aus Bild 5 ermöglicht die Auswahl eines oder mehrere Felder für die Zeilenüberschriften. Wenn Sie mehrere Felder wählen, werden diese nebeneinander dargestellt. In unserem Fall sollen die beiden Felder **KundeID** und **Firma** angezeigt werden. Danach legen Sie im Dialog aus Bild 6 fest, welches Feld die

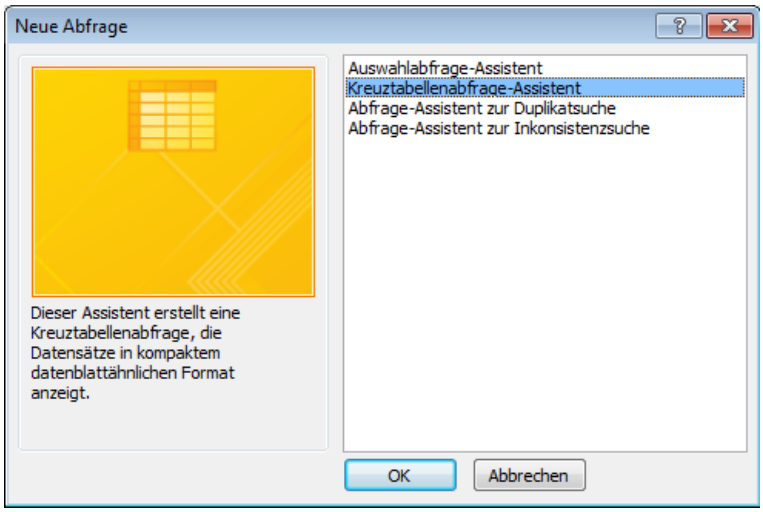


Bild 4: Start des Assistenten

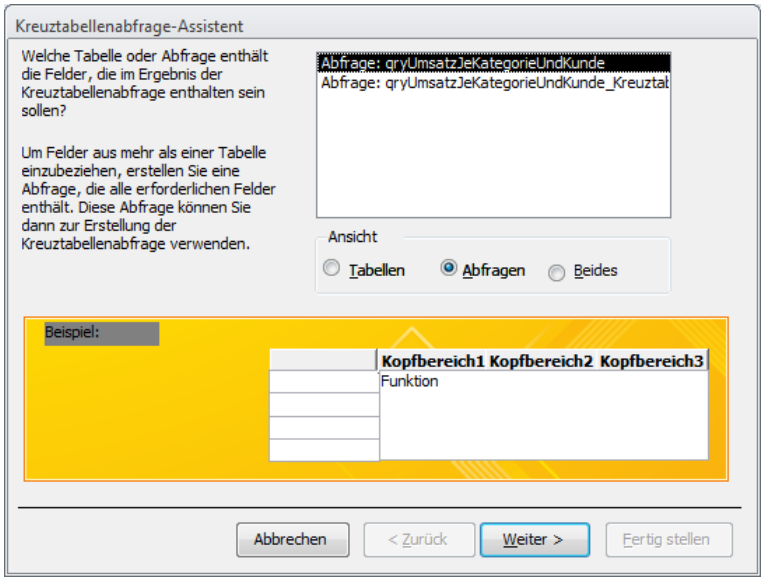


Bild 5: Auswählen der Quellabfrage

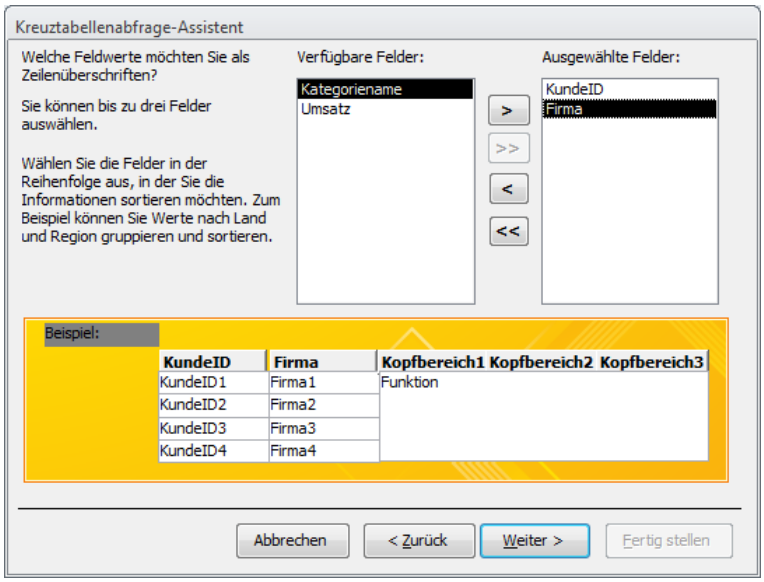


Bild 6: Festlegen des Feldes für die Zeilenköpfe

Werte für die Spaltenüberschriften liefern soll. Hier können Sie nur ein Feld auswählen. Der Inhalt dieses Feldes landet im Ergebnis der Kreuztabellenabfrage in den Spaltenköpfen (siehe Bild 7).

Schließlich teilen Sie dem Assistenten noch mit, welches Feld nach den für die Spalten- und Zeilenköpfe festgelegten Gruppierungen berechnet werden und welche Berechnung die Abfrage dabei durchführen soll. Sie wählen zwischen Einträgen wie Anzahl, Summe et cetera (sieBild 8). In diesem Schritt geben Sie außerdem noch an, ob die Gesamtsumme je Zeile neben den Zeilenköpfen ausgegeben werden soll – wir entscheiden uns für dieses Beispiel für die Anzeige der Gesamtsumme. Das Ergebnis sieht bereits gut aus: Es zeigt die richtigen Zeilen- und Spaltenköpfe sowie die Gesamtsumme und die gruppierten Summen an. Es überzeugt aber nicht vollständig: Es zeigt nämlich viel mehr Nachkommastellen an, als für die Berechnung von Währungsbeträgen nötig ist (siehe Bild 9).

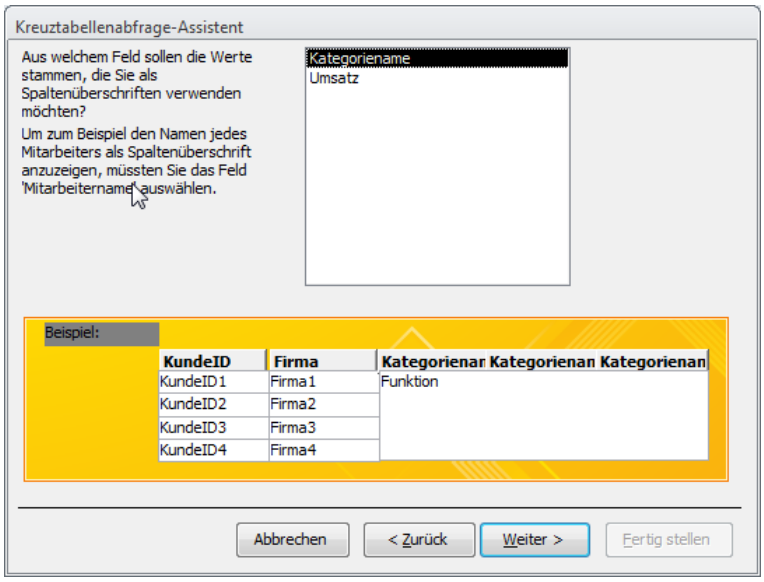


Bild 7: Festlegen des Feldes für die Spaltenköpfe

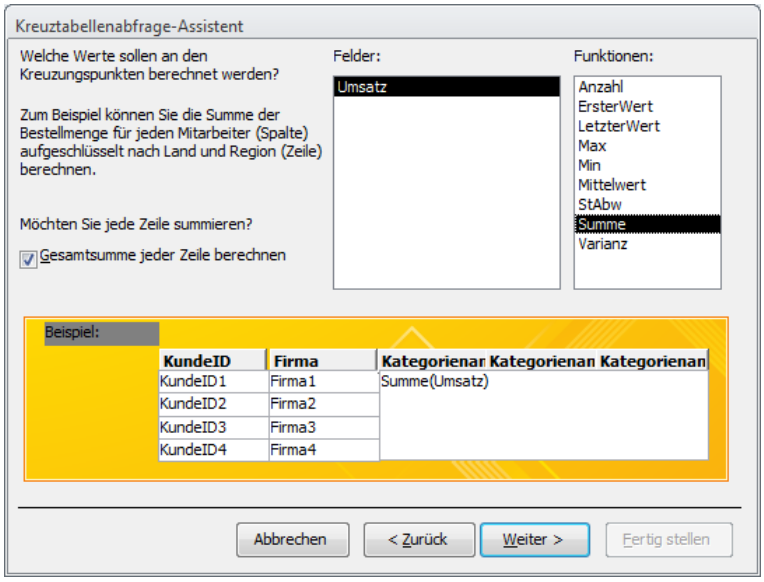
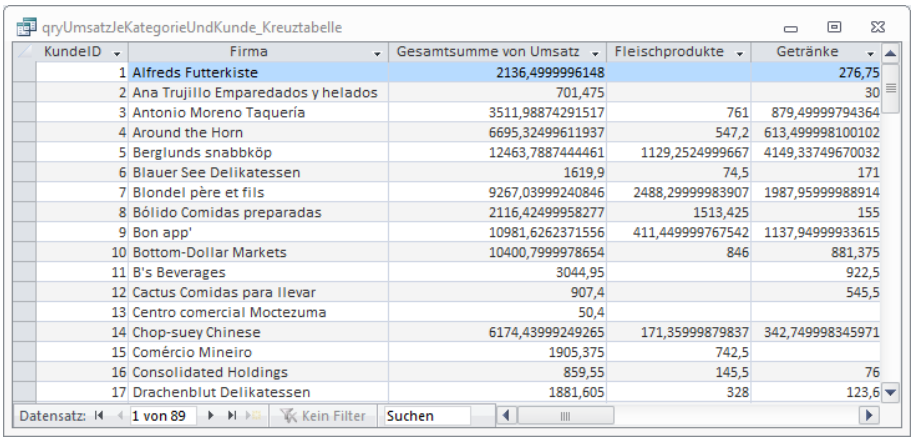


Bild 8: Auswählen des zu berechnenden Feldes

Finetuning

Kümmern wir uns also zunächst um die Anzahl der Nachkommastellen im Ergebnis. Dazu öffnen Sie die Entwurfsansicht der Abfrage, die etwas anders aussieht als der Entwurf einer einfachen Auswahlabfrage (sieBild 10). Sollten Sie einmal manuell eine Kreuztabellenabfrage erstellen wollen, legen Sie zunächst eine einfache Auswahlabfrage an und ändern den Typ dann in Kreuztabellenabfrage um. Dies erledigen Sie versionsübergreifend mit dem Kontextmenüeintrag **Abfragetyp|Kreuztabellenabfrage** des Abfrageentwurfs. Zusätzlich zu der bereits von gruppierten Abfragen bekannten Zeile **Funktion** halten Kreuztabellenabfragen im Entwurf noch eine Zeile namens **Kreuztabelle** bereit. Mit dieser Zeile legen Sie die Funktion der beteiligten Felder in der Kreuztabelle fest. In unserem Beispiel


qryUmsatzJeKategorieUndKunde_Kreuztabelle

KundeID	Firma	Gesamtsumme von Umsatz	Fleischprodukte	Getränke
1	Alfreds Futterkiste	2136,4999996148		276,75
2	Ana Trujillo Emparedados y helados	701,475		30
3	Antonio Moreno Taquería	3511,98874291517	761	879,49999794364
4	Around the Horn	6695,32499611937	547,2	613,499998100102
5	Berglunds snabbköp	12463,7887444461	1129,2524999667	4149,33749670032
6	Blauer See Delikatessen	1619,9	74,5	171
7	Blondel père et fils	9267,03999240846	2488,29999983907	1987,95999988914
8	Bólido Comidas preparadas	2116,42499958277	1513,425	155
9	Bon app'	10981,6262371556	411,449999767542	1137,94999933615
10	Bottom-Dollar Markets	10400,7999978654	846	881,375
11	B's Beverages	3044,95		922,5
12	Cactus Comidas para llevar	907,4		545,5
13	Centro comercial Moctezuma	50,4		
14	Chop-suey Chinese	6174,43999249265	171,35999879837	342,749998345971
15	Comércio Mineiro	1905,375	742,5	
16	Consolidated Holdings	859,55	145,5	76
17	Drachenblut Delikatessen	1881,605	328	123,6

Datensatz: 1 von 89 | Kein Filter | Suchen

Bild 9: Datenblattansicht der vom Assistenten erstellten Kreuztabellenabfrage

Schnellsuche im Listenfeld

Listenfelder eignen sich prima zur Anzeige von Daten in Listenform – vor allem, wenn der Benutzer diese Daten nicht an Ort und Stelle ändern soll. Im Gegensatz zur Datenblattansicht lässt es aber auch eigene Filter- oder Sortiermöglichkeiten vermissen. Dieser Artikel zeigt, wie Sie zumindest eine komfortable Suche nach bestimmten Einträgen realisieren.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1205_Schnellsuche.mdb**.

Schnellsuche

Warum heißt die in diesem Artikel vorgestellte Suchfunktion eigentlich Schnellsuche? Nun: Im Gegensatz zu einer Suche, bei der Sie den Suchbegriff in ein Suchfeld eingeben und die Suche dann etwa durch Betätigen der Eingabetaste oder per Klick auf eine entsprechende Schaltfläche anstoßen, soll die Suche hier nach Eingabe jedes einzelnen Zeichens ausgelöst werden. Sprich: Wenn Sie für eine Suche im Feld **Artikelname** der Tabelle **tblArtikel** den Buchstaben **A** eingeben, soll das Listenfeld gleich alle Artikel anzeigen, die mit diesem Buchstaben beginnen. Geben Sie einen weiteren Buchstaben wie etwa **L** ein, zeigt das Listenfeld nur noch Einträge an, die mit **AL** beginnen.

Wenn Sie nach allen Artikelnamen suchen wollen, die einen Buchstaben oder eine Zeichenfolge an irgendeiner Stelle enthalten, verwenden Sie entsprechende Platzhalter wie etwa das Sternchen. Der Suchbegriff ***A*** liefert dann alle Artikel, die den Buchstaben **A** im Namen enthalten (siehe Bild 1).

Alternativ können Sie die Suche auch so gestalten, dass automatisch alle Artikelnamen ausgegeben werden, die eine bestimmte Zeichenfolge enthalten – dazu später mehr.

Formular mit Schnellsuche erstellen

Legen Sie ein neues Formular an und speichern Sie es unter dem Namen **frmSchnellsuche**.

Fügen Sie dem Formular dann ein Listenfeld zur Anzeige des Suchergebnisses an.

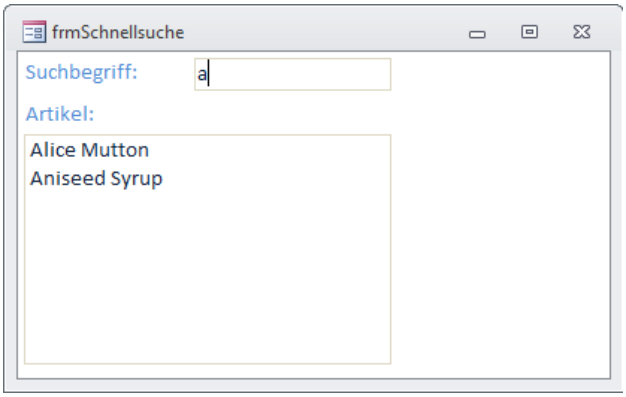


Bild 1: Schnellsuche im Listenfeld

Das Listenfeld soll den Namen **lstArtikel** erhalten. Es soll nur zur Auswahl eines Artikels dienen, daher reicht es, wenn es die Artikelnamen anzeigt. Damit der ausgewählte Datensatz jedoch eindeutig identifiziert werden kann, muss die **Datensatzherkunft** des Listenfeldes neben dem Feld **Artikelname** auch noch das Feld **ArtikelID** enthalten. Außerdem kann eine Sortierung nach dem Feld **Artikelname** nicht schaden (siehe Bild 2).

Damit das Listenfeld nur den Artikelnamen, nicht aber das Feld **ArtikelID** anzeigt, stellen Sie außerdem

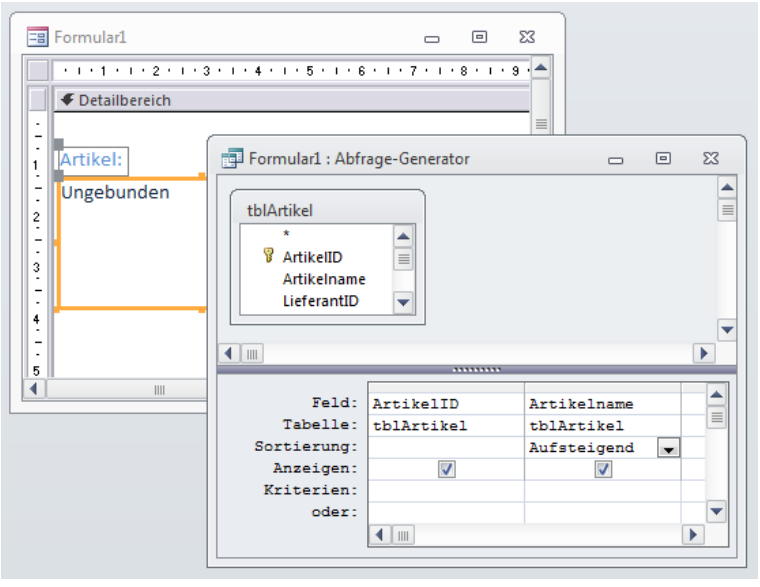


Bild 2: Datensatzherkunft des Listenfeldes

die Eigenschaften **Spaltenanzahl** und **Spaltenbreiten** des Listenfeld-Steuerelements auf die Werte **2** und **0cm** ein. Wie ein Wechsel in die Formularansicht zeigt, liefert das Listenfeld bereits die gewünschten Daten (siehe Bild 3). Bevor wir uns um das Textfeld zur Eingabe des Suchergebnisses kümmern, stellen wir noch einige Formulareigenschaften ein. Die Eigenschaften **Datensatzmarkierer**, **Navigationsschaltflächen**, **Trennlinien** und **Bildlaufleisten** stellen Sie auf den Wert **Nein** ein. Das Formular wirkt dadurch etwas aufgeräumter, außerdem benötigen wir all diese Elemente aktuell nicht.

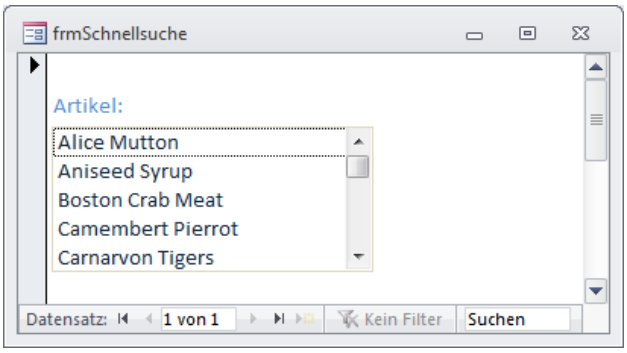


Bild 3: Das Listenfeld wird wie gewünscht gefüllt.

Suchfeld anlegen

Als Suchfeld verwenden wir ein herkömmliches Textfeld. Es soll **txtSuche** heißen und soll über dem Listenfeld platziert werden. Die Eingabe von Suchbegriffen ist kein Problem, aber wie sorgen wir dafür, dass die im Listenfeld angezeigten Artikelnamen nach jeder Änderung des Suchbegriffs aktualisiert werden?

Dazu benötigen wir ein Ereignis und eine entsprechende Ereignisprozedur, die nach jeder Änderung des Textfeldinhalts ausgelöst wird. Habe ich Änderung gesagt? Nun, da gibt es doch eine Ereigniseigenschaft mit dem Namen **Bei Änderung**. Prüfen Sie also, ob dieses Ereignis das Passende ist und legen dieses an. Dazu markieren Sie das Textfeld, aktivieren das Eigenschaftsfenster und wählen für die Eigenschaft **Bei Änderung** den Eintrag **[Ereignisprozedur]** aus (siehe Bild 4). Die nach einem Klick auf die Schaltfläche mit den drei Punkten (...) im VBA-Editor angezeigte Prozedur ergänzen Sie wie folgt:

```
Private Sub txtSuche_Change()
    Debug.Print "Suchbegriff: " & Me!txtSuche
End Sub
```

Damit soll zu Testzwecken der aktuelle Inhalt des Textfeldes im Direktfenster ausgegeben werden. Das Ergebnis ist eher ernüchternd: Dort erscheint nämlich nur der Text **Suchbegriff:**, aber nicht der in das Suchfeld eingegebene Text. Die Lösung ist einfach: **Me!txtSuche** ist eine Abkürzung für **Me!txtSuche.Value**. Das lässt sich so abkürzen, weil **Value** die Standardeigenschaft des Textfeldes ist. Zwischen dem Wert der Eigenschaft **Value** und dem im Textfeld angezeigten Text gibt es jedoch einen Unterschied: **Value** wird nämlich erst auf den enthaltenen Text eingestellt, wenn Sie den Inhalt etwa mit der Eingabetaste oder durch den Wechsel zu einem anderen Steuerelement bestätigen.

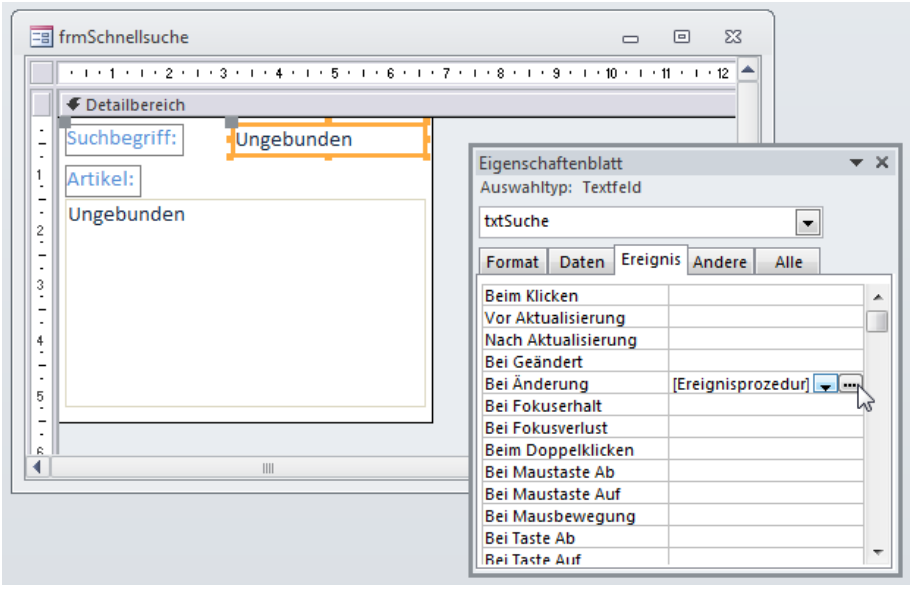


Bild 4: Anlegen der Ereignisprozedur, die durch das Ereignis **Bei Änderung** ausgelöst wird.

Dies können Sie leicht prüfen, indem Sie eine Zeichenfolge wie abc eingeben, auf das Listenfeld klicken, dann wieder die Einfügemarke im Textfeld platzieren und ein weiteres Zeichen eintippen. Im Direktfenster erscheint dann die Zeichenfolge **Suchbegriff: abc**. Zum Glück bietet das Textfeld auch eine Eigenschaft, mit der Sie auf den aktuell angezeigten Text zugreifen können. Diese Eigenschaft heißt **Text**. Mit der folgenden Änderung der Prozedur wird jeweils der aktuelle Text im Direktfenster ausgegeben:

Ereignisse in Formularen: Öffnen und Schließen

Nachdem Sie im ersten Teil dieser Artikelreihe gelernt haben, wie Sie die Reihenfolge der auftretenden Ereignisse beim Öffnen und Schließen von Formularen samt Unterformular ermitteln, schauen wir uns in diesem Teil genauer an, was beim Öffnen eines Formulars geschieht und welche Aktionen Sie in den verschiedenen Ereignisprozeduren unterbringen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1205_EreignisseInFormularen_2.mdb**.

Ereignisse beim Öffnen eines Formulars

Schauen wir uns nun im Detail an, was beim Öffnen eines Formulars geschieht und was Sie mit den ausgelösten Prozeduren anfangen können.

Das Ereignis Beim Öffnen

Dieses Ereignis wird beim Öffnen eines Formulars als Erstes ausgelöst. Damit dies geschieht, stellen Sie den Wert der Ereigniseigenschaft **Beim Öffnen** auf den Wert **[Ereignisprozedur]** ein und klicken auf die Schaltfläche mit den drei Punkten, um die durch das Ereignis ausgelöste Ereignisprozedur anzulegen. Diese sieht zunächst so aus:

```
Private Sub Form_Open(Cancel As Integer)
End Sub
```

Es bietet einen Parameter namens **Cancel**, mit dem sich das Öffnen abbrechen lässt, und das ist einer der wesentlichen Unterschiede zum Ereignis **Form_Load**, das wir gleich vorstellen.

Das Ereignis Beim Laden

Dieses Ereignis wird nach dem Ereignis **Form_Open** ausgelöst. Der wesentliche Unterschied zwischen beiden Ereignissen ist, dass Sie das Öffnen des Formulars im Ereignis **Beim Öffnen** abbrechen können. Außerdem bietet das **Form_Load**-Ereignis die Möglichkeit, die angezeigten Daten bereits beim Laden per VBA-Code anzupassen.

Öffnungsvorgang abbrechen

Wenn Sie den Parameter **Cancel** auf den Wert **True** einstellen, wird das Formular gar nicht erst geöffnet – probieren Sie es aus:

```
Private Sub Form_Open(Cancel As Integer)
    Cancel = True
End Sub
```

Natürlich macht es keinen Sinn, ein Formular mit einer Prozedur auszustatten, die dieses sofort wieder schließt. Dafür sollte es schon eine entsprechende Bedingung geben – zum Beispiel, dass das Formular keine Daten enthält. Dies lässt sich wie folgt realisieren:

```
Private Sub Form_Open(Cancel As Integer)
    If Me.Recordset.EOF Then
        MsgBox "Das Formular enthält keine Daten."
        Cancel = True
    End If
End Sub
```

Das **Recordset**-Objekt ermöglicht den Zugriff auf die Daten des Formulars. Wenn die Eigenschaft **EOF** wahr ist, bedeutet dies, dass der Datensatzzeiger sich hinter dem letzten Datensatz befindet, was gleichbedeutend mit einer leeren Datenherkunft ist. In diesem Fall soll eine Meldung erscheinen und der Parameter **Cancel** auf den Wert **True** eingestellt werden, damit das Formular nach der Anzeige der Meldung gar nicht erst geöffnet wird.

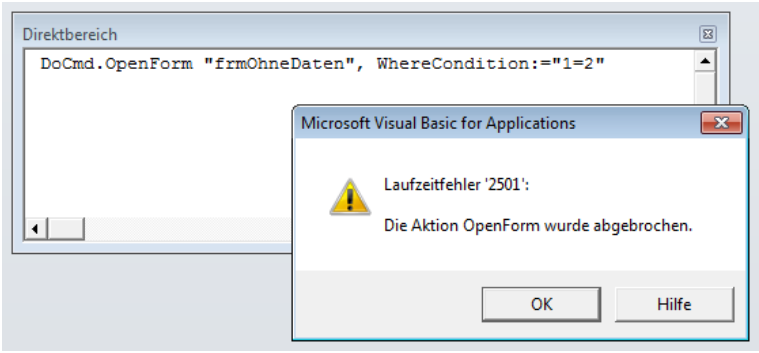


Bild 1: Fehler beim Abbruch des Öffnungsvorgangs

Wie können Sie dies ausprobieren? Indem Sie das Formular mit der **DoCmd.OpenForm**-Methode öffnen und dabei eine **WhereCondition** übergeben, die garantiert für keinen Datensatz der Daten-

herkunft gilt. Ein passender Kandidat für eine solche Bedingung lautet **1=2**. Setzen Sie die folgende Anweisung im Direktbereich des VBA-Editors ab:

```
DoCmd.OpenForm "frmOhneDaten", WhereCondition:="1=2"
```

Die Meldung erscheint wie erwartet, aber leider zieht der Aufruf auch noch einen Fehler nach sich (siehe Bild 1). Um diesen Fehler zu verhindern, müssen Sie die **DoCmd.OpenForm**-Methode in eine kleine Fehlerbehandlung integrieren. Diese sorgt dafür, dass im Falle des Fehlers **2501** keine Meldung angezeigt wird, sondern nur bei anderen Fehlercodes:

```
Public Sub FormularOeffnenAbbrechen()
    On Error Resume Next
    DoCmd.OpenForm "frmOhneDaten", _
        WhereCondition:="1=2"
    Select Case Err.Number
        Case 0, 2501
        Case Else
            MsgBox "Fehler"
    End Select
    On Error GoTo 0
End Sub
```

Datenherkunft einstellen

Meistens ist die Tabelle oder Abfrage, welche die Daten für ein Formular liefert, bereits vor dem Öffnen bekannt und wird für die Eigenschaft **Datenherkunft** des Formulars eingestellt. Gegebenfalls geben Sie beim Öffnen mit **Docmd.OpenForm** noch eine **WhereCondition** oder einen **Filter** mit, um die Daten einzuschränken.

Es gibt jedoch auf Fälle, in denen Sie die Datenherkunft erst beim Öffnen des Formulars einstellen und somit die Eigenschaft **Recordsource** per VBA festlegen.

Nach den Angaben in der Onlinehilfe wird das Formular zwischen den beiden Ereignissen **Form_Open** und **Form_Load** mit Daten gefüllt, sodass aus Performancegründen ein Einstellen der Datenherkunft in der Ereignisprozedur **Form_Open** sinnvoller ist.

Das Gleiche gilt für die Eigenschaften **Filter**, **FilterOn**, **OrderBy** und **OrderByOn**, mit denen Sie einen Filter oder die Reihenfolge für die die anzuzeigenden Daten einstellen.

Standardwerte einstellen

Auch den Standardwert für ein an ein Feld der Datenherkunft gebundenes Steuerelement können Sie in den beiden Ereignisprozeduren **Form_Open** und **Form_Load** einstellen:

```
Private Sub Form_Load()
    Me.Controls("LieferantID").DefaultValue = 1
End Sub

Private Sub Form_Open(Cancel As Integer)
    Me.Controls("LieferantID").DefaultValue = 1
End Sub
```

Dies gelingt allerdings auch im Ereignis **Beim Anzeigen**, das bei jedem Datensatzwechsel ausgelöst wird.

Daten ändern

Nehmen wir an, Sie möchten in einem Detailformular geöffnete Datensätze mit einem Datum des letzten Öffnungsvorgangs versehen. Dann fällt zumindest das Ereignis **Form_Open** weg, denn auch wenn es Anzeichen dafür gibt, dass zu diesem Zeitpunkt bereits Daten vorliegen, so können Sie diese noch nicht bearbeiten.

Den Beweis liefern das Formular **frmZuletztGeoeffnet** und der Fehler, der beim Versuch auftritt, einen Datensatz beim Ereignis **Beim Öffnen** zu ändern (sie-

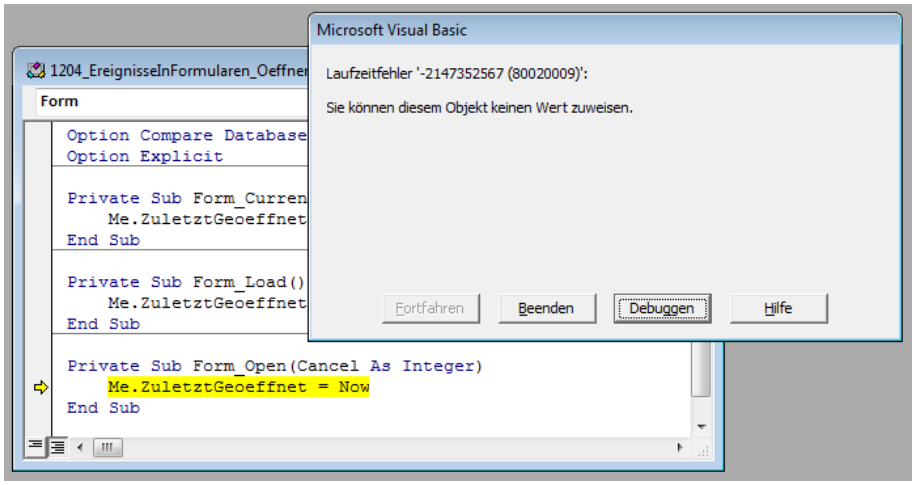


Bild 2: Beim Öffnen ist noch keine Änderung der Daten möglich.

Kreuztabellenabfragen nach Datum

Wenn Sie Kreuztabellenabfragen erstellen, die Daten nach dem Datum gruppieren, gibt es einige Besonderheiten gegenüber einfachen Kreuztabellenabfragen. Dazu gehört etwa die Festlegung der Datumsintervalle oder das Anpassen der Spaltenüberschriften. Außerdem zeigen wir in diesem Artikel, wie Sie in einer Kreuztabellenabfrage mit Parametern etwa zum Einschränken des betroffenen Zeitraums arbeiten.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1206_KreuztabellenabfragenDatum.mdb**.

Kreuztabellenabfragen mit Datum als Spaltenkopf

Im Artikel **Kreuztabellenabfragen** haben wir Ihnen bereits die Grundlagen zur Erstellung von Kreuztabellenabfragen gezeigt. Dort haben Sie auch den Kreuztabellenassistenten kennengelernt, der einem für einfache Fälle die notwendigen Schritte zur Definition der Kreuztabelle abnimmt.

In diesem Beitrag starten wir wieder mit dem Kreuztabellenassistenten. Dieser setzt, wie Sie bereits wissen, eine Abfrage mit allen zu erfassenden Daten voraus. Sie können nicht erst im Assistenten mehrere Tabellen mit den benötigten Feldern zusammenfügen. Zu diesem Zweck haben wir die Abfrage aus Bild 1 vorbereitet. Diese enthält Daten aus den Tabellen **tblPersonal**, **tblBestellungen** und **tblBestelldetails**. Sie soll Informationen über die Umsätze der einzelnen Mitarbeiter nach dem Datum liefern. Dazu fügen Sie dem Entwurf die Felder **PersonalID**, das zusammengesetzte Feld **Personal** mit dem Ausdruck **Nachname & ", " & Vorname**, das Feld **Umsatz** mit dem Ausdruck **Einzelpreis*Anzahl*(1-Rabatt)** sowie das Feld **Bestelldatum** hinzu.

Mit dieser Abfrage starten Sie nun den Kreuztabellenassistenten. Die einzelnen Schritte haben wir bereits im oben erwähnten Artikel detailliert beschrieben. Wenn Sie jedoch im Dialog zur Auswahl der Spaltenköpfe das Feld **Bestelldatum** auswählen (siehe Bild 2), ergibt sich eine Änderung im Vergleich zum gewohnten Ablauf: Im folgenden Dialog fordert der Assistent Sie nämlich auf, das Intervall auszuwählen. Es stehen die folgenden Intervalle zur Verfügung: **Jahr**, **Quartal**, **Monat**, **Datum** und **Datum/Zeit**.

In diesem Beispiel wählen wir den Eintrag **Monat** aus. In der Vorschau zeigt der Assistent direkt an, wie die Spaltenüberschriften aussehen werden (siehe Bild 3). Nach dem Fertigstellen der Kreuztabellenabfrage ändern Sie gleich die Darstellung der Zahlenwerte, damit diese mit maximal zwei Nachkommastellen erscheinen. Dazu ändern Sie beispielsweise die Spalte **Umsatz** in den folgenden Ausdruck:

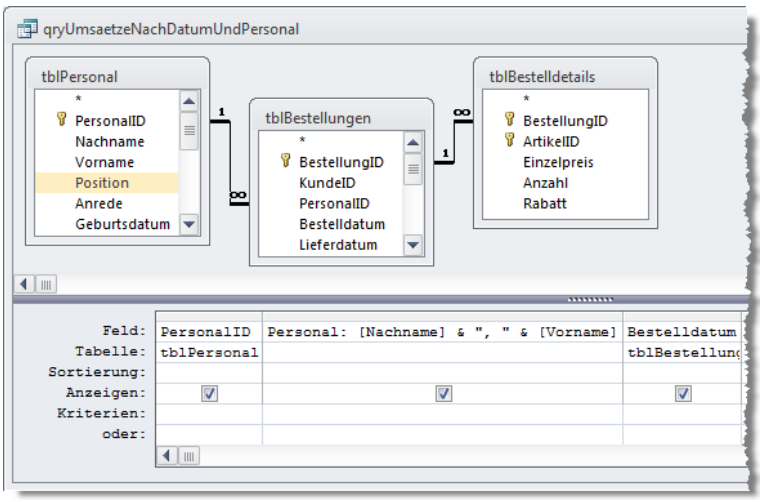


Bild 1: Umsätze nach Datum und Personal

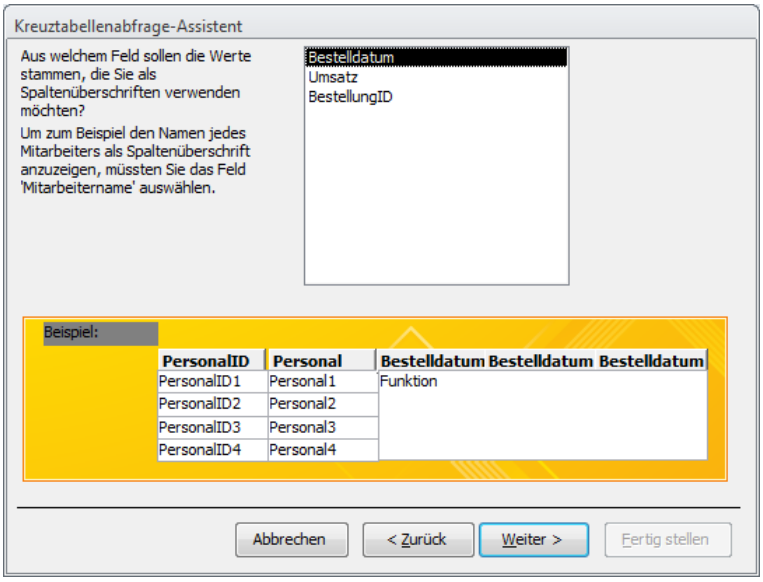


Bild 2: Auswahl des Feldes **Datum**

Dateibefehle

VBA bietet eine Reihe von Dateibefehlen, mit denen Sie Verzeichnisse anlegen, Verzeichnisse entfernen, das aktuelle Arbeitsverzeichnis ermitteln, das Arbeitsverzeichnis ändern, Dateien löschen und sogar komplette Verzeichnishierarchien neu anlegen. Dieser Artikel liefert die Grundlagen zu diesen Befehlen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1206_Dateibefehle.mdb**.

Dateizugriffe

Der Artikel **Textdateien** stellt die Möglichkeiten zum Erstellen und zum Lesen von Textdateien vor. Dort gibt es eine Methode namens **Open**, die je nach Dateimodus eine bestehende Textdatei öffnet oder eine neue Datei erstellt. Was hat das mit dem vorliegenden Artikel zu tun? Ganz einfach: Die **Open**-Anweisung erwartet die Angabe des Dateinamens. Dabei können Sie wählen zwischen der Angabe des Dateinamens inklusive Verzeichnis oder nur des Dateinamens. Wenn Sie das Verzeichnis mit angeben, ist genau definiert, auf welche Datei die **Open**-Anweisung sich beziehen soll. Falls aber wie im folgenden Beispiel nur ein Dateiname angeben wurde, ermittelt VBA selbst den Pfad:

```
Open "Test.txt" For Input As #1
```

Aktuelles Verzeichnis ermitteln

Und woher bezieht VBA den Ort der betroffenen Datei? Dieser wird intern gespeichert und kann über die Funktion **CurDir** ausgelesen werden. Ein Aufruf im Direktfenster des VBA-Editors sieht beispielsweise wie folgt aus:

```
Debug.Print CurDir
C:\Users\Andre\Documents
```

Das bedeutet, dass die **Open**-Anweisung dem Dateinamen ohne Angabe eines Verzeichnisses die intern gespeicherte und mit **CurDir** zu ermittelnde Pfadbezeichnung voranstellt.

Aktuelles Verzeichnis ändern

Wenn sich die Dateioperationen innerhalb einer Anwendung immer auf das gleiche Verzeichnis beziehen, ist es sinnvoll, dieses Verzeichnis an die eigenen Bedürfnisse anzupassen. Da trifft es sich gut, dass VBA mit **ChDir** auch eine Funktion zum Festlegen eines neuen aktuellen Verzeichnisses liefert. Sie können den Wert von **CurDir** beispielsweise auf den Pfad zur aktuell geöffneten Datenbank einstellen:

```
ChDir Currentproject.Path
```

Die anschließende Prüfung liefert das erwartete Ergebnis:

```
Debug.Print CurDir
C:\Daten\Fachartikel\AiUBasic\Basics\2012\06\Dateibefehle
```

CurDir liefert für jedes Laufwerk ein eigenes aktuelles Verzeichnis. Um das aktuelle Verzeichnis eines anderen Laufwerks als **c:** zu ermitteln, geben Sie den Laufwerksbuchstaben in Anführungszeichen als Parameter der **CurDir**-Funktion an:

```
? CurDir("h")
h:\Daten
```

Wenn Sie das aktuelle Verzeichnis für ein bestimmtes Laufwerk ändern möchten, geben Sie einfach den kompletten Pfad samt Laufwerksbuchstaben an:

```
ChDir("h:\Daten\Accessprojekte")
```

Wie die folgende Prüfung ergibt, funktioniert es:

```
? CurDir("h")
h:\Daten\Accessprojekte
```

CurDir liefert ohne Parameter standardmäßig das aktuelle Verzeichnis des Laufwerks **c:**. Wenn Sie möchten, dass **CurDir** ohne Parameter das aktuelle Verzeichnis eines anderen Laufwerks liefert, stellen Sie das Standardlaufwerk entsprechend mit der Funktion **ChDrive** ein:

```
ChDrive "h"
```

CurDir liefert dann Folgendes:

```
? CurDir
h:\Daten\Accessprojekte
```

Dateien löschen

Während die im Artikel **Textdateien** vorgestellte **Open**-Anweisung nicht nur zum Öffnen, sondern auch zum Anlegen von Textdateien genutzt werden kann, fehlt noch ein Pendant zum Löschen von Dateien – in diesem Fall nicht nur von Textdateien. Diese Methode heißt **Kill**. Diese Methode erwartet lediglich den Dateinamen als Parameter:

```
Kill CurrentProject.Path & "\test.txt"
```

Wenn Sie keinen Pfad angeben, verwendet auch die **Kill**-Methode das mit **CurDir** ermittelte Verzeichnis.

Verzeichnis erstellen

Neue Verzeichnisse erstellen Sie mit der Methode **MkDir**. Wenn Sie **MkDir** nur den Namen des zu erstellenden Verzeichnisses übergeben, also etwa wie im folgenden Aufruf, erzeugt VBA – Sie ahnen es bereits – das neue Verzeichnis im mit **CurDir** zu ermittelnden aktuellen Verzeichnis:

```
MkDir "Test"
```

Obiger Aufruf würde also das Verzeichnis **c:\Daten\Test** erzeugen, wenn **CurDir** das Verzeichnis **c:\Daten** liefert. Dabei können übrigens kaum Fehler auftreten – im Gegensatz zur Variante mit Angabe des Verzeichnisses. Wenn Sie beispielsweise zwei verschachtelte Unterverzeichnisse namens **Test1** und **Test2** unter **c:\Daten** erzeugen möchten, gelingt dies nicht in einem einzigen Schritt – Sie erhalten dann nämlich den Fehler aus Bild 1.

Stattdessen erledigen Sie diese Aufgabe in zwei Schritten:

```
MkDir "c:\Daten\Test1"
MkDir "c:\Daten\Test1\Test2"
```

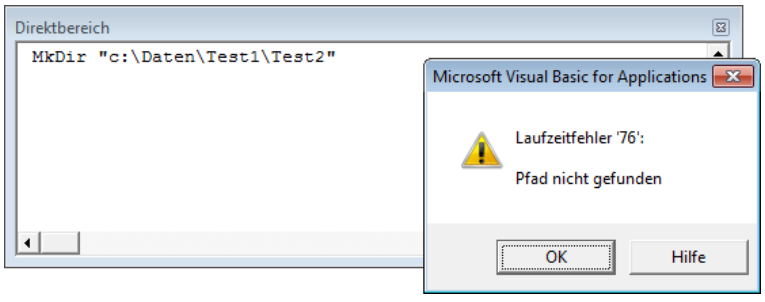


Bild 1: Fehler beim Versuch, zwei Verzeichnisse auf einmal zu erstellen

Löschen von Verzeichnissen

Das Löschen eines Verzeichnisses erledigen Sie mit der **RmDir**-Methode. Auch diese erlaubt nur das Löschen eines Verzeichnisses gleichzeitig, genau genommen sogar nur das Löschen leerer Verzeichnisse.

Wenn Sie die zuvor angelegten Verzeichnisse wie folgt löschen wollten, würde folgende Anweisung einen Laufzeitfehler **75**, **Fehler bei Zugriff auf Datei**, auslösen:

```
RmDir "c:\Daten\Test1\"
```

Sie benötigen also wieder zwei Anweisungen, wobei erst das zuletzt angelegte Verzeichnis entfernt werden muss:

```
RmDir "c:\Daten\Test1\Test2"
RmDir "c:\Daten\Test1\"
```

Den Fehler **75** erhalten Sie übrigens auch, wenn Sie versuchen, ein bereits vorhandenes Verzeichnis anzulegen.

Vorhandensein von Dateien und Verzeichnissen prüfen

Da ist es doch hilfreich, dass es eine Funktion gibt, mit der Sie das Vorhandensein von Dateien und Verzeichnissen prüfen können. Wenn Sie beispielsweise wissen wollen, ob das Verzeichnis **c:\Daten\Test1** bereits vorhanden ist, setzen Sie die folgende Anweisung ab:

```
? Dir("c:\Daten\Test1", vbDirectory)
Test1
```

Der zweite Parameter muss beim Prüfen eines Verzeichnisses unbedingt den Wert **vbDirectory** aufweisen, der erste den zu prüfenden Verzeichnisnamen ohne folgendes Backslash-Zeichen (\). Ist das Verzeichnis vorhanden, liefert **Dir** den Namen des gesuchten Unterverzeichnisses zurück.

Auf Laufwerk prüfen

Gegebenenfalls möchten Sie prüfen, ob ein Laufwerk verfügbar ist. Das ist insbesondere hilfreich, wenn Sie einen Dateinamen samt Laufwerk und Pfad in einer An-

Textdateien schreiben und lesen

VBA bietet einige einfache Methoden, um Textdateien zu erstellen, diese mit Texten zu füllen und auch die enthaltenen Texte zu lesen. Dieser Artikel stellt diese Methoden vor und erläutert, wie Sie eine Datei komplett oder zeilenweise lesen, wie Sie neue Dateien erstellen und mit Texten füllen oder Texte zu bestehenden Dateien hinzufügen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1206_Textdateien.mdb**.

Wozu Textdateien verwenden?

Auch wenn Sie mit Access auf strukturierte Dokumenttypen wie XML-Dokumente oder die Office-Dateitypen zugreifen können, hat die gute alte Textdatei dennoch ihre Berechtigung. Immerhin gibt es noch zahlreiche Anwendungen, die Ihre Daten beispielsweise zeilenweise im CSV-Format ausgeben – und dies lässt sich prima mit den Methoden zum Einlesen von Textdateien auswerten. Aus dem gleichen Grund ist auch das Erstellen und Beschreiben von Textdateien noch längst nicht aus der Mode gekommen.

Inhalt von Textdateien

Textdateien speichern in der Regel Buchstaben, Zahlen, Sonderzeichen und Steuerzeichen. ASCII legt für jedes dieser Zeichen eine Ganzzahl zur digitalen Darstellung fest. Da der ASCII-Standard weltweit verbreitet ist, kann man auch damit rechnen, dass eine im Textformat gespeicherte Datei überall dort gelesen werden kann, wo es auch Computer gibt.

Ein ASCII-Zeichen benötigt ein Byte Speicherplatz in der Textdatei. Mit einem Byte lassen sich Zahlen von 0 bis 255 darstellen. Die ersten 128 decken einen Standardzeichensatz ab, die übrigen stehen für Sonderzeichen und länderspezifische Zeichen wie etwa die deutschen Umlaute.

Kleine Buchstaben belegen die ASCII-Codes 97-122, große Buchstaben die Codes 65-90. Die Zahlen finden sich in den Codes 48-57.

Textdateien mit und ohne Zeilenumbruch

Es gibt zwei Möglichkeiten, Daten in einer Textdatei zu speichern: in Zeilen, die durch spezielle Steuerzeichen voneinander getrennt werden und in Zeilen fester Zeichenlänge. Im ersten Fall geben die beiden Zeichen **Line Feed** (**10**) und **Carriage Return** (**13**) den Zeilenumbruch an, im zweiten Fall wird jede Zeile bis zur angegebenen Zeichenanzahl aufgefüllt. Dieser Artikel stellt die Methoden zur ersten Variante vor.

Datei vorbereiten

Für die unterschiedlichen Zugriffsarten auf eine Textdatei gibt es verschiedene Parameter. Wie behandeln die folgenden Möglichkeiten:

- Schreiben in eine neue Datei
- Schreiben in eine vorhandene Datei
- Lesen aus einer Datei

Die Vorbereitung der Datei, also das Öffnen oder Anlegen, übernimmt in allen Fällen die **Open**-Anweisung. Die grundlegende Syntax sieht wie folgt aus:

```
Open <Pfadname> For <Modus> As <Dateinummer>
```

Angeben des Dateinamens

Sie können entweder den kompletten Dateinamen inklusive Laufwerk und Pfad angeben oder nur einen Dateinamen. Der erste Fall ist klar, im zweiten Fall ergänzt Access die Datei um das aktuelle, mit der Funktion **CurDir** auszulesende und mit **ChDir** zu setzende Verzeichnis (siehe auch Artikel **Dateibefehle**).

Je nach Öffnungsmodus muss die Datei bereits vorhanden sein (**Output**), anderenfalls wird diese gegebenenfalls beim Öffnen erstellt (**Input**, **Append**).

Öffnungsmodus

In Zusammenhang mit Textdateien gibt es drei verschiedene Modi zum Öffnen einer Datei:

- **Append**: Öffnet oder erstellt eine Datei zum Anhängen weiterer Inhalte
- **Input**: Öffnet oder erstellt eine Datei und fügt Inhalte ein, vorhandene Inhalte werden gelöscht

- **Output**: Öffnet eine bestehende Datei und bereitet diese zum Lesen der Inhalte vor.

Output und Input – wie soll man sich das merken? Ganz einfach: Gehen Sie immer von der VBA-Prozedur aus, die auf die Datei zugreift.

Liest diese die Daten aus der Textdatei ein, verwenden Sie den **Input**-Modus (Daten fließen zur Prozedur), sonst den **Output**-Modus (Daten fließen zur Datei).

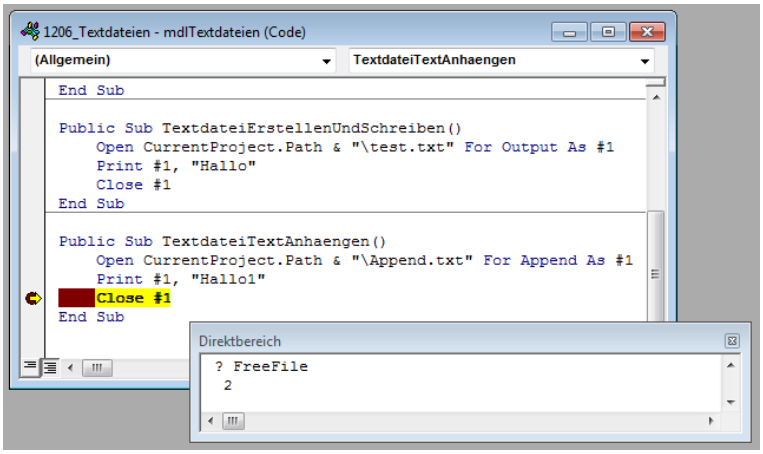


Bild 1: Aufruf von **FreeFile** während bereits eine Datei geöffnet ist

Dateinummer

Üblicherweise gibt man als Dateinummer einfach den Wert **#1** an. Dies kann jedoch in die Hose gehen, wenn die gleiche Access-Instanz an anderer Stelle bereits eine Datei mit dieser Nummer geöffnet hat.

Wenn es verschiedene Prozeduren innerhalb einer Anwendung gibt, die Dateien öffnen und bei denen es vorkommen kann, dass die eine Datei noch offen ist, während eine andere geöffnet werden soll, können Sie für jeden Ort innerhalb der Datenbank eine andere Dateinummer angeben, also **#1**, **#2** und so weiter.

Noch einfacher geht es, wenn Sie die Funktion FreeFile nutzen, um sich eine aktuell freie Dateinummer geben zu lassen. Normalerweise gibt **FreeFile** die kleinste verfügbare Dateinummer zurück, was sich einfach im Direktfenster prüfen lässt:

```
? FreeFile
 1
```

Ob diese Funktion funktioniert, können Sie prüfen, indem Sie zuvor eine Datei mit der Nummer **1** öffnen und dann nochmals **FreeFile** im Direktfenster aufrufen (siehe Bild 1).

Wozu benötigen Sie nun eine solche Dateinummer? Nun: Sie vergeben diese beim Öffnen oder Anlegen der Datei und müssen diese auch beiden folgenden Operationen angeben, damit VBA weiß, aus welcher Datei Daten gelesen oder in welche Datei Daten geschrieben werden sollen.

Wir schauen uns an einem einfachen Beispiel an, wie eine feste Dateinummer im Vergleich zu einer per **FreeFile** ermittelten Dateinummer eingesetzt wird.

Die folgende kleine Prozedur öffnet eine Datei zum Beschreiben und trägt eine Zeile mit Text ein. Dabei wird in jeder Anweisung die #1 als Dateinummer für den Zugriff verwendet:

```
Public Sub TextdateiErstellenUndSchreiben()
    Open CurrentProject.Path & "\test.txt" _
        For Output As #1
    Print #1, "Hallo"
    Close #1
End Sub
```

Mit der **FreeFile**-Funktion benötigen Sie eine Variable, welche die Dateinummer speichert, um diese für alle Zugriffe bereitzustellen – hier **intFile** genannt. Dieser Variablen weisen Sie den Wert der Funktion **FreeFile** zu, also beispielsweise die Zahl **2**. In den folgenden drei Anweisungen wird statt **1** der Variablenname **intFile** angegeben:

```
Public Sub TextdateiErstellenUndSchreiben_FreeFile()
    Dim intFile As Integer
    intFile = FreeFile
    Open CurrentProject.Path & "\test.txt" For _
        Output As #intFile
    Print #intFile, "Hallo"
    Close #intFile
End Sub
```

Sie können an allen drei Stellen das Raute-Zeichen beibehalten, aber Sie müssen es mit der **Print**-Anweisung verwenden – anderenfalls löst dies einen Fehler aus.

Damit haben Sie gleichzeitig schon eine einfache Variante zum Schreiben einer Zeile Text in eine Textdatei kennengelernt.

Lesen einer kompletten Textdatei

Wenn Sie den kompletten Inhalt einer Textdatei in eine Variable einlesen möchten, verwenden Sie eine Prozedur wie die folgende:

```
Public Sub TextdateiLesen()
    Dim strText As String
    Dim lngAnzahlZeichen As Long
    Open CurrentProject.Path & "\test.txt" _
        For Input As #1
    lngAnzahlZeichen = LOF(1)
    strText = Input(lngAnzahlZeichen, #1)
    Debug.Print strText
    Close #1
End Sub
```

Die Prozedur öffnet zunächst die Datei für den lesenden Zugriff. Die Datei muss unbedingt vorhanden sein, sonst löst die **Open**-Anweisung einen Fehler aus.

Nach dem Öffnen der Datei ermitteln Sie mit der Funktion **LOF** die Anzahl der enthaltenen Zeichen. Der Parameterwert **1** entspricht der verwendeten Dateinummer. Sollten Sie also mit einer mit **FreeFile** ermittelten Dateinummer arbeiten, ersetzen Sie **1** durch die Variable mit der Dateinummer.

Danach liest die Input-Methode aus der mit **#1** referenzierten Datei die ersten n Zeichen ein, wobei n dem Inhalt der Variablen **lngAnzahlZeichen** entspricht – in diesem Fall der kompletten Datei. Die **Debug.Print**-Anweisung gibt die ermittelte Zeichenkette mangels echtem Verwendungszweck im Direktfenster aus, bevor die **Close**-Anweisung die Datei wieder schließt.

Text an eine Textdatei anhängen

Die folgende Prozedur öffnet eine Textdatei im **Append**-Modus. Das bedeutet, dass die Datei entweder neu erstellt wird (sofern diese noch nicht existiert) oder dass eine bestehende Datei geöffnet wird. Dieser Datei hängt die Prozedur eine neue Zeile mit dem angegebenen Inhalt an:

```
Public Sub TextdateiTextAnhaengen()
    Open CurrentProject.Path & "\Append.txt" _
        For Append As #1
    Print #1, "Hallo"
    Close #1
End Sub
```

Wenn Sie diese Prozedur mehrfach aufrufen, finden Sie die Zeile mit dem Text **Hallo** entsprechend der Anzahl der Aufrufe in der Datei vor.

Datei zeilenweise einlesen

Sie können eine Datei auch zeilenweise einlesen. Dazu öffnen Sie die Datei im **Input**-Modus und prüfen dann mit der **EOF**-Funktion, ob das Ende der Datei erreicht ist. Die **EOF**-Funktion erwartet wiederum die Dateinummer als Parameter. Eine **Do While**-Schleife wird solange durchlaufen, bis **EOF(1)** den Wert **False** annimmt.

Innerhalb der Schleife verwenden Sie die **Line Input**-Anweisung mit der Dateinummer und einer String-Variablen als Parameter. Die String-Variable, hier **strLine**, nimmt dabei den Inhalt der aktuell eingelesenen Zeile entgegen. Nach dem Einlesen wird ein Zeiger auf die nächste Zeile verschoben – solange, bis der Zeiger sich hinter der letzten Zeile befindet und **EOF(1)** den Wert **False** liefert:

```
Public Function DateiZeilenweiseLesen()
    Dim strLine As String
    Open CurrentProject.Path & "\Append.txt" _
        For Input As #1
    Do While Not EOF(1)
        Line Input #1, strLine
        Debug.Print strLine
    Loop
    Close 1
End Function
```

Wenn Sie beispielsweise nur auf die erste Zeile zugreifen möchten, können Sie dies noch einfacher erledigen:

```
Public Function DateiErsteZeileLesen()
    Dim strLine As String
    Open CurrentProject.Path & "\Append.txt" _
        For Input As #1
    If Not EOF(1) Then
        Line Input #1, strLine
        Debug.Print strLine
    End If
    Close #1
End Function
```

Datei zeichenweise einlesen

Sie können eine Datei auch zeichenweise einlesen. Dazu verwenden Sie eine Prozedur wie die folgende:

Rechnungsbericht

In Ausgabe 2/2012 haben wir bereits Tabellen und Formulare zum Verwalten von Bestellungen vorgestellt. Nun liefern wir einen Bericht nach, mit dem Sie pro Bestellung eine eigene Rechnung erstellen können. Mit einer Rechnung treffen wir dabei auf ein typisches Beispiel für den Einsatz von Berichten.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1206_Rechnungsbericht.mdb**.

Aufbau des Berichts

Der Bericht hat zunächst folgenden Aufbau, später gehen wir etwas mehr ins Detail: Der Berichtskopf soll den Briefkopf des Absenders inklusive Name und Anschrift sowie die Anschrift des Rechnungsempfängers enthalten. Der Detailbereich enthält die einzelnen Rechnungspositionen, also die Position, die Artikelbezeichnung, die Menge, den Einzelpreis, die Mehrwertsteuer, den Rabatt sowie den Endpreis. Der Berichtsfuß enthält allgemeine Informationen wie etwa eine Grußformel. Fehlen noch die Summen mit und ohne Mehrwertsteuer: Diese landen in einem eigenen Bereich, der durch die Gruppierung nach der Bestellnummer erzeugt wird. Nach der Fertigstellung soll der Bericht wie in Bild 2 aussehen.

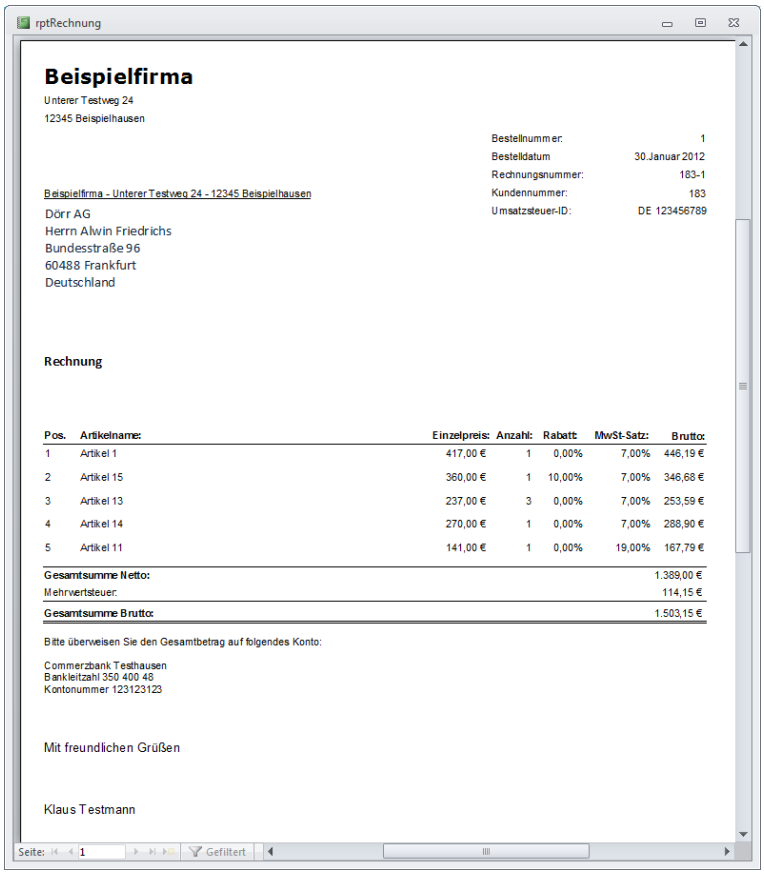


Bild 2: Der Bericht in der Vorschauansicht

Aufruf des Berichts

Auch wenn wir den Bericht noch gar nicht programmiert haben: Von irgendeiner Stelle aus müssen Sie diesen später aufrufen. Im Rahmen einer Bestellverwaltung macht es natürlich Sinn, diesen Bericht von dem Formular aus aufzurufen, das auch die Bestelldaten anzeigt. Dieses Formular haben wir bereits im Artikel **Bestellungen in Formularen verwalten** programmiert, die Daten zum Bericht stammen aus Tabellen, die in **Bestellungen in Tabellen verwalten** beschrieben wurden. Wir greifen das Formular auf und fügen diesem zwei Schaltflächen hinzu, mit dem der Benutzer den Bericht in der Vorschauansicht betrachten und mit der er den Bericht auch gleich drucken kann (siehe Bild 1). Die beiden Schaltflächen heißen **cmdRechnungVorschau** und **cmdRechnungDrucken**. Für die Schaltfläche **cmdRechnungVorschau** hinterlegen Sie die folgende Ereignisprozedur:

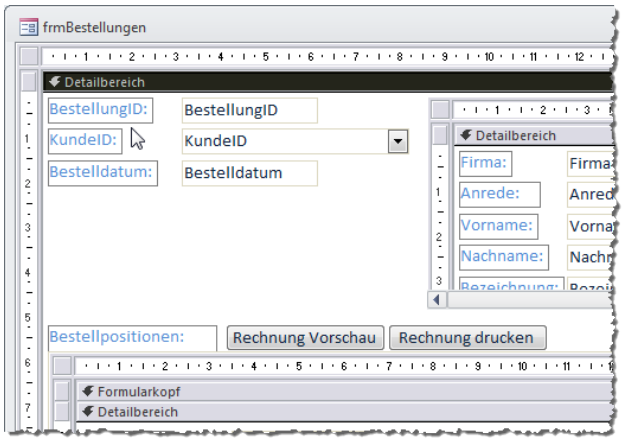


Bild 1: Aufruf des Berichts vom Bestellformular aus

```
Private Sub cmdRechnungVorschau_Click()
    DoCmd.OpenReport "rptRechnung", _
        WhereCondition:="BestellungID = " _
        & Me!BestellungID, View:=acViewPreview
End Sub
```

Diese **OpenReport**-Anweisung öffnet einen Bericht namens **rptRechnung**, wobei dieser nur solche Datensätze anzeigen soll, deren Feld **BestellungID** den Wert des gleichnamigen Feldes der Datenherkunft des Formulars aufweist. Außerdem übergeben Sie für den Parameter **View** den Wert **acViewPreview**, wodurch der Bericht in der Vorschauansicht geöffnet wird.

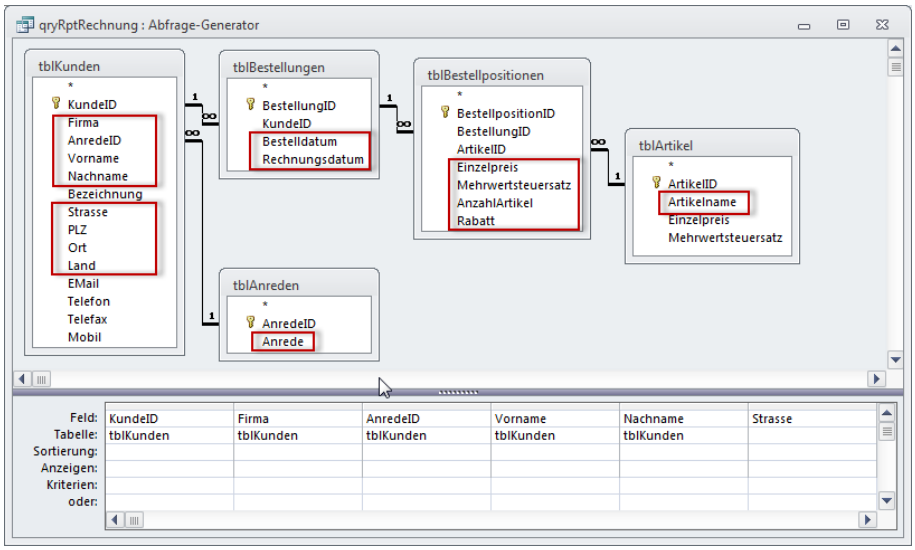


Bild 3: Datenherkunft des Berichts

Die einzige Anweisung der Ereignisprozedur für die zweite Schaltfläche, **cmdRechnungDrucken**, unterscheidet sich nur durch den Wert des Parameters View von der oben beschriebenen Prozedur. Hier verwenden wir den Wert **acViewNormal**, was den Bericht gleich an den Standarddrucker sendet.

Änderungen am Datenmodell

Da wir nun nicht mehr nur Bestellungen aufnehmen, sondern auch noch Rechnungen schreiben, haben wir der Tabelle **tblBestellungen** noch ein Feld namens **Rechnungsdatum** hinzugefügt.

Bericht anlegen

Erstellen Sie einen neuen Bericht und speichern Sie diesen unter dem Namen **rptRechnung**.

Datenherkunft

Wir wissen schon einmal, dass die Datenherkunft des Berichts das Feld **BestellungID** enthalten muss, damit wir die Bestelldaten nach der gewünschten Bestellung filtern können. Der Bericht enthält jedoch insgesamt sogar fünf Tabellen, die Sie in Bild 3 abgebildet sehen. Zur besseren Übersicht haben wir die Felder eingerahmt, die wir zum Entwurfsraster hinzugefügt haben. Um die Datenherkunft festzulegen, öffnen Sie den Bericht in der Entwurfsansicht und zeigen Sie mit **F4** die Berichtseigenschaften an. Klicken Sie auf die Eigenschaft **Datenherkunft** (ab Access 2007 **Datensatzquelle**) und dann auf die Schaltfläche mit den drei Punkten. Es erscheint eine leere Abfrage, die Sie wie in der Abbildung füllen. In der Beispieldatenbank finden Sie die Abfrage übrigens unter dem Namen **qryRepRechnung** als eigenes Objekt im Datenbankfenster beziehungsweise im Navigationsbereich. Dies hat keinen tieferen Sinn – außer, dass dadurch die Gestaltung der Abfrage, also etwa die Anordnung der Tabellen und ihrer Beziehungen sowie der Spaltenbreiten, auch nach dem Schließen und erneuten Öffnen der Abfrage erhalten bleibt.

Bericht mit Steuerelementen füllen

Mit diesen Informationen ausgestattet, können wir uns an den Berichtsentwurf begeben. Dabei unterteilen wir die folgenden Schritte in Pflicht und Kür. Die Pflicht sind die Angabe der Empfängeranschrift, der Rechnungspositionen, des Rechnungsdatums, der Rechnungssummen sowie diverser Nummern wir Rechnungsnummer, Kundennummer, Bestellnummer et cetera. Die Kür machen Elemente wie der Briefkopf, die rechte Seite mit allgemeinen Angaben sowie der Berichtsfuß aus, wo Sie Bankverbindung, Grußformel und andere Dinge nach Gusto unterbringen können.

Als erstes fügen wir der Rechnung die Rechnungsanschrift hinzu. Diese wird, wie bereits im Artikel **Adressblöcke in Berichten** beschrieben, als ein Textfeld angelegt, das seine Informationen aus einer Formel bezieht. Diese Formel lautet wie folgt:

```
=Wenn([Firma];[Firma];Wenn([Anrede]="Herr";"He
rrn";"Frau")) & Zchn(13) & Zchn(10) & Wenn([Fi
rma];Wenn([Anrede]="Herr";"Herrn";"Frau"))+" "
& [Vorname]+" " & [Nachname]+Zchn(13)+Zchn(10)
& [Strasse]+Zchn(13)+Zchn(10) & [PLZ]+"
"+[Ort]+Zchn(13)+Zchn(10) & [Land]
```

Kurz zusammengefasst prüft diese Formel zunächst, ob für diesen Kunden ein Eintrag im Feld **Firma** vorliegt. Falls ja, landet die Firma in der ersten Zeile und der Kundenname samt Anrede in der zweiten. Anderenfalls wird die Anrede in die erste und der Kundenname in der zweiten Zeile angelegt. Die übrigen Zeilen werden mit Straße, PLZ, Ort und Land gefüllt. Sollte ein Feld leer sein, verhindert die Formel eventuell auftretende leere Zeilen und überflüssige Leerzeichen. Das Textfeld zur Anzeige der Anschrift landet im Berichtskopf. Dort fügen Sie außerdem noch nach eigenen Vorlieben den Briefkopf mit Firmenname und -sitz sowie weitere Felder wie Bestellnummer, Bestelldatum, Rechnungsnummer, Kundennummer und Ust.-IdNr. ein. Bestelldatum, Bestellnummer und Kundennummer holen Sie sich aus der Datenherkunft des Berichts, die Rechnungsnummer setzen wir aus Kundennummer und Bestellnummer zusammen. Dazu fügen Sie ein Textfeld namens **txtRechnungsnummer** hinzu und hinterlegen den folgenden Ausdruck für seine Eigenschaft **Steuerelementinhalt**:

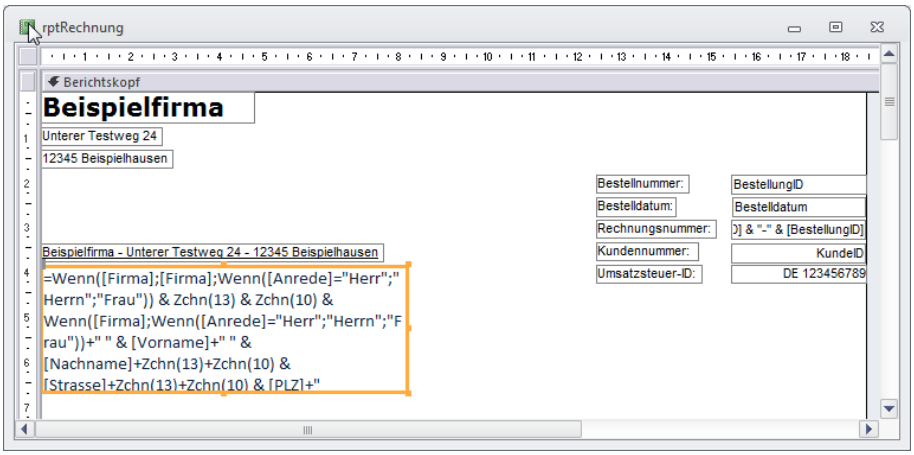


Bild 4: Kopfbereich des Berichts mit Anschrift

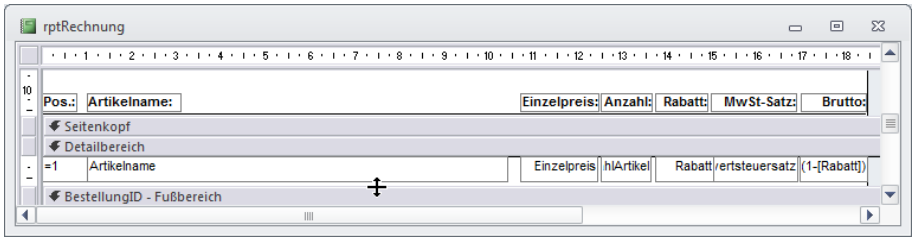


Bild 5: Detailbereich des Berichts mit den Rechnungspositionen

```
=[KundeID] & "-" & [BestellungID]
```

Der Kopf des Berichts sieht somit wie in Bild 4 aus.

Rechnungspositionen

Als nächstes kümmern wir uns um die Rechnungspositionen, die den Datensätzen der Tabelle **tblBestellpositionen** entsprechen. Wir zeigen nur eine Rechnung pro Bericht an, also sind die Rechnungspositionen die einzigen Elemente, die sich wiederholen könnten. Folgerichtig landen die Felder mit den einzelnen Daten der Rechnungspositionen im Detailbereich des Berichts.

Der Detailbereich befindet sich direkt unterhalb des Berichtskopfes. Damit die Überschriften für Position, Artikelname, Einzelpreis, Anzahl, Rabatt, Mehrwertsteuersatz und Bruttopreis nur einmal angezeigt werden, fügen Sie diese am unteren Rand des Berichtskopfes ein. Der Detaibereich nimmt die Felder **Artikelname**, **Einzelpreis**, **AnzahlArtikel**, **Rabatt** und **Mehrwertsteuersatz** auf (siehe Bild 5). Außerdem fügen Sie ganz vorne ein Feld namens **txtPosition** hinzu und ganz hingen ein Feld namens **txtBrutto** hinzu.

Das Feld **txtBrutto** soll den Gesamtbruttopreis für die aktuelle Position anzeigen, also das Produkt aus dem Einzelpreis, der Anzahl, der Mehrwertsteuer und dem Rabatt. Die dazu benötigte Formel tragen Sie als Steuerelementinhalt für das Textfeld **txtBrutto** ein:

```
=[Einzelpreis]*(1+[Mehrwertsteuersatz])*(1-
[Rabatt]*[AnzahlArtikel])
```

Interessanter wird es noch, wenn es um das Füllen des Steuerelements **txtBrutto** geht. Dieses soll die Positionen durchnummerieren, damit Sie eine Position bei eventuellen Rückfragen des Kunden direkt referenzieren können. Dazu verwenden wir einen kleinen Trick: Legen Sie als **Steuerelementinhalt** des Feldes **txtPosition** den Wert **1** fest und für die Eigenschaft **Laufende Summe** den Eintrag **Über Gruppe**. Access berechnet somit die Summe der Werte des Steuerelements **txtPosition** vom ersten bis zum aktuellen Datensatz. Da der Wert immer **1** ist, liefert die laufende Summe für die erste Position den Wert

Die Dir-Funktion

Die Dir-Funktion ist ein echter Tausendsassa: Sie ermöglicht sowohl das Prüfen des Vorhandenseins von Dateien als auch das Ermitteln von Dateien innerhalb eines Verzeichnisses. Dabei gilt es jedoch, einige Klippen zu umschiffen: Anderenfalls treten schnell nicht erklärbare Fehler in Ihrer Datenbank auf. Dieser Beitrag zeigt, wie Sie die Dir-Funktion für verschiedene Anwendungszwecke einsetzen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1207_Dir.mdb**.

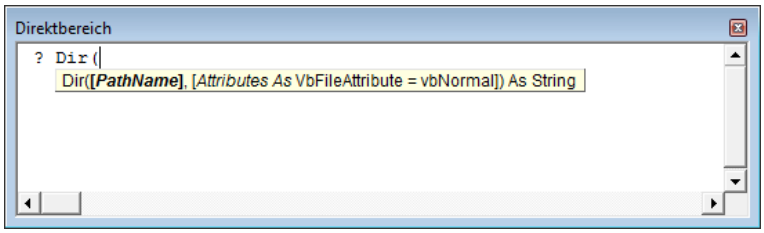


Bild 1: IntelliSense zeigt die Parameter als optional an

Vorhandensein von Verzeichnissen oder Dateien prüfen

Ein einfacher Aufruf der **Dir**-Funktion, beispielsweise im Direktfenster des VBA-Editors abgesetzt (zu öffnen mit **Strg + G**, siehe Bild 1), löst einen Fehler aus:

```
? Dir()
```

Und das, obwohl IntelliSense die Parameter dieser Funktion bei der Eingabe im Direktfenster in eckigen Klammern anzeigt, was schlicht bedeutet, dass es sich dabei um optionale Parameter handelt (siehe Bild 1).

Der Grund ist, dass diese Funktion zumindest beim ersten Aufruf einen Ausdruck für den ersten Parameter erwartet. Hier geben Sie im einfachsten Fall eine leere Zeichenkette an:

```
? Dir("")
```

Dies liefert die erste Datei des aktuellen Verzeichnisses. Welches Verzeichnis hiermit gemeint ist, erfahren Sie durch den Aufruf der Funktion **CurDir**. Dies liefert beispielsweise folgendes Ergebnis:

```
? CurDir
C:\Users\Andre\Documents
```

Auf das in **CurDir** hinterlegte Verzeichnis sollten Sie sich jedoch nicht verlassen, sondern stattdessen explizit angeben, auf welches Verzeichnis oder auf welche Datei sich die **Dir**-Funktion beziehen soll (Sie können natürlich auch die **ChDir**-Funktion nutzen, um das aktuelle Verzeichnis festzulegen).

Wenn Sie prüfen möchten, ob ein Verzeichnis oder eine Datei existiert, geben Sie den Verzeichnis- oder Dateinamen inklusive Pfad als Parameter an. Um zum Beispiel herauszufinden, ob eine Datei existiert, geben Sie den kompletten Pfad wie folgt an:

```
? Dir("c:\Test.txt")
```

Sollte es eine entsprechende Datei im Verzeichnis **c:** geben, liefert diese Funktion den Namen der Datei ohne Pfad als Ergebnis, also etwa **Text.txt**.

Wenn Sie hingegen die Existenz eines Verzeichnisses prüfen möchten, müssen Sie zusätzlich den Wert **vbDirectory** für den zweiten Parameter der **Dir**-Funktion übergeben:

```
? Dir(CurrentProject.Path, vbDirectory)
VBA_Dir
```

Da **CurrentProject.Path** in diesem Fall den Wert **C:\Daten\Fachartikel\AiUBasic\Basics\2012\07\VBA_Dir** enthält, liefert die Funktion den Namen des aktuellen Verzeichnisses zurück, also **VBA_Dir**.

Sie können **vbDirectory** allerdings auch angeben, wenn Sie das Vorkommen eines Dateinamens prüfen möchten – dies untersucht sowohl Datei- als auch Verzeichnisnamen.

Dateien und Verzeichnisse ermitteln

Etwas anders sieht die Vorgehensweise aus, wenn Sie die Dateien und Verzeichnisse innerhalb eines vorgegebenen Verzeichnisses ermitteln möchten.

Um etwa die erste gefundene Datei im Verzeichnis **c:\Beispiel** zu ermitteln, müssen Sie den Verzeich-

Binärzahlen in VBA-Eigenschaften

Jede VBA-Eigenschaft enthält oder erwartet nur einen einzigen Wert. Manchmal handelt es sich dabei jedoch um Zahlenwerte, die mehr als einen Eigenschaftswert repräsentieren. Ein gutes Beispiel dafür ist die MsgBox-Anweisung, für die Sie mit einem einzigen Zahlenwert gleichzeitig die anzuzeigenden Schaltflächen und das Symbol festlegen. Diese Festlegung ist einfach – schwieriger wird es, wenn Sie solche Eigenschaften auswerten müssen. Dieser Artikel liefert das notwendige Know-how zu diesem Thema.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1207_Binaerzahlen.mdb**.

Was sind Binärzahlen?

Im Gegensatz zu den hierzulande gebräuchlichen Dezimalzahlen, die mit Ziffern von 0 bis 9 zusammengesetzt werden, gibt es im Binärsystem nur die beiden Zahlen 0 und 1. Wie im Dezimalsystem entscheidet die Stelle über die Wertigkeit des jeweiligen Teils. **123** entspricht einem Hunderter, zwei Zehnern und drei Einern. Im Binärsystem gibt es Zahlen wie **1101**. In diesem Fall heißt dies: Ein Achter, ein Vierer, kein Zweier und ein Einer, in Summe: 13. Jede Stelle wird also gegenüber der Stelle rechts davon nicht um den Faktor 10, sondern lediglich um den Faktor 2 erhöht.

Was soll man als Access-Entwickler damit anfangen? Die Frage ist berechtigt, denn in den meisten Fällen werden solche Zahlenwerte als Parameter für VBA-Funktionen verwendet oder als Ergebnisse von Eigenschaften geliefert – und zwar gut kaschiert.

Binärzahlen als Parameter der MsgBox-Anweisung

Ein gutes Beispiel ist die **MsgBox**-Anweisung (siehe Artikel **Meldungsfenster anzeigen und auswerten**). Diese Funktion erwartet als zweiten Parameter Informationen darüber, welche Schaltflächen es anzeigen soll, welches Icon erscheint und welche Schaltfläche als Standardschaltfläche markiert wird.

Ein Beispiel ist die folgende Anweisung:

```
MsgBox "Beispielmeldung", vbOKCancel + vbCritical + vbDefaultButton2
```

Wie nur ermittelt VBA nun, welche Schaltflächen und Icons angezeigt werden und welche Schaltfläche als Standardschaltfläche dient? Es ist ganz einfach: Jeder der Ausdrücke, also **vbOKCancel**, **vbCritical** oder **vbDefaultButton2**, entspricht einem Zahlenwert. Durch den **Plus**-Operator **(+)** werden die Zahlenwerte addiert und später von VBA wieder auseinandergenommen. Damit dies funktioniert, dürfen die Zahlenwerte jedoch nicht willkürlich eingesetzt werden.

Die Zahlenwerte hinter **vbDefaultButton1** bis **vbDefaultButton4** lauten beispielsweise **0**, **256**, **512** und **768**. Die Schaltflächen geben Sie mit diesen Werten an:

- **vbOKOnly**: **0**
- **vbOKCancel**: **1**
- **vbAbortRetryIgnore**: **2**
- **vbYesNoCancel**: **3**
- **vbYesNo**: **4**
- **vbRetryCancel**: **5**

Fehlen noch die Icons, die durch folgende Konstanten angegeben werden:

- **vbCritical**: **16**
- **vbQuestion**: **32**
- **vbExclamation**: **48**
- **vbInformation**: **64**

Wenn Sie nun etwa ein Meldungsfenster mit den Schaltflächen **OK** und **Abbrechen**, der zweiten Schaltfläche als Standardschaltfläche und dem Icon mit dem Fragezeichen (**vbQuestion**) anzeigen möchten, verwenden Sie als zweiten Parameter diesen Ausdruck:

```
vbOKCancel + vbDefaultButton2 + vbQuestion
```

Dies entspricht wiederum den folgenden Zahlenwerten:

```
1 + 256 + 32
```

Sie könnten also auch schreiben:

```
MsgBox "Beispielmeldung", 289
```

Statt dem Plus-Zeichen können Sie auch den Operator **Or** verwenden:

```
vbOKCancel Or vbDefaultButton2 Or vbQuestion
```

Das Plus-Zeichen hat die gleiche Bedeutung wie **Or**? Wäre hier nicht eher **And** das passende Pendant? Nein – dies können Sie leicht prüfen. Wenn Sie das Ergebnis von der Variante mit **Or** im Direktfenster ausgeben, erhalten Sie dieses Ergebnis:

```
? vbOKCancel Or vbDefaultButton2 Or vbQuestion
 289
```

Die **And**-Variante liefert hingegen:

```
? vbOKCancel And vbDefaultButton2 And vbQuestion
 0
```

Was führt zu diesen Ergebnissen für **Or** und **And**? Nun: **Or** steht für eine logische Disjunktion. Hier wird jede Position der binären Zahlen miteinander verglichen und mit **Or** verknüpft. Dabei gilt:

- **1 Or 1= 1**
- **1 Or 0 = 1**
- **0 Or 1= 1**
- **0 Or 0 = 0**

Um dies für das obige Beispiel umzusetzen, berechnen Sie zunächst die Binärzahlen:

1 + 32 + 256 entspricht somit **1 + 100000 + 100000000**. Dies ergibt **100100001**, umgerechnet **289**.

Die And-Verknüpfung wiederum entspricht einer logischen Konjunktion, das heißt, dass nur solche Bits erhalten bleiben, die in allen verknüpften Zahlen vorkommen. Hier lauten die Regeln:

- **1 And 1= 1**
- **1 And 0 = 0**
- **0 And 1= 0**
- **0 And 0 = 0**

Deshalb ergibt auch **1 + 32 + 256**, also **1 + 100000 + 100000000**, den Wert **0**.

Nun wissen Sie, wie sich der Wert für die Konfiguration des Meldungsfensters zusammensetzt. Wie aber ermittelt VBA, welche Schaltflächen und welche Icons anzuzeigen sind?

Dazu wird der mit **Or** zusammengestellte Wert per **And**-Operator mit allen verfügbaren Einzelwerten verglichen. Das einfachste Beispiel sind die Schaltflächen. VBA wird etwa wie folgt vorgehen und nacheinander alle Werte von **0** bis **5** mit dem übergebenen Wert vergleichen. Angenommen, der zweite Parameter hat den Wert **1**, was der Konstanten **vbOKCancel** entspricht.

Die folgende Beispielfunktion verdeutlicht dies:

```
Public Function SchaltflaechenAuswerten( _
        intParameter As VbMsgBoxStyle) As Integer
    Dim intTemp As Integer
    If (intParameter And 1) = 1 Then _
        intTemp = intTemp + 1
    If (intParameter And 2) = 2 Then _
        intTemp = intTemp + 2
    If (intParameter And 3) = 3 Then _
        intTemp = intTemp + 3
    If (intParameter And 4) = 4 Then _
        intTemp = intTemp + 4
    If (intParameter And 5) = 5 Then _
        intTemp = intTemp + 5
    If intTemp > 5 Then intTemp = 0
    SchaltflaechenAuswerten = intTemp
End Function
```

Die Prozedur nimmt eine Konstante für die Meldungsfensterkonfiguration entgegen, also beispielsweise **vbOKCancel** mit dem Wert **1**. Sie verarbeitet aber auch Parameter wie **vbOKCancel Or vbDefaultButton2** (mit dem Wert **257**). Beim Durchlaufen der Prozedur wird die erste **If...Then**-Bedingung erfüllt, denn **1** ist binär in **257** enthalten. Die übrigen Bedingungen werden nicht erfüllt und die Funktion gibt den Wert **1** für die zu verwendende Kombination von Schaltflächen zurück. Wenn der Benutzer versehentlich **vbOKCancel** und **vbAbortRetryIgnore** angibt (Wert **1 + 2 = 3**), interpretiert die Funktion dies genau wie die **MsgBox**-Funktion als **vbYesNoCancel** (Wert **3**).

Kombinations- und Listenfelder mit Wertliste füllen

Neben der Zuweisung einer Tabelle oder Abfrage als Datensatzherkunft können Sie Kombinations- und Listenfelder auch mit einfachen Wertlisten füllen. Dabei handelt es sich um Zeichenketten, die durch Semikola voneinander getrennt werden. Dieser Artikel zeigt, wie Sie Kombinations- und Listenfelder mit Wertlisten füllen und was Sie dabei beachten müssen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1207_Kombinationsfeld-Wertliste.mdb**.

Einfache Wertlisten

Wenn Sie in einem Formular ein Kombinationsfeld angelegt haben, dass schlicht und einfach wenige Einträge zur Auswahl anbieten soll, können Sie diese Werte mithilfe einer Wertliste zuweisen. Dazu stellen Sie einfach zwei Eigenschaften des Kombinationsfeldes wie in Bild 1 ein. Für die Eigenschaft **Herkunftsart** wählen Sie den Eintrag **Wertliste** aus und für die Eigenschaft **Datensatzherkunft** tragen Sie etwa den folgenden Ausdruck ein:

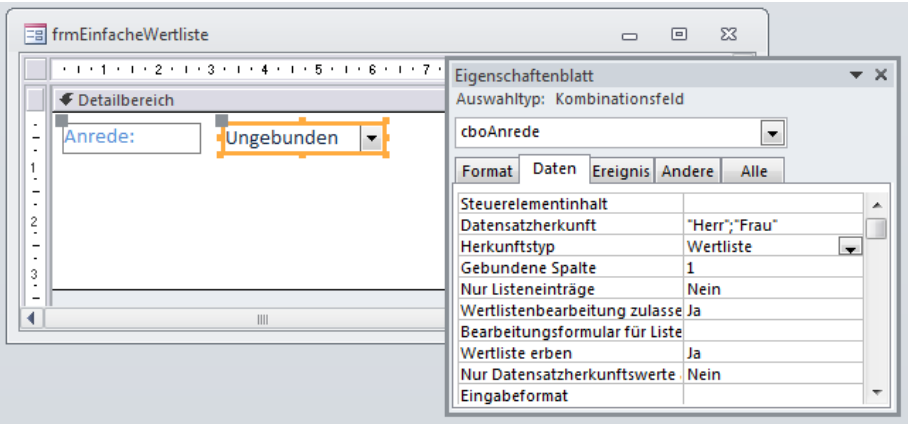


Bild 1: Kombinationsfeld mit einer einfachen Wertliste als Datensatzherkunft

```
"Herr";"Frau"
```

Das Ergebnis sieht dann etwa wie in Bild 2 aus. Wenn Sie den Inhalt des Kombinationsfeldes auswerten, erhalten Sie etwas mit folgenden Ausdruck über das Direktfenster den gewählten Eintrag:

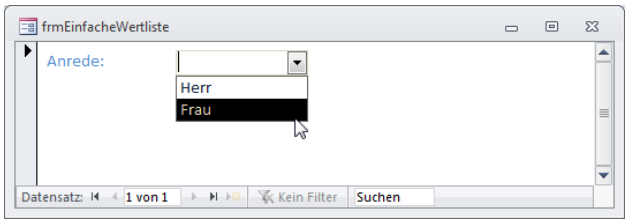


Bild 2: Auswahl eines Eintrags der Wertliste

```
? Forms!frmEinfacheWertliste!cboAnrede
Herr
```

Mehrspaltige Wertlisten

Nun möchten Sie vielleicht einen nicht angezeigten, aber dem gewählten Eintrag entsprechenden Ausdruck verwenden (siehe Kombinationsfeld **cboAnredeMitID** im Beispielformular **frmEinfacheWertliste**).

Der angezeigte Ausdruck **Herr** soll beispielsweise den Wert **1**, der Ausdruck **Frau** den Wert **2** liefern. Dazu muss das Kombinationsfeld zwei Spalten enthalten, von denen die erste die gebundene, nicht angezeigte Spalte ist.

Dazu verwenden Sie ein ähnliches Kombinationsfeld, allerdings mit etwas anderen Eigenschaften. Zunächst stellen Sie die Eigenschaft **Spaltenanzahl** auf **2** und die Eigenschaft **Spaltenbreiten** auf **0cm** ein. Dadurch zeigt das Kombinationsfeld seine Daten nun in zwei Spalten an, wobei die erste jedoch eine Breite von **0cm** aufweist – die zweite nimmt dadurch automatisch den übrigen Platz ein.

Fehlt noch die Eingabe der Zahlenwerte für die Wertliste. Dazu weisen Sie der Eigenschaft **Datensatzherkunft** den folgenden Ausdruck zu:

```
1;"Herr";2;"Frau"
```

Da die Eigenschaft **Spaltenbreiten** den Wert **2** enthält, liest Access automatisch jeweils zwei Werte für jede Zeile ein, also die ersten beiden für die erste Zeile und die letzten beiden für die zweite Zeile. Die Zahlenwerte werden jeweils mit der Breite **0cm** angezeigt und somit ausgeblendet, die Anreden werden weiterhin angezeigt.

Kombinations- und Listenfelder per Callback füllen

Es gibt verschiedene Arten, Kombinations- und Listenfelder mit Daten zu füllen: Sie können diese an Tabellen oder Abfragen binden, einfach mit einer Wertliste füllen oder die Namen der Felder einer Tabelle darin anzeigen. Es gibt jedoch noch eine weitgehend unbekannte Methode, die Nachteile der übrigen Möglichkeiten ausbügelt, aber auch etwas aufwendiger zu implementieren ist. Dabei kommt eine sogenannte Callback-Funktion zum Einsatz, welche die anzuzeigenden Daten zum Kombinationsfeld hinzufügt.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1207_KombinationsfeldCallback.mdb**.

Kombinationsfelder und Listenfelder füllen

Der Optimalfall beim Füllen eines Kombinationsfeldes ist das Vorhandensein einer Tabelle oder Abfrage als Datenherkunft. In diesem Fall behalten Sie den Wert **Tabelle/Abfrage** für die Eigenschaft **Herkunftstyp** bei.

Dies ist jedoch nicht immer gegeben: Wenn Sie beispielsweise eine Liste aller in einem Verzeichnis enthaltenen Dateien in einem Kombinationsfeld anzeigen wollen, liegen die Daten zunächst nicht in einer Tabelle vor. Nun gibt es die Möglichkeit, die Daten einfach in Form einer Wertliste aneinanderzureihen.

Dazu fügen Sie alle Dateinamen durch Semikola voneinander getrennt zu einer Zeichenkette zusammen, stellen die Eigenschaft **Herkunftsart** des Kombinationsfeldes auf **Wertliste** ein und weisen die Zeichenkette per VBA der Eigenschaft **RowSource** zu. Dies hat allerdings den Nachteil, dass die Wertliste eines Kombinationsfeldes nicht beliebig lang sein darf – genau genommen sind es weniger als 40.000. Welche Anzahl genau möglich ist, können Sie mit einem kleinen Testformular ermitteln. Dieses finden Sie in der Beispieldatenbank unter dem Namen **frmLangeWertliste**. Beim Laden löst das Formular die folgende Prozedur aus:

```
Private Sub Form_Load()
    Dim l As Long
    For l = 1 To 100000
        On Error Resume Next
        Me!cboLangeWertliste.RowSource = _
            String(l, "a")
        If Not Err.Number = 0 Then
            MsgBox "Maximal " & l & " Zeichen."
            Exit Sub
        End If
    Next l
End Sub
```

Die Prozedur fügt schlicht und einfach eine immer längere Zeichenkette für die Eigenschaft **Wertliste** ein – bis dies irgendwann einen Fehler auslöst. Auf dem Testsystem lieferte dies beispielsweise eine zulässige Länge von 32.751 Zeichen. Übrigens: Unter älteren Access-Versionen war die Anzahl der Zeichen auf 2.048 festgelegt – dort war der Einsatz von Callback-Funktionen noch wesentlich öfter gefragt.

Wenn Sie nun den Inhalt von Verzeichnissen mit vielen Dateien darstellen möchten, kann dies einen Fehler auslösen, wenn die Anzahl der erlaubten Zeichen für die Eigenschaft **RowSource** überschritten wird.

Liste per Callback-Funktion füllen

Die in diesem Artikel vorgestellte Funktion verwendet eine weitere Alternative, bei der statt eines der verfügbaren Werte **Tabelle/Abfrage**, **Wertliste** oder **Feldliste** einfach der Name einer VBA-Funktion als Wert der Eigenschaft **Herkunftsart** angegeben wird. Die Angabe eines Wertes für die Eigenschaft **Datensatzherkunft** ist in diesem Fall nicht nötig.

Die verwendete Callback-Funktion hat einen speziellen Aufbau, der wie folgt aussieht:

```
Function DateienEinlesen(ctl As Control, _
        lngID As Long, lngRow As Long, _
        lngColumn As Long, intCode As Integer) _
        As Variant
    Select Case intCode
        Case acLBInitialize
            DateienEinlesen = True
        Case acLBOpen
            DateienEinlesen = Timer
        Case acLBGetRowCount
            DateienEinlesen = 1
        Case acLBGetColumnCount
            DateienEinlesen = 1
        Case acLBGetColumnWidth
            DateienEinlesen = -1
        Case acLBGetValue
            DateienEinlesen = "Test"
        Case acLBGetFormat
```

```
            DateienEinlesen = Null
        Case Else
            Debug.Print intCode
    End Select
End Function
```

Dies wirkt auf den ersten Blick etwas unübersichtlich, was sich aber gleich noch ändern wird – zu Ungunsten der Übersicht, denn die obige Variante enthält nur die einfachste Variante der Prozedur, die lediglich einen einzigen Wert in das Ziel-Listenfeld schreibt. Schauen wir uns jedoch zunächst den groben Aufbau an.

Die Funktion wird mehrfach aufgerufen und soll dabei verschiedene Anweisungen ausführen. Diese Anweisungen tragen Sie unter den entsprechenden Verzweigungen der **Select Case**-Bedingung ein. Welcher Teil der **Select Case**-Bedingung ausgeführt wird, entscheidet der beim Aufruf übergebene Wert für den Parameter **intCode**. Dieser kann einen Zahlenwert annehmen, der einer der folgenden Konstanten entspricht:

- **acLBInitialize**: Dieser Wert für **intCode** wird beim ersten Aufruf der Callback-Funktion übergeben. Hier wird geprüft, ob das Kombinations- oder Listenfeld überhaupt mit dieser Funktion gefüllt werden soll. Falls ja, muss der Wert **True** als Rückgabewert der Funktion festgelegt werden.

- **acLBOpen**: Beim Aufruf der Funktion mit dem Wert **acLBOpen** für den Parameter **intCode** erwartet Access eine Zahl als Rückgabewert, die das Steuerelement, von dem aus die Callback-Funktion aufgerufen wurde, eindeutig identifiziert. Hintergrund ist, das man ja theoretisch mehrere Steuerelemente mit der gleichen Callbackfunktion ausstatten kann. Wenn sichergestellt ist, dass nur ein Steuerelement diese Funktion als Wert für die Eigenschaft **Herkunftsart** verwendet, können Sie hier einfach den Wert **1** als Funktionswert angeben. Anderenfalls geben Sie das Ergebnis der Funktion **Timer** zurück, welche die Zeit in Millisekunden berechnet – dies geht nur schief, wenn die Funktion zweimal in der gleichen Millisekunde aufgerufen wird, was unwahrscheinlich ist.

- **acLBGetRowCount**: Dieser Aufruf der Funktion soll die Anzahl der zu füllenden Zeilen ermitteln. Dies ist der geeignete Ort, um die anzuzeigenden Daten zusammenzustellen.

- **acLBGetColumnCount**: Dieser Funktionsaufruf erwartet die Übergabe der Anzahl der zu füllenden Spalten.

- **acLBGetColumnWidth**: Dieser Funktionsaufruf erwartet die Übergabe der Breiten der zu füllenden Spalten.

- **acLBGetValue**: Die Callback-Funktion wird für jedes anzuzeigende Elemente einmal mit dem Wert **acLBGetValue** für den Parameter **intCode** aufgerufen. Die Anzahl der Elemente ergibt sich aus dem Produkt aus Zeilen- und Spaltenzahl. Welche Zeile und Spalte gerade abgefragt wird, können Sie den weiteren Parametern **lngRow** und **lngColumn** entnehmen.

- **acLBGetFormat**: Auch der Aufruf mit diesem Parameterwert erfolgt für jedes anzuzeigende Element einmal. Hier können Sie die Formatierung der Listeneinträge vornehmen (standardmäßig **-1**).

- **acLBClose**, **acLBEnd**: Diese beiden Parameter werden in weiteren Aufrufen der Callback-Funktion nach dem Füllen des Steuerelements übergeben. Damit können Sie beispielsweise den Zeitpunkt erfassen, zu dem das Steuerelement komplett gefüllt ist.

Wann wird die Callback-Funktion aufgerufen?

Wenn Sie experimentell ermitteln möchten, wann welcher Wert für den Parameter **intCode** übergeben wird, fügen Sie einfach für jeden Zweig der **Select Case**-Anweisung eine Anweisung wie die folgenden ein:

```
Case acLBInitialize
    Debug.Print "acLBInitialize"
    DateienEinlesen = True
Case acLBOpen
    Debug.Print "acLBOpen"
    DateienEinlesen = Timer
```

Sie können dem Direktfenster dann die Reihenfolge der Aufrufe entnehmen. Auf diese Weise finden Sie auch leicht heraus, dass die Aufrufe mit den beiden Parametern **acLBGetValue** und **acLBGetFormat** nicht nur beim ersten Füllen des Kombinations- oder Listenfeldes erfolgen, sondern auch zu weiteren Gelegenheiten – beispielsweise, wenn das Formular

den Fokus erhält, wenn Sie auf einen der Einträge im Steuerelement klicken oder wenn Sie im Falle eines Listenfeldes die Bildlaufleiste betätigen.

Im Falle eines Kombinationfeldes werden die Elemente übrigens erst beim ersten Aufklappen angelegt. Dies ändert sich auch nicht, wenn Sie das Kombinationsfeld gleich beim Öffnen des Formulars aktualisieren (mit **Me!<Formularname>.Requery**).

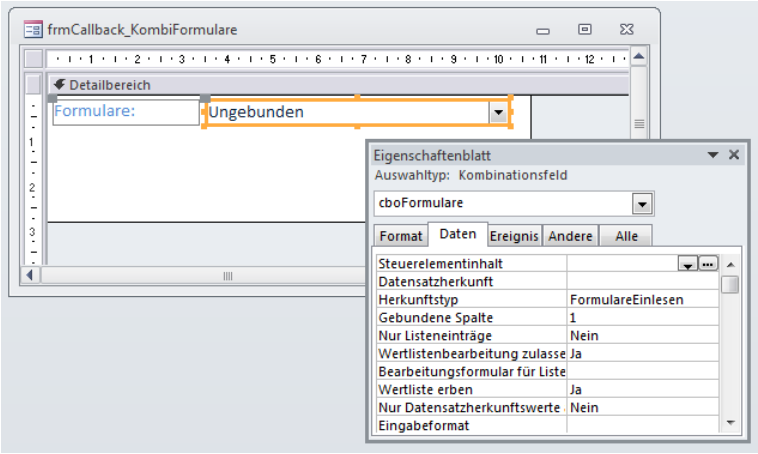


Bild 1: Einstellen der Callback-Funktion für ein Kombinationsfeld

Beispielanwendung

Nun benötigen wir ein passendes Beispiel, um ein Kombinations- oder Listenfeld mithilfe einer Callback-Funktion mit Daten zu füllen. Um die Vorgehensweise für ein- und mehrspaltige Darstellungen zu erläutern, verwenden wir gleich zwei Beispiele. Im ersten soll ein Kombinationsfeld mit den Namen aller Formulare der aktuellen Datenbank geöffnet werden, im zweiten ein Listenfeld mit den Dateinamen eines ausgewählten Verzeichnisses und deren Dateigröße.

Formularliste

Das Formular **frmCallback_KombiFormulare** soll alle Formularnamen der aktuellen Datenbank zur Auswahl anbieten. Dazu fügen Sie diesem zunächst ein Kombinationsfeld namens **cboFormulare** hinzu. Stellen Sie für die Eigenschaft **Datensatzherkunft** den Wert **FormulareEinlesen** ein (siehe Bild 1) und legen Sie die Callback-Funktion **FormulareEinlesen** im Klassenmodul des Formulars an. Gegebenenfalls müssen Sie zunächst noch ein Klassenmodul erzeugen – dies erledigen Sie durch Einstellen der Eigenschaft **Hat Modul** auf den Wert **Ja**.

Die Callback-Funktion sieht wie in Listing 1 aus. Die Funktion gibt beim ersten Aufruf zunächst den Wert **True** als Funktionswert zurück, damit die Funktion mit den übrigen Parameterwerten aufgerufen wird.

Da dieses Kombinationsfeld das einzige Steuerelement ist, das diese Callback-Funktion aufruft, geben Sie für den Aufruf mit dem Wert **acLBOpen** für den Parameter **intCode** den Wert **1** zurück.

Für den Parameterwert **acLBGetRowCount** ermitteln Sie die Anzahl der in der Datenbank enthaltenen Formulare über die Eigenschaft **Count** der Auflistung

```
Function FormulareEinlesen(ctl As Control, lngID As Long, lngRow As Long, lngColumn As Long, _
        intCode As Integer) As Variant
    Select Case intCode
        Case acLBInitialize
            FormulareEinlesen = True
        Case acLBOpen
            FormulareEinlesen = 1
        Case acLBGetRowCount
            FormulareEinlesen = CurrentProject.AllForms.Count
        Case acLBGetColumnCount
            FormulareEinlesen = 1
        Case acLBGetColumnWidth
            FormulareEinlesen = -1
        Case acLBGetValue
            FormulareEinlesen = CurrentProject.AllForms.Item(lngRow).Name
        Case acLBGetFormat
            FormulareEinlesen = Null
    End Select
End Function
```

Listing 1: Einlesen der Namen der in der aktuellen Datenbank vorhandenen Formulare

Das ListView-Steuerelement – Grundlagen

Neben dem Listenfeld gibt es noch ein weiteres Steuerelement zur Anzeige von Listen: das ListView-Steuerelement. Dieses Steuerelemente ist allerdings kein in Access integriertes Steuerelement, sondern ein ActiveX-Steuerelement. Sie fügen es auf etwas anderem Wege zu einem Formular hinzu und füllen es nicht durch einfaches Zuweisen einer Datensatzherkunft, sondern per VBA. Darüber hinaus bietet es allerdings eine Reihe Möglichkeiten, die das herkömmliche Listenfeld nicht aufweist.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1207_ListView.mdb**.

Eigenschaften des ListView-Steuerelements

Das ListView-Steuerelement bietet die folgenden Möglichkeiten:

- Anzeigen von Icons und Texten
- Anzeigen von Bildern
- Ändern der Spaltenbreite zur Laufzeit
- Verwendung von Drag and Drop
- Manuelles Ändern des Inhalts (nur erste Spalte)
- Sortierung per Mausklick

In dieser Artikelreihe sehen wir uns diese und weitere Techniken rund um das ListView-Steuerelement an.

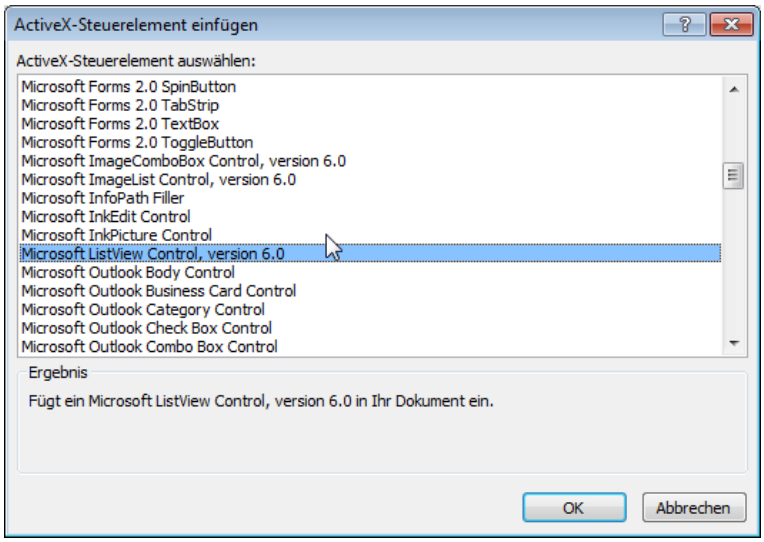


Bild 1: Auswählen des ListView-Steuerelements

ListView-Steuerelement anlegen

Das ListView-Steuerelement fügen Sie über den Eintrag ActiveX-Steuerelement der Liste der Steuerelemente hinzu. Dies öffnet den Dialog aus Bild 1, wo Sie den Wert **Microsoft ListView Control, version 6.0** auswählen. Das ListView-Steuerelement wird zunächst recht klein angelegt (siehe Bild 2), sodass Sie dieses zunächst auf die gewünschte Größe ziehen. Vergeben Sie gleich einen Namen für das neu hinzugefügte Steuerelement. In diesem Beispiel wollen wir die Daten der Tabelle **tblPersonal** im ListView-Steuerelement anzeigen, also nennen wir das Steuerelement **lvwPersonal**.

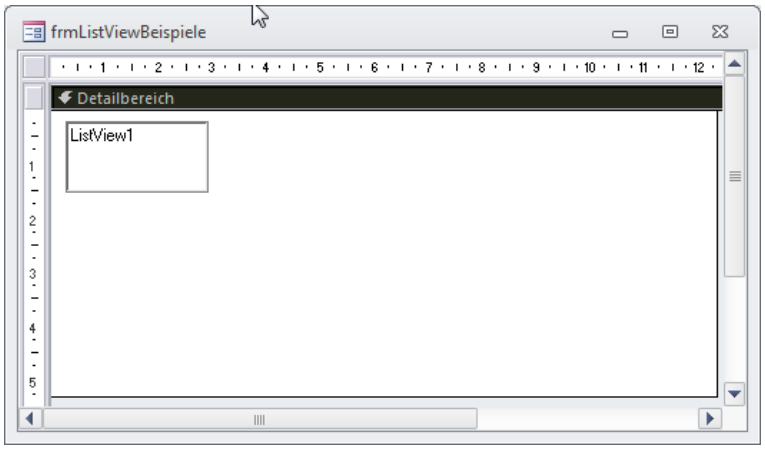


Bild 2: Ein ListView-Steuerelement direkt nach dem Hinzufügen

Handhabung des ListView-Steuerelements unter VBA

Einer der wichtigsten Unterschiede zwischen eingebauten Steuerelementen und ActiveX-Steuerelementen ist folgender: Wenn Sie unter VBA mit einem eingebauten Steuerelement arbeiten und dieses etwa mit **Me!txtVorname** referenzieren, können Sie über IntelliSense gleich auf alle Eigenschaften zugreifen, die für dieses Steuerelement vorgesehen sind. Bei ActiveX-Steuerelementen sieht dies etwas anders aus. Wenn Sie etwa mit **Me!lvwPersonal** auf das soeben angelegte ListView-Steuerelement zugreifen und

dann einen Punkt eintippen, zeigt IntelliSense nur die Eigenschaften an, die alle ActiveX-Steuerelemente gemeinsam aufweisen (siehe Bild 3). Um per VBA und IntelliSense direkt auf die ListView-spezifischen Eigenschaften zuzugreifen, sind einige zusätzliche Schritte nötig. Der erste ist, dass Sie im Klassenmodul des Formulars eine Objektvariable deklarieren, die genau den Typ des ListView-Steuerelements aufweist – etwa so:

```
Dim objLvwPersonal As MSComctlLib.ListView
```

Diese Variable müssen Sie nun nur noch mit einem entsprechenden Verweis füllen. Das erledigen Sie direkt beim Öffnen des Formulars, aber nicht in der Ereignisprozedur, die durch das Ereignis **Beim Öffnen** ausgelöst wird, sondern im Ereignis **Beim Laden**. **Beim Öffnen** ist möglicherweise zu früh, gegebenenfalls führt der Zugriff auf ein ActiveX-Steuerelement zu diesem Zeitpunkt zu Problemen. Also legen Sie eine Ereignisprozedur an, die durch das Ereignis **Beim Laden** des Formulars ausgelöst wird. Dieser Prozedur fügen Sie zunächst nur eine einzige Zeile hinzu, welche die Zuweisung übernimmt:

```
Private Sub Form_Load()
    Set objLvwPersonal = Me!lvwPersonal.Object
End Sub
```

Im Gegensatz zu den eingebauten Steuerelementen, bei denen die Zuweisung einfach über einen Verweis wie **Me!txtVorname** erfolgt, wird der Verweis hier gezielt über die Eigenschaft **Object** des ActiveX-Steuerelements hinzugefügt. Danach können Sie dann per IntelliSense auf die eigentlichen Eigenschaften des ListView-Steuerelements zugreifen (siehe Bild 4).

Einträge zum ListView-Steuerelement hinzufügen

Nachdem dies erledigt ist, fügen Sie den ersten Eintrag zum ListView-Steuerelement hinzu – zunächst noch nicht auf Basis der Tabelle **tblPersonal**, sondern durch Übergabe eines Literals mit dem Namen. Fügen Sie dazu die folgenden Zeilen zur obigen Prozedur hinzu:

```
With objLvwPersonal
    .ListItems.Add 1, "a1", "André Minhorst"
End With
```

Das ListView-Steuerelement verfügt also über eine Auflistung namens **ListItems**, der Sie mit der **Add**-Eigenschaft einen Wert hinzufügen können.

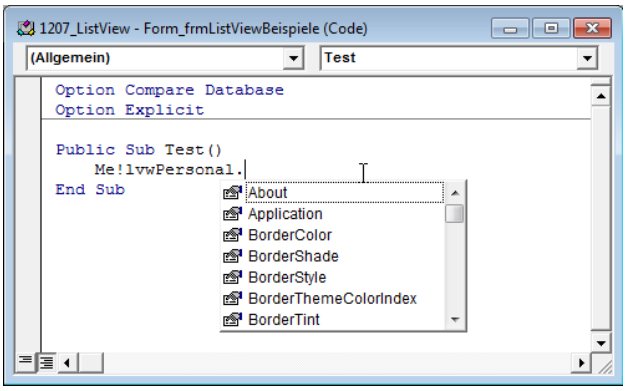


Bild 3: Standardeigenschaften von ActiveX-Steuerelementen

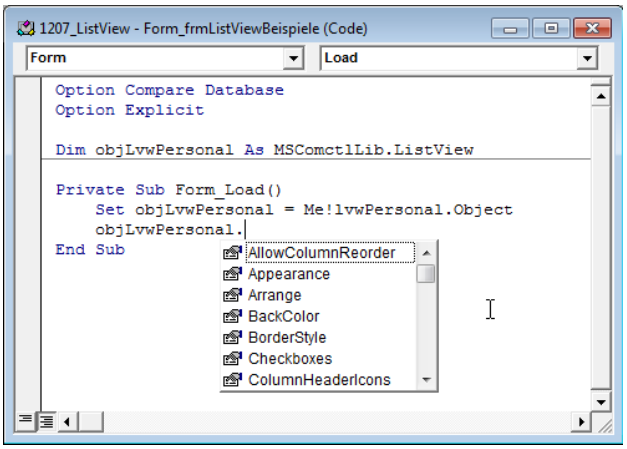


Bild 4: Spezielle Eigenschaften des ListView-Steuerelements

Diese Methode hat die folgenden Parameter:

- **Index**: Index-Wert für das Element. Gibt die Position an, an welcher der neue Eintrag hinzugefügt wird.
- **Key**: Eindeutiger Wert, muss mit einem Buchstaben beginnen
- **Text**: Anzuzeigender Text
- **Icon**: Name eines Icons im zugewiesenen **ImageList**-Steuerelement
- **SmallIcon**: Name eines Icons im zugewiesenen **ImageList**-Steuerelement

Die beiden letzten Parameter sind an dieser Stelle noch nicht relevant, wir kümmern uns später darum.

Reihenfolge beim Anlegen per Index

Zunächst betrachten wir die Wirkung verschiedener Werte für den Parameter **Index**. Dieser Index beginnt

immer mit **1**, weshalb Sie beim Hinzufügen des ersten Elements entweder keinen Wert für den Parameter Index angeben oder den Wert **1**. Grundsätzlich ist der Wert des Parameters **Index** für das spätere Einfügen von Elementen sinnvoll. Wenn Sie also etwa bereits ein Element angelegt haben und wollen ein weiteres vor diesem Element hinzufügen, übergeben Sie dem Parameter **Index** den Wert **1** – dies fügt den neuen Eintrag an der ersten Position ein und verschiebt die übrigen Einträge nach unten.

Wie dies funktioniert, zeigen die folgenden beiden Codezeilen:

```
objLvwPersonal.ListItems.Add , "a1", _
    "Zuerst hinzugefügt"
objLvwPersonal.ListItems.Add 1, "a2", _
    "Später vorn eingeschoben"
```

Das Ergebnis zeigt Bild 5. Auf die gleiche Weise können Sie Elemente an beliebiger Stelle einschieben.

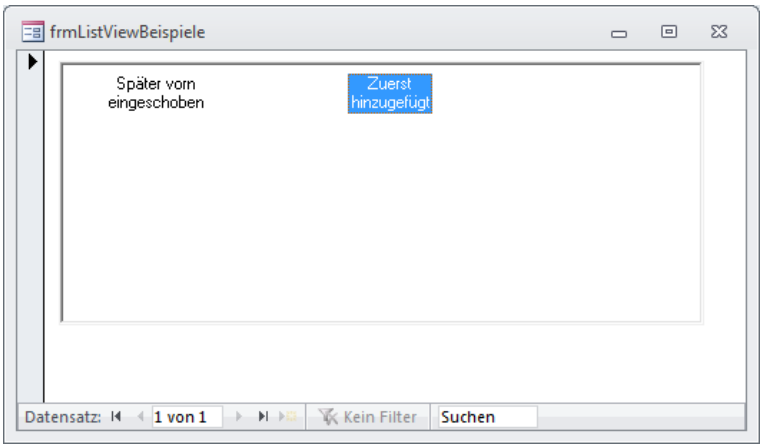


Bild 5: Einschieben von Elementen im ListView-Steuerelement

Eigenschaften des ListView-Steuerelements

Nach diesen ersten Schritten werfen wir einen Blick auf das Eigenschaftsfenster eines ListView-Steuerelements. Um dieses zu öffnen, wechseln Sie zur Entwurfsansicht des Formulars und klicken mit der rechten Maustaste auf das ListView-Steuerelement. Wählen Sie aus dem nun erscheinenden Kontextmenü den Eintrag **ListViewCtrl-Object|Properties** aus, um den Dialog aus Bild 6 zu öffnen. Hier können Sie einige Eigenschaften einstellen. Alle Eigenschaften können Sie allerdings auch per VBA einstellen. Die Auswirkungen der verschiedenen Einstellungen für die Eigenschaften schauen wir uns zu gegebener Zeit an. Zunächst wollen wir nur dafür sorgen, dass das ListView-Steuerelement unsere Daten in Listenform und nicht nebeneinander anzeigt. Dazu stellen Sie die Eigenschaft **View** auf den Wert **lvwReport** ein.

Nach dem Schließen des Eigenschaftsfensters und einem Wechsel in die Formularansicht bleibt das Listenfeld jedoch leer. Das Problem ist, dass diese Ansicht nur Daten anzeigt, wenn entsprechende Spaltenüberschriften vorliegen – und diese fehlen an dieser Stelle noch. Also öffnen wir nochmals den Eigenschaften-Dialog und wechseln dort zur Registerseite **ColumnHeaders**. Hier klicken Sie auf die Schaltfläche **InsertColumn** und tragen für die Eigen-

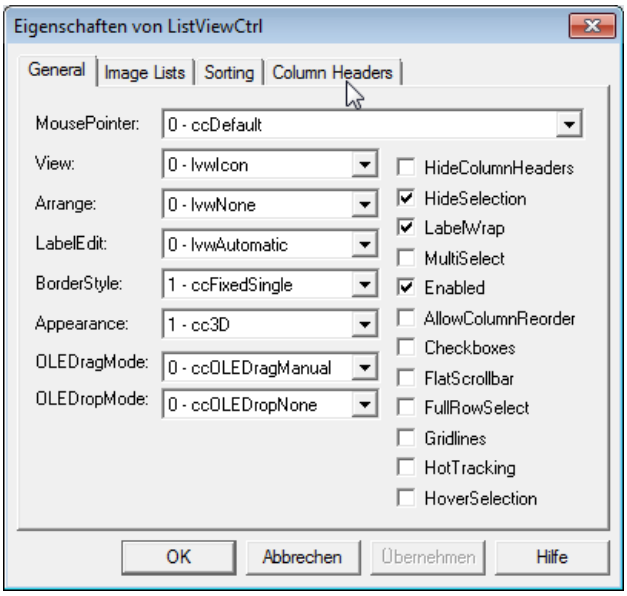


Bild 6: Eigenschaften eines ListView-Steuerelements

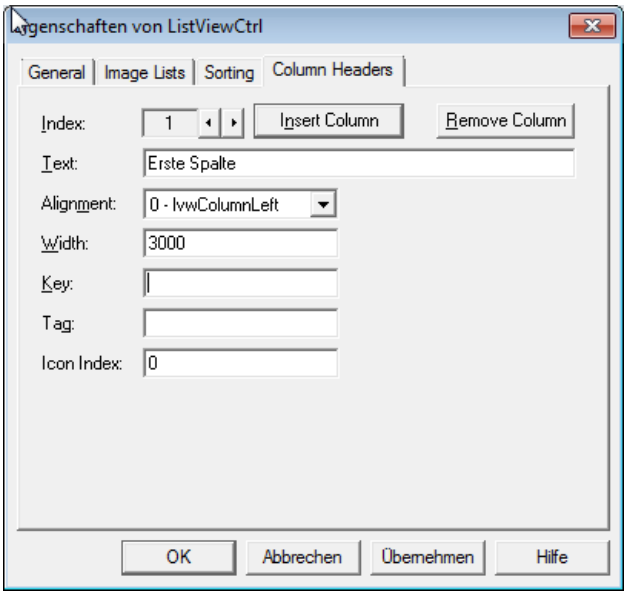


Bild 7: Hinzufügen eines Spaltenkopfes zum ListView-Steuerelement

Access und Excel: Import und Export

Die Zusammenarbeit von Access und Excel ist schon immer fruchtbar gewesen – egal, ob Daten aus Access nach Excel oder umgekehrt importiert oder verknüpft werden oder ob die beiden Anwendungen sich gegenseitig fernsteuern. In den Fällen, in denen der Benutzer es richtig macht, profitieren die für Einsteiger so ähnlich erscheinenden und dennoch so unterschiedlichen Anwendungen voneinander. In dieser Artikelreihe zeigen wir Ihnen, wie Access und Excel zusammenarbeiten. Im ersten Teil geht es um den Import und Export von Daten.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1208_AccessUndExcel.mdb** und der Datei **tblArtikel.xls**.

Von Access nach Excel

Als Erstes schauen wir uns die verschiedenen Möglichkeiten an, wie Sie Daten von Access nach Excel verschieben oder verknüpfen. Dabei kommen auch durchaus profane Methoden zum Zug – wie zum Beispiel die Zwischenablage.

Kopieren über die Zwischenablage

Der einfachste Weg, Daten aus einer Access-Tabelle in ein Excel-Sheet zu übertragen, ist die Zwischenablage. Wenn Sie alle Datensätze einer Tabelle wie etwa der Tabelle **tblArtikel** kopieren möchten, markieren Sie die zu kopierenden Daten, indem Sie auf die linke obere Ecke der Tabelle klicken (siehe Bild 1). Um die markierten Daten in die Zwischenablage zu übertragen, betätigen Sie die Tastenkombination **Strg + C**.

Bild 1: Markieren aller Datensätze einer Tabelle per Mausklick auf die linke obere Ecke

Bild 2: Excel-Tabelle mit frisch eingefügten Daten aus einer Access-Tabelle

Danach aktivieren Sie Excel und platzieren die Einfügemarke an die Stelle, welche das linke obere Element der zu kopierenden Daten aufnehmen soll. Betätigen Sie die Tastenkombination **Strg + V**, um die Daten einzufügen.

Das Ergebnis sieht (nach der Anpassung der Zeilenhöhen und Spaltenbreiten) wie in Bild 2 aus – die Daten wurden samt Spaltenüberschriften übernommen.

Auch die Einheiten wurden soweit übertragen – lediglich das die Werte des **Ja/Nein**-Feldes **Auslaufartikel** wurden mit **WAHR** und **FALSCH** in die Excel-Tabelle eingetragen.

Auf die gleiche Art können Sie auch einen oder mehrere zusammenhängende Datensätze nach Excel übernehmen. Dazu markieren Sie alle zu kopierenden Datensätze und gehen wie oben beschrieben vor (siehe Bild 3). Auch hier werden die Spaltenköpfe übernommen.

Und schließlich können Sie auch noch den Inhalt einzelner Spalten kopieren. Im Beispiel aus Bild 4 soll nur die Spalte mit den Artikelnamen kopiert werden.

Dazu klicken Sie einfach auf dem Kopf der zu kopierenden Spalte. Die Daten werden dann ebenfalls

inklusive Spaltenkopf in die Excel-Tabelle eingefügt.

Wenn Sie bestimmte Felder einiger Datensätze kopieren möchten, kommen Sie allerdings nicht um die Erstellung einer entsprechenden Abfrage herum. Zeigen Sie diese in der Datenblattansicht an und kopieren Sie die daraus gewünschten Daten.

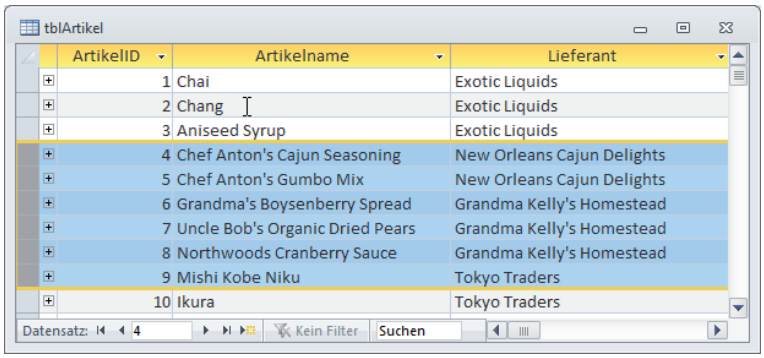


Bild 3: Kopieren nur einiger Zeilen einer Access-Tabelle

Kopieren von Excel nach Access

Der umgekehrte Weg ist nicht ganz so komfortabel. Das liegt aber auch daran, dass Access natürlich ein festes Schema vorgibt und Excel nicht – hier können Sie quasi beliebige Inhalte einfügen. Andersherum ist es weitaus schwieriger: Wenn Sie Daten von Excel in die Zwischenablage und von dort in eine bestehende Access-Tabelle kopieren wollen, müssen schon alle Felddatentypen genau passen, die Anzahl der Felder übereinstimmen et cetera.

Auf gut deutsch: Es funktioniert so gut wie gar nicht. Anders sieht es natürlich aus, wenn Sie nur einmal den Inhalt eines Feldes von Excel nach Access übertragen möchten. In diesem Fall muss der hinzugefügte Ausdruck lediglich den passenden Datentyp aufweisen.

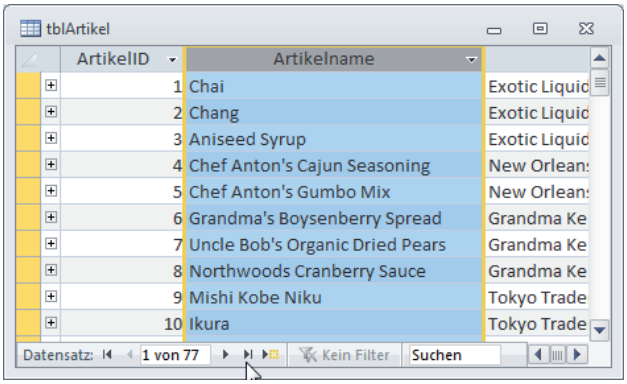


Bild 4: Kopieren einer einzelnen Spalte

Exportieren von Access nach Excel

Wenn Sie eine komplette Tabelle oder Abfrage in eine neue Excel-Datei exportieren möchten, verwenden Sie den Kontextmenü-Eintrag **Exportieren|Excel** des zu exportierenden Objekts aus dem Datenbankfenster beziehungsweise dem Navigationsbereich (siehe Bild 5). Es erscheint der Dialog **Exportieren – Excel-Tabelle**, mit dem Sie die Zieltabelle angeben, das Dateiformat auswählen und angeben, ob die Ausgabe formatiert erfolgen soll. Außerdem können Sie noch festlegen, ob Access das neue Excel-Dokument gleich nach dem Erstellen öffnen soll. Nach einem Mausklick auf die Schaltfläche **OK** wird die gewünschte Datei erstellt.

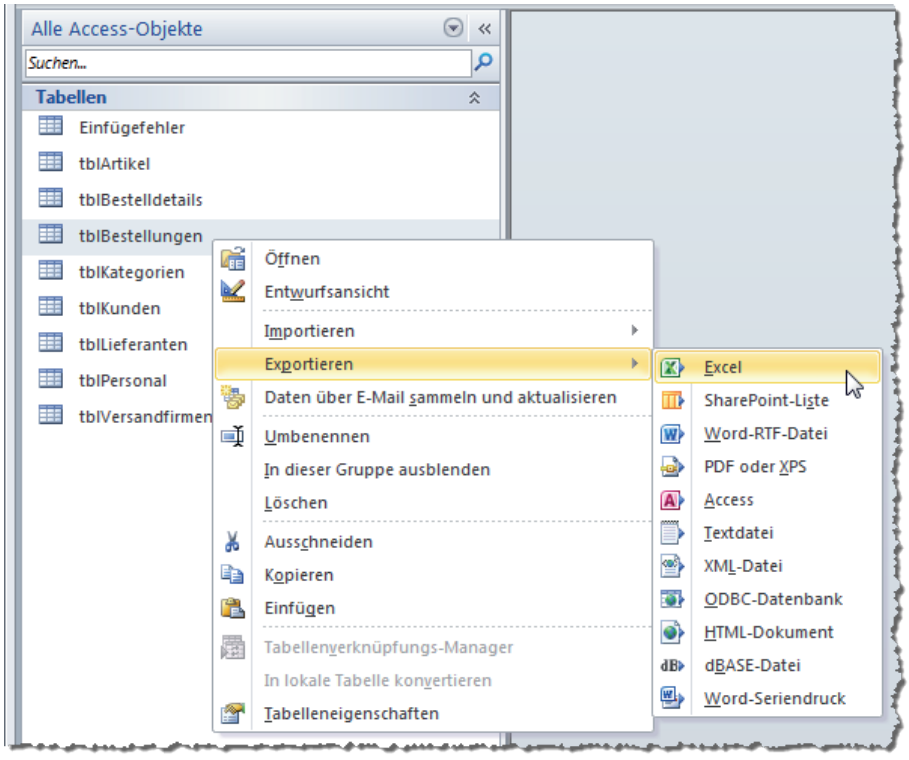


Bild 5: Exportieren einer kompletten Tabelle von Access nach Excel

Auch mit dieser Methode können Sie einzelne Datensätze speichern. Dazu öffnen

Sie die zu exportierende Tabelle oder Abfrage, markieren die betroffenen Datensätze (naturgemäß können hier nur zusammenhängende Datensätze markiert werden) und rufen dann über das Kontextmenü der Tabelle im Datenbankfenster beziehungsweise Navigationsbereich den Export-Eintrag auf. Nun wird eine weitere Option aktiviert, mit der Sie festlegen können, dass nur die markierten Datensätze exportiert werden sollen.

Diese Variante hat einen Nachteil: Sie können damit immer nur eine neue Excel-Datei erstellen oder eine bestehende überschreiben. Das Einfügen als neues Tabellenblatt ist nicht möglich.

Beachten Sie auch, dass beim Export von Tabellen mit Nachschlagefeldern die angezeigten Werte exportiert werden und nicht die tatsächlichen Werte der Fremdschlüsselfelder. Dies trifft in der Tabelle **tblArtikel** beispielsweise auf die Felder **KategorieID** und **LieferantID** zu – deren angezeigte Inhalte werden als Text in die entsprechende Spalte der Excel-Tabelle exportiert.

Achtung, Shortcut!

Dies gelingt hingegen auf eine andere Weise überraschend einfach (getestet nur unter Access 2010):

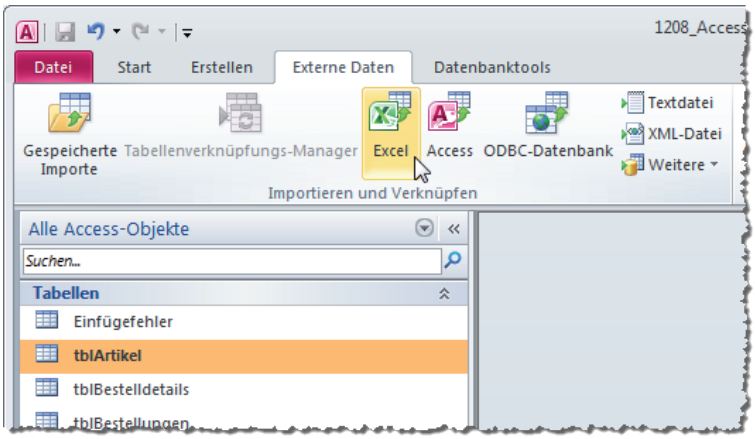


Bild 6: Aufrufen des Imports oder der Verknüpfung von Daten aus Excel von Access aus

Öffnen Sie einfach die Access-Datenbank mit aktiviertem Datenbankfenster oder Navigationsbereich und daneben die Excel-Tabelle. Nun ziehen Sie einfach eine Tabelle oder Abfrage aus einem dieser Bereiche auf die Zelle in der Excel-Tabelle, welche die erste Spaltenüberschrift enthalten soll. Fertig - dies kopiert ebenfalls die komplette Tabelle.

Importieren von Daten aus Excel

Wenn Sie Daten aus einer Excel-Tabelle nach Access importieren möchten, haben Sie mehrere Möglichkeiten. Einige davon werden von Access aus initiiert, andere von Excel aus.

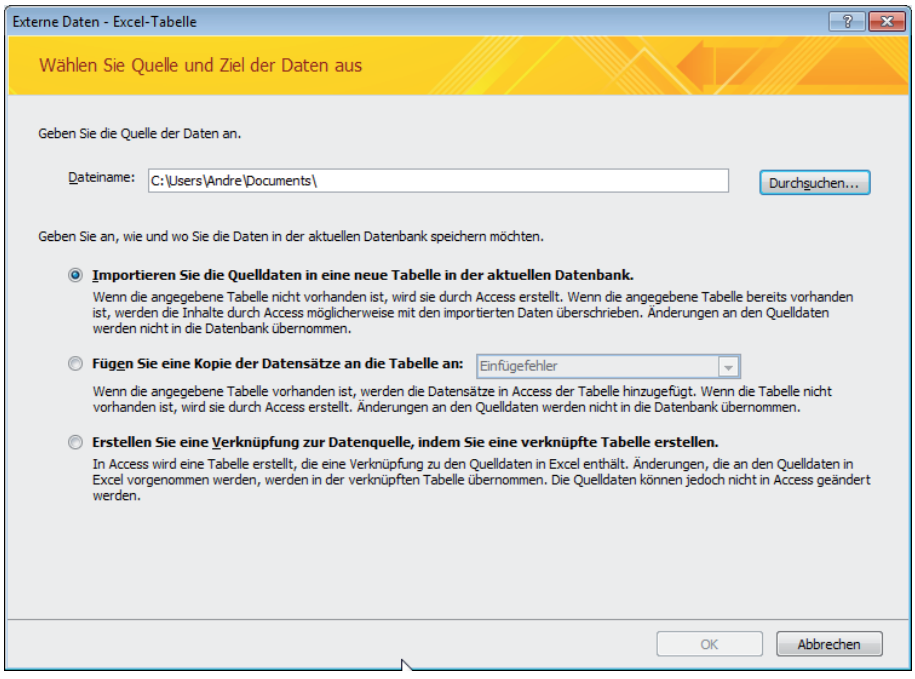


Bild 7: Dialog zum Steuern von Import- und Verknüpfungsvorgängen

Bei den von Access aus gesteuerten Methoden gibt es die Möglichkeit zum Import und zum Verknüpfen der benötigten Daten. Unter Access 2010 rufen Sie diese Methoden beide über den Ribbon-Eintrag **Externe Daten|Importieren und Verknüpfen|Excel** auf (siehe Bild 6).

Dies öffnet den Dialog aus Bild 7. Hier finden Sie drei Möglichkeiten: den Import in eine neue Tabelle, das Anhängen an eine bestehende Tabelle oder das Verknüpfen der Excel-Tabelle.

Wir haben zu Testzwecken einfach den Inhalt der Ta-

Die Umschaltfläche

Neben der normalen Schaltfläche fristet die Umschaltfläche bei den meisten Entwicklern das Dasein eines Stiefkindes. Oder sollte man eher sagen, neben dem Kontrollkästchen und der Optionsgruppe samt Optionsfeldern? Denn eigentlich ist die Aufgabe dieses Steuerelements eher die zur Auswahl einer von zwei oder mehr Optionen. Dazu gehört natürlich auch das An- oder Ausschalten bestimmter Zustände.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1208_Umschaltflaeche.mdb**.

Umschaltfläche als Einzelgänger

Im einfachsten Fall verwenden Sie die Umschaltfläche als einfaches, ungebundenes Steuerelement. Ein Beispiel zeigt Bild 1. Die dortige Umschaltfläche im Formular **frmToggleButton** der Beispieldatenbank heißt schlicht **tblAnAus**. In diesem ersten Beispiel wollen wir uns zunächst ansehen, welche Werte die Umschaltfläche abhängig von ihrem Zustand annimmt.

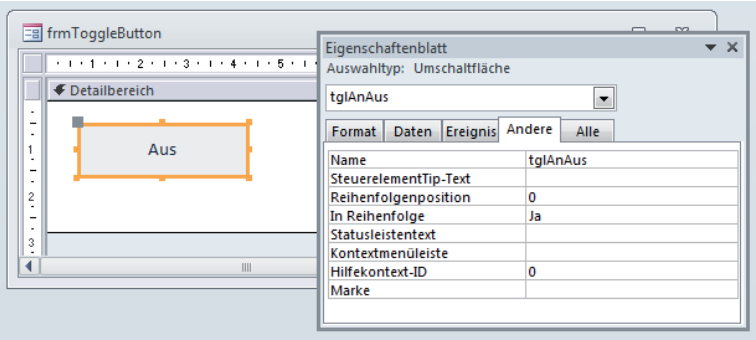


Bild 1: Eine einfache Umschaltfläche

Dazu geben Sie einfach im Direktfenster des VBA-Editors (**Strg + G**, am besten platzieren Sie beide Fenster nebeneinander) die folgende Anweisung ein:

```
? Forms!frmToggleButton!tglAnAus.Value
Null
```

Dies war der Zustand direkt nach dem Öffnen des Formulars für eine ungebundene Umschaltfläche – die Eigenschaft **Value** liefert den Wert **Null**. Sie können die Angabe der Eigenschaft **Value** auch weglassen, da dies die Standardeigenschaft dieses Steuerelements ist. Klicken Sie nun einmalig auf die Umschaltfläche und führen die Abfrage für den gedrückten Status aus, sieht das Ergebnis so aus:

```
? Forms!frmToggleButton!tglAnAus
-1
```

Der gedrückte Zustand liefert also den Wert **-1** beziehungsweise **True**. Ein erneutes Anklicken liefert schließlich den Wert **0**, also **False**.

Ereignisse der Umschaltfläche

Anschließend prüfen wir, wie wir auf das Anklicken der Umschaltfläche reagieren und den aktuellen Wert verarbeiten können. Ein Blick auf die Registerseite **Ereignis** des Eigenschaftsfensters liefert alle zur Verfügung stehenden Ereigniseigenschaften (siehe Bild 2). Davon sind drei interessant: **Beim Klicken**, **Vor Aktualisierung** und **Nach Aktualisierung**.

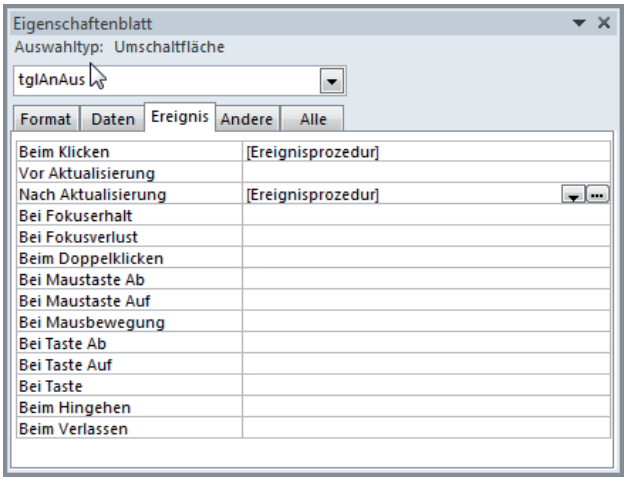


Bild 2: Ereigniseigenschaften der Umschaltfläche

Beim Anklicken der Umschaltfläche werden alle drei Eigenschaften in der folgenden Reihenfolge ausgelöst:

- **Vor Aktualisierung**
- **Nach Aktualisierung**
- **Beim Klicken**

Dies können Sie ganz einfach testen, indem Sie für alle drei Eigenschaften je eine Ereignisprozedur anlegen und diesen einen Haltepunkt hinzufügen. Die Ereig-

niseigenschaft fügen Sie am schnellsten hinzu, indem Sie den Wert **[Ereigniseigenschaft]** für die jeweilige Eigenschaft auswählen und auf die Schaltfläche mit den drei Punkten klicken. Einmal im VBA-Editor angelangt, können Sie weitere Ereignisprozeduren über die beiden Kombinationsfelder oben im Code-Fenster mit dem Klassenmodul **Form_frmToggleButton** des Beispielformulars hinzufügen. Dort klicken Sie dann links auf den grauen Rand neben dem Prozedurkopf, um Haltepunkte hinzuzufügen (siehe Bild 3).

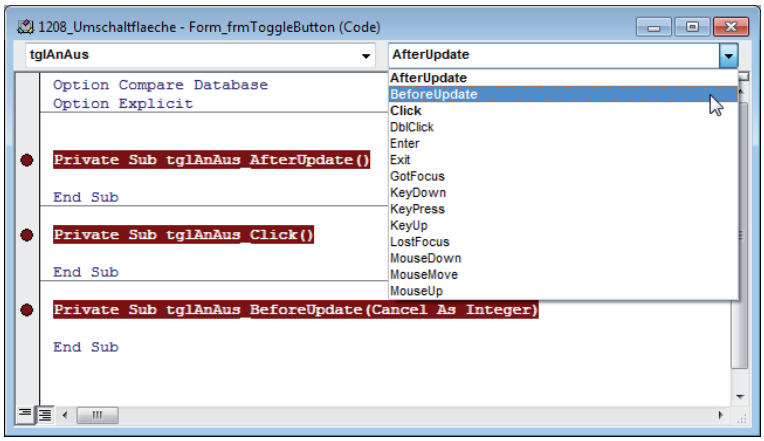


Bild 3: Wichtige Ereignisprozeduren einer Umschaltfläche

Schauen wir, ob es Unterschiede zwischen den Prozeduren gibt – gegebenenfalls wird ja das Ereignis **Vor Aktualisierung** so früh ausgelöst, dass Value noch nicht den geänderten Wert liefert, dies aber bei den übrigen Ereignissen der Fall ist. Dazu lassen wir bei jedem Ereignis den Wert der Umschaltfläche ausgeben:

```
Private Sub tglAnAus_AfterUpdate()
    Debug.Print "AfterUpdate: ", tglAnAus.Value
End Sub

Private Sub tglAnAus_Click()
    Debug.Print "Click: ", tglAnAus.Value
End Sub

Private Sub tglAnAus_BeforeUpdate(Cancel As Integer)
    Debug.Print "BeforeUpdate: ", tglAnAus.Value
End Sub
```

Nach dem Anklicken der Umschaltfläche liefern jedoch alle drei Ereignisse den gleichen Wert, wie die folgende Ausgabe im Direktfenster bestätigt:

```
BeforeUpdate:               -1
AfterUpdate:   -1
Click:         -1
```

Der einzige offensichtliche Unterschied ist somit, neben der Reihenfolge, der Parameter **Cancel** der Prozedur, die durch das Ereignis **Vor Aktualisierung** ausgelöst wird. Mit diesem können Sie festlegen, dass die Aktualisierung abgebrochen wird. Vorsicht ist hier bei der Abfrage des Wertes in der durch die Eigenschaft **Vor Aktualisierung** ausgelösten Prozedur geboten: Diese zeigt nämlich immer den Wert an, den die Umschaltfläche annehmen würde, wenn die Prozedur nicht durch Einstellen des **Cancel**-Parameters auf **True** abgebrochen wird.

Bild 4: Aktivieren und Deaktivieren weiterer Steuerelemente in Abhängigkeit vom Zustand der Umschaltfläche

Aktionen beim Umschalten

Das folgende Beispiel soll zeigen, wie Sie in Abhängigkeit vom Zustand der Umschaltfläche weitere Aktionen anstoßen. Dazu fügen wir dem Formular **frmToggleButton** eine einfache Schaltfläche namens **cmdSchaltflaeche** hinzu (siehe Bild 4). Ein Klick auf die Umschaltfläche soll die Schaltfläche aktivieren oder deaktivieren, je nachdem, ob die Umschaltfläche den Wert **True** oder **False** enthält.

Da anscheinend alle oben geprüften Ereignisprozeduren den gleichen Zustand liefern, können wir einfach die durch das **Beim Klicken**-Ereignis ausgelöste Prozedur für unsere Zwecke einsetzen.

```
Private Sub tglAnAus_Click()
    If Me!tglAnAus = True Then
        Me!cmdSchaltflaeche.Enabled = True
    Else
        Me!cmdSchaltflaeche.Enabled = False
    End If
End Sub
```

Wenn der Benutzer nun die Umschaltfläche heruterdrückt, wird die Schaltfläche **cmdSchaltflaeche** aktiviert. Wenn die Umschaltfläche gelöst wird, deaktiviert dies die Umschaltfläche. Die **If...Then**-Abfrage untersucht dazu den Ausdruck **Me!tblAnAus**

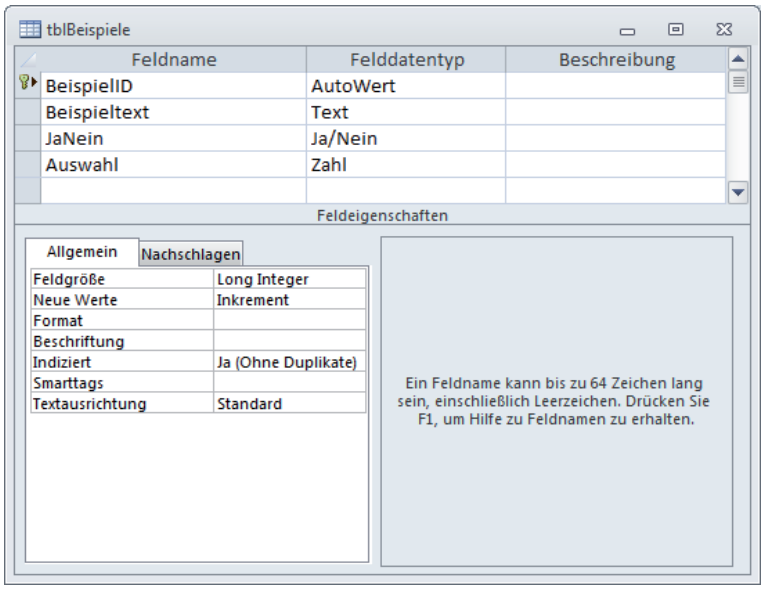


Bild 5: Wichtige Ereignisprozeduren einer Umschaltfläche

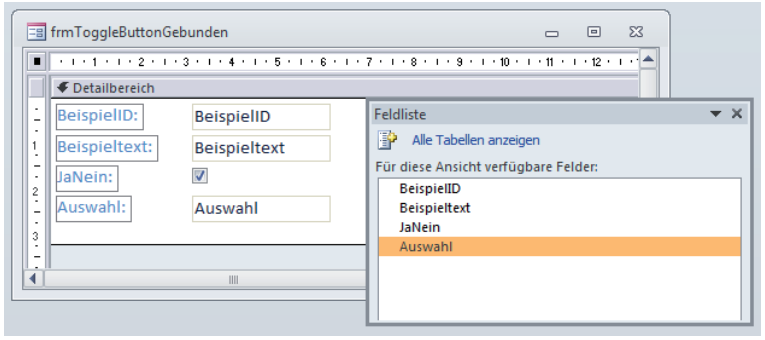


Bild 6: Hinzufügen der Beispielfelder zum Formular **frmToggleButtonGebunden**

auf den Wert **True** und weist der Eigenschaft **Enabled** der Schaltfläche **cmdSchaltflaeche** gegebenenfalls ebenfalls den Wert **True** zu, anderenfalls den Wert **False**.

Das schreit nach einer Vereinfachung: Sollten nicht noch weitere Aktionen innerhalb der **If...Then**-Bedingung ausgeführt werden, können Sie diese durch eine einzige Anweisung ersetzen:

```
Private Sub tglAnAus_Click()
    Me!cmdSchaltflaeche.Enabled = _
        Me!tglAnAus
End Sub
```

Gebundene Umschaltflächen

Umschaltflächen lassen sich anstelle von Kontrollkästchen und Optionsfeldern einsetzen und können somit zur Auswahl des Inhalts von **Ja/Nein**-Feldern oder auch von Feldern mit mehreren möglichen, aber in der Anzahl eingeschränkten Werten dienen. Dies bezieht sich vor allem auf das Bearbeiten von in Tabellen gespeicherten Werten. Zu Testzwecken enthält die Beispieldatenbank eine Tabelle namens **tblBeispiele**, die neben einem Textfeld auch ein **Ja/Nein**-Feld und ein Feld namens **Auswahl** mit dem Datentyp **Zahl** enthält (siehe Bild 5).

Wenn Sie diese Tabelle als Datenherkunft des Formulars **frmToggleButtonGebunden** festlegen und alle Felder dieser Tabelle in den Formularentwurf ziehen, sieht dies zunächst wie gewohnt aus (siehe Bild 6).

Wir wollen zunächst das **Ja/Nein**-Feld **JaNein**, das standardmäßig als Kontrollkästchen ausgeführt wird, in eine Umschaltfläche umwandeln. Dazu wählen Sie aus dem Kontextmenü dieses Steuerelements den Eintrag **Ändern zu|Umschaltfläche** aus (siehe Bild 7).

Damit haben wir ein Zwischenziel erreicht: Wenn Sie nun in die Formularansicht des Formulars wechseln, können Sie den Wert des **Ja/Nein**-Feldes **JaNein** durch Aktivieren und Deaktivieren der Umschaltfläche auf den gewünschten Wert einstellen. Und wenn Sie mehr als einen Datensatz mit angelegt

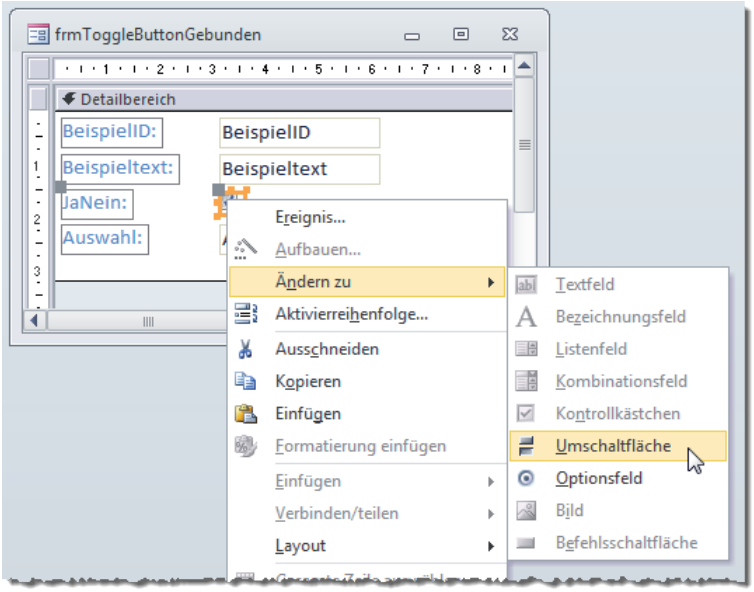


Bild 7: Umwandeln des Kontrollkästchens in eine Umschaltfläche

ListView-Steuerelement mit Daten füllen

Das ListView-Steuerelement soll in den meisten Fällen Daten aus einer Tabelle oder Abfrage anzeigen. Nachdem Sie im Artikel Das ListView-Steuerelement – Grundlagen bereits die grundlegenden Techniken im Umgang mit dem ListView-Steuerelement kennengelernt haben, schauen wir uns in diesem Artikel an, wie Sie ein ListView-Steuerelement mit den Daten einer Tabelle füllen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1208_ListView.mdb**.

ListView mit Daten der Personal-Tabelle füllen

Wie bereits im Artikel **Das ListView-Steuerelement – Grundlagen** erläutert, können Sie dem **ListView**-Steuerelement nicht einfach eine Tabelle oder Abfrage als Datenherkunft zuweisen. Dies ist natürlich erst einmal ein Nachteil gegenüber einem Unterformular in der Datenblattansicht oder einem herkömmlichen Listenfeld, aber wenn Sie den notwendigen Code erst einmal kennen, ist dies auch kein Aufwand mehr.

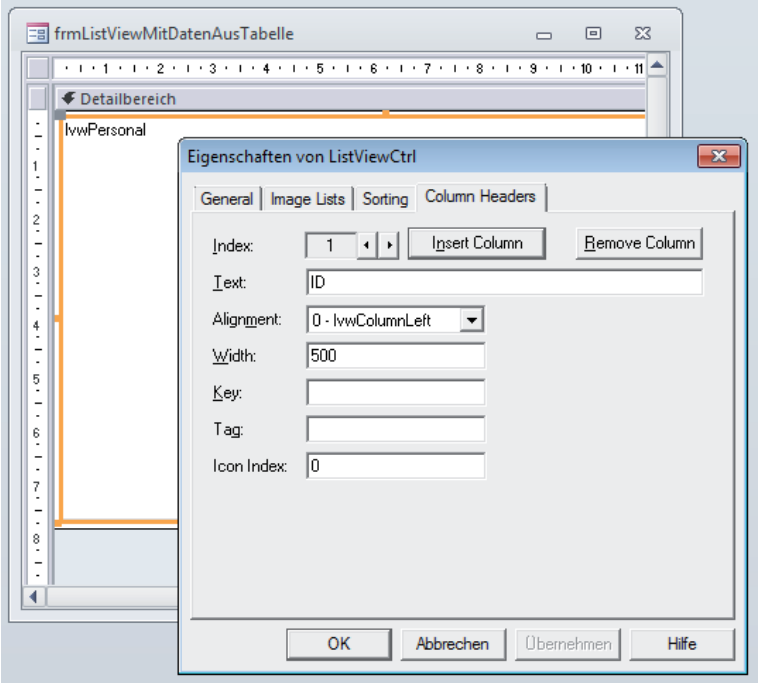


Bild 1: Einstellen der Spalten im ListView-Steuerelement

Spalten einrichten

Wichtig ist zunächst, dass Sie festlegen, welche Spalten der Datenherkunft im **ListView**-Steuerelements angezeigt werden sollen. Im oben erwähnten Artikel haben Sie auch erfahren, dass es zwei Wege gibt, die Eigenschaften des ListView-Steuerelements zu definieren: über den entsprechenden Dialog oder gleich per VBA.

Wir wollen im ersten Ansatz den Wert des Feldes **PersonalID** der Tabelle **tblPersonal** der Beispieldatenbank ausgeben sowie die beiden Felder **Vorname** und **Nachname**. Wir benötigen also drei Spalten, die Sie auf zwei Arten anlegen können.

Doch zunächst legen Sie ein neues Formular namens **frmListViewMitDatenAusTabelle** an und fügen eines neues ListView-Steuerelement hinzu. Geben Sie diesem den Namen **lvwPersonal**.

Wenn Sie die Spalten über den Eigenschaften-Dialog des **ListView**-Steuerelements erstellen möchten, klicken Sie mit der rechten Maustaste auf das Steuerelement und wählen den Eintrag **ListViewCtrl-Objekt|Properties** aus. In diesem Dialog wechseln Sie zur Registerseite **ColumnHeaders** und legen die Eigenschaften für die erste Spalte an – in diesem Fall den Text und die Spaltenbreite (siehe Bild 1).

Die übrigen Eigenschaften kommen hier noch nicht zum Einsatz. Fügen Sie dann mit der Schaltfläche **Insert Column** zwei weitere Spalten hinzu, die Sie mit den Beschriftungen **Vorname** und **Nachname** sowie jeweils mit der Spaltenbreite **3.000** ausstatten.

Das Ergebnis sieht nun wie in Bild 2 aus – Spaltenüberschrift und Spaltenbreite entsprechen den Vorgaben.

Wenn Sie nun jedoch ein ListView-Steuerelement mit einigen weiteren Spalten ausstatten möchten, wird dies schnell unübersichtlich. Deshalb ist das Einstellen der Spalten und der übrigen Eigenschaften per VBA eine gute Idee: Sie haben dort nämlich die Spalteneigenschaften gleich im Überblick und müssen nicht mühsam im Eigenschaftsfenster navigieren.

Bild 2: **ListView**-Steuerelement mit Spaltenköpfen, aber noch ohne Inhalt

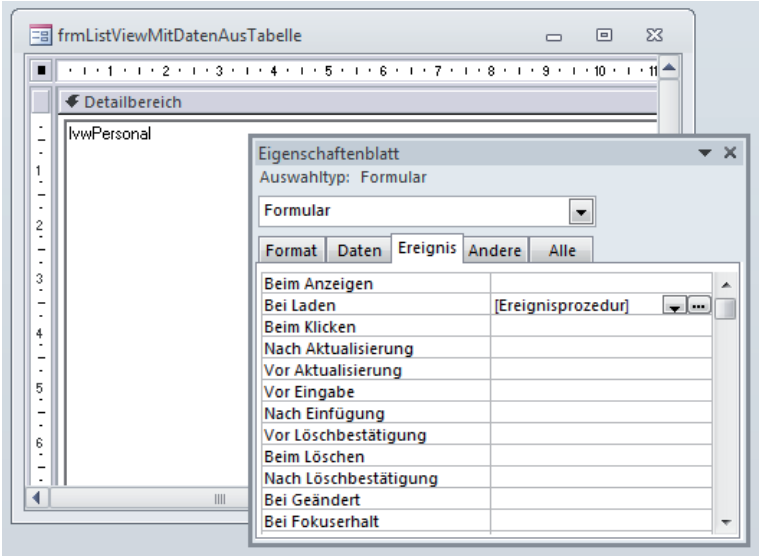


Bild 3: Hinzufügen einer Ereignisprozedur für das **Beim Laden**-Ereignis des Formulars mit dem **ListView**-Steuerelement

Um dies per VBA zu erledigen, wählen Sie im Eigenschaftsfenster des Formulars den Wert **[Ereignisprozedur]** für die Eigenschaft **Beim Laden** aus und klicken Sie auf die Schaltfläche mit den drei Punkten (siehe Bild 3). Ergänzen Sie die nun im VBA-Editor erscheinende Prozedur wie in Listing 1.

Legen Sie außerdem noch die folgende Deklaration im Modulkopf des Klassenmoduls **Form_frmListViewMitDatenAusTabelle** an:

```
Dim objLvwPersonal As MSComctlLib.ListView
```

Die ersten Eigenschaften stellen das Aussehen des **ListView**-Steuerelements ein (**View** sorgt für die Listenansicht, **Appearance** stellt ein, dass das Steuerelement nicht vertieft angezeigt wird, **BorderStyle** entfernt den Rahmen und **FlatScrollbar** aktiviert die Bildlaufleisten, falls der Inhalt nicht auf einmal angezeigt werden kann. **Gridlines** aktiviert die Anzeige eines Rasters.

Danach kümmert sich die Prozedur um die Spalten. Diese können mit der Auflistung **ColumnHeaders** angesprochen, erstellt oder auch gelöscht werden.

Die **Clear**-Methode entfernt zunächst alle vorhandenen Spalten. Dann fügt die **Add**-Methode nacheinander die drei gewünschten Spalten hinzu und stellt mit dem dritten und vierten Parameter die Spaltenüberschrift sowie die Spaltenbreite ein.

Mit dem vierten Parameter können Sie noch die Ausrichtung festlegen. Dieser Parameter entspricht der Eigenschaft **Alignment**, für die Sie die Werte **lvwColumnCenter**, **lvwColumnLeft** und **lvwColumnRight** einstellen können.

ListView-Steuerelement mit Daten füllen

Nun sollen die Daten der drei Felder **PersonalID**, **Vorname** und **Nachname** im **ListView**-Steuerelement landen. Die dafür notwendigen Anweisungen fügen wir der Übersicht halber in eine eigene Prozedur ein.

```
Private Sub Form_Load()
    Set objLvwPersonal = Me!lvwPersonal.Object
    With objLvwPersonal
        .View = lvwReport
        .Appearance = ccFlat
        .BorderStyle = ccNone
        .FlatScrollBar = False
        .Gridlines = True
        .ColumnHeaders.Clear
        .ColumnHeaders.Add , , "ID", 500
        .ColumnHeaders.Add , , "Vorname", 3000
        .ColumnHeaders.Add , , "Nachname", 3000
        .ListItems.Clear
    End With
End Sub
```

Listing 1: Einstellen der Eigenschaften und Hinzufügen der Spalten zum **ListView**-Steuerelement

Zugriff auf Daten im ListView-Steuerelement

Genau wie beim Listenfeld oder auch in einem Datenblatt gibt es Möglichkeiten, Einträge anzuklicken, den markierten Eintrag auszuwerten oder auch per Code bestimmte Einträge zu markieren. Wie dies gelingt, zeigt dieser Artikel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1208_ListView_Zugriff.mdb**.

Eigenschaften zur Markierung

Zunächst schauen wir uns die Eigenschaft **FullRowSelect** an. Mit dieser legen Sie fest, auf welchen Bereich der Benutzer klicken muss, um einen Eintrag zu markieren und wie der markierte Eintrag hervorgehoben wird.

Im Beispiel im Formular **frmListViewMarkieren** können Sie diese Eigenschaft mit der Umschaltfläche **tglKompletteZeile** ein- und ausschalten. Das Formular basiert auf dem Formular **frmListViewMitDatenAusTabelle**, das Listenfeld wird wie im Artikel **ListView-Steuerelement mit Daten füllen** mit den Daten der Tabelle **tblPersonal** gefüllt.

Über dem **ListView**-Steuerelement haben wir eine Umschaltfläche platziert, welche die Eigenschaft **FullRowSelect** des **ListView**-Steuerelements abwechselnd auf die Werte **True** und **False** einstellt.

Damit dies geschieht, legen Sie für die Umschaltfläche mit dem Namen **tblKompletteZeile** eine Ereignisprozedur an, die durch das Ereignis **Nach Aktualisierung** ausgelöst wird (Markieren der Eigenschaft, Auswahl von **[Ereignisprozedur]**, Klick auf die Schaltfläche mit den drei Punkten). Diese Prozedur füllen Sie wie folgt:

```
Private Sub tglKompletteZeile_AfterUpdate()
    Select Case Me!tglKompletteZeile
        Case -1
            objLvwPersonal.FullRowSelect = True
        Case 0
            objLvwPersonal.FullRowSelect = False
    End Select
End Sub
```

Beim Laden des Formulars wird die Eigenschaft **FullRowSelect** auf den Wert **False** eingestellt, was die folgende Zeile in der Prozedur **Form_Load** bewirkt:

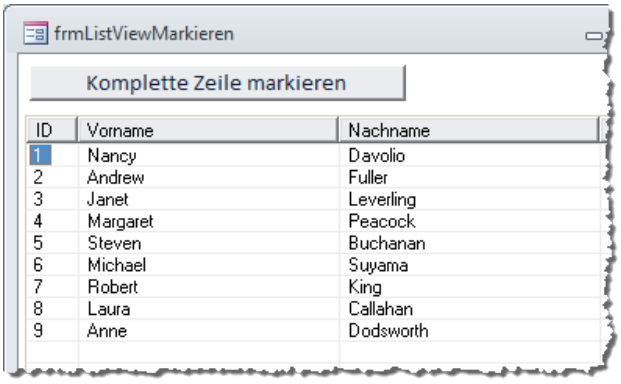


Bild 1: Einstellen der Spalten im ListView-Steuerelement

```
Private Sub Form_Load()
    Set objLvwPersonal = Me!lvwPersonal.Object
    With objLvwPersonal
        ...
        .FullRowSelect = False
        ...
End Sub
```

Dies sorgt außerdem dafür, dass nur ein Klick auf den Text in der ersten Spalte des **ListView**-Steuerelements den gewünschten Eintrag markiert – und hier auch nur den Inhalt der ersten Spalte (siehe Bild 1).

Die übrigen hier nicht abgebildeten Anweisungen finden Sie im Klassenmodul **Form_frmListViewMarkieren** der Beispieldatenbank, die Erläuterung dazu im Artikel **ListView-Steuerelement mit Daten füllen**.

Da die Umschaltfläche beim Öffnen des Formulars standardmäßig den Wert **Null** enthält, ändert ein erster Klick den Wert auf **-1**, also **True**. Dies löst die Ereignisprozedur **tblKompletteZeile_AfterUpdate** aus und stellt die Eigenschaft **FullRowSelect** ebenfalls auf **True** ein. Ein weiterer Klick stellt die Eigenschaft wieder auf **False** ein. Wenn **FullRowSelect** den Wert **True** enthält, kann der Benutzer den Eintrag an einer beliebigen Stelle anklicken. Dies markiert dann auch die komplette Zeile (siehe Bild 2).

Den markierten Datensatz ermitteln

Wenn der Benutzer einen Datensatz markiert, dann zum Durchführen einer bestimmten Aktion: Viel-

leicht möchte er die Detailansicht aufrufen, den Datensatz löschen oder etwas anderes damit erledigen. In jedem Fall benötigen wir dann einen eindeutigen Hinweis auf diesen Datensatz – in der Regel den Wert des Primärschlüsselfeldes für diesen Eintrag.

Diesen haben wir beispielsweise in der ersten Spalte des **ListView**-Steuerelements abgebildet, aber er befindet sich auch in der **Key**-Eigenschaft eines jeden Eintrags. Wir schauen uns erst an, wie Sie den Wert der ersten Spalte ermitteln und kümmern uns dann um die **Key**-Eigenschaft. Letztere wird vermutlich öfter eingesetzt, da es nicht immer gewünscht ist, die Werte der Primärschlüsselfelder anzuzeigen.

Führen wir die ersten Versuche über das Direktfenster aus (Tastenkombination **Strg + G**) – bei in der Formularansicht geöffnetem Formular **frmListViewMarkieren** und markiertem Datensatz. Dann referenzieren Sie das **ListView**-Steuerelement wie folgt – hier zur Ausgabe des Namens zur Prüfung, ob der Bezug stimmt:

```
? Forms!frmListViewMarkieren!lvwPersonal.Name
lvwPersonal
```

Dies gelingt bereits. Übrigens ist es eine gute Idee, sich zum Ermitteln von Bezügen auf Steuerelemente oder Eigenschaften in einem Formular Schritt für Schritt vorzuarbeiten. Auf diese Weise können Sie potenzielle Fehlerquellen nacheinander eliminieren.

Wenn wir uns dann weiter vorarbeiten, verwenden wir die Eigenschaft **SelectedItem**, um auf den gewünschten Eintrag zuzugreifen. Dies liefert in folgendem Beispiel den Wert **3**:

```
? Forms!frmListViewMarkieren!lvwPersonal.SelectedItem
3
```

Nun stellt sich nur noch die Frage: Liefert dies den Wert der ersten Spalte oder den Index der markierten Spalte? Leider sind beide Werte gleich, sodass wir dies aktuell nicht ermitteln können.

Wie finden wir also heraus, ob die Eigenschaft den Index oder den Wert der ersten Spalte liefert? Ganz einfach: Wir fügen die Daten der Datenherkunft so hinzu, dass diese in der umgekehrten Reihenfolge angezeigt werden. Dazu brauchen Sie nur den ersten Parameter der **Add**-Anweisung in der Prozedur **ListViewFuellen** im Klassenmodul **Form_frmListViewMarkieren** auf **1** einstellen:

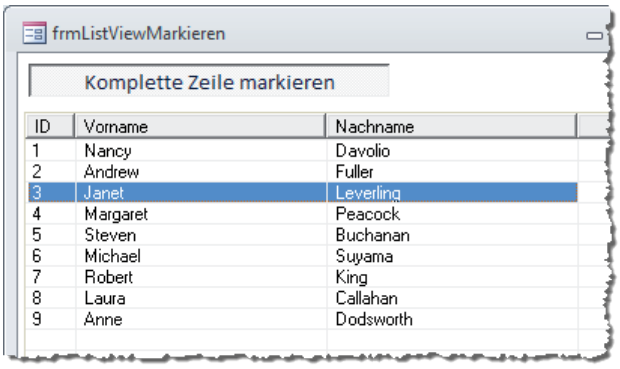


Bild 2: Markieren der kompletten Zeile im **ListView**

```
Set objListitem = objLvwPersonal.ListItems.Add(1, _
    "p" & rst!PersonalID, rst!PersonalID)
```

Der neueste Eintrag wird dann jeweils an der ersten Position eingefügt, sodass die Datensätze wie in Bild 3 hinzugefügt werden. Eine erneute Abfrage liefert dann Gewissheit, dass **SelectedItem** den Wert des ersten Feldes zurückgibt. Aber ist das die ganze Wahrheit? Nein. Dies finden Sie heraus, wenn Sie einmal mit **SelectedItem** den aktuellen Wert abfragen möchten, wenn gar kein Eintrag markiert ist. Dies liefert die Fehlermeldung aus Bild 4.

Wenn **SelectedItem** den Typ **Variant**, **String** oder einen Zahlendatentyp hätte, würde dies nicht passieren. Zum Glück können wir den Typ des von **Selected-**

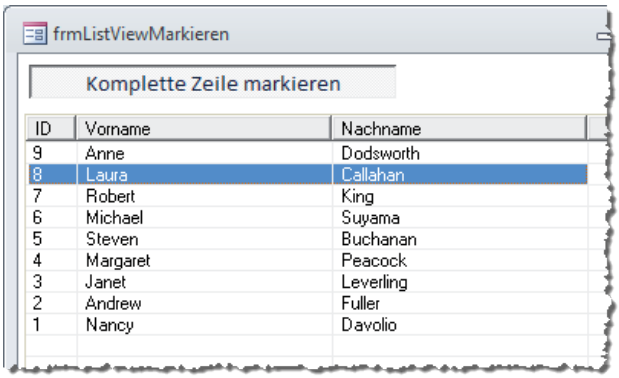


Bild 3: Datensätze in umgekehrter Reihenfolge

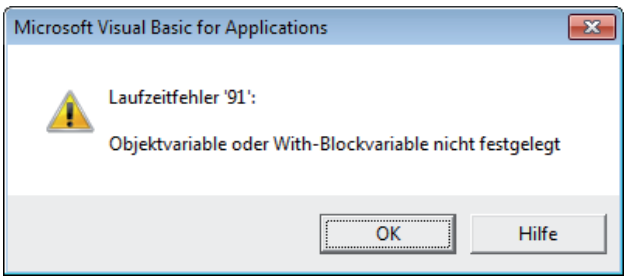


Bild 4: Fehler beim Zugriff auf **SelectedItem**, wenn kein Datensatz markiert ist

Item gelieferten Wertes mit der Funktion **TypeName** ermitteln – zumindest, wenn ein Eintrag markiert ist:

```
? Typename(Forms!frmListViewMarkieren!lvwPersonal.SelectedItem)
IListItem
```

SelectedItem liefert also ein Objekt zurück, das auf der Schnittstelle **IListItem** basiert – in diesem Fall mit dem Datentyp **ListItem** (wäre hier nun kein Eintrag markiert gewesen, hätte **Typename** den Wert **Nothing** geliefert, weil **SelectedItem** schlicht leer ist). Dies nur als theoretischer Hintergrund. Es stellt sich hier die Frage, warum **SelectedItem** bei einem markierten Element den Wert der ersten Spalte des markierten Elements zurückliefert.

Der Grund ist, dass **SelectedItem** wie auch andere Objekte wie etwa ein Textfeld eine Standardeigenschaft besitzen, die abgefragt werden, wenn der Entwickler keine Eigenschaft angibt. In diesem Fall handelt es sich um die **Text**-Eigenschaft, die Sie auch direkt abfragen können – am einfachsten wieder im Direktfenster (Anweisung in einer Zeile):

```
? Forms!frmListViewMarkieren!lvwPersonal.SelectedItem.Text
3
```

Primärschlüsselwert aus Key ermitteln

Nun enthält die erste Spalte nicht immer den Primärschlüsselwert, der aber so wichtig ist, um etwas mit dem betroffenen Datensatz anzustellen. Wie können wir noch auf den Primärschlüsselwert zugreifen? Es hat sich als gängige Praxis herausgestellt, den Primärschlüsselwert mit einem führenden Buchstaben (oder auch mehreren) zu versehen und diesen für die Eigenschaft **Key** anzugeben.

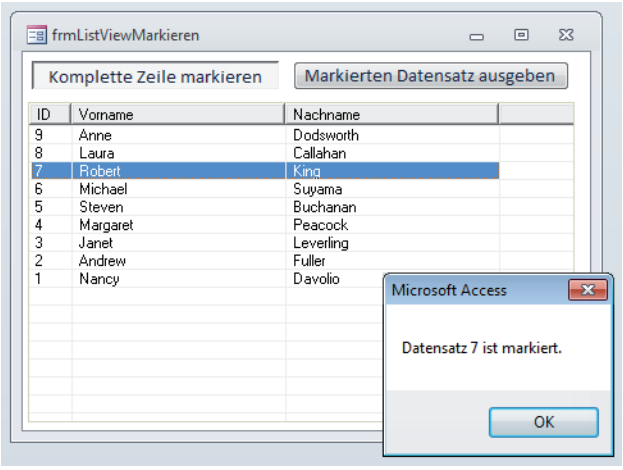


Bild 5: Ausgabe des Primärschlüsselwertes für einen markierten Eintrag

Auf diese Weise schlagen Sie zwei Fliegen mit einer Klappe: Erstens ist jedes Element im **ListView**-Steuerelement eindeutig identifizierbar und zweitens können Sie darüber den Primärschlüsselwert ermitteln.

In diesem Beispiel erhält die **Key**-Eigenschaft jeweils den Buchstaben **p** und den Primärschlüsselwert des abgebildeten Datensatzes zugewiesen, also beispielsweise **p3**. Dies geschieht gleich beim Hinzufügen der Elemente:

```
Set objListitem = objLvwPersonal.ListItems.Add(1, _
                  "p" & rst!PersonalID, rst!PersonalID)
```

Den **Key**-Wert des aktuell markierten Datensatz können Sie nun mit folgender Zeile ausgeben:

```
? Forms!frmListViewMarkieren!lvwPersonal.SelectedItem.Key
p3
```

Nun benötigen wir aber nur den Primärschlüsselwert. Außerdem soll gleich noch geprüft werden, ob überhaupt ein Eintrag markiert wurde, da sonst ein Fehler ausgelöst wird. Dies erledigen wir mit einer Schaltfläche namens **cmdMarkierterDatensatz**. Für die **Beim Klicken**-Ereigniseigenschaft hinterlegen wir die folgende Ereignisprozedur:

```
Private Sub cmdMarkierterDatensatz_Click()
    If Me!lvwPersonal.SelectedItem Is Nothing Then
        MsgBox "Kein Eintrag markiert."
    Else
        MsgBox "Datensatz " & Mid(Me!lvwPersonal. _
            SelectedItem.Key, 2) & " ist markiert."
    End If
End Sub
```

Die Prozedur prüft zunächst die **SelectedItem**-Eigenschaft auf den Wert **Nothing**. In diesem Fall wird eine Meldung ausgegeben, dass kein Eintrag markiert ist. Anderenfalls liefert eine Meldung den Primärschlüsselwert des markierten Datensatzes. Dazu liest sie die Eigenschaft **Key** ein und ermittelt mit der **Mid**-Funktion den Teil ab dem zweiten Zeichen. Wenn **Key** den Wert **p3** liefert, heißt das Ergebnis also **3**.

Dies zeigt wie in Bild 5 den Primärschlüsselwert des aktuell markierten Eintrags an.

AutoKeys: Globale Tastenkombinationen

Makros werden von professionellen VBA-Entwicklern gern als Spielerei abgetan und links liegen gelassen. Es gibt jedoch zwei Ausnahmen: das Autoexec-Makro, mit dem Sie Aktionen direkt beim Start einer Anwendung ausführen können, und das AutoKeys-Makro, mit dem sich globale Tastenkombinationen definieren lassen. Das Autoexex-Makro haben wir bereits in einem früheren Artikel beleuchtet, jetzt schauen wir uns das AutoKeys-Makro an.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1209_AutoKeys.mdb**.

Tastenkombinationen in Access

Grundsätzlich gibt es zwei Möglichkeiten, unter Access bestimmte Aktionen durch Tastenkombinationen auszulösen: Ereignisse in Formularen und das **AutoKeys**-Makro. Der wesentliche Unterschied ist, dass das **AutoKeys**-Makro global wirkt. Es spielt keine Rolle, ob gerade ein Formular geöffnet ist oder nicht, die im **AutoKeys**-Makro definierte Tastenkombination funktioniert immer.

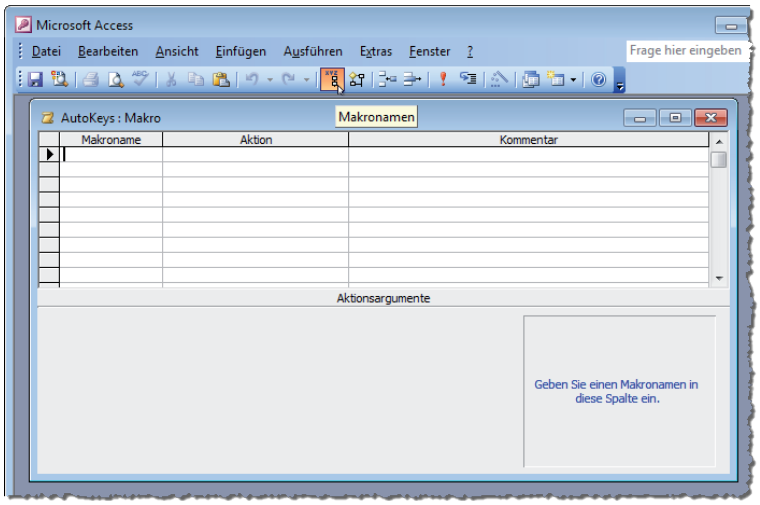


Bild 1: Anlegen eines neuen **AutoKeys**-Makros

Formular-Ereignisse auf Basis von Tastenanschlägen werden nur ausgelöst, wenn das Formular gerade den Fokus besitzt, und auch dann gibt es je nach Einstellung verschiedene Einschränkungen. Zu den Tastenereignissen in Formularen kommen wir in einem weiteren Artikel, hier kümmern wir uns zunächst nur um das **AutoKeys**-Makro.

Das AutoKeys-Makro

Dieses Makro hat gegenüber anderen Makros zwei Besonderheiten: Es besitzt den Namen **AutoKeys** und Sie legen zusätzlich zu den auszuführenden Anweisungen noch einen Ausdruck an, der die auslösende Tastenkombination beschreibt.

Im folgenden Beispiel erfahren Sie, wie Sie durch die Tastenkombination **Strg + Umschalt + M** ein Meldungsfenster anzeigen. Dies zeigt die grundlegende Funktion, anschließend lernen Sie weitere Tastenkombinationen kennen.

AutoKeys-Makro unter Access 2003

Unter Access 2003 und älter legen Sie zunächst ein neues Makro an, indem Sie bei aktiviertem Datenbankfenster den Menübefehl **Einfügen|Makro** ausführen oder im Datenbankfenster zur Registerseite **Makros** wechseln und dort auf die Schaltfläche **Neu** klicken.

Beides erstellt ein neues, leeres Makro, dass Sie zunächst unter dem Namen **AutoKeys** speichern sollten (mehr zu diesem Namen und eventuell damit verbundenen Problemen weiter unten).

Sie finden hier nun zwei Zeilen vor: **Aktion** und **Kommentar**. Die Zeile **Aktion** nimmt den Befehl auf, der beim Aufrufen der Tastenkombination ausgeführt werden soll, unter **Kommentar** können Sie entsprechende Anmerkungen unterbringen.

Fehlt noch eine Spalte, in der Sie die Tastenkombination einfügen können, die den Befehl auslösen soll. Dazu wird die Spalte **Makroname** zweckentfremdet, die Sie mit dem Eintrag **Makroname** der Symbolleiste einblenden (siehe Bild 1).

Tragen Sie nun in der ersten Zeile die folgenden Werte ein:

- **Makroname**: **^+P** (**^** steht für **Strg**, **+** für **Umschalt**)

Access und Excel: Verknüpfungen

Im Artikel »Access und Excel: Import und Export« haben Sie bereits verschiedene Möglichkeiten kennengelernt, Daten von Access nach Excel zu übertragen und umgekehrt. Im vorliegenden Artikel sehen wir uns an, wie Sie Daten aus einer Excel-Tabelle für den Zugriff in einer Access-Datenbank bereitstellen. Und dies gelingt auch andersherum: Dabei verlinken wir eine Access-Tabelle innerhalb von Excel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1209_AccessUndExcel_II.mdb** und in der Datei **tblArtikel.xls**.

Verknüpfen von Excel-Tabellen in Access

Im Artikel **Access und Excel: Import und Export** haben wir bereits einen Assistenten verwendet, der das Importieren und Exportieren von Daten von Access aus erlaubte. Die dritte Möglichkeit dieses Assistenten ist die Verknüpfung mit den in einer Excel-Tabelle gespeicherten Daten. Dazu wählen Sie im Dialog **Externe Daten – Excel-Tabelle** die dritte Option aus. Die danach erscheinenden Schritte des Assistenten kennen Sie bereits aus dem oben genannten Artikel. In diesem Fall können Sie damit lediglich festlegen, ob die Spaltenüberschriften als Feldnamen verwendet werden sollen.

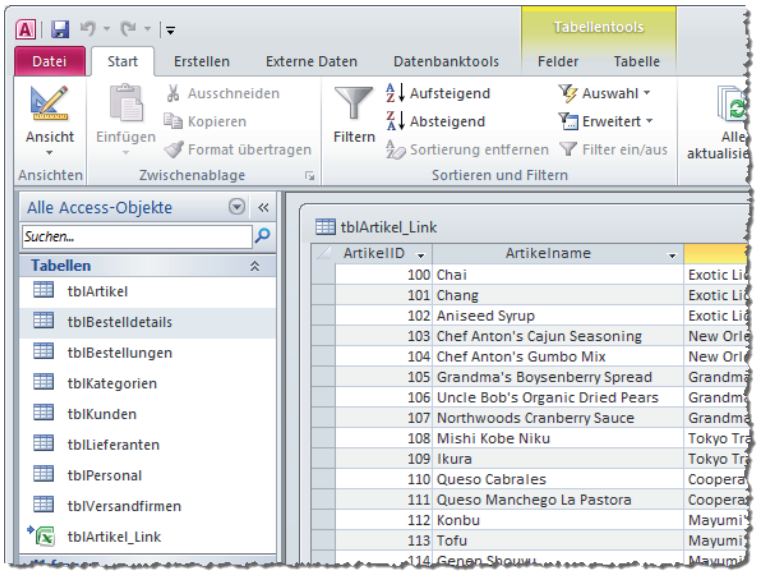


Bild 1: Verknüpfte Tabelle im Navigationsbereich und in der Datenblattansicht

Der Unterschied zum Import besteht darin, dass Access nun keine neue Tabelle erstellt oder die Daten an eine bestehende Tabelle anhängt, sondern dass lediglich eine Verknüpfung auf die Excel-Tabelle erstellt wird. Das Ergebnis können Sie Bild 1 entnehmen.

Dort erkennen Sie links den Eintrag für die Verknüpfung im Navigationsbereich der Datenbank,rechts erscheint die in der Datenblattansicht geöffnete verknüpfte Tabellen mit den aktuell in der Excel-Tabelle enthaltenen Daten.

Beispiel Excel-Daten in Access verfügbar machen

In vielen Unternehmen liegen Daten in Excel-Tabellen vor, die eigentlich auch in einer Datenbank-Tabelle gespeichert werden könnten. Gerade für den Zugriff mehrerer Mitarbeitern auf verschiedene Datensätze der gleichen Excel-Tabelle wäre dies interessant. Allerdings bietet sich diese Möglichkeit manchmal einfach nicht und die Daten verbleiben in einer Excel-Datei. Das ist allerdings auch kein Problem: Durch eine Verknüpfung wie die oben beschriebene können Sie die Daten in einer Access-Datenbank verfügbar machen und von dort aus über ein komfortables Formular zugreifen anstatt die Daten etwa in eine Tabellen mit vielen Spalten einzutragen.

Eine per Verknüpfung eingebundene Excel-Tabelle können Sie auch in Abfragen als Datenherkunft verwenden. Es lassen sich sogar Beziehungen herstellen, allerdings nicht mit referenzieller Integrität.

Einen Nachteil haben die verknüpften Excel-Tabellen: Die darin angezeigten Daten lassen sich nicht bearbeiten.

Aktualisieren der Verknüpfung

Früher oder später verschieben Sie die verknüpfte Excel-Datei vielleicht einmal oder Sie erhalten eine Access-Datenbank mit einer ungültigen Verknüpfung auf eine Excel-Tabelle. Dies wird zum Beispiel der Fall sein, wenn Sie die Beispieldaten erstmals in einem Verzeichnis auf Ihrem Rechner speichern. Sofern Sie nicht zufällig genau die gleiche Verzeich-

Beispieldaten-Assistent

Jeder Access-Entwickler steht irgendwann vor der Situation, mal eben einige Beispieldatensätze zu seiner Anwendung hinzuzufügen. Wer keine Lust auf Fantasie-Einträge wie ASDF oder QWER hat, die sich mal eben schnell mit wenigen Fingerbewegungen in ein Tabellenfeld einfügen lassen, kann dazu meinen Beispieldatenassistent verwenden. Dieser ist auf Adresstabellen spezialisiert, füllt aber auch andere Datenfelder. Dieser Artikel zeigt, wie Sie den Assistenten installieren und damit arbeiten.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1209_Beispieldaten.mdb** und **Beispieldaten-Assistent.mda**.

Installation

Der Beispieldatenassistent ist ein Access-Add-In. Solche Datenbanken haben eine spezielle Dateiendung (**.mda** unter Access 2003 und älter und **.accda** unter Access 2007 und jünger).

Access bietet eine einfache Funktion, um solche Access-Add-Ins schnell und zuverlässig zu installieren und diese somit unter Access sofort verfügbar zu machen. Im vorliegenden Beispiel entpacken Sie einfach die Add-In-Datei in ein Verzeichnis Ihrer Wahl.

Starten Sie dann Access, öffnen Sie eine beliebige Datenbankdatei (etwa die Beispieldatenbank zu diesem Artikel) und wählen Sie dann abhängig von der Access-Versionen einen der folgenden Menü- beziehungsweise Ribbonbefehle aus, um zunächst den Add-In-Manager zu starten:

- Access 2003 und älter: Wählen Sie den Menübefehl **Extras|Add-Ins|Add-In-Manager** aus (siehe Bild 1).
- Access 2007: Klicken Sie auf den Ribbon-Eintrag **Datenbanktools|Datenbanktools|Add-Ins|Add-In-Manager**.
- Access 2010: Betätigen Sie den Ribbon-Eintrag **Datenbanktools|Add-Ins|Add-In-Manager**.

Der Assistent sieht in allen Access-Versionen ähnlich aus, daher schauen wir uns hier nur den der aktuellen Version Access 2010 an. Dieser zeigt sich wie in Bild 2. Auf dem hier verwendeten Rechner wurden bereits einige Access-Add-Ins installiert. Für das markierte Add-In zeigt der Assistent einige Eigenschaften an. Ein noch nicht installiertes Access-Add-In wie etwa den Beispieldaten-Assistenten installie-

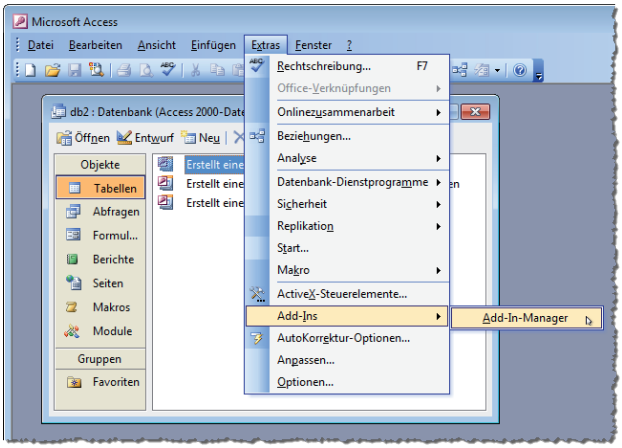


Bild 1: Installieren eines Access-Add-Ins unter Access 2003

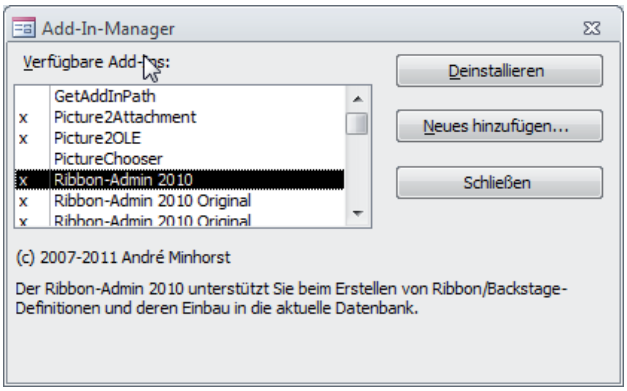


Bild 2: Der Add-In-Manager in Aktion

ren Sie nun, indem Sie auf die Schaltfläche **Neues hinzufügen** klicken. Es erscheint ein Dateiauswahl-Dialog, der standardmäßig das Verzeichnis anzeigt, in dem sich die bereits installierten Access-Add-Ins befinden. Dieses können Sie sich merken, wenn Sie später neue Versionen über ein bestehendes Access-Add-In kopieren möchten – Sie brauchen dann nicht den Umweg über den Add-In-Manager zu nehmen. Beachten Sie jedoch, dass das Access-Add-In bereits einmal installiert worden sein muss, damit es funktioniert.

Navigieren Sie nun zu dem Verzeichnis, in dem sich die zu installierende Datei **Beispieldaten-Assistent.**

mda befindet und klicken Sie auf die Schaltfläche **Öffnen**.

Danach erledigt der Add-In-Manager gleich eine ganze Reihe von Aufgaben – manche lassen sich direkt erkennen, andere verlaufen im Hintergrund:

- Im Add-In-Manager erscheint ein neuer, mit einem **x** gekennzeichneter Eintrag namens **Beispieldaten-Assistent** (siehe Bild 3).
- Das Add-In wird nun in der Liste der verfügbaren Add-Ins aufgeführt. Unter Access 2003 etwa unter **Extras|Add-Ins|Beispieldaten-Assistent**, unter Access 2007 unter **Datenbanktools|Datenbanktools|Add-Ins|Beispieldaten-Assistent** und unter Access 2010 wie in Bild 4 zu erkennen unter **Datenbanktools|Add-Ins|Beispieldaten-Assistent**.

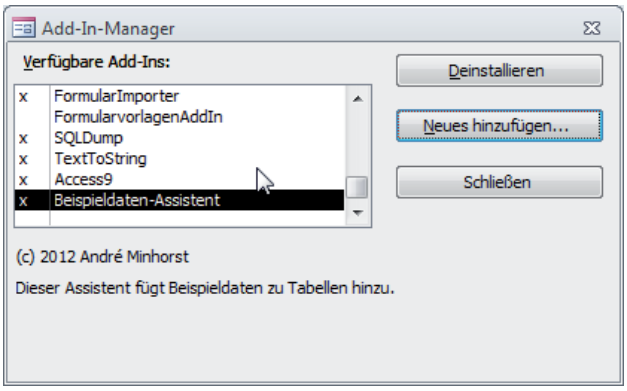


Bild 3: Ein frisch hinzugefügtes Access-Add-In

- Damit der Assistent so reibungslos funktioniert, erledigt der Add-In-Manager außerdem zwei Schritte im Hintergrund: Erstens wir die Add-In-Datei Beispieldaten-Assistent in das Add-In-Verzeichnis kopiert (im Falle von Windows 7 beispielsweise nach **C:\Users\<Benutzername>\AppData\Roaming\Microsoft\AddIns**). Und zweitens legt der Assistent einige Einträge in der Registry an, die Access beim Starten ausliest. Dabei erfährt Access, welche Access-Add-Ins verfügbar sind und dementsprechend in der Liste der Add-Ins angeboten werden sollen.

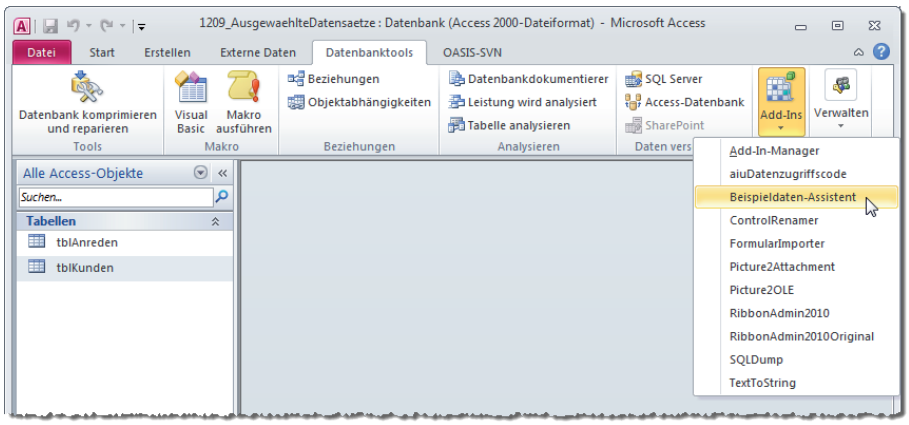


Bild 4: Aufruf des Add-Ins über die Liste der installierten Add-Ins

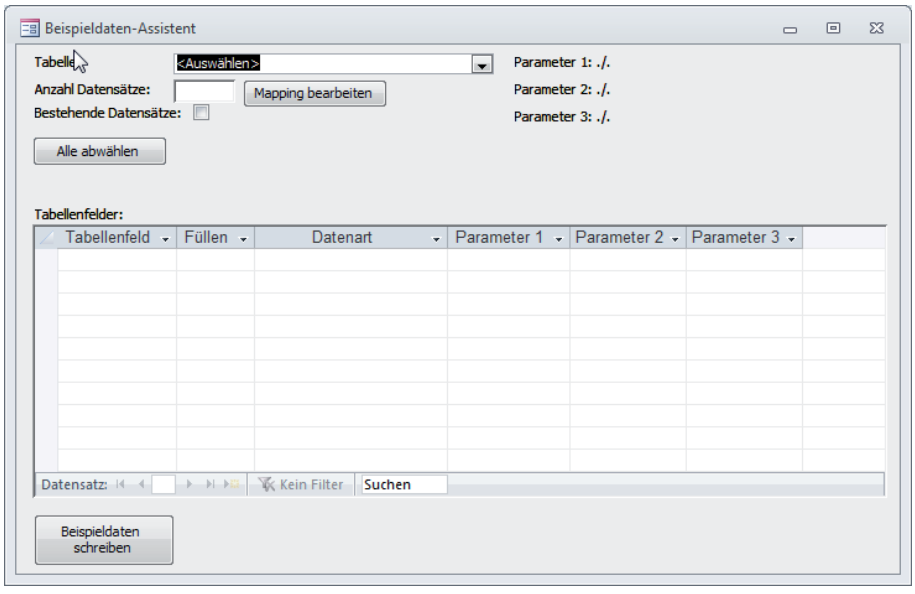


Bild 5: Beispieldaten-Assistent gleich nach dem Aufruf

Beispieldaten-Assistent einsetzen

Den Beispieldaten-Assistent starten Sie nun über einen Klick auf den entsprechenden Menü- beziehungsweise Ribbon-Eintrag. Der Assistent erscheint zunächst wie in Bild 5. Die wenigen Bedienelemente sind schnell erklärt: Mit dem Kombinationsfeld oben wählen Sie eine der Tabellen der Datenbank aus, beispielsweise **tblKunden**. Unmittelbar danach liest der Beispieldaten-Assistent die Felder dieser Tabelle ein, listet diese unter **Tabellenfelder** auf und

Early Binding und Late Binding

Wenn Sie per VBA mit Objekten wie Word, Excel, Outlook interagieren oder auf die Elemente externe Bibliotheken zugreifen möchten, die Objekte, Eigenschaften, Methoden und Ereignisse bereitstellen, können Sie dies auf zwei Arten erledigen: mit Early Binding oder mit Late Binding. Dieser Artikel erklärt die Vor- und Nachteile dieser beiden Vorgehensweisen anhand des Zugriffs auf eine Excel-Tabelle.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1209_EarlyUndLateBinding.mdb**.

Beispiel

Das Beispiel für diesen Artikel ist einfach: Wir wollen einfach nur eine Excel-Instanz erstellen, ein neues Dokument anlegen und in der ersten Zelle der ersten Tabellen des Dokuments einen Wert eintragen. Dies erledigen auf zwei Arten: mit Early Binding und mit Late Binding. Beides bezieht sich auf die Art, wie Sie die Excel-Instanz deklarieren instanzieren und die im VBA-Code verwendeten Objekte deklarieren.

Early Binding

Beim Early Binding fügen Sie dem VBA-Projekt der Datenbank-Datei einen Verweis auf die Bibliothek **Microsoft Excel x.0 Object Library** hinzu. Dazu öffnen Sie den VBA-Editor (zum Beispiel mit **Alt + F11**), betätigen den Menübefehl **Extras|Verweise** und finden dann den **Verweise**-Dialog vor. Dort finden Sie, wenn Sie Office beziehungsweise Excel installiert haben, einen entsprechenden Eintrag in der Liste der verfügbaren Verweise vor und markieren diesen (siehe Bild 1).

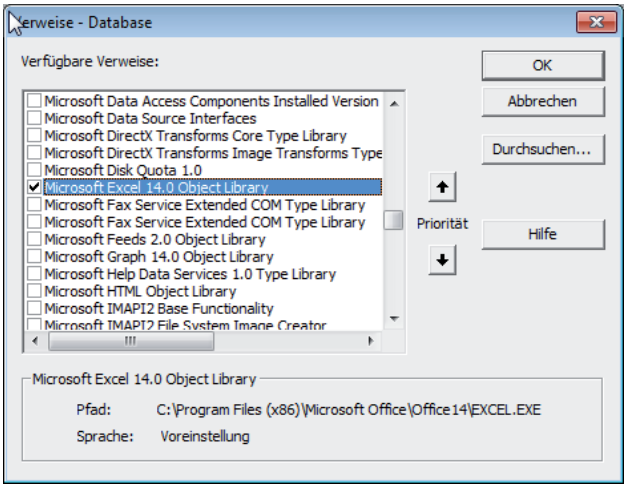


Bild 1: Hinzufügen eines Verweises auf die Excel-Bibliothek

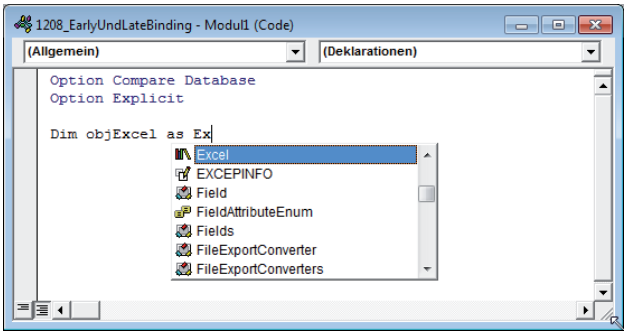


Bild 2: Beim Early Binding stehen die Elemente der referenzierten Bibliothek per IntelliSense zur Auswahl bereit

Damit teilen Sie dem VBA-Editor mit, dass Sie die in dieser Bibliothek definierten Objekte, Eigenschaften, Methoden und Ereignisse verfügbar machen wollen. Im VBA-Editor macht sich das Hinzufügen dieses Verweises zum Beispiel dadurch bemerkbar, dass das Excel-Objekt und seine Elemente beim Deklarieren verfügbar sind.

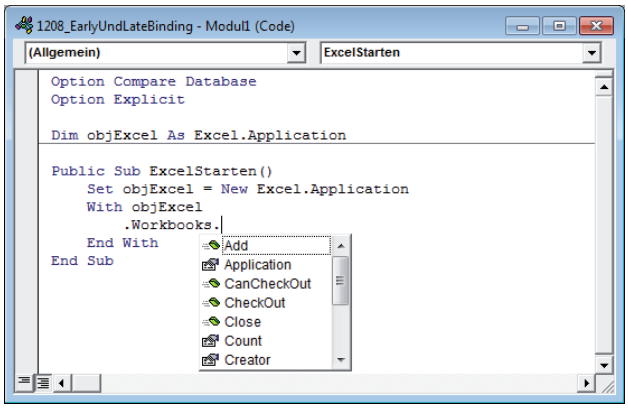


Bild 3: Auch die Eigenschaften und Methoden der Objektvariablen werden per IntelliSense angezeigt.

Dies probieren wir in einem neuen Standardmodul aus, das Sie mit dem Menübefehl **Einfügen|Modul** anlegen – speichern Sie dieses unter dem Namen **mdlExcel**.

Wenn Sie dort gleich oben eine Deklarationszeile wie die folgende Anlegen, bietet IntelliSense gleich das **Excel**-Objekt zur Auswahl an (siehe Bild 2):

```
Dim objExcel As Excel.Application
```

Auch die übrigen Elemente dieser Bibliothek stehen nun zur Deklaration bereit. Doch damit nicht genug: Wenn Sie nun ein Objekt auf Basis der Klasse **Excel.Application** erstellen, können Sie auch noch per IntelliSense auf seine Eigenschaften und Methoden zugreifen (siehe Bild 3).

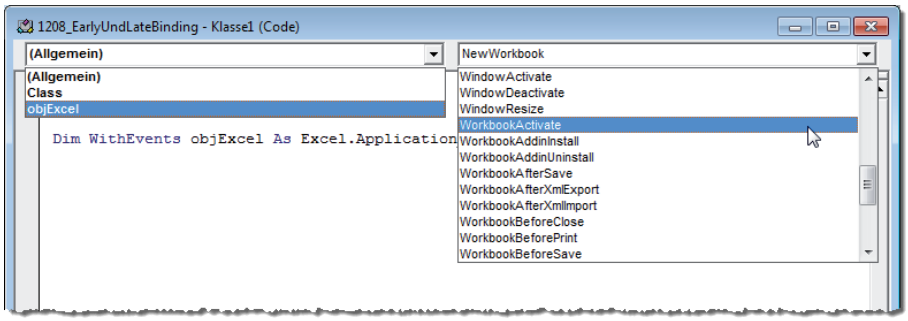


Bild 4: Einsatz von Ereignissen eines mit **WithEvents** deklarierten Objekts

Wenn Sie das **Excel**-Objekt in einem Klassenmodul deklarieren, können Sie dort sogar noch das Schlüsselwort **WithEvents** mit angeben (siehe Klassenmodul **clsExcel** – anlegen mit **Einfügen|Klassenmodul**, speichern unter **clsExcel**):

```
Dim WithEvents objExcel As Excel.Application
```

Sie können dann, genau wie bei den Ereignissen von Formularen oder den darin enthaltenen Steuerelementen, Ereignisprozeduren definieren, die bei bestimmten Ereignissen des entsprechenden Objekts ausgelöst werden.

Nur gibt es in diesem Fall kein Eigenschaftsfenster, mit dem Sie den Wert der Ereigniseigenschaft auf **[Ereignisprozedur]** festlegen und dann per Mausklick auf die Schaltfläche mit den drei Punkten die entsprechende Ereignisprozedur anlegen. Stattdessen erledigen Sie dies, indem Sie im linken Kombinationsfeld des Code-Fensters des entsprechenden Klassenmoduls das Objekt auswählen (das zwingend mit dem Schlüsselwort **WithEvents** deklariert sein muss) und dann im rechten Fenster das zu implementierende Ereignis selektieren (siehe Bild 4).

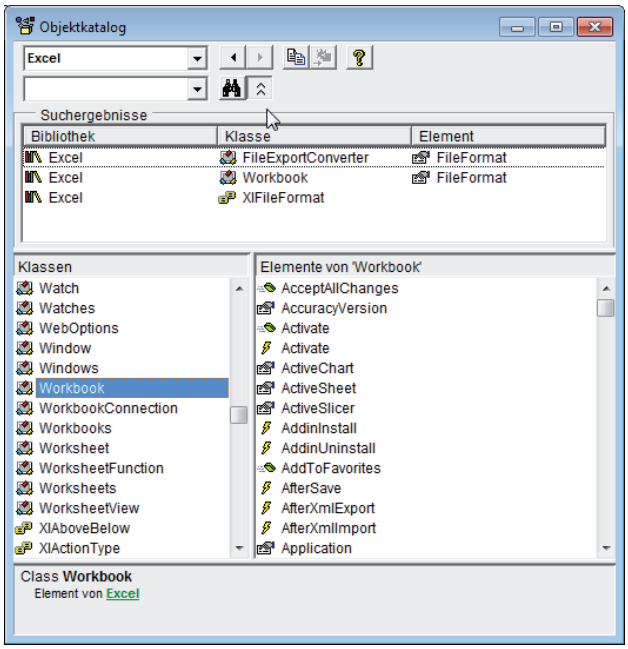


Bild 6: Der Objektkatalog bietet Einblick in alle Elemente, Eigenschaften, Methoden und Ereignisse von Objekten

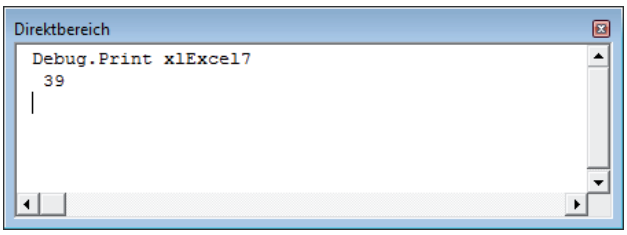


Bild 5: Ausgabe des tatsächlichen Wertes einer Konstanten der Excel-Bibliothek

Schließlich bietet der Einsatz einer per Early Binding referenzierten Bibliothek für viele Einsatzfälle Konstanten an, welche die eigentlich als Parameter zu verwendenden Zahlen besser lesbar machen.

Ein Beispiel ist das Format, unter dem ein frisch erstelltes Excel-Workbook gespeichert werden soll. In den folgenden Zeilen finden Sie die **SaveAs**-Methode des aktuellen Workbooks, die als zweiten und letzten Parameter die Konstante **xlExcel7** verwendet:

```
Set objExcel = New Excel.Application
With objExcel
    .Workbooks.Add
    .ActiveCell.Value = "A1"
    .ActiveWorkbook.SaveAs CurrentProject.Path _
        & "\test.xls", xlExcel7
    .Visible = True
End With
```

Welchen Wert diese Konstante tatsächlich hat, können Sie beispielsweise herausfinden, indem Sie sich den Wert im Direktfenster ausgeben lassen – dies gelingt allerdings nur, wenn ein Verweis auf die Excel-Bibliothek existiert (siehe Bild 5).

ListView-Steuerelement: Daten bearbeiten

In einem gewissen Rahmen bietet das ListView-Steuerelement auch die Möglichkeit, Daten zu bearbeiten. Dazu können Sie beispielsweise den Inhalt der ersten Spalte anklicken, den Eintrag ändern und die Änderung übernehmen. Außerdem können Sie mit dem ListView-Steuerelement ein Kontrollkästchen vor jedem Eintrag anzeigen, um die Einträge auszuwählen oder damit ein Ja/Nein-Feld abzubilden. Wie dies gelingt, zeigt der vorliegende Artikel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1209_ListView.mdb**.

Erste Spalte ändern

Im folgenden Beispiel soll der Benutzer direkt im ListView-Steuerelement die in der ersten Spalte angezeigten Daten ändern können. Das setzt voraus, das die erste Spalte auch Daten anzeigt, die man am ehesten schnell mal ändern möchte. Die Folgerung ist, dass die erste Spalte nun nicht mehr den Primärschlüsselwert der zugrunde liegenden Tabelle anzeigen kann. Dieser soll aber in den meisten Fällen ohnehin nicht erscheinen.

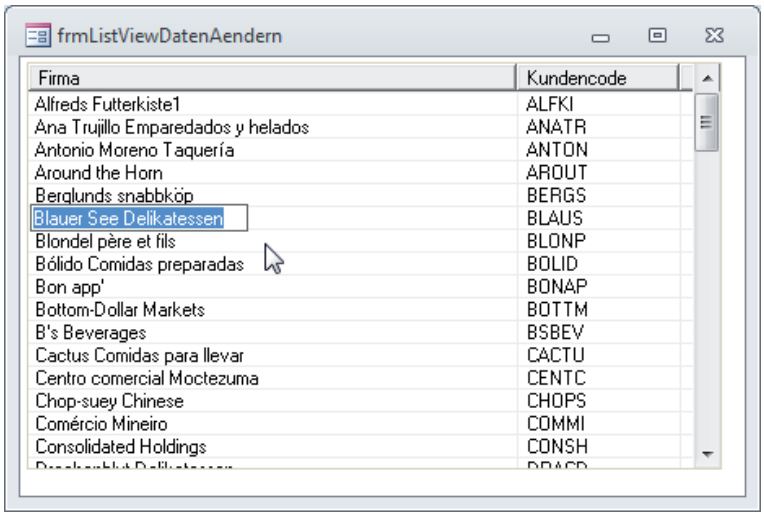


Bild 1: Ändern der in der ersten Spalte angezeigten Daten

Um den Wert des in der ersten Spalte angezeigten Wertes zu ändern, soll der Benutzer diesen einfachen anklicken, damit er wie etwa ein Dateiname im Windows Explorer zur manuellen Bearbeitung vorbereitet wird (siehe Bild 1). Der Benutzer ersetzt dann den vorhandenen Text durch den gewünschten Text und bestätigt die Eingabe durch das Betätigen der Eingabetaste oder anderweitigem Verschieben des Fokus auf ein anderes Element des Formulars.

Damit das Formular **frmListViewDatenAendern** mit den angezeigten Daten der Tabelle **tblKunden** gefüllt wird, deklarieren wir zunächst eine Objektvariable, mit der wir auf die Eigenschaften und Methoden des **ListView**-Steuerelements zugreifen, im Klassenmodul **Form_frmListViewDatenAendern**:

```
Dim objLvwKunden As MSComctlLib.ListView
```

Dieses wird im Ereignis **Form_Load** mit den benötigten Eigenschaften versehen (weitere Informationen hierzu in den Artikeln **Das ListView-Steuerelement – Grundlagen** und **ListView-Steuerelement mit Daten füllen**). Die Prozedur **Form_Load** legen Sie an, indem Sie die Formulareigenschaft **Beim Laden** auf **[Ereignisprozedur]** einstellen und dann auf die Schaltfläche mit den drei Punkten klicken. Füllen Sie die nun erscheinende Prozedur im VBA-Editor wie folgt auf, um die Basiseinstellungen und Spalteneigenschaften vorzunehmen:

```
Private Sub Form_Load()
    Set objLvwKunden = Me!lvwKunden.Object
    With objLvwKunden
        .View = lvwReport
        .Appearance = ccFlat
        .BorderStyle = ccNone
        .FlatScrollBar = False
        .GridLines = True
        .ColumnHeaders.Clear
        .ColumnHeaders.Add , , "Firma", 4500
        .ColumnHeaders.Add , , "Kundencode", 1500
        .ListItems.Clear
    End With
    ListViewFuellen objLvwKunden
End Sub
```

Das Füllen des **ListView**-Steuerelements mit den Daten übernimmt die folgende Prozedur. Diese wird von der Prozedur **Form_Load** aufgerufen und liest alle Datensätze der Tabelle **tblKunden** in das Listenfeld ein. Fügen Sie diese Prozedur ebenfalls in das Klassenmodul **Form_frmListViewDatenAendern** ein (siehe Listing 1).

```
Private Sub ListViewFuellen(objLvwKunden As ListView)
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim objListitem As MSComctlLib.ListItem
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT KundeID, Firma, Kundencode FROM tblKunden", dbOpenDynaset)
    Do While Not rst.EOF
        Set objListitem = objLvwKunden.ListItems.Add(, "k" & rst!KundeID, rst!Firma)
        With objListitem
            .ListSubItems.Add , "p1" & rst!KundeID, rst!Kundencode
        End With
        rst.MoveNext
    Loop
    rst.Close
    Set rst = Nothing
    Set db = Nothing
End Sub
```

Listing 1: Füllen des **ListView**-Steuerelements mit den Daten der Tabelle **tblKunden**

Inhalt der ersten Spalte ändern

Wenn Sie den Inhalt der ersten Spalte des Listenfeldes ändern möchten, ist dies kein Problem – es muss lediglich die Eigenschaft **LabelEdit** entweder im Eigenschaften-Dialog des **ListView**-Steuerelements im Entwurf auf **0-lvwAutomatic** einzustellen (siehe Bild 2) oder dies mit der folgenden Zeile in der Prozedur **Form_Load** durchzuführen:

```
.LabelEdit = lvwAutomatic
```

Verwenden Sie den Wert **1-lvwManual**, kann der Wert in der ersten Zeile nicht im Bearbeitungsmodus angezeigt werden.

Wenn die Eigenschaft **LabelEdit** den Wert **lvwAutomatic** hat, können Sie den Wert der ersten Spalte nach einmaligem Anklicken eines bereits markierten Wertes ändern (sonst müssen Sie zweimal mit etwas zeitlichem Abstand klicken – genau wie beim Ändern von Einträgen im Windows Explorer).

Nach dem Ändern und dem Bestätigen der Änderung etwa mit der Eingabetaste wird diese übernommen.

Allerdings nicht in die zugrunde liegende Tabelle: Das Listenfeld ist schließlich nicht an die Tabelle gebunden und weiß dementsprechend auch gar nicht, welches Feld mit dem neuen Wert gefüllt werden sollte.

Also müssen Sie, wie bereits beim Füllen des ListView-Steuerelements, auch für eine einfache Aktion wie dem Ändern eines Feldinhalts eine VBA-Prozedur hinzuziehen.

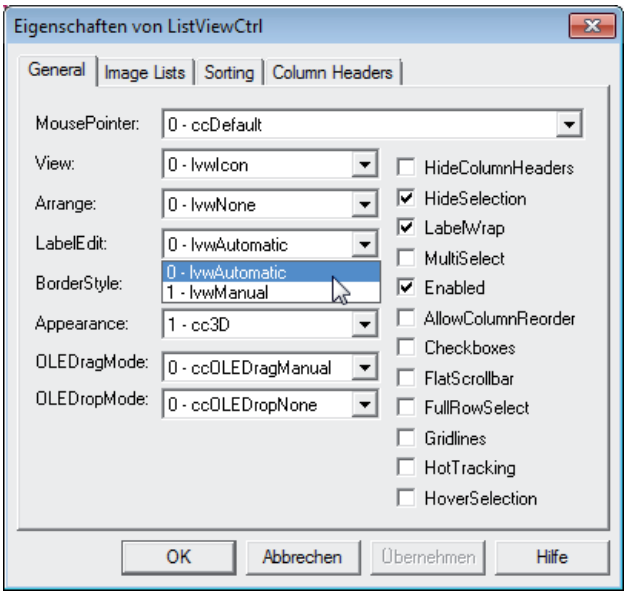


Bild 2: Eigenschaft für das Verhalten beim Ändern der ersten Spalte

Ereignisse des ListView-Steuerelements nutzen

Dazu nutzen wir erstmals in der lockeren Folge von Artikeln zum Thema **ListView** eine Ereignisprozedur des **ListView**-Steuerelements. Einen Überblick über die verfügbaren Ereignisse erhalten Sie, wie bereits erwähnt, nicht wie bei den übrigen Steuerelementen über die Registerseite **Ereignis** des Eigenschaftsfensters in der Entwurfsansicht. Das heißt: Eigentlich zeigt dieses schon einige Eigenschaften an, aber nicht die speziell für das **ListView**-Steuerelement verfügbaren (siehe Bild 3).

Um diese zu erhalten, müssen Sie in den VBA-Editor wechseln. Dort wählen Sie im linken Kombinations-

Datensätze individuell auswählen

Es gibt verschiedene Gelegenheiten, zu denen Datensätze nicht nach verschiedenen, auf den enthaltenen Feldern basierenden Kriterien selektiert werden können. In einem solchen Fall ist die individuelle Selektion der Datensätze gefragt. Dazu gibt es verschiedene Varianten. Dieser Artikel zeigt, wie Sie Daten eines Unterformulars in der Datenblattansicht mithilfe eines Kontrollkästchens zur Auswahl hinzufügen oder daraus entfernen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1210_DatensaetzeAuswaehlen.mdb**.

Beispieldaten

Als Beispieldaten verwenden wir die Tabelle **tblKunden** der Beispieldatenbank. Diese Tabelle enthält allgemeine Kundendaten wie Name, Anschrift, Telefonnummern und die E-Mail-Adresse. Den Entwurf dieser Tabelle entnehmen Sie Bild 1.

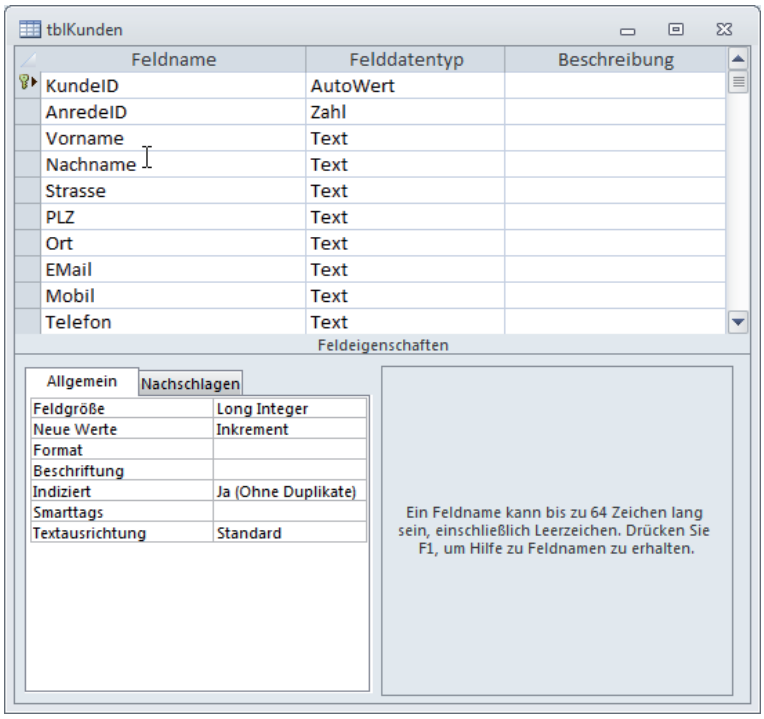


Bild 1: Beispieltabelle für die Auswahl von Datensätzen

Ausgewählte Datensätze markieren

Während Sie die gewünschten Datensätze in den meisten Fällen durch den Einsatz entsprechender Filterkriterien ermitteln, ist dies hier nicht der Fall.

Wie aber sollen die individuell ermittelten Datensätze markiert werden? Der einfachste Weg ist, einfach ein Feld etwa namens **Markiert** und mit dem Felddatentyp **Ja/Nein** zum Tabellenentwurf hinzuzufügen.

Das Ergebnis sieht dann in der Datenblattansicht wie in Bild 2 aus. Der Benutzer kann nun die gewünschten Datensätze markieren und etwa in einer Abfrage alle Datensätze zutage liefern, deren Feld **Markiert** den Wert **True** enthält.

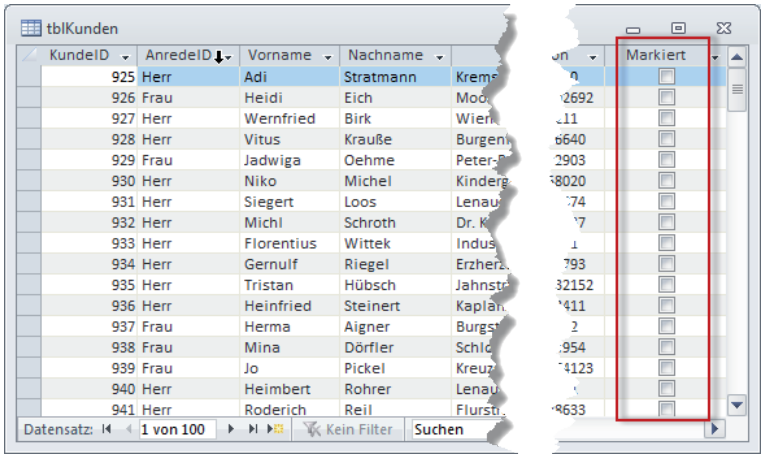


Bild 2: Datenblattansicht mit **Markiert**-Feld

Nun soll der Benutzer aber nicht direkt mit den Tabellen arbeiten, sondern in einem Formular, und dieses soll sowohl die Möglichkeit zum Markieren der Daten als auch zum Ausführen bestimmter Aktionen auf Basis dieser Daten bieten.

Also erstellen Sie zwei Formulare, wobei das Hauptformular **frmDatenauswahl** das Unterformular **sfmDatenauswahl** aufnimmt. Das Unterformular wiederum verwendet die Tabelle **tblKunden** als Datenherkunft und zeigt all ihre Felder in der Datenblattansicht an.

In der Entwurfsansicht sehen die beiden Formulare zunächst wie in Bild 3 aus. Um dies zu erreichen,

erstellen Sie zunächst das Unterformular **sfmDatenauswahl**, weisen der Eigenschaft **Datenherkunft** die Tabelle **tblKunden** zu, stellen die Eigenschaft **Standardansicht** auf **Datenblatt** ein und fügen die anzuzeigenden Felder zur Detailansicht des Formularentwurfs hinzu.

Speichern Sie das Unterformular unter dem Namen **sfmDatenauswahl**. Erstellen Sie dann das Hauptformular **frmDatenauswahl** und ziehen Sie das Unterformular aus dem Datenbankfenster (bis Access 2003) beziehungsweise dem Navigationsbereich (ab Access 2007) in den Entwurf des Hauptformulars.

Da wir im Hauptformular außerdem keine Navigationsschaltflächen, Bildlaufleisten, Trennlinien oder Datensatzmarkierer benötigen, stellen Sie die entsprechenden Eigenschaften jeweils auf den Wert **Nein** ein. Außerdem erhält die Eigenschaft **Automatisch zentrieren** den Wert **Ja**.

Das Ergebnis zeigt sich dann nach dem Wechsel in die Formularansicht des Hauptformulars (siehe Bild 4). Um die Nutzbarkeit zu erhöhen, verschieben Sie das Feld zum Markieren der Datensätze ganz nach vorn.

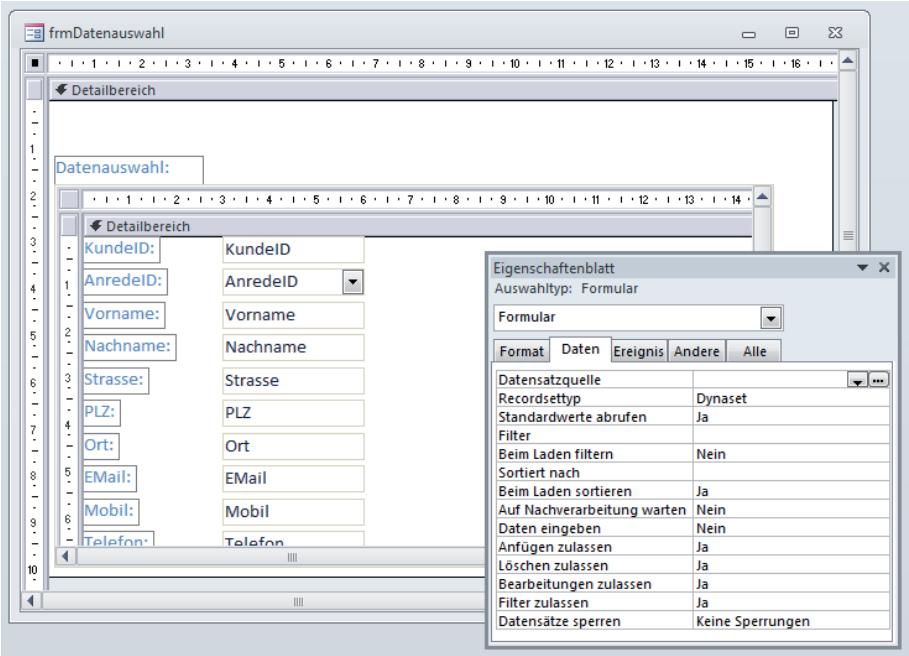


Bild 3: Haupt- und Unterformular der Beispieldatenbank

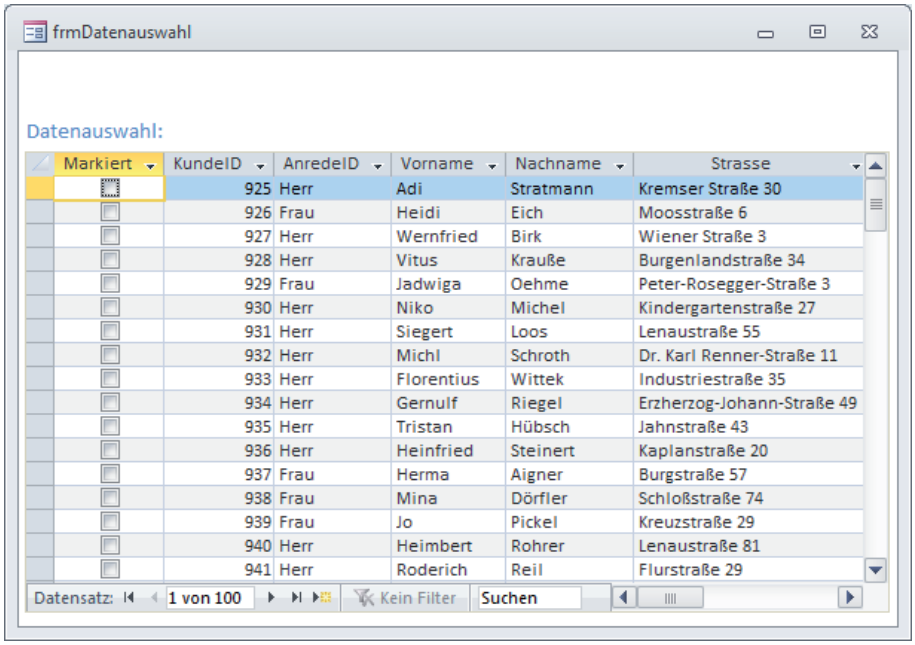


Bild 4: Anzeige der auszuwählenden Datensätze im Formular **frmDatenauswahl**

Dies gelingt ganz einfach, indem Sie den Spaltenkopf des Feldes anklicken und diesen per Drag and Drop nach vorn verschieben.

Damit sind Sie grundsätzlich fertig – die Datensätze können nun markiert und über eine entsprechende Abfrage für verschiedene Aktionen wie etwa den Versand einer Rechnung, einer E-Mail et cetera ausgelesen werden.

Aus- und Abwählen aller Datensätze

So leicht wollen wir es uns allerdings nicht machen: Etwas mehr Komfort als in der Datenblattansicht der Tabelle könnte die Lösung schon gebrauchen. Also fügen wir zunächst Schaltflächen hinzu, mit denen Sie alle Datensätze auswählen und mit denen Sie alle Einträge wieder abwählen können.

Diese ordnen Sie etwa so wie in Bild 5 an. Benennen Sie die Schaltflächen **cmdAlleAuswaehlen** und

Synchrone Unterformulare

Es kommt oft vor, dass ein Unterformular Daten anzeigen soll, die mit dem im Hauptformular angezeigten Datensatz in Beziehung stehen. Dies geschieht beispielsweise, wenn das Hauptformular Kunden und das Unterformular die dem Kunden zugeordneten Projekte anzeigt. Es geht jedoch auch noch eine Stufe weiter: Dann soll ein Unterformular Daten in Abhängigkeit vom Hauptformular anzeigen und ein weiteres Daten, die vom aktuellen Datensatz des Unterformulars abhängen. Wie Sie dies bewerkstelligen, zeigt dieser Artikel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1210_SynchroneUnterformulare.mdb**.

Beispieldaten

Als Beispieldaten verwendet dieser Artikel die Tabellen der Südsturm-Datenbank, einer abgewandelten Variante der Nordwind-Datenbank. Bild 1 zeigt alle Tabellen, deren Daten im Beispielformular angezeigt werden sollen. Das Hauptformular soll dabei die Kundendaten aus der Tabelle **tblKunden** liefern und dabei jeweils einen Datensatz anzeigen.

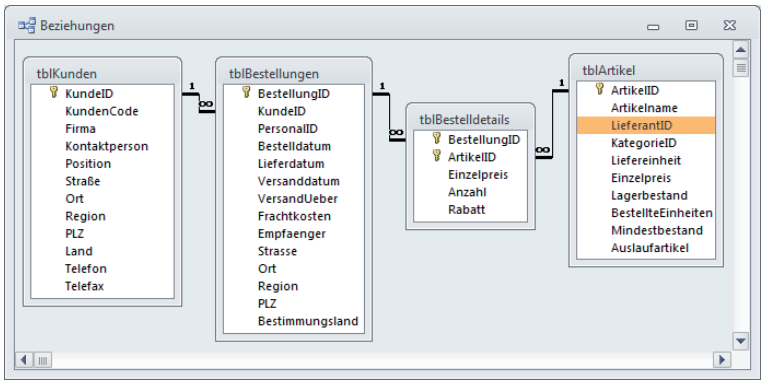


Bild 1: Diese Tabellen sollen im Formular angezeigt werden.

Ein erstes Unterformular zeigt alle Datensätze der Tabelle **tblBestellungen** an, die zum aktuell im Hauptformular angezeigten Datensatz der Tabelle **tblKunden** passen, deren Fremdschlüsselfeld **KundeID** also den Wert des Feldes **KundeID** des Hauptformulars enthält.

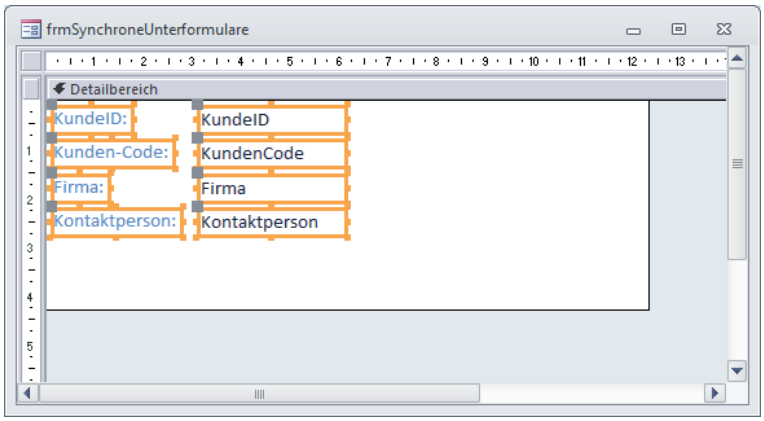


Bild 2: Ausgangsbasis: Das Formular **frmSynchroneUnterformulare** mit Daten aus der Tabelle **tblKunden**

Außerdem wollen wir noch die Bestelldetails zur jeweils im ersten Unterformular ausgewählten Bestellung sehen, und zwar in einem zweiten Unterformular.

Dieses zeigt prinzipiell nur Daten der Tabelle **tblBestelldetails** an, wobei das Nachschlagefeld **ArtikelID** seine angezeigten Daten (hier den Artikelnamen) aus der Tabelle **tblArtikel** bezieht.

Hauptformular frmSynchroneUnterformulare

Das Hauptformular wird in der Formularansicht als Standardansicht angezeigt. Es verwendet die Tabelle **tblKunden** als Datenherkunft, wobei wir der Übersicht halber nur einige wenige Felder dieser Tabelle im Entwurf platzieren (siehe Bild 2).

Unterformular sfmBestellungen

Das Unterformular zur Anzeige der Bestellungen soll **sfmBestellungen** heißen. Es verwendet die Tabelle **tblBestellungen** als Datenherkunft. Fügen Sie etwa die Felder **BestellungID**, **PersonalID**, **Bestelldatum** und **Lieferdatum** zum Detailbereich des Formularentwurfs hinzu (siehe Bild 3). Stellen Sie außerdem die Eigenschaft **Standardansicht** auf den Wert **Datenblatt** ein, damit das Formular die Daten in tabellarischer Form anzeigt.

Schließen Sie das Unterformular und vergeben Sie beim Speichern den Namen **sfmBestellungen**. Öffnen Sie das Hauptformular **frmSynchroneUnterformular** in der Entwurfsansicht und ziehen Sie den Ein-

trag **sfmBestellungen** aus dem Datenbankfenster (bis Access 2003) beziehungsweise dem Navigationsbereich (ab Access 2007) in den Entwurf des Formulars **frmSynchroneUnterformulare**.

Das Ergebnis sieht schließlich wie in Bild 4 aus. Das Formular **sfmBestellungen** wird in ein Unterformular-Steuerelement eingebettet und wird in der Entwurfsansicht innerhalb des Hauptformulars angezeigt.

Wichtig ist an dieser Stelle, dass das Eigenschaftsfenster für das Unterformular-Steuerelement die beiden Eigenschaften **Verknüpfen von** und **Verknüpfen nach** automatisch mit dem Ausdruck **KundeID** gefüllt wird. Die Eigenschaften des Unterformular-Steuerelements werden gern mit denen des im Unterformular-Steuerelement enthaltenen Formulars verwechselt. Damit Sie gleich die richtigen Eigenschaften prüfen können, klicken Sie an einer beliebigen Stelle in das Hauptformular (nur nicht auf das Unterformular) und wählen dann aus dem Kombinationsfeld mit der Liste der Elemente im Eigenschaftsfenster den Eintrag aus, der mit dem Namen des eingefügten Unterformulars übereinstimmt (hier **sfmBestellungen**). Nach der Auswahl erscheinen auf der Registerseite **Daten** die gewünschten Eigenschaften.

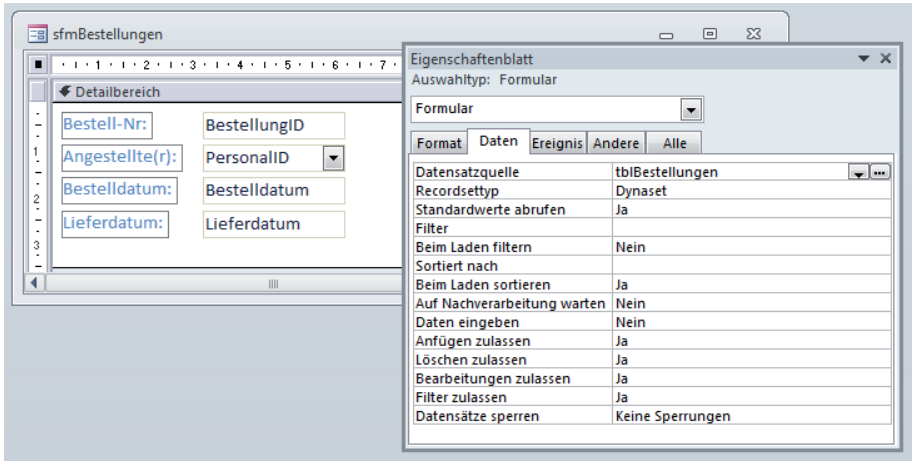


Bild 3: Das erste Unterformular zeigt die Daten der Tabelle **tblBestellungen** an.

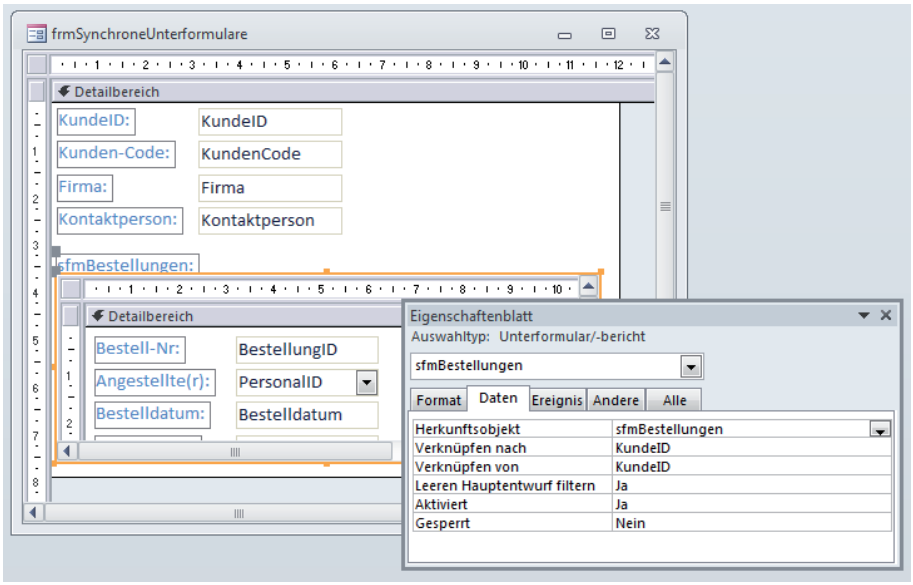


Bild 4: Einbau des ersten Unterformulars

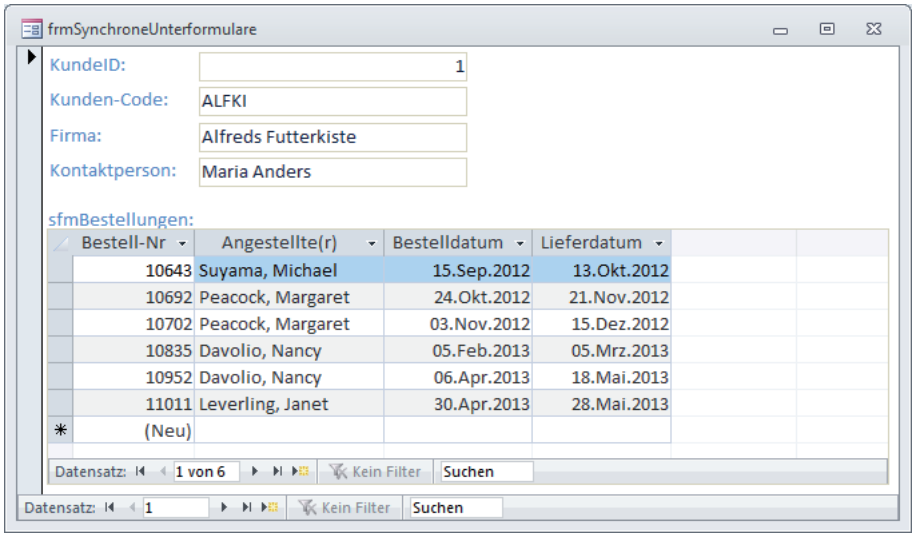


Bild 5: Haupt- und Unterformular in Aktion

Access hat hier automatisch erkannt, dass das Unterformular ein Fremdschlüsselfeld enthält, dass mit dem im Hauptformular enthaltenen Primärschlüsselfeld verknüpft ist und eine ent-

Internetmarke mit Access nutzen

Im Artikel CSV-Export für Adressen haben wir uns mit der Vorbereitung von Adressdaten für den Einsatz mit dem Versanddienstleister Hermes beschäftigt. Dieser versendet allerdings nur Pakete und zweitens wollen wir uns hier ja nicht einseitig mit dem Angebot eines einzelnen Unternehmens beschäftigen. Deshalb und weil ich damit gerade einige Erfahrungen sammeln durfte, schauen wir uns im vorliegenden Artikel die Möglichkeiten an, die Internetmarke der Deutschen Post für unsere Zwecke einzusetzen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1210_Internetmarke.mdb**.

Internetmarke nutzen

Die Internetmarke ist ein relativ neues Angebot der Deutschen Post, um online verschiedene Medien mit Porto zu bedrucken. Wer einen solchen Dienst von Access aus nutzt, der hat nicht nur im Sinn, gelegentlich eine Briefmarke auszudrucken, sondern wird auf Basis der Adressen in seiner Datenbank eine oder mehrere Adressetiketten erstellen wollen.

Das Ergebnis wird in Form einer ein- oder mehrseitigen PDF-Datei geliefert. In meinem Fall wurden Etiketten mit den Maßen 62 x 100 mm für einen Brother-Etikettendrucker benötigt. Das Ergebnis sah beispielsweise wie in Bild 1 aus.

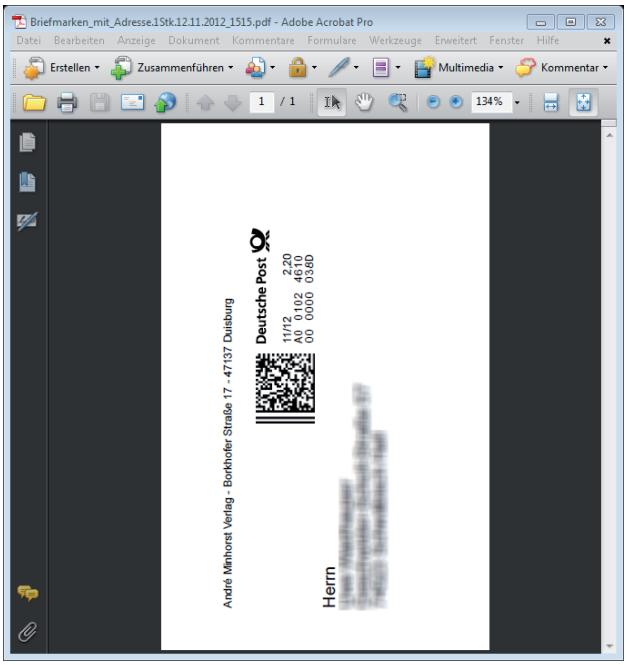


Bild 1: Beispiel für ein Adressetikett

Dazu sind folgende Dinge nötig:

- Eine Abfrage, welche die Adressdaten in einem geeigneten Format liefert,
- das auf Basis dieser Abfrage exportierte Ergebnis in Form einer **.csv**-Datei,
- eine Portokasse bei der deutschen Post sowie
- ein Zugang zum Angebot **Internetmarke**.

Wir kümmern uns in diesem Artikel zunächst um die Gestaltung der zur Übergabe von Adressen benötigten **.csv**-Datei, schauen uns dann die die Übergabe dieser Datei an das Online-Angebot der Post an und erstellen dann schließlich die gewünschte **.csv**-Datei mit Access.

Ich habe bislang zwei verschiedene Formate für die Übergabe von Adressdaten zum Erstellen von Internetmarken ausprobiert. Das erste bezieht sich auf reguläre deutsche oder ausländische Empfänger, das zweite auf Sendungen, die an Packstationen geschickt werden sollen.

Es gibt noch zwei weitere Formate für den Versand an ein Postfach und für den Versand an Großempfänger – mehr Informationen hierzu finden Sie unter **https://internetmarke.deutschepost.de/internetmarke/content/help.do**.

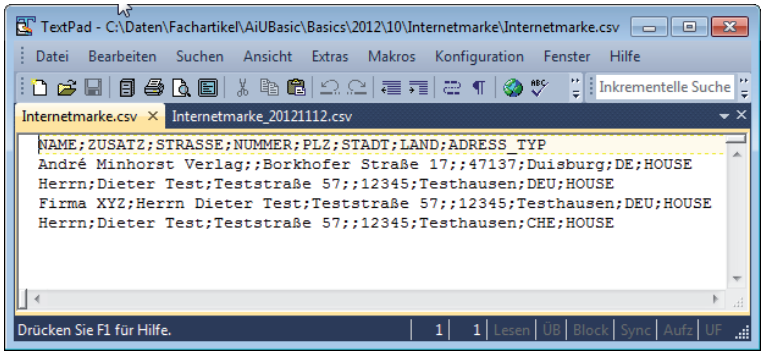
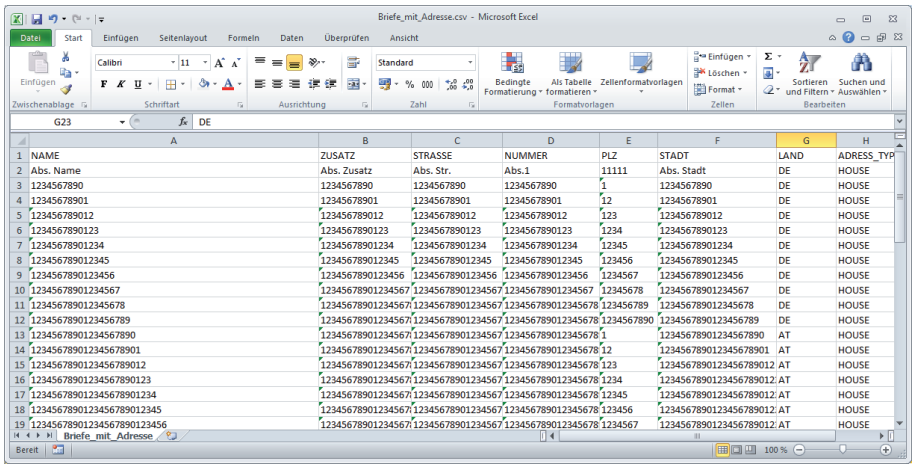


Bild 2: .csv-Datei für den Upload von Adressdaten

Die erste Variante ist wie in Bild 2 aufgebaut. Die einzelnen Zeilen enthalten in dieser Reihenfolge folgende Informationen:

- Feldnamen (eine Zeile),
- Daten für den Versender (eine Zeile) und
- Empfängerdaten (eine oder mehrere Zeilen).



	A	B	C	D	E	F	G	H
1	NAME	ZUSATZ	STRASSE	NUMMER	PLZ	STADT	LAND	ADRESS_TYP
2	Abs. Name	Abs. Zusatz	Abs. Str.	Abs.1	11111	Abs. Stadt	DE	HOUSE
3	1234567890	1234567890	1234567890	1234567890	1	1234567890	DE	HOUSE
4	12345678901	12345678901	12345678901	12345678901	12	12345678901	DE	HOUSE
5	123456789012	123456789012	123456789012	123456789012	123	123456789012	DE	HOUSE
6	1234567890123	1234567890123	1234567890123	1234567890123	1234	1234567890123	DE	HOUSE
7	12345678901234	12345678901234	12345678901234	12345678901234	12345	12345678901234	DE	HOUSE
8	123456789012345	123456789012345	123456789012345	123456789012345	123456	123456789012345	DE	HOUSE
9	1234567890123456	1234567890123456	1234567890123456	1234567890123456	1234567	1234567890123456	DE	HOUSE
10	12345678901234567	12345678901234567	12345678901234567	12345678901234567	12345678	12345678901234567	DE	HOUSE
11	123456789012345678	123456789012345678	123456789012345678	123456789012345678	123456789	123456789012345678	DE	HOUSE
12	1234567890123456789	1234567890123456789	1234567890123456789	1234567890123456789	1234567890	1234567890123456789	DE	HOUSE
13	12345678901234567890	12345678901234567890	12345678901234567890	12345678901234567890	1	12345678901234567890	AT	HOUSE
14	123456789012345678901	123456789012345678901	123456789012345678901	123456789012345678901	12	123456789012345678901	AT	HOUSE
15	1234567890123456789012	1234567890123456789012	1234567890123456789012	1234567890123456789012	123	1234567890123456789012	AT	HOUSE
16	12345678901234567890123	12345678901234567890123	12345678901234567890123	12345678901234567890123	1234	12345678901234567890123	AT	HOUSE
17	123456789012345678901234	123456789012345678901234	123456789012345678901234	123456789012345678901234	12345	123456789012345678901234	AT	HOUSE
18	1234567890123456789012345	1234567890123456789012345	1234567890123456789012345	1234567890123456789012345	123456	1234567890123456789012345	AT	HOUSE
19	12345678901234567890123456	12345678901234567890123456	12345678901234567890123456	12345678901234567890123456	1234567	12345678901234567890123456	AT	HOUSE

Bild 3: Test-CSV-Datei zur Ermittlung der Zeichenzahl je Feld

Die zulässige Länge der Daten für die einzelnen Felder, die übrigens durch Semikola voneinander getrennt werden, entsprach in der Praxis nicht den Vorgaben in der Hilfe.

Damit Sie zuverlässige Informationen erhalten, haben wir geprüft, welche Felder wieviele Zeichen aufnehmen können. Dabei hilft eine **.csv**-Datei, die verschieden lange Einträge je Feld enthält (siehe Bild 3).

Mit dieser Testdatei öffnen wir nun das Internetangebot der Post (**http://www.post.de**) und klicken dann links auf **Briefmarken kaufen|Internetmarke**.

Es erscheint der Dialog aus Bild 4. Hier können Sie sich, sofern Sie bereits über einen Zugang verfügen, einloggen (oben über den Link **Login**). Dies ist vorteilhaft, da einige zuletzt verwendete Einstellungen wie etwa das Papierformat übernommen werden. Danach klicken Sie auf **Internetmarken für|Briefe**.

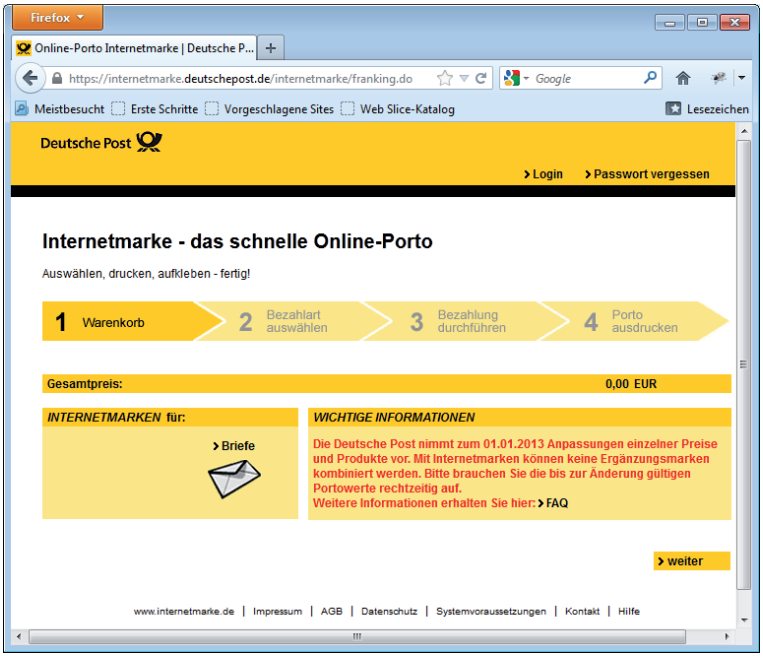


Bild 4: Startpunkt beim Erstellen von Internetmarken

Dies zeigt eine andere Ansicht des vorherigen Schrittes an (siehe Bild 5). Sie können hier nun einfach das Porto drucken lassen, aber das ist ja nicht unser Ziel. Stattdessen klicken Sie auf Adresse hinzufügen, um weitere Optionen zu erhalten – zum Beispiel über den Eintrag **Adress-Datenimport** (siehe Bild 6).

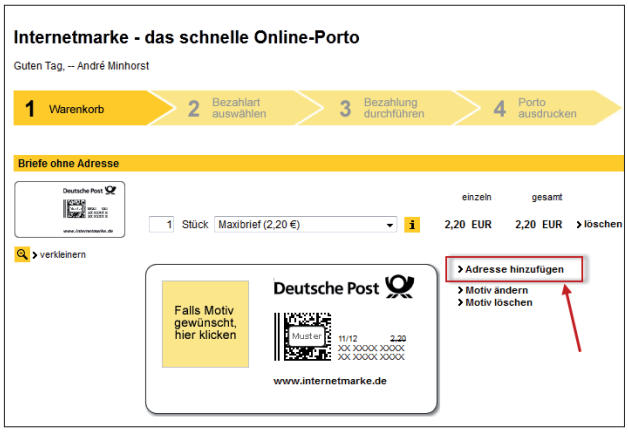


Bild 5: Einblenden der Befehle zum Hinzufügen von Adressen

Klicken Sie auf diesen Eintrag, erscheint ein HTML-Popup, das einen Dateiauswahl-Dialog anbietet, um die **.csv**-Datei auf Ihrem Rechner hochzuladen.

Nach der Auswahl klicken Sie auf **Datei hochladen**, um die Daten auf den Server der Post zu übertragen (siehe Bild 7).

Nach dem Hochladen zeigt ein Dialog das Ergebnis an. Sollten Fehler aufgetreten sein (was bei unserer Testdatei zu erwarten war), zeigt der Dialog diese nun an (siehe Bild 8).

Leider gibt es keine explizite Fehlerbeschreibung, sondern nur den Hinweis auf den jeweils falschen Eintrag. Bei der Testdatei haben wir beispielsweise die Werte **1**, **12**, **123** und so weiter für das Feld **PLZ** angegeben.

Ist gleichzeitig Deutschland als Empfängerland festgelegt, führen alle Werte, die nicht aus fünf Ziffern bestehen, zu einem Fehler. Bei ausländischen Adressen ist der Dienst etwas lässiger: hier können Sie auch andere Werte eintragen. Sie müssen also schon genau kontrollieren, ob die PLZ zum angegebenen Länderkennzeichen passt.

Eigentlich wollten wir mit der Testdatei ermitteln, wie lang die einzelnen Einträge für die verschiedenen Felder sein dürfen. Das Ergebnis sieht so aus:

- Name: 50 Zeichen
- Zusatz: 50 Zeichen
- Straße: 35 Zeichen
- Nummer: 10 Zeichen
- PLZ: je nach Land, maximal 59 Zeichen
- Ort: 35 Zeichen

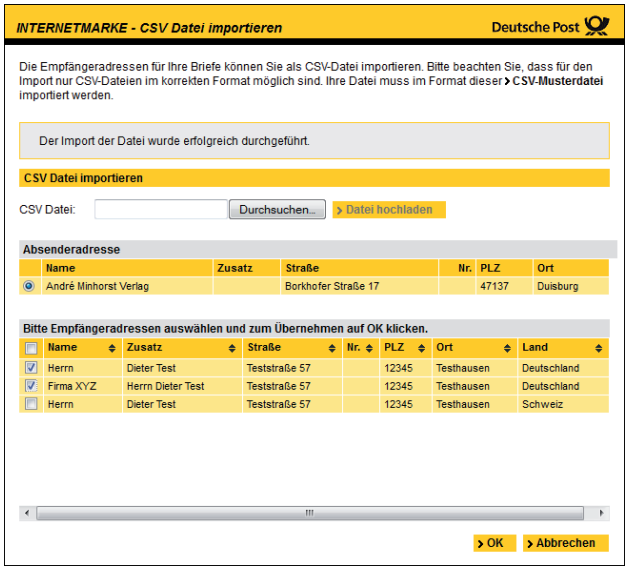


Bild 9: Auswählen der Adressaten

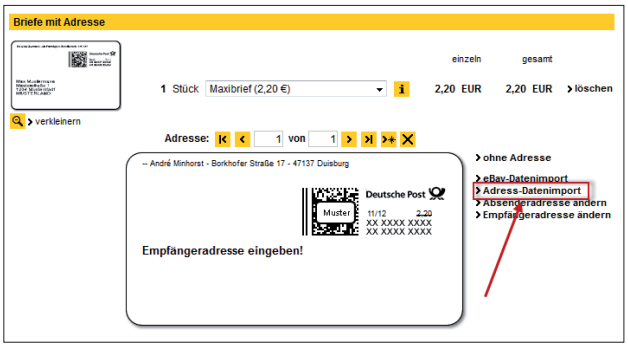


Bild 6: Hinzufügen von Adressen per Datenimport

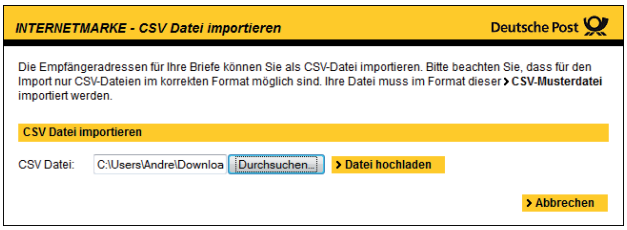


Bild 7: Auswahl einer Datei zum Hochladen

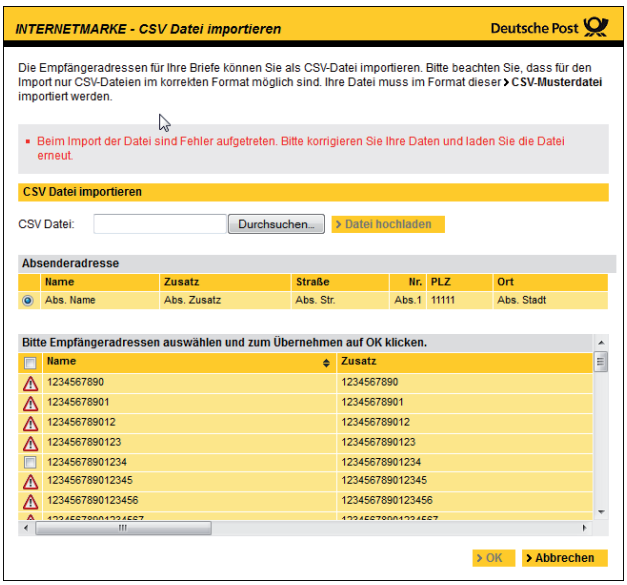


Bild 8: Prüfen des Ergebnisses des Uploads

Länderkennzeichen

Die Länderkennzeichen sind zweistellig. Eine Liste der Kennzeichen (zum Beispiel DE für Deutschland) finden Sie unter folgendem Link:

`http://de.wikipedia.org/wiki/ISO-3166-1-Kodierliste`

Etiketten erstellen

Davon ausgehend, dass alle Einträge in Ihrer **.csv**-Datei korrekt sind, können Sie nun die gewünschten Einträge auswählen. Ein Tipp vorneweg: Wenn Sie Sendungen mit verschiedenen Beträgen frankieren

möchten, wählen Sie zunächst die Adressen für den ersten Betrag aus (siehe Bild 9). Nachdem Sie den Dialog mit **OK** geschlossen haben, können Sie sich die Vorschau für die Etiketten ansehen.

Legen Sie dann oben fest, ob Sie einen Standardbrief, einen Kompaktbrief oder eine andere Sendung frankieren möchten. Geben Sie dort außerdem die Anzahl ein – wobei diese bei der Erstellung von Frankierungen für mehrere Kunden wohl **1** lauten dürfte.

Außerdem müssen Sie noch das Papierformat auswählen. Hier ist standardmäßig **DIN A4 Normalpapier** eingestellt. Wenn Sie auf ein anderes Medium drucken möchten, klicken Sie rechts auf **Papierformat ändern**.

Die Angabe des anfänglich angegebenen Mediums wird in ein Auswahlfeld umgewandelt, sodass Sie eines der verfügbaren Formate auswählen können. Es gibt eine ganze Reihe Formate – in meinem Fall etwa (siehe Screenshot oben) kam der Eintrag **Brother DK-11202 Versand-Etikett 62 x 100** zum Einsatz.

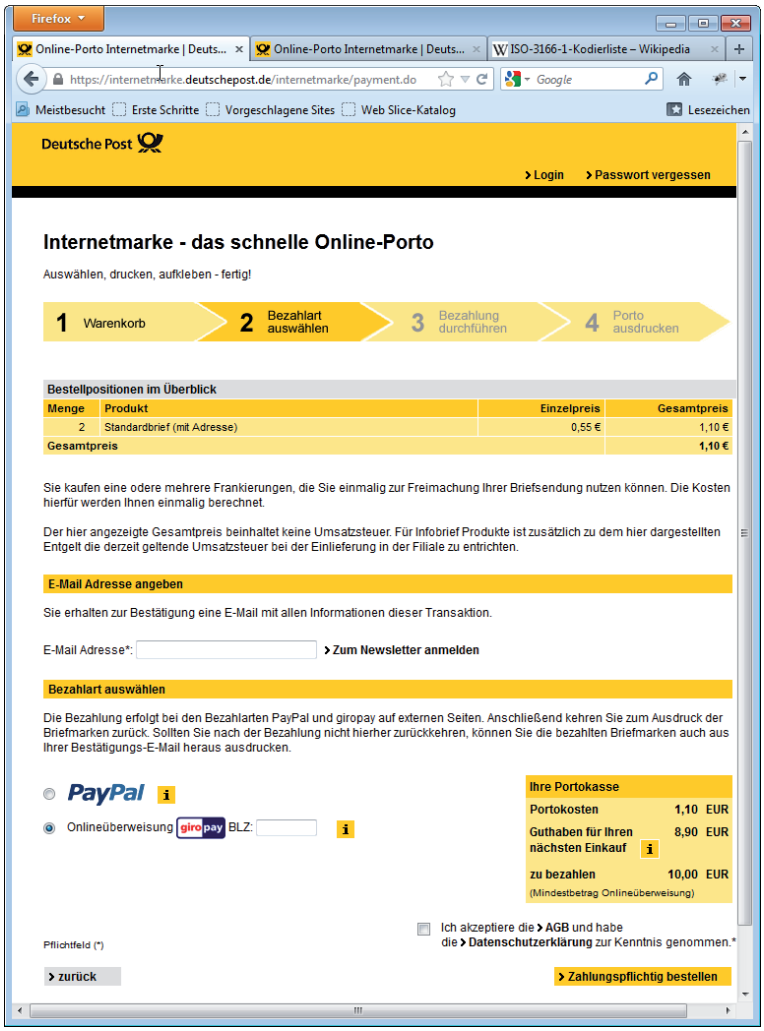


Bild 10: Übernahme der Daten und Angabe der Zahlungsmethode

Sie können im gleichen Zuge auch noch einen Testdruck ausführen, indem Sie auf den Link **kostenloser Testdruck der ersten Seite** klicken. In diesem Fall erhalten Sie die Möglichkeit, ein entsprechendes PDF-Dokument herunterzuladen, zu betrachten und gegebenenfalls testweise auszudrucken.

Weitere Frankierungen

Wenn Sie verschiedene Frankierungen mit einem Bezahlvorgang abwickeln möchten, klicken Sie unter den Daten für die soeben angelegten Frankierungen wieder auf den Eintrag **Internetmarken für|Briefe**. Es erscheint erneut der oben bereits vorhandene Bereich, sodass Sie nun erneut die Adressen einlesen, die Adressaten auswählen und die Portoart sowie das Papierformat festlegen können.

Bezahlart

Danach klicken Sie auf die Schaltfläche **Weiter** und gelangen zum zweiten Schritt (siehe Bild 10). Hier erhalten Sie eine Übersicht über die Bestellung. Wenn Sie noch nicht angemeldet sind, finden Sie hier nur die beiden Bezahlarten **PayPal** und **Onlineüberweisung** vor. Wenn Sie sich einloggen, erscheint außerdem noch der Eintrag **Lastschrift**.

Gegebenenfalls haben Sie bereits eine Portokasse bei der Post eingerichtet. Diese können Sie mit dem gewünschten Betrag aufladen, beispielsweise per Lastschrift. Sollte das aktuell bestellte Porto durch das Guthaben der Portokasse gedeckt sein, können Sie dieses gleich bestellen. Anderenfalls laden Sie die Portokasse auf. Das Einrichten der Portokasse und weitere Details der Bezahlung klammern wir an dieser Stelle aus Platzgründen jedoch aus.

Wichtig ist nur Folgendes: Sollten Sie mal in die Verlegenheit kommen, kurzfristig sehr viel Porto zu drucken, könnten Sie scheitern: Sie können nämlich nur Porto in Höhe eines begrenzten Betrags pro Zeitraum ordern. Sollten Sie mehr benötigen, können Sie

das Budget jedoch per Anruf bei der Hotline erhöhen lassen.

Und noch ein wichtiger Hinweis: Nach meiner Erinnerung (meine Anmeldung liegt nun bereits eine Weile zurück) erhält man nicht umgehend Zugriff auf die Portokasse. Gegebenenfalls müssen Sie dann eine der übrigen Bezahlarten wie PayPal oder Onlineüberweisung nutzen.

Porto herunterladen

Schließlich folgt noch der letzte Schritt des Porto-Assistenten (siehe Bild 11). Hier können Sie zwei Dokumente herunterladen: erstens ein PDF-Dokument mit dem Porto (im Beispiel für den Ausdruck eines Etiketts von 62 x 100 mm) sowie den Lieferschein.

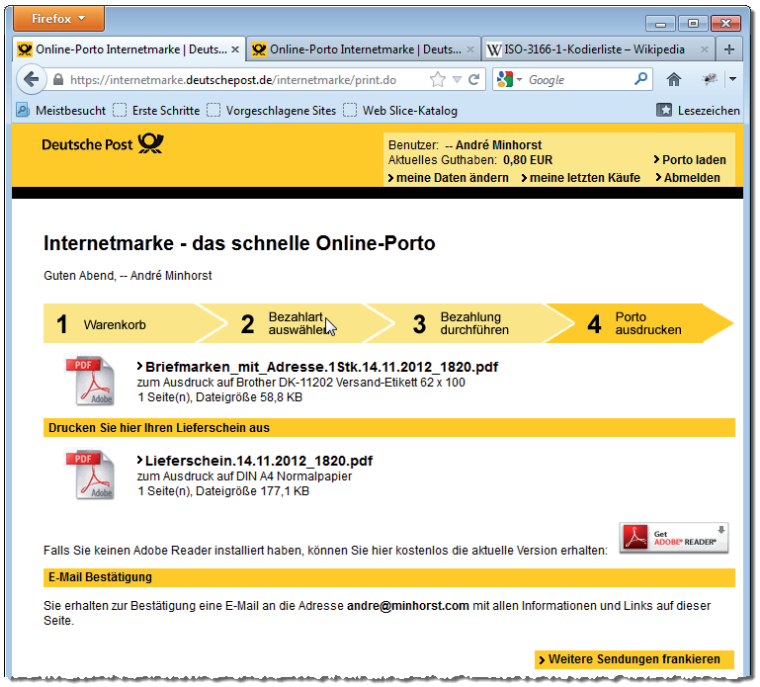


Bild 11: Download des Portos und des Lieferscheins

Nach dem Herunterladen des Portos können Sie dieses mit dem gewünschten Drucker ausdrucken und weiterverwerten.

Adressdaten bereitstellen

Nun kommt der Access-Part dieses Artikels: Sie wissen nun zwar, wie Sie auf Basis einer fertigen **.csv**-Datei einen Porto-Ausdruck erstellen können. Es fehlt allerdings noch ein Weg, aus einer üblichen Adresstabelle die benötigten Daten zu extrahieren und in Form einer **.csv**-Datei bereitzustellen.

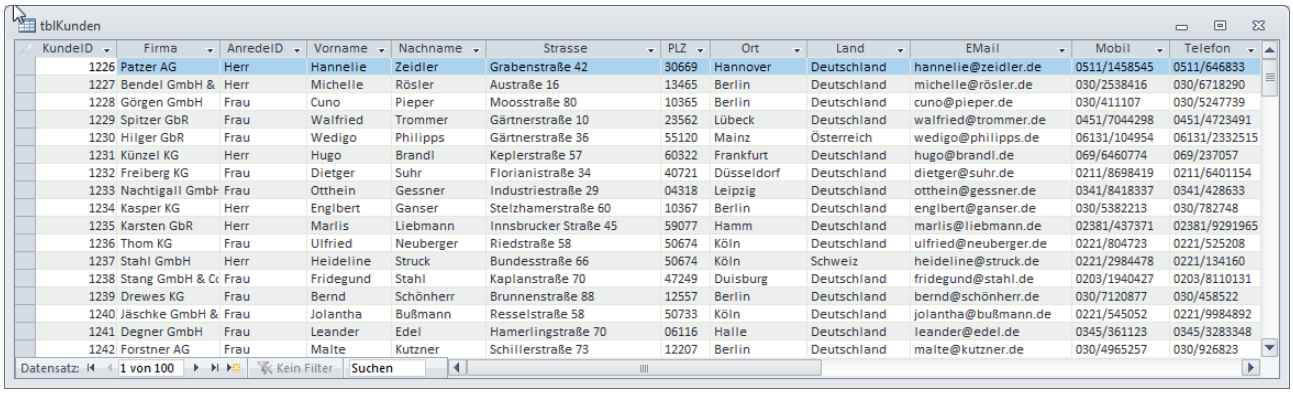
Die Beispieldatenbank liefert die Adressen in einer Tabelle namens **tblKunden**, wobei diese noch über das Feld **AnredeID** mit der Tabelle **tblAnreden** verknüpft ist (siehe Bild 12).

Länderkürzel

Die Länderkürzel könnten Sie in einem weiteren Feld der Tabelle **tblKunden** speichern, allerdings würde dies redundante Daten liefern.

Also erstellen Sie eine weitere Tabelle namens **tblLaender**, welche die drei Felder **LandID** (Primärschlüsselfeld mit Autowert), **Land** und **Kuerzel** enthält (siehe Bild 13).

In der gleich zu erstellenden Abfrage werden wir die Tabelle **tblKunden** über das Feld **Land** mit dem gleichnamigen Feld der Tabelle **tblLaender** verknüpfen, um das jeweilige Länderkürzel zu ermitteln


tblKunden

KundeID	Firma	AnredeID	Vorname	Nachname	Strasse	PLZ	Ort	Land	EMail	Mobil	Telefon
1226	Patzer AG	Herr	Hannelie	Zeidler	Grabenstraße 42	30669	Hannover	Deutschland	hannelie@zeidler.de	0511/1458545	0511/646833
1227	Bendel GmbH &	Herr	Michelle	Rösler	Austraße 16	13465	Berlin	Deutschland	michelle@rösler.de	030/2538416	030/6718290
1228	Görgen GmbH	Frau	Cuno	Pieper	Moosstraße 80	10365	Berlin	Deutschland	cuno@pieper.de	030/411107	030/5247739
1229	Spitzer GbR	Frau	Walfried	Trommer	Gärtnerstraße 10	23562	Lübeck	Deutschland	walfried@trommer.de	0451/7044298	0451/4723491
1230	Hilger GbR	Frau	Wedigo	Philipps	Gärtnerstraße 36	55120	Mainz	Österreich	wedigo@philipps.de	06131/104954	06131/2332515
1231	Künzel KG	Herr	Hugo	Brandl	Keplerstraße 57	60322	Frankfurt	Deutschland	hugo@brandl.de	069/6460774	069/237057
1232	Freiberg KG	Frau	Dietger	Suhr	Florianistraße 34	40721	Düsseldorf	Deutschland	dietger@suhr.de	0211/8698419	0211/6401154
1233	Nachtigall GmbH	Frau	Otthein	Gessner	Industriestraße 29	04318	Leipzig	Deutschland	otthein@gessner.de	0341/8418337	0341/428633
1234	Kasper KG	Herr	Englbert	Ganser	Stelzhamerstraße 60	10367	Berlin	Deutschland	englbert@ganser.de	030/5382213	030/782748
1235	Karsten GbR	Herr	Marlis	Liebmann	Innsbrucker Straße 45	59077	Hamm	Deutschland	marlis@liebmann.de	02381/437371	02381/9291965
1236	Thom KG	Frau	Ulfried	Neuberger	Riedstraße 58	50674	Köln	Deutschland	ulfried@neuberger.de	0221/804723	0221/525208
1237	Stahl GmbH	Herr	Heideline	Struck	Bundesstraße 66	50674	Köln	Schweiz	heideline@struck.de	0221/2984478	0221/134160
1238	Stang GmbH & Co	Frau	Fridegund	Stahl	Kaplanstraße 70	47249	Duisburg	Deutschland	fridegund@stahl.de	0203/1940427	0203/8110131
1239	Drewes KG	Frau	Bernd	Schönherr	Brunnenstraße 88	12557	Berlin	Deutschland	bernd@schönherr.de	030/7120877	030/458522
1240	Jäschke GmbH &	Frau	Jolantha	Bußmann	Resselstraße 58	50733	Köln	Deutschland	jolantha@bußmann.de	0221/545052	0221/9984892
1241	Degner GmbH	Frau	Leander	Edel	Hamerlingstraße 70	06116	Halle	Deutschland	leander@edel.de	0345/361123	0345/3283348
1242	Forstner AG	Frau	Malte	Kutzner	Schillerstraße 73	12207	Berlin	Deutschland	malte@kutzner.de	030/4965257	030/926823

Datensatz: 1 von 100 | Kein Filter | Suchen

Bild 12: Die Tabelle **tblKunden** dient als Datenherkunft

Das Registersteuerelement

Mit dem Registersteuerelement können Sie Daten, die nicht gleichzeitig sichtbar sein müssen, platzsparend auf mehrere Registerseiten verteilen. Per Mausklick wechseln Sie dann zwischen den einzelnen Registerseiten und zeigen die gewünschten Steuerelemente an. Diese Artikel zeigt, wie Sie das Registersteuerelement anlegen, Registerseiten hinzufügen, entfernen und anordnen und diese mit Steuerelementen füllen. Außerdem lernen Sie die wichtigsten Einstellungen für das Aussehen des Registersteuerlements kennen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1211_Registersteuerelement.mdb**.

Registersteuerelement anlegen

Das Registersteuerelement finden Sie etwa unter Access 2010 im Ribbon unter **Entwurf|Steuerelemente|Registersteuerelement** (siehe Bild 1).

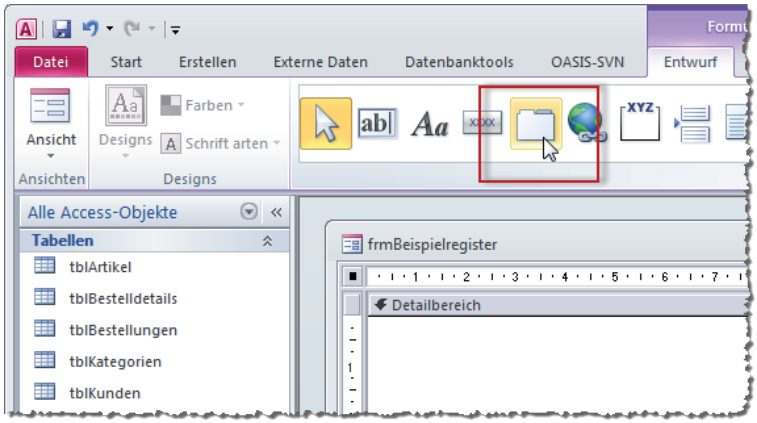


Bild 1: Einfügen eines Registersteuerelements

Nach dem Anklicken dieser Schaltfläche klicken Sie an die Stelle des Formulars in der Entwurfsansicht, an der sich die linke obere Ecke des Registersteuerelements befinden soll. Alternativ ziehen Sie einen Rahmen mit der gewünschten Größe auf. Das Ergebnis sieht dann wie in Bild 2 aus. Access legt gleich zwei Beispielseiten an, die Sie nach Wunsch weiterverarbeiten oder löschen können.

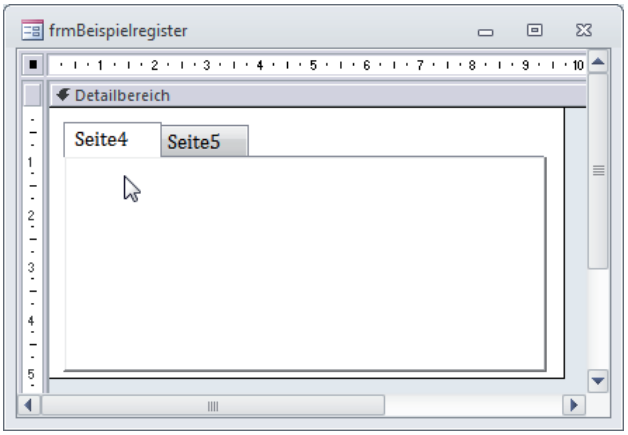


Bild 2: Ein frisch angelegtes Registersteuerelement

Werfen wir zunächst einen Blick auf einige wichtige Eigenschaften, die Sie im Eigenschaftsfenster des Registersteuerelements einstellen können. Als erstes passen wir den Namen des Steuerelements an, damit wir es später in Ausdrücken oder per VBA einfach referenzieren können.

Dazu müssen Sie das Steuerelement selbst erst einmal markieren. Dies gelingt durch einen Mausklick auf die Stelle rechts neben den Registerreitern oder durch einen Klick auf den entsprechenden Eintrag im Kombinationsfeld des Eigenschaftsfensters (etwa auf **RegisterStr3**).

Danach können Sie auf der Seite **Andere** des Eigenschaftsfensters zunächst den Namen des Registersteuerelements einstellen (siehe Bild 3) – zum Beispiel auf **tabBeispiel**.

Anschließend ändern wir gleich noch die Namen der einzelnen Registerseiten, bei denen es sich prinzipiell um eigene Steuerelemente handelt.

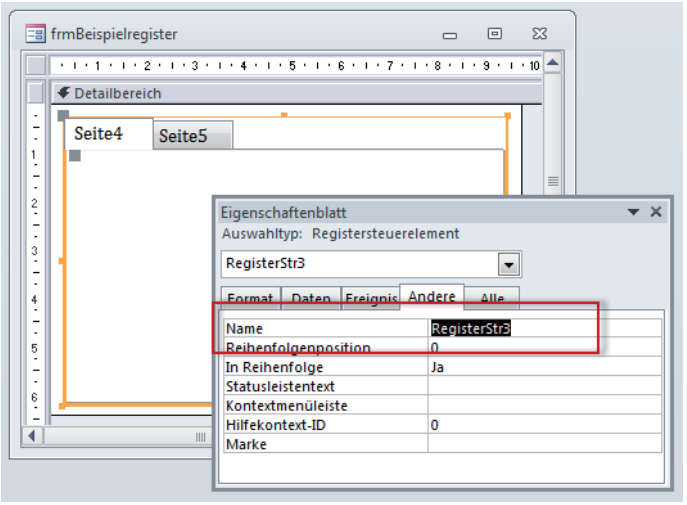


Bild 3: Einstellen des Namens des Registersteuerelements

Dazu klicken Sie jeweils auf den Registerreiter und stellen dann den Wert der Eigenschaft **Name** auf entsprechende Werte ein, zum Beispiel **pge1** und **pge2** (**pge** steht für **Page**).

Normalerweise sollten die Registerreiter nun gleichzeitig **pge1** und **pge2** als Beschriftung anzeigen. Das ist natürlich nicht gewünscht und rührt daher, dass die Eigenschaft Beschriftung der Registerseiten noch nicht angepasst wurde.

Dies können Sie ebenfalls im Eigenschaftsfenster erledigen. Für uns reichen vorerst die Beschriftungen **Beispielseite 1** und **Beispielseite 2** aus. Die Breite der Registerreiter passt Access automatisch an, darum brauchen Sie sich also nicht zu kümmern.

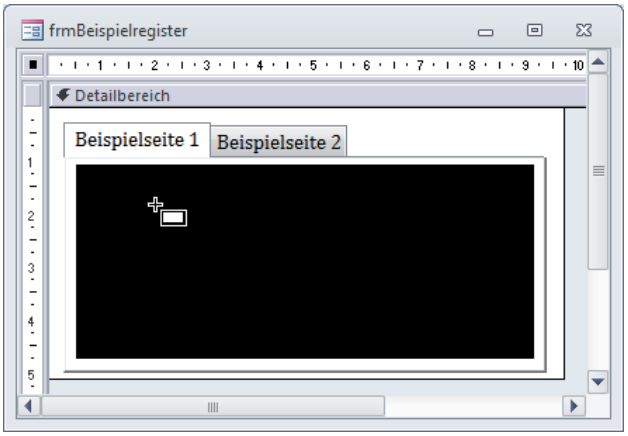


Bild 4: Hinzufügen eines Steuerelements

Steuerelemente hinzufügen

Anschließend fügen wir einige Steuerelemente hinzu, zum Beispiel Schaltflächen. Wann immer Sie ein Steuerelement zum Hinzufügen aus der entsprechenden Liste ausgewählt haben und anschließend mit der Maus über die eigentliche Registerseite fahren, wird diese dunkel hervorgehoben.

Sie können das Steuerelement dann über der Seite fallen lassen (siehe Bild 4).

Um zu prüfen, ob das Steuerelement auch auf der Registerseite gelandet ist, wechseln Sie kurz zur nächsten Seite – dort sollte es dann nicht erscheinen.

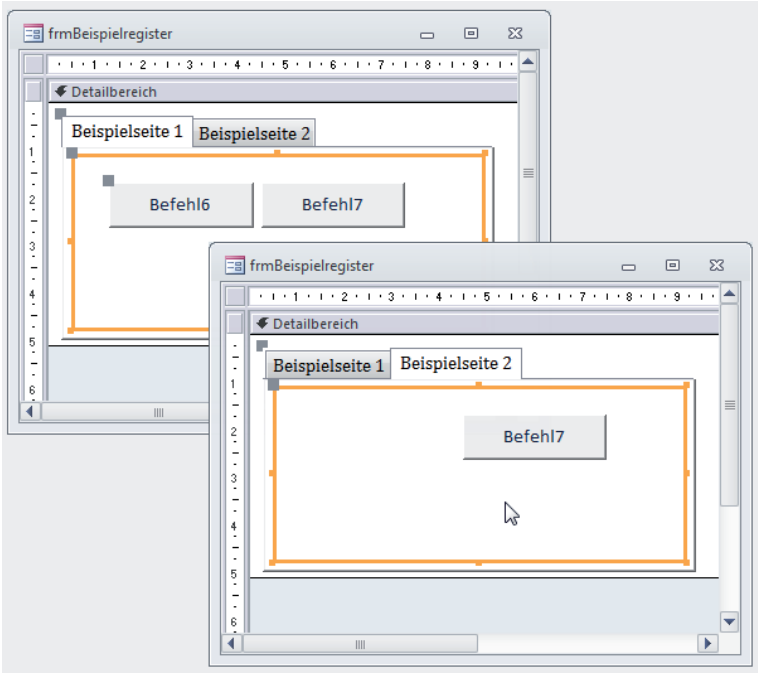


Bild 5: Die Schaltfläche liegt über dem Registersteuerelement, nicht darauf.

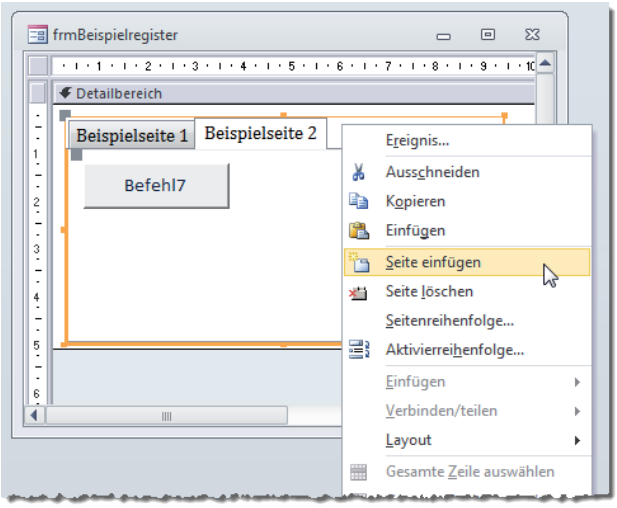


Bild 6: Hinzufügen einer Registerseite

Falls das trivial erscheint, probieren Sie mal Folgendes aus: Legen Sie eine neue Schaltfläche neben dem Registersteuerelement an und ziehen Sie es anschließend per Drag and Drop auf das Registersteuerelement.

Wenn Sie dann zur nächsten Registerseite wechseln, sollte die zuletzt angelegte Schaltfläche eigentlich verschwinden – tut sie aber nicht. Sie wird weiterhin angezeigt, weil Sie nicht auf der ersten Registerseite, sondern über dem Registersteuerelement angelegt wurde (siehe Bild 5).

Wenn Sie ein oder mehrere bestehende Steuerelemente auf eine Registerseite verschieben möchten, gelingt dies nur durch Ausschneiden und Einfügen.

ListView-Steuerelement: Kontrollkästchen nutzen

Im Gegensatz zum in Access eingebauten Listenfeld-Steuerelement kann das ListView-Steuerelement auch ein Kontrollkästchen vor jeder Zeile anzeigen. Das ist eine wirklich praktische Funktion, wenn Sie Datensätze für verschiedene Aktionen auswählen möchten oder einfach einen bestimmten Status darstellen wollen. Dieser Artikel zeigt, welche Varianten es zur Darstellung des Kontrollkästchens gibt und wie Sie dieses aktivieren oder deaktivieren und es an Daten eines Feldes binden.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1211_ListView.mdb**.

Kontrollkästchen anzeigen

Für das erste Beispiel fügen Sie einfach einem Formular ein **ListView**-Steuerelement hinzu und stellen seine Eigenschaft **Name** auf den Wert **lvwKontrollkaestchen** ein (siehe Formular **frmKontrollkaestchenEinfach** in der Beispieldatenbank).

Die Eigenschaften stellen wir, da sich das ListView-Steuerelement ohnehin nur per VBA-Code füllen lässt, direkt komplett per VBA ein – das Eigenschaftsfenster dieses Steuerelements lassen wir damit außen vor. Wenn Sie das Formular in der Entwurfsansicht öffnen, wählen Sie im Eigenschaftsfenster den Eintrag **Formular** aus, stellen für die Eigenschaft **Beim Laden** den Wert **[Ereignisprozedur]** ein und klicken auf die Schaltfläche mit den drei Punkten.

Die nun im VBA-Editor erscheinende leere Prozedur ergänzen Sie wie folgt und tragen außerdem die Deklarationszeile für das Objekt zum Referenzieren des ListView-Steuerelements oben im Modul ein:

```
Dim objLvwKontrollkaestchen As MSComctlLib.ListView

Private Sub Form_Load()
    Set objLvwKontrollkaestchen = _
        Me!lvwKontrollkaestchen.Object
    With objLvwKontrollkaestchen
        .View = lvwReport
        .Appearance = ccFlat
        .BorderStyle = ccNone
        .FlatScrollBar = False
        .GridLines = True
        .ColumnHeaders.Clear
        .Checkboxes = True
        .ColumnHeaders.Add , , "Listeneinträge", 5000
        .ListItems.Clear
        .ListItems.Add , , "Eintrag 1"
        .ListItems.Add , , "Eintrag 2"
        .ListItems.Add , , "Eintrag 3"
    End With
End Sub
```

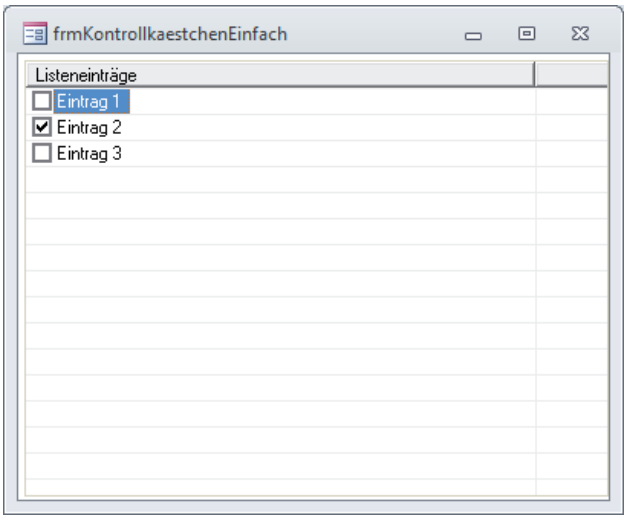


Bild 1: Kontrollkästchen und erstes Feld in einer Spalte

Ein Wechseln in die Formularansicht liefert nun das **ListView**-Steuerelement aus Bild 1. Die Prozedur stellt dazu beim Laden des Formulars zunächst einige Eigenschaften zur Definition des Aussehens ein und legt die Anzahl der Spalten sowie die Spaltenüberschrift und -breite fest.

Danach folgt bereits die entscheidende Eigenschaft: Mit **.Checkboxes = True** aktivieren Sie die Anzeige der Kontrollkästchen in der linken Spalte. Danach fügt die Prozedur noch drei Beispieleinträge zum **ListView**-Steuerelement hinzu.

Kontrollkästchen per Code aktivieren

Wenn Sie gleich beim Öffnen des Formulars einen Eintrag ankreuzen möchten, müssen Sie den betroffenen Eintrag mit einer wie folgt deklarierten Objektvariablen referenzieren:

```
Dim objListItem As MSComctlLib.ListItem
```

Dieser Variablen weisen Sie den Eintrag dann gleich beim Anlegen zu:

Aktionsabfragen per VBA

Mit Aktionsabfragen ändern Sie Daten – ganz im Gegensatz zu Auswahlabfragen, mit denen Sie Daten selektieren. Die Grundlagen zur Erstellung von Aktionsabfragen mit der Entwurfsansicht für Abfragen haben wir in früheren Artikeln bereits behandelt, aber wir haben uns noch nicht im Detail angesehen, wie Sie solche Abfragen unter VBA ausführen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1211_AktionsabfragenPerVBA.mdb**.

Varianten

Es gibt mehrere Möglichkeiten, eine Aktionsabfrage unter VBA auszuführen. Die zwei bekanntesten sind die folgenden: Die erste ist die **RunSQL**-Methode des **DoCmd**-Objekts. Diese erwartet als Parameter den Namen einer gespeicherten Aktionsabfrage oder den entsprechenden SQL-Ausdruck. Optional kann eine Transaktion angegeben werden, in der die Aktionsabfrage ausgeführt wird. Die zweite Variante ist die **Execute**-Methode des **Database**-Objekts der DAO-Bibliothek. Sie bietet ebenfalls einen Parameter an, dem Sie den SQL-Ausdrucks der auszuführenden Abfrage zuweisen – jedoch keine gespeicherten Abfragen, die Sie mit der Abfrage-Entwurfsansicht erstellt haben! Der zweite Parameter jedoch dient der Übergabe von Optionen.

Die **Execute**-Anweisung bietet wesentlich mehr Möglichkeiten als die **RunSQL**-Methode. Sie können beispielsweise Fehler abfragen, die beim Ausführen von Aktionsabfragen mit der **Execute**-Anweisung auftreten, oder Sie fragen die Anzahl der von der Aktionsabfrage betroffenen Datensätze ab. Schließlich können Sie auch noch den Primärschlüsselwert des zuletzt hinzugefügten Datensatzes ermitteln, sofern dieser mit einer Aktionsabfrage hinzugefügt wurde.

Einsatz der DoCmd.RunSQL-Methode

Die **DoCmd.RunSQL**-Methode ist prinzipiell mit dem Menübefehl zum Ausführen der aktuell im Datenbankfenster beziehungsweise im Navigationsbereich markierten Aktionsabfrage zu vergleichen. Der Unterschied ist, dass Sie bei der VBA-Anweisung noch den Namen der Anweisung angeben müssen. Um eine einfache Abfrage zum Ändern eines Datensatzes auszuführen, legen Sie diese zunächst wie in Bild 1 an und speichern sie unter dem Namen **qryLagerbestandErhoehen**. Die Abfrage erhöht den Bestand des Artikels mit dem Wert **1** im Feld **ArtikelID** um **1**.

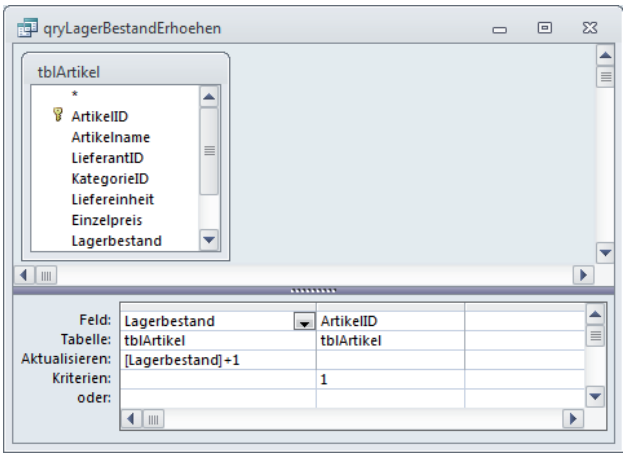


Bild 1: Aktionsabfrage zum Erhöhen des Lagerbestandes eines Artikels um 1

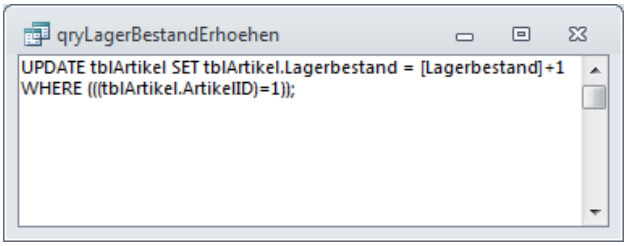


Bild 2: SQL-Ansicht der Abfrage **qryLagerbestandErhoehen**

Wechseln Sie dann in die SQL-Ansicht der Abfrage (am einfachsten durch einen Rechtsklick auf die Titelleiste der Abfrage und Auswahl des Eintrags **SQL-Ansicht** aus dem Kontextmenü) und kopieren Sie den dort angezeigten SQL-Ausdruck in die Zwischenablage (siehe Bild 2). Diese fügen Sie dann in Anführungszeichen als ersten Parameter der **RunSQL**-Methode im Direktfenster des VBA-Editors ein (**Strg + G**):

```
DoCmd.RunSQL "UPDATE tblArtikel SET Lagerbestand =
[Lagerbestand]+1 WHERE ArtikelID=1"
```

Das Ausführen dieser Abfrage durch Betätigen der Eingabetaste führt zum Anzeigen der Meldung aus Bild 3. Wenn Sie auf **Ja** klicken, wird der Datensatz geändert, anderenfalls löst dies einen Fehler aus.

Anhand eines weiteren Beispiels überzeugen wir uns noch davon, dass **RunSQL** tatsächlich nicht mit dem Namen einer gespeicherten Abfrage als Parameter

Dynamische Datenherkunft mit Formularbezug

Sicher haben Sie bereits einige Abfragen erstellt und als Datenherkunft von Formularen, Berichten oder Steuerelementen eingesetzt. Solange das jeweilige Element alle Datensätze der zugrunde liegenden Abfrage anzeigen soll, ist dies einfach. Wenn Sie die Daten jedoch zur Laufzeit nach bestimmten Kriterien filtern möchten, gibt es zahlreiche Möglichkeiten, dies zu tun. Dieser Artikel stellt einige davon vor.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1211_DynamischeDatenherkunftMitFormularbezug.mdb**.

Artikel filtern

Im ersten Beispiel sollen die Daten der Tabelle **tblArtikel** in einem Unterformular in der Datenblattansicht angezeigt werden. Um die Datensätze nach dem Artikelnamen zu filtern, enthält das Hauptformular ein Textfeld namens **txtArtikelsuche**. Wenn der Benutzer einen Text in dieses Feld eingegeben und die Schaltfläche **cmdSuchen** angeklickt hat, soll das Unterformular nur noch die passenden Datensätze anzeigen.

Dazu erstellen Sie zunächst ein geeignetes Unterformular. Diesem weisen Sie zuerst die Tabelle **tblArtikel** als **Datenherkunft** zu, damit Sie die Felder der Tabelle aus der Feldliste in den Formularentwurf ziehen können.

Das Ergebnis sieht etwa wie in Bild 1 aus. Stellen Sie außerdem die Eigenschaft **Standardansicht** des Formulars auf **Datenblatt** ein. Speichern Sie das Formular unter dem namen **sfmArtikelsuche** und schließen Sie es.

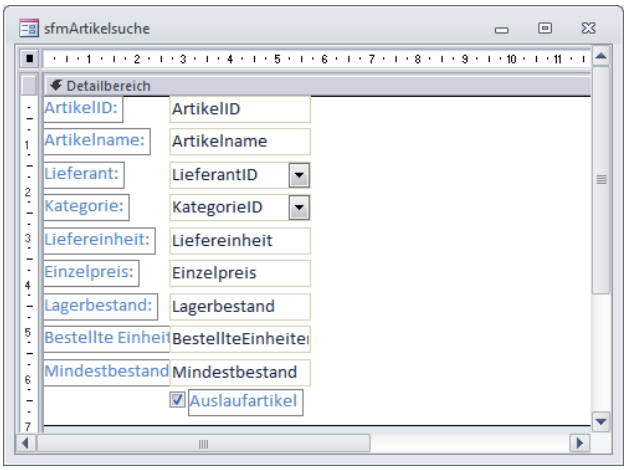


Bild 1: Zu filterndes Unterformular

Nun erstellen Sie das Hauptformular **frmArtikelsuche**. Fügen Sie diesem ein Textfeld namens **txtArtikelsuche** hinzu sowie eine Schaltfläche namens **cmdSuchen**.

Diese Schaltfläche soll den Inhalt des Unterformulars aktualisieren. Dazu hinterlegen Sie für das Ereignis **Beim Klicken** die folgende Prozedur (Klick auf die Eigenschaft **Beim Klicken**, Auswahl von **[Ereignisprozedur]**, Klick auf die Schaltfläche mit den drei Punkten):

```
Private Sub cmdSuchen_Click()
    Me!sfmArtikelsuche.Form.Requery
End Sub
```

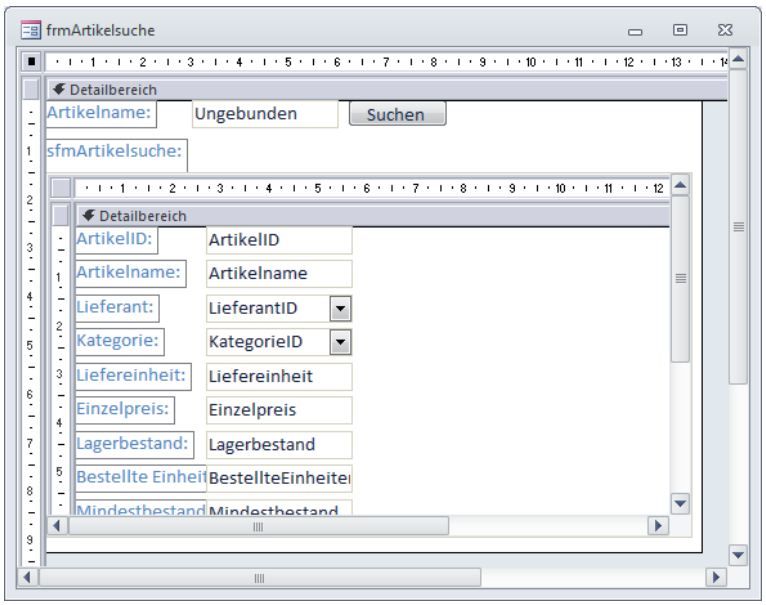


Bild 2: Haupt- und Unterformular in der Entwurfsansicht

Ziehen Sie außerdem das Unterformular **sfmArtikelsuche** aus dem Datenbankfenster beziehungsweise aus dem Navigationsbereich in den Detailbereich des Entwurfs des Formulars **frmArtikelsuche**. Das Ergebnis sieht wie in Bild 2 aus.

Schließlich stellen Sie die Eigenschaften Datensatzmarkierer, **Navigationsschalt-**

flächen, **Trennlinien** und **Bildlaufleisten** des Hauptformulars auf **Nein** und **Automatisch zentrieren** auf **Ja** ein.

Datenherkunft anpassen

Nun passen Sie die bislang statische Datenherkunft des Unterformulars an. Dies können Sie gleich über die Entwurfsansicht das Hauptformulars erledigen. Klicken Sie zwei Mal oben links im Unterformular auf den Schnittpunkt der beiden Lineals, um das Unterformular zu markieren. Der erste Klick aktiviert das Unterformular-Steuerelement, der zweite das darin enthaltene Unterformular.

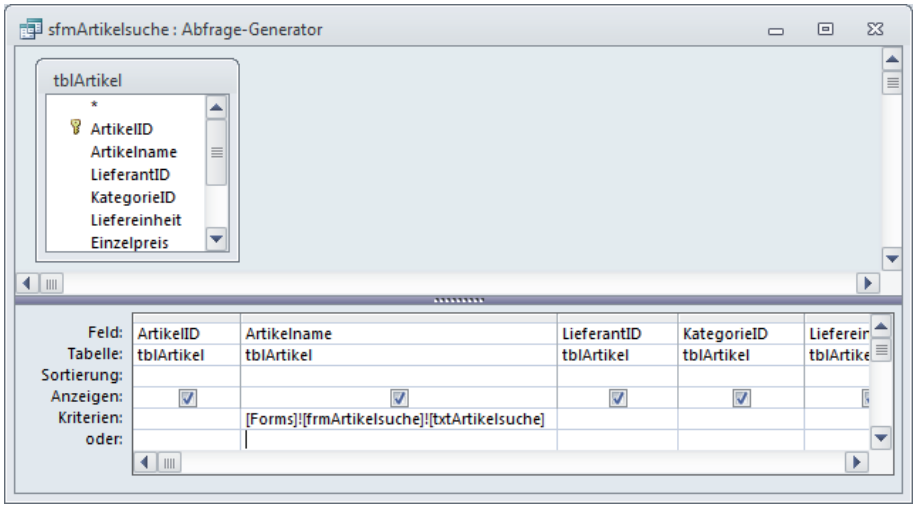


Bild 3: Neue Datenherkunft des Unterformulars

Aktivieren Sie das Eigenschaftsfenster und wechseln Sie zur Registerseite Daten. Klicken Sie erst auf die Eigenschaft **Datenherkunft** (ältere Access-Versionen) beziehungsweise Datensatzquelle und dann auf die Schaltfläche mit den drei Punkten am rechten Rand. Access fragt nun, ob Sie eine Abfrage erstellen möchten, die auf der als Datenherkunft gewählten Tabelle basiert – klicken Sie hier auf **Ja**.

Nun erscheint der Entwurf einer Abfrage, die im oberen Bereich bereits die Tabelle **tblArtikel** enthält.

Im Unterformular sind bereits Steuerelemente zur Anzeige aller Felder der Tabelle angelegt, also muss auch die neue Datenherkunft alle Felder der Tabelle zurückliefern. Markieren Sie alle Felder in der Feldliste unter **tblArtikel**, indem Sie zunächste das oberste Feld und dann bei gedrückter Umschalttaste das unterste Feld anklicken.

Lassen Sie die Umschalttaste los und ziehen Sie die markierten Felder per Drag and Drop in das Entwurfsraster im unteren Bereich.

Um alle Felder anzuzeigen, könnten Sie auch einfach das Sternchen aus der Feldliste in das Entwurfsraster ziehen. Allerdings müssten Sie dann solche Felder, für die Sie Sortierungen oder Kriterien festlegen möchten, erneut zum Entwurf hinzufügen und dann deren Anzeige deaktivieren, damit diese nicht zweimal im Abfrageergebnis erscheinen. Wenn Sie gleich alle Felder in das Entwurfsraster ziehen, ist es einfacher.

Im nächsten Schritt legen Sie das Kriterium für das Abfrageergebnis fest. Dieses soll nur diejenigen Datensätze liefern, deren Artikelname mit dem im Textfeld **txtArtikelsuche** des Hauptformulars angegebenen Ausdruck übereinstimmt.

Im einfachsten Fall gehen wir davon aus, dass es sich um eine hundertprozentige Übereinstimmung handeln muss. Dies erreichen Sie, indem Sie den folgenden Ausdruck als Kriterium in die Spalte für das Feld **Artikelname** eintragen (siehe Bild 3):

```
Forms!frmArtikelsuche!txtArtikelsuche
```

Schließen Sie den Abfrageentwurf und bestätigen Sie die Änderungen. Anschließend zeigen Sie das Formular in der Formularansicht an.

Das Ergebnis überrascht kaum: Das Unterformular zeigt keinerlei Datensätze an – schließlich enthält das als Kriterium angegebene Textfeld **txtArtikelsuche** ja auch keinen Wert. Es geschieht erst etwas, wenn Sie etwa den Artikelnamen **Chai** in das Suchfeld eintragen und auf **Suchen** klicken.

Fehlersuche

Wie finden Sie nun heraus, was schiefläuft, wenn kein Suchbegriff angegeben wurde? Am einfachsten, indem Sie zunächst die für die Eigenschaft Datenherkunft des Unterformulars festgelegte Abfrage als eigene Abfrage speichern und den Namen der gespeicherten Abfrage als Datenherkunft angeben. Damit

Internetmarke mit Access nutzen, Teil II

In Ausgabe 10/2012 haben Sie erfahren, wie Sie Daten aus einer Adresstabelle so aufbereiten, dass diese vom Internetdienst der deutschen Post zum Erstellen von Frankieretiketten genutzt werden können. Ein weiterer Artikel dieser Ausgabe zeigte, wie Sie Adressen aus einer Liste von Adressen auswählen. Der vorliegende Artikel führt beide Lösungen zusammen und zeigt außerdem, wie Sie die Export-Datei nach dem Markieren der betroffenen Adressdatensätze mit nur einem Mausklick erstellen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1211_Internetmarke.mdb**.

Ausgangspunkt: Auswahlformular

Wir beginnen mit dem Formular **frmDatenauswahl**, das im Artikel **Datensätze individuell auswählen** detailliert beschrieben wurde. Diesem fügen Sie lediglich noch eine Schaltfläche namens **cmdExport** hinzu (siehe Bild 1). Diese soll alle notwendigen Befehle enthalten, um die im Unterformular des Formulars markierten Einträge in eine Export-Datei zum Erstellen von Internetmarken zu übertragen.

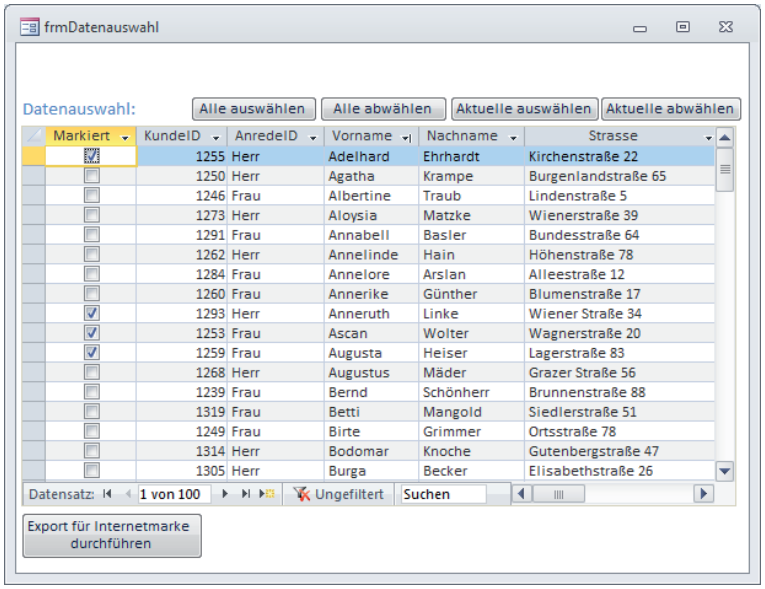


Bild 1: Ausgangspunkt für das Erstellen von Internetmarken

Dazu sind im Detail drei Schritte nötig:

- Die im ersten Teil dieses Artikels vorgestellte Abfrage muss ihre Werte nach dem Feld **Markiert** der Tabelle **tblKunden** filtern.
- Wir benötigen eine Export-Spezifikation, die festlegt, wie Daten in der **.csv**-Datei angezeigt werden.
- Schließlich brauchen wir die notwendigen Anweisungen, welche den Export und somit die Erstellung der **.csv**-Datei durchführen.

Datenherkunft anpassen

Die bisherige Datenherkunft wurde so aufgebaut:

- Eine Basisabfrage namens **qryInternetporto** führt die Daten der Tabellen **tblKunden**, **tblAnreden** und **tblLaender** so zusammen, dass die gewünschten Felder mit entsprechenden Feldnamen versehen werden.
- Eine **UNION**-Abfrage namens **qryInternetportoMitAbsender** fügt zu dieser Abfrage einen weiteren Datensatz hinzu, der die Daten des Absenders enthält. Diese enthält noch die Werte des Feldes **KundeID** der Tabelle **tblKunden** beziehungsweise den Wert **0** für den als Absender hinzugefügten Datensatz.
- Die Abfrage **qryInternetportoMitAbsenderSortiert** greift schließlich dieses Abfrageergebnis auf, sortiert nach dem Wert des Feldes **KundeID** (wodurch der Absender mit dem Wert **0** ganz nach vorn rutscht). Dieses Feld wird jedoch nicht in das Abfrageergebnis übernommen, da es von der entsprechenden Internetseite nicht interpretiert werden kann.

Wo setzen wir nun an, um nur diejenigen Kundendatensätze zu übergeben, die im Formular **frmDatenauswahl** markiert wurden? Ganz einfach: Bei der ersten Abfrage **qryInternetporto**. Wir könnten zwar das Kriterium zwar in jeder der Abfragen einfügen, aber je früher wir das tun, desto früher reduzieren wir auch die zu händelnde Datenmenge. Also öffnen Sie die Abfrage **qryInternetporto** in der Entwurfs-

ansicht und ziehen das Feld **Markiert** der Tabelle **tblKunden** in das Entwurfsraster. Stellen Sie als Kriterium den Wert **True** ein (siehe Bild 2). Wenn Sie bereits einige Datensätze im Formular **frmDatenauswahl** markiert haben, können Sie nun testen, ob die drei Abfragen auch nur die ausgewählten Datensätze liefern.

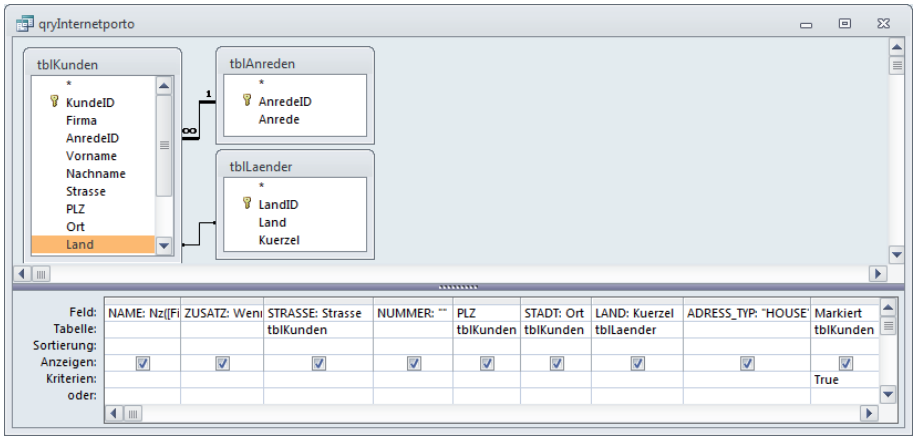


Bild 2: Einfügen des Kriteriums, das nur die ausgewählten Datensätze anzeigt

Export-Spezifikation erstellen

Im ersten Teil dieser Artikelreihe haben wir ja bereits beschrieben, welche Einstellungen im Export-Assistenten für die Erstellung der **.csv**-Datei durchgeführt werden müssen.

Dort waren – zumindest unter Access 2010 – nur zwei Änderungen gegenüber der Standardkonfiguration nötig. Die erste war das Aktivieren der Option **Feldnamen in die erste Zeile einbeziehen**, die zweite war das Einstellen der Option **Textbegrenzungszeichen** auf den Wert **{kein}**.

Diese Einstellungen wollen wir nun in einer Export-Spezifikation speichern, damit Sie diese nicht immer wieder neu vornehmen müssen. Dazu klicken Sie im Textexport-Assistenten nach dem Vornehmen der Einstellungen auf die Schaltfläche **Erweitert...** und dann auf **Speichern unter...** (siehe Bild 3). Im nun erscheinenden Dialog **Import/Export-Spezifikation speichern** geben Sie den Namen für die Spezifikation ein, beispielsweise **Internetmarke**. Dieser Name muss lediglich innerhalb einer Access-Datei eindeutig sein. Bei Angabe eines bereits vorhandenen Namens können Sie die vorhandene Spezifikation überschreiben lassen.

Bevor wir zum Finale kommen, noch ein Hinweis zum Thema Spezifikationen: In der Tat nimmt diese nicht alle Einstellungen auf, die Sie für den aktuellen Export festgelegt haben. So müssen Sie beispielsweise immer wieder die Zieldatei angeben und auch die Eigenschaft **Feldnamen in die erste Zeile einbeziehen** merkt sich die Spezifikation nicht.

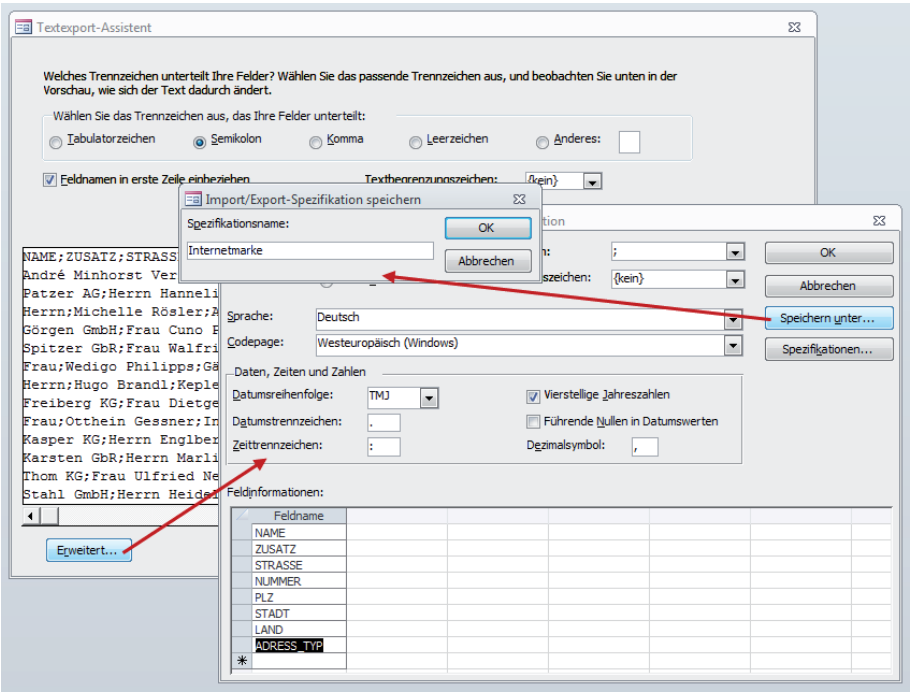


Bild 3: Speichern der Export-Spezifikation

Das ist für uns in diesem Fall aber auch kein Problem, da wir diese Informationen beim Exportieren der Daten per VBA angeben können.

Export per VBA

Und damit kommen wir zum dritten Schritt bei der Automatisierung des Exports: Dem Anlegen einer Ereignisprozedur, die durch das Anklicken der Schaltfläche **cmdExport** ausgelöst wird und

Das Registersteuerelement in der Praxis

Bereits ohne den Einsatz von VBA können Sie das Registersteuerelement prima nutzen, um die Steuerelemente eines Formulars platzsparend auf mehrere Registerseiten aufzuteilen. Aber erst VBA entfaltet die vollen Möglichkeiten dieses Steuerelements: Sie können damit programmgesteuert ermittelten, welche Registerseite gerade aktiv ist, diese automatisch einstellen oder sogar per VBA Registerseiten hinzufügen und entnehmen, wenn Sie dies nicht von Hand erledigen möchten.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1212_RegistersteuerelementPraxis.mdb**.

Registersteuerelement programmieren

Bevor wir zu den Praxisbeispielen kommen, schauen wir uns an, wie Sie das Aussehen, das Verhalten und die Interaktion des Registersteuerelements per VBA beeinflussen können.

Aktuelle Seite abfragen

Wenn Sie für ein Registersteuerelement namens **tabBeispielVBA** in einem Formular wie **frmBeispielVBA** aus Bild 1 die aktuell angezeigte Seite abfragen möchten, erledigen Sie dies etwa im Direktfenster mit folgender Anweisung:

```
Debug.Print Forms!frmBeispielVBA!tabBeispielVBA.Value
 0
```

Ist die erste Seite aktiviert, liefert dies den Index der Seite, nämlich **0**. Für die zweite Seite würde dies den Wert **1** liefern und so weiter. Da Value die Standardeigenschaft dieses Steuerelements ist, können Sie auch die Kurzform ohne Angabe der Eigenschaft verwenden:

```
Debug.Print Forms!frmBeispielVBA!tabBeispielVBA
```

Aktuelle Seite einstellen

Wenn Sie per VBA die aktuell sichtbare Registerseite einstellen möchten, gelingt dies auf ähnliche Weise, wie dieser Aufruf zeigt – ebenfalls testweise im Direktfenster abzusetzen:

```
Forms!frmBeispielVBA!tabBeispielVBA = 0
```

Damit aktivieren Sie die erste Registerseite, durch die Angabe anderer Werte die entsprechend indizierte Seite.

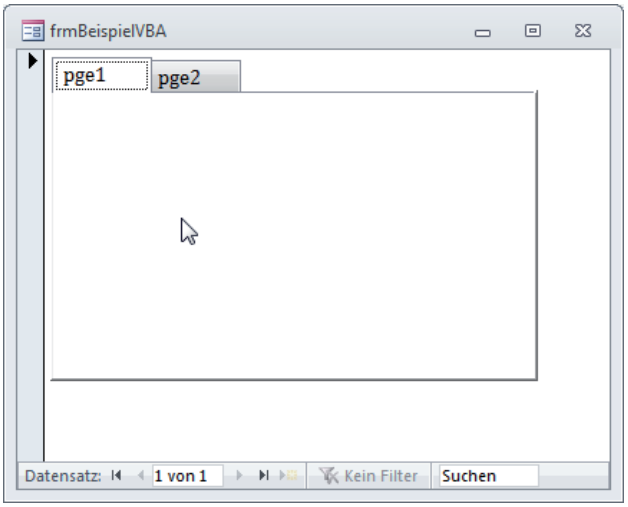


Bild 1: Beispielregistersteuerelement

Informationen einer Seite ermitteln

Wenn Sie Informationen wie Name, Beschriftung et cetera einer bestimmten Seite des Registersteuerelements ermitteln möchten, können Sie dies mit der **Pages**-Auflistung erledigen. Damit greifen Sie über den Index oder den Namen der Registerseite auf alle enthaltenen Seiten zu.

Den Namen der ersten Registerseite erhalten Sie über das Direktfenster etwa so:

```
? Forms!frmBeispielVBA!tabBeispielVBA.Pages(0).Name
pge1
```

Die Beschriftung erhalten Sie auf ähnliche Weise:

```
? Forms!frmBeispielVBA!tabBeispielVBA.Pages(0).
Caption
```

Die Beschriftung etwa der ersten Seite des Registersteuerelements stellen Sie wie folgt ein:

```
Forms!frmBeispielVBA!tabBeispielVBA.Pages(0)._
    Caption = "Seite 1"
```

Informationen zu allen Seiten, die sich aktuell im Registersteuerelement befinden, können Sie mit einer

For Each-Schleife durchlaufen und ausgeben (diese Prozedur finden Sie im Beispielformular **frmBeispielVBA**):

```
Private Sub cmdAlleSeitennamen_Click()
    Dim pge As Page
    For Each pge In Me.tabBeispielVBA.Pages
        Debug.Print pge.Name
    Next pge
End Sub
```

Aktuelle Seite beim Klicken ausgeben

Wenn Sie beim Wechseln der Registerseite auswerten möchten, welche die aktuelle Seite ist, legen Sie dafür eine entsprechende Ereignisprozedur an. Aber für welches Ereignis? Das Registersteuerelement liefert zwei, die dafür in Frage kommen könnten – **Beim Klicken** und **Bei Änderung**.

Um herauszufinden, welches Ereignis beim Wechseln der Registerseite ausgelöst wird, legen Sie einfach für beide Ereigniseigenschaften den Wert **[Ereignisprozedur]** an und klicken je einmal auf die Schaltfläche mit den drei Punkten, um die entsprechende Ereignisprozedur anzulegen (siehe Bild 4).

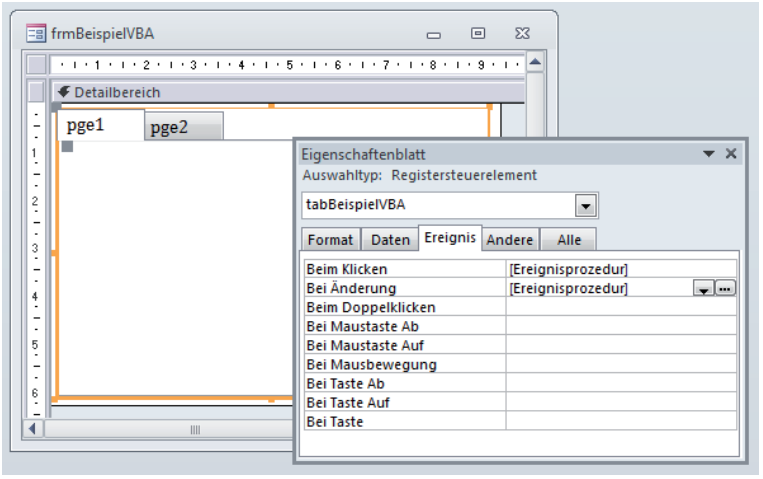


Bild 4: Anlegen von Ereignisprozeduren für die Ereigniseigenschaften des Registersteuerelements

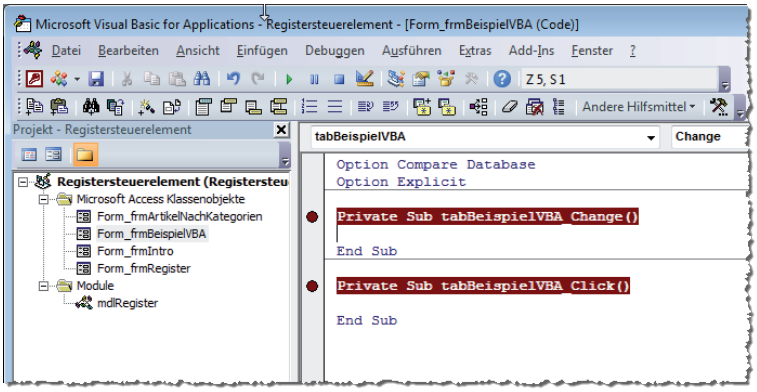


Bild 2: Ereignisprozeduren mit Haltepunkten

Fügen Sie im VBA-Editor für beide Prozeduren einen Haltepunkt hinzu, indem Sie auf die graue Leiste links vom Code klicken (siehe Bild 2).

Zeigen Sie das Formular dann in der Formularansicht an und wechseln Sie die Registerseite. Dies öffnet den VBA-Editor, wobei die Prozedur **tabBeispielVBA_Change** gelb markiert ist (siehe Bild 3). Ein Klick auf die Taste F5 lässt den Code weiterlaufen. Die Prozedur **tabBeispielVBA_Click** wird jedoch nicht mehr gestartet. Dies bedeutet, dass nur das Ereignis **Beim Ändern** durch den Wechsel der Registerseite ausgelöst wird.

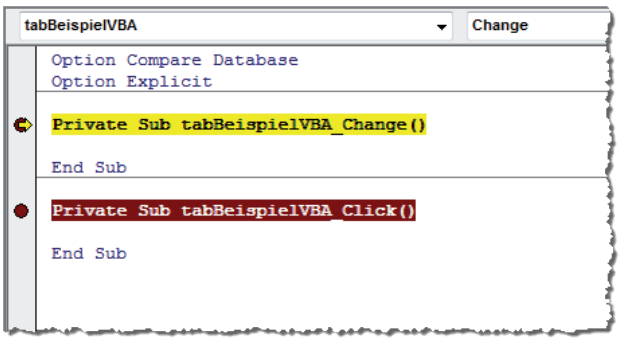


Bild 3: Nur diese Prozedur wird beim Wechsel der Registerseite ausgelöst.

Fügen wir dieser Prozedur doch eine Anweisung hinzu, welche den Index der aktuellen Registerseite in einem Meldungsfenster ausgibt:

```
Private Sub tabBeispielVBA_Change()
    MsgBox Me!tabBeispielVBA
End Sub
```

Danach wollen wir aus dem Indexwert noch den Namen der Registerseite sowie die Beschriftung entnehmen. Dies gelingt durch die folgende Erweiterung – diesmal mit **Debug.Print** zur Ausgabe im Direktfenster:

```
Private Sub tabBeispielVBA_Change()
    Debug.Print Me!tabBeispielVBA, _
        Me!tabBeispielVBA.Pages(Me!tabBeispielVBA).Name
End Sub
```

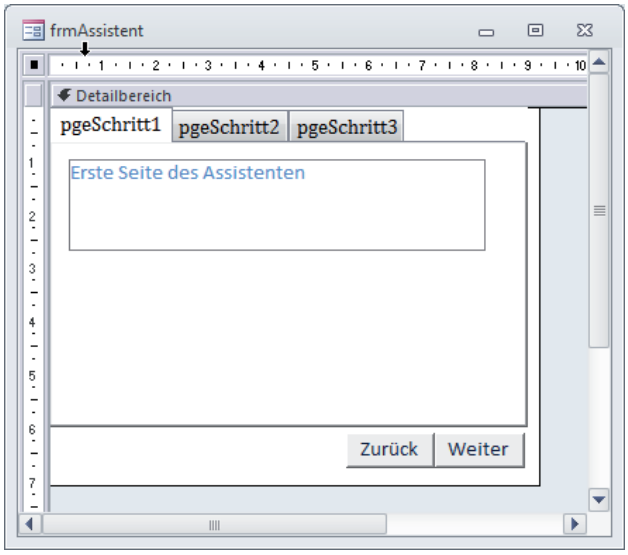


Bild 5: Beispiel für den Einsatz in einem Assistenten

Der erste Ausdruck liefert den Index (also den Wert der **Value**-Eigenschaft), der zweite Ausdruck den Namen der Registerseite. Dazu verwendet er die **Pages**-Auflistung, über die Sie mit der Angabe des Indexwertes auf alle Registerseiten zugreifen können. **Name** wiederum gibt den Namen des Registerseite-Elements aus.

Wann aber wird die **Beim Klicken**-Ereigniseigenschaft des Registersteuerelements ausgelöst? Dies gelingt ausschließlich, wenn Sie auf den freien Bereich im Kopf des Registersteuerelements klicken.

Registersteuerelement als Assistent

Die einzelnen Seiten des Registersteuerelements eignen sich hervorragend, um die einzelnen Schritte eines Assistenten abzubilden. Das gilt insbesondere, da Sie die Steuerelemente zum Abarbeiten der einzelnen Schritte so bequem auf verschiedenen Seiten des Registersteuerelements unterbringen und Schritt für Schritt bearbeiten können.

Bild 5 zeigt ein schematisches Beispiel für einen solchen Assistenten. Hier wird vereinfachend nur je ein Text auf jeder Seite des Assistenten angezeigt. Der Benutzer soll mit den beiden Schaltflächen **Zurück** und **Weiter** durch die einzelnen Schritte navigieren können. Die Besonderheit ist, dass nicht nur das Anklicken der beiden Schaltflächen die Anzeige der aktuellen Seite des Registersteuerelements beeinflussen soll. Andersherum soll die **Zurück**-Schaltfläche bei aktivierter erster Registerseite deaktiviert werden, denn weiter zurück kann der Benutzer an dieser Stelle nicht mehr gehen (siehe Bild 6). Und die **Weiter**-Schaltfläche soll, wenn der Benutzer die letzte Seite des Assistenten aufgerufen hat, den Text **Beenden** erhalten und umgekehrt (siehe Bild 7).

Damit dies gleich mit dem Laden des Formulars reibungslos klappt, ruft die durch die Ereigniseigenschaft **Beim Laden** ausgelöste Prozedur **Form_Load** eine weitere Prozedur namens **SchaltflaechenEinstellen** auf:

```
Private Sub Form_Load()
    Call SchaltflaechenEinstellen
End Sub
```

Diese kümmert sich dann um die korrekte Einstellung der Eigenschaften der betroffenen Steuerelemente:

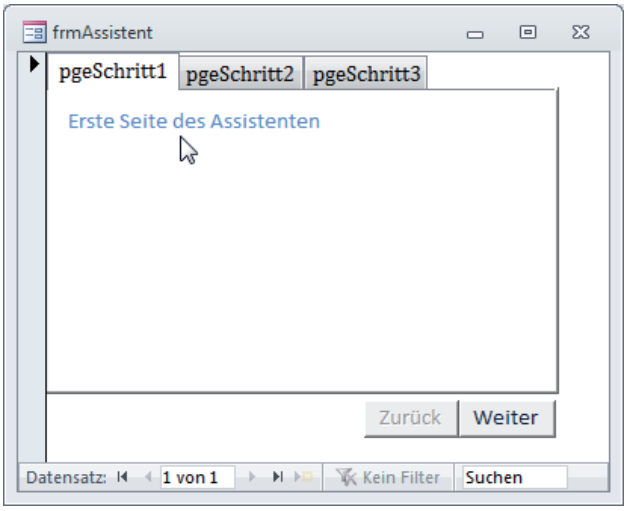


Bild 6: Ausblenden der **Zurück**-Schaltfläche für den ersten Schritt

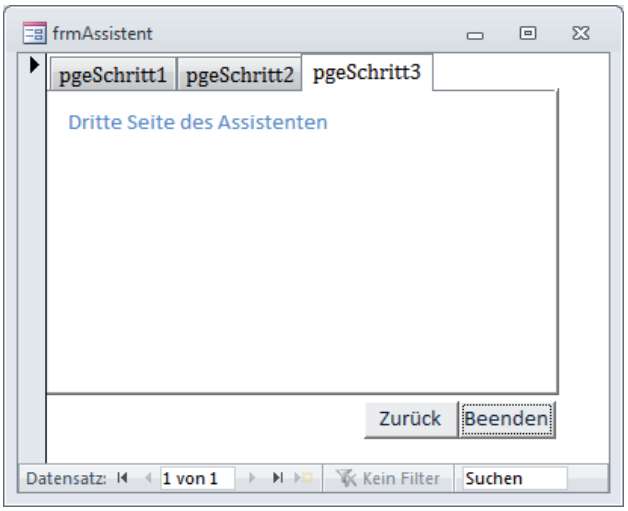


Bild 7: Ändern der Beschriftung der **Weiter**-Schaltfläche im letzten Schritt des Assistenten

SQL-Grundlagen: Aktionsabfragen

Wer schon etwas länger mit Access arbeitet, kennt das: Sie möchten mal eben den Wert eines Feldes für alle Datensätze ändern oder auch eine bestimmte Gruppe von Datensätzen löschen. Manuell in der Datenblattansicht ist das aber nicht zu erledigen, weil die Anzahl der Datensätze dafür zu groß ist. Dafür extra eine Aktionsabfrage erstellen und ausführen? Nein, das ist auch zuviel Aufwand. Also bleibt nur eine Möglichkeit: Sie schaffen sich einige Grundlagen zum Thema Aktionsabfragen unter SQL drauf und erledigen die genannten Aufgaben schnell vom Direktfenster aus.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1211_SQLAktionsabfragen.mdb**.

Aufruf

Zum Ausführen einer Aktionsabfrage über das Direktfenster können Sie zwei verschiedene Methoden verwenden. Beide werden im Artikel Aktionsabfragen per VBA ausführlich erläutert. Im vorliegenden Artikel verwenden wir ausschließlich die **Execute**-Methode des mit **CurrentDb** gelieferten Objekts. Dies sieht, am einfachen Beispiel einer Löschabfrage, etwa so aus:

```
CurrentDb.Execute "DELETE FROM tblTest",
dbFailOnError
```

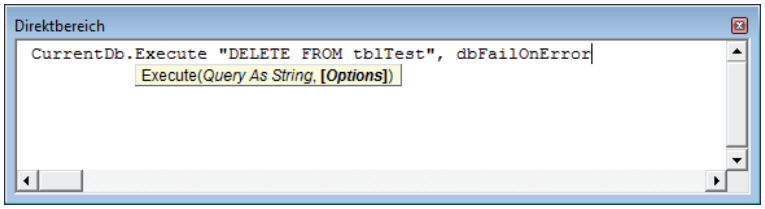


Bild 1: Absetzen einer SQL-Anweisung per Direktfenster

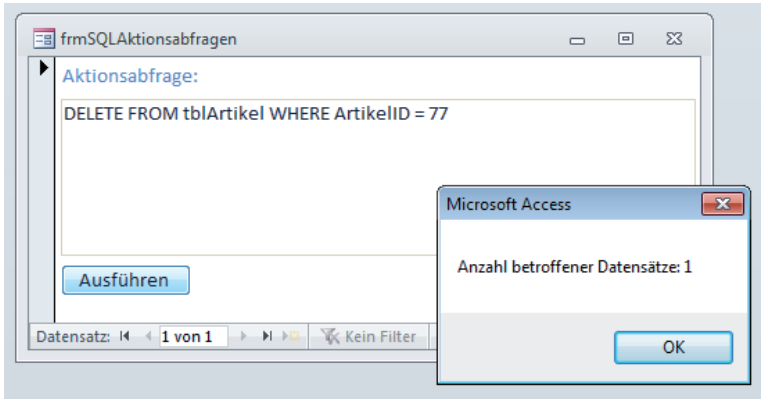


Bild 2: Hilfsformular zum Ausführen von SQL-Aktionsabfragen

Mit dieser Anweisung, abgesetzt im Direktfenster des VBA-Editors, das Sie mit der Tastenkombination **Strg + G** öffnen, würden Sie alle Datensätze der Tabelle **tblTest** löschen (siehe Bild 1). Die **Execute**-Anweisung erwartet die SQL-Variante der Aktionsabfrage als Parameter in Anführungszeichen. Der Wert **dbFailOnError** für den zweiten Parameter sorgt dafür, dass eventuell auftretenden Fehler gemeldet werden.

Damit Sie etwas Spaß an den Beispielen des Artikels haben, finden Sie in der Beispieldatenbank ein Formular namens **frmSQLAktionsabfragen**. Dort können Sie Aktionsabfragen zum Ausprobieren eingeben und auf Ausführen klicken – dies ist erstens etwas einfacher als der Weg über das Direktfenster und zweitens erhalten Sie auch noch Feedback, wieviele Datensätze von der Änderung betroffen sind (siehe Bild 2).

Das Formular ist außerdem an eine Tabelle namens **tblAktionsabfragen** gebunden, die alle Beispielaufrufe dieses Artikels speichert – Sie können diese also im Formular durchlaufen und ausprobieren. Der notwendige Code hinter der Schaltfläche ist einfach:

```
Private Sub cmdAusfuehren_Click()
    Dim db As DAO.Database
    Set db = CurrentDb
    db.Execute Me!txtAktionsabfrage, dbFailOnError
    MsgBox "Anzahl betroffener Datensätze: " _
        & db.RecordsAffected
    Set db = Nothing
End Sub
```

Er führt lediglich die Aktionsabfrage aus und fragt nachher mit der Eigenschaft **RecordsAffected** ab, wieviele Datensätze von der Änderung betroffen sind.

Die tatsächlich für die Schaltfläche hinterlegte Prozedur ist etwas komplizierter, weil diese es erlaubt, die durchgeführten Änderungen wieder zu verwerfen – dieses Thema werden wir jedoch in einem ande-

ren Artikel behanden, aktuell dient dies nur Ihrer Bequemlichkeit.

Syntax ermitteln

In den folgenden Abschnitten erhalten Sie grundlegende Informationen zu den vier möglichen Aktionsabfragen. Damit können Sie wahrscheinlich die meisten Konstellationen abbilden. Allerdings kann es auch sein, dass Sie doch einen kleinen Denkanstoß benötigen – in diesem Falle ziehen Sie dann doch die Entwurfsansicht für Abfragen zuhilfe und erstellen damit die gewünschte Aktionsabfrage. Anschließend können Sie den SQL-Code für diese Abfrage ganz einfach ermitteln.

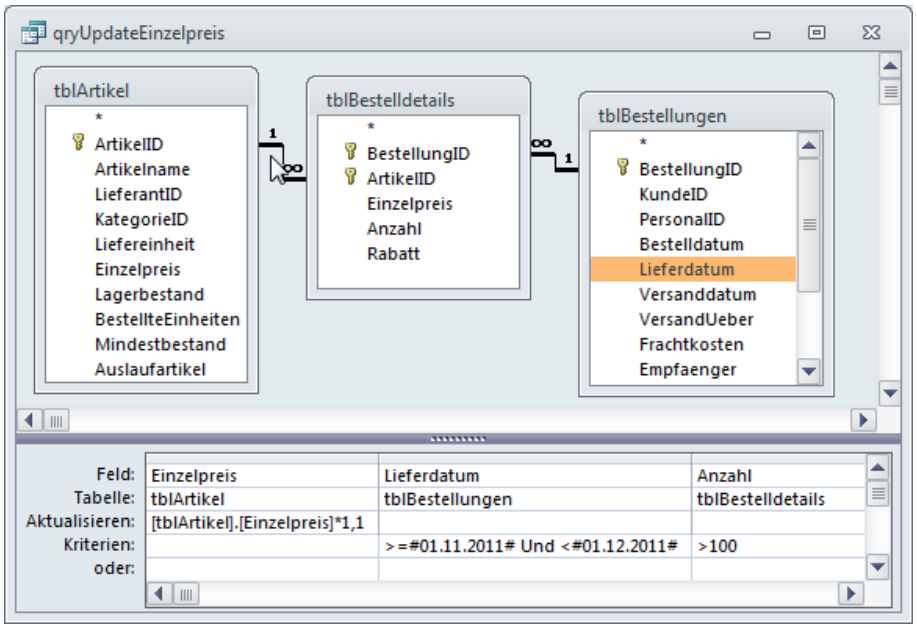


Bild 3: Komplizierte Abfrage im Entwurf

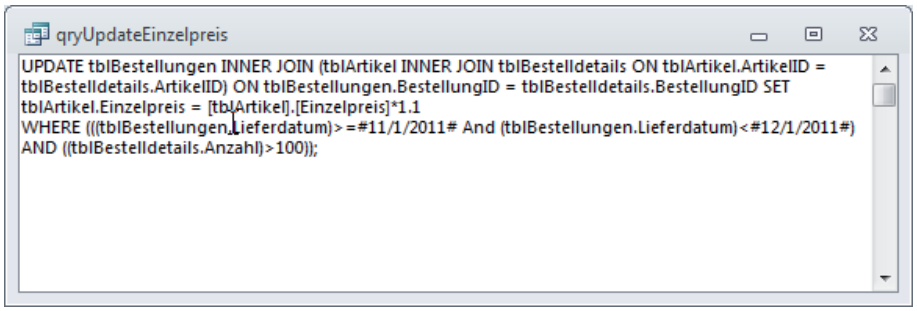


Bild 4: Komplizierte Abfrage – unter SQL noch verwirrender

Die Abfrage aus Bild 3 etwa soll die Einzelpreise aller Datensätze um 10% erhöhen, von denen im November 2011 mehr als 100 verkauft wurden. Nachdem Sie diese Abfrage eingegeben haben, können Sie mit einem Rechtsklick auf die Titelleiste des Abfrageentwurfs und dem anschließenden Auswählen des Eintrags **SQL-Ansicht** des Kontextmenüs die SQL-Ansicht betrachten. Einen solchen Ausdruck werden Sie erst nach einiger Übung aus dem Kopf entwerfen können ... (siehe Bild 4). Dennoch: Auch diese Ausdruck können Sie kopieren und mit dem oben vorgestellten Formular ausführen lassen. Sie dürfen die SQL-Ansicht natürlich auch für die Ermittlung der SQL-Ausdrücke einfacherer Aktionsabfragen heranziehen. In den folgenden Abschnitten lernen Sie nun die SQL-Syntax der vier verschiedenen SQL-Aktionsabfragen kennen.

Daten löschen

Löschabfragen haben die einfachste Syntax der Aktionsabfragen, da hier keine Angabe eines Zielfeldes notwendig ist:

```
DELETE FROM <Tabelle> [WHERE-Klausel]
```

Um beispielsweise alle Artikel aus der Artikeltabelle zu löschen, deren Preis höher als **10 EUR** ist, verwenden Sie die folgende Abfrage:

```
DELETE FROM tblArtikel WHERE Einzelpreis>10;
```

Es kann aber auch sein, dass die Syntax etwas komplizierter wird: Wenn die Datenherkunft der Abfrage mehr als eine Tabelle enthält, weil Sie etwa ein Feld einer verknüpften Tabelle als Kriterium für die zu löschenden Datensätze verwenden möchten, müssen Sie zwischen **DELETE** und **FROM** noch angeben, aus welcher der angegebenen Tabelle die zu löschenden Datensätze stammen. Dann schreiben Sie beispielsweise folgende Aktionsabfrage:

```
DELETE tblArtikel.* FROM tblArtikel INNER JOIN ...
```

Daten aktualisieren

Zum Aktualisieren von Daten sind Aktionsabfragen gerade dann interessant, wenn größere Mengen Da-

DAO: Querydefs und Parameterabfragen unter VBA

Mit der QueryDefs-Auflistung greifen Sie per VBA auf die in einer Datenbank gespeicherten Abfragen zu, legen neue Abfragen an oder öffnen Abfragen mit Parametern per VBA. Sie können damit sogar temporäre Abfragen erstellen, die Sie nur einmal benötigen, und diese anschließend wieder löschen. Dieser Artikel zeigt alle Tricks rund um die Arbeit mit dem QueryDef-Objekt.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1212_QueryDefs.mdb**.

Das QueryDef-Objekt

Ein **QueryDef**-Objekt ist die Repräsentation einer Abfrage unter VBA beziehungsweise DAO. Sie können mit DAO beispielsweise die Namen aller in der Datenbank gespeicherten Abfragen ausgeben, deren SQL-Code ansehen, neue Abfragen anlegen oder bestehende Abfragen löschen.

Wenn Sie beispielsweise alle Abfragen der Datenbank im Direktfenster auflisten möchten, erstellen Sie eine kleine VBA-Prozedur. Dazu benötigen Sie zunächst ein Standardmodul. Öffnen Sie mit **Strg + G** oder **Alt + F11** den VBA-Editor und wählen Sie den Menüeintrag **Einfügen|Modul**. Speichern Sie das Modul mit **Strg + S** unter dem Namen **mdlQuerydefs**.

Nun fügen Sie dem Modul die folgende Prozedur hinzu:

```
Public Sub AbfragenAuflisten()
    Dim db As DAO.Database
    Dim qdf As DAO.QueryDef
    Set db = CurrentDb
    For Each qdf In db.QueryDefs
        Debug.Print qdf.Name
    Next qdf
    Set db = Nothing
End Sub
```

Platzieren Sie die Einfügemarke innerhalb der Prozedur und betätigen Sie die Taste **F5**, um die Prozedur auszuführen. Was geschieht? Die Prozedur gibt alle in der Datenbank enthaltenen Abfragen im Direktfenster aus (siehe Bild 1).

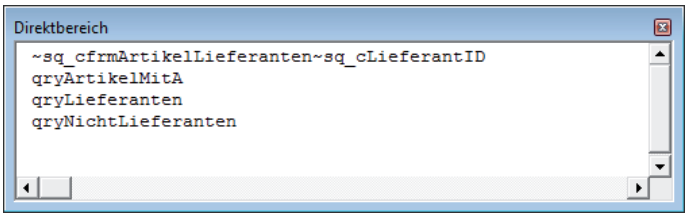


Bild 1: Ausgabe aller Abfragen der Datenbank

Das Ergebnis ist interessant: Dort finden sich nämlich nicht nur die im Navigationsbereich beziehungsweise im Datenbankfenster sichtbaren Abfragen, die normalerweise mit dem Präfix **qry** versehen werden, sondern auch noch Einträge, die mit dem Tilde-Zeichen beginnen (**~**).

Dabei handelt es sich um solche Abfragen, die als Datenherkunft etwa für Formular, Berichte oder Kombinationsfelder angelegt wurden.

Wenn Sie beispielsweise ein Formular namens **frmArtikel** anlegen und diesem eine Abfrage als Datenherkunft zuweisen, wird diese unter dem Namen **~sq_ffrmArtikel** ausgegeben. Wenn Sie ein Formular namens **frmMitCboArtikel** und einem Kombinationsfeld namens **cboArtikel** anlegen und dem Kombinationsfeld eine Abfrage als Datensatzherkunft zuweisen, finden Sie unter den Abfrage mit obiger Prozedur einen Eintrag namens **~sq_cfrmMitCboArtikel~sq_ccboArtikel** vor. Die Namen dieser versteckt gespeicherten Abfragen werden also aus dem Formularnamen und, wenn ein Kombinationsfeld oder ein Listenfeld im Spiel ist, auch noch aus dem Namen des Steuerelements gebildet – dieser Exkurs nur als Erläuterung diese kryptisch anmutenden Abfragenamen.

Was genau erledigt nun die oben vorgestellte Prozedur? Sie deklariert zunächst eine Objektvariable namens **db**, welche einen Verweis auf das **Database**-Objekt zur aktuellen Datenbank aufnimmt. Eine weitere Objektvariable soll später mit Verweisen auf die **QueryDef**-Objekte der Datenbank gefüttert werden, was der gespeicherten Definition einer Abfrage entspricht. Und dies bezieht sich, wie oben angeführt, nicht nur auf die durch den Benutzer gespeicherten Abfragen, sondern auch auf die Datenherkünfte und datensatzherkünfte von Formularen, Berichten, Kombinationsfeldern und Listenfeldern.

Die Variable **db** wird nun mit einem Verweis auf das **Database**-Objekt gefüllt. Die Variable **qdf** wird in einer **For Each**-Schleife nacheinander mit Verweisen auf alle in der **QueryDefs**-Auflistung enthaltenen **QueryDef**-Objekte versehen.

Die einzige Anweisung innerhalb der **For Each**-Schleife gibt den Namen des jeweiligen **QueryDef**-Objekts im Direktfenster aus.

Mit der **QueryDefs**-Auflistung können Sie noch weitere Dinge tun – zum Beispiel die Anzahl der **QueryDef**-Objekte der aktuellen Datenbank ermitteln. Das geht ganz schnell über das Direktfenster, wie Bild 2 zeigt. Die **QueryDefs**-Auflistung können Sie dabei auch direkt als Elemente der **CurrentDB**-Funktion verwenden, ein vorheriges Referenzieren mit einer **Database**-Variablen wie etwa **db** ist gar nicht nötig.

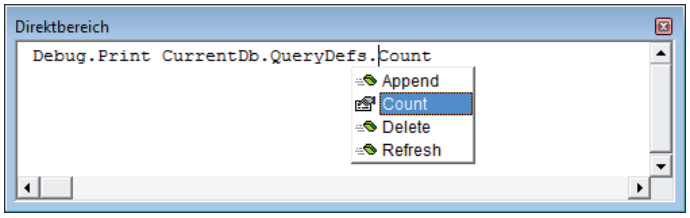


Bild 2: Abfrage der Anzahl der **QueryDef**-Objekte

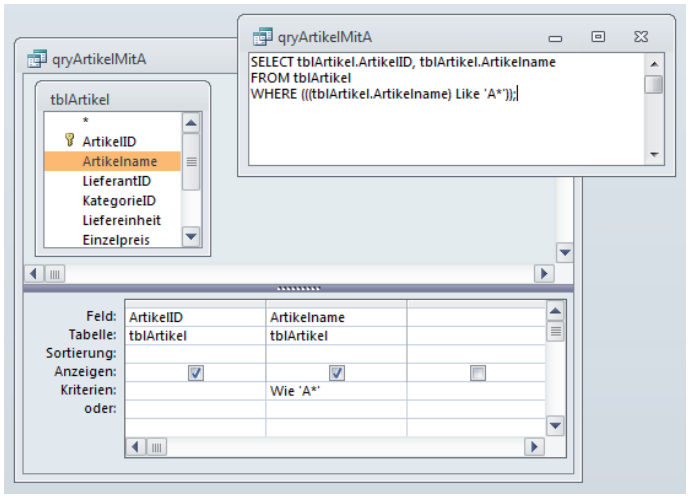


Bild 3: SQL-Ansicht einer Abfrage

SQL-Ansicht anzeigen

Wenn Sie eine Abfrage bearbeiten, verwenden Sie dazu in der Regel die Entwurfsansicht (siehe Bild 3). Sie können dies allerdings auch in der SQL-Ansicht erledigen, die Sie ebenfalls in der Abbildung vorfinden. Den dort dargestellten SQL-Code liefert auch das QueryDef-Objekt.

Wenn Sie schnell einmal den SQL-Ausdruck einer Abfrage ermitteln möchten, können Sie dies mit der folgenden Anweisung im Direktfenster erledigen:

```
Debug.Print CurrentDb.QueryDefs("qryArtikelMitA").SQL
```

Das Ergebnis sieht dann genauso aus wie in der SQL-Ansicht:

```
SELECT tblArtikel.ArtikelID, tblArtikel.Artikelname
FROM tblArtikel
WHERE (((tblArtikel.Artikelname) Like 'A*'));
```

In diesem Fall geben Sie in Klammern hinter der **QueryDefs**-Auflistung den Namen der gewünschten Abfrage an und fragen den Wert der Eigenschaft **SQL** ab. Natürlich können Sie diese Abfrage auch in die eingangs vorgestellte Prozedur einbauen und so nicht nur die Namen, sondern auch den SQL-Ausdruck aller Abfragen ausgeben – hier der geänderte Code-Ausschnitt:

```
For Each qdf In db.QueryDefs
    Debug.Print qdf.Name
    Debug.Print qdf.SQL
Next qdf
```

Eine Abfrage anlegen

Wenn Sie mit VBA eine Abfrage anlegen möchten, verwenden Sie dazu die **CreateQueryDef**-Methode des **Database**-Objekts. Die folgende Beispielprozedur erstellt zunächst mit **db** einen Verweis auf die aktuelle Datenbank und führt dann die **CreateQueryDef**-Methode aus. Dann legt sie die beiden Eigenschaften **Name** und **SQL** fest. Dies sind alle notwendigen Informationen. Allerdings erscheint die Abfrage auf diese Weise noch nicht im Datenbankfenster beziehungsweise im Navigationsbereich – und sie wird auch noch nicht in der Datenbank gespeichert. Dies erledigt erst der Aufruf der **Append**-Methode der **QueryDefs**-Auflistung mit der Objektvariablen **qdf** als Parameter:

```
Public Sub AbfrageAnlegen_Lang()
    Dim db As DAO.Database
    Dim qdf As DAO.QueryDef
    Set db = CurrentDb
    Set qdf = db.CreateQueryDef
    With qdf
        .Name = "qryTest_II"
        .SQL = "SELECT * FROM tblKategorien"
    End With
```

ListView-Steuerelement: Drag and Drop, Teil I

ListView-Steuerelemente unterstützen im Gegensatz zum Listenfeld sogar den Einsatz von Drag and Drop. Damit können Sie nicht nur Einträge von einem ListView-Steuerelement zum nächsten ziehen, sondern auch die Reihenfolge der Einträge eines ListView-Steuerelements einstellen. Im ersten Teil dieser Mini-Artikelreihe zeigen wir, wie Sie Datensätze zwischen zwei ListView-Steuerelementen hin- und herziehen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1212_ListView.mdb**.

Von ListView zu ListView per Drag and Drop

Einträge von einem ListView-Steuerelement zum nächsten Ziehen und fort fallenlassen – und umgekehrt? Wozu soll das gut sein? Nun: Beispielsweise können Sie so die Zuordnung von Elementen in einer m:n-Beziehung verwalten. Das Hauptformular zeigt dann die Datensätze der einen Seite der Beziehung an, die beiden ListView-Steuerelemente die diesem Datensatz zugeordneten und die nicht zugeordneten Datensätze der anderen Seite der Beziehung.

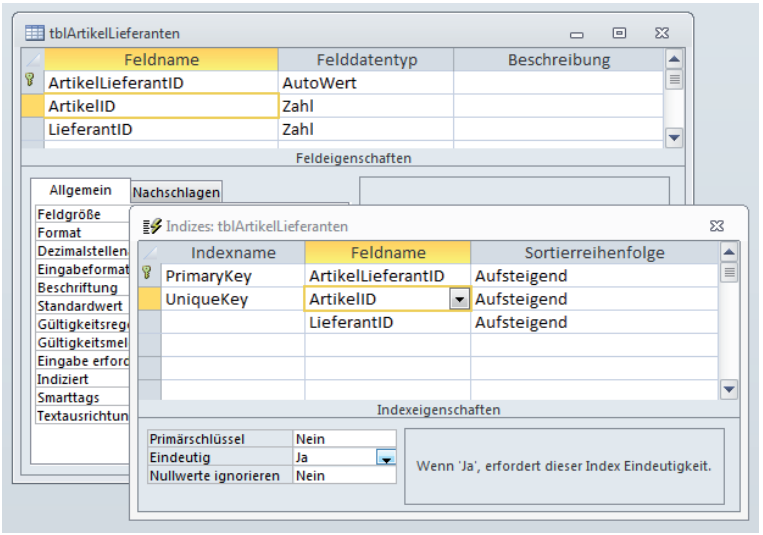


Bild 1: Tabelle zur Herstellung einer m:n-Beziehung

Die bisher in den Beispielen zum ListView-Steuerelement verwendeten Tabellen der Südsturm-Beispieldatenbank enthalten leider nur eine einzige m:n-Beziehung (zwischen den Bestellungen und den Artikeln). Diese ist leider nicht für dieses Beispiel geeignet, da ja hier noch einige weitere Daten wie Menge, Einzelpreis et cetera verwaltet werden müssen.

Also fingieren wir einfach ein geeignetes Beispiel. Die Tabelle **tblArtikel** enthält bislang ein Fremdschlüsselfeld, mit dem der Lieferant des Artikels eingestellt werden kann. Der Nachteil dieser Verknüpfung ist, dass Sie für jeden Artikel nur einen einzigen Lieferanten bestimmen können. In der Praxis gibt es jedoch meist noch weitere Lieferanten, die herangezogen werden, wenn der erste Lieferant nicht liefern kann oder ein anderer Lieferant einen günstigeren Preis anbietet.

Wir benötigen also eine Tabelle, mit der Sie einem Artikel einen oder mehrere Lieferanten zuordnen können. Diese sieht wie in Bild 1 aus und enthält neben dem Primärschlüsselfeld noch zwei weitere Felder namens **ArtikelID** und **LieferantID**. Für diese beden Felder legen Sie außerdem einen zusammengesetzten Schlüssel fest, dessen Eigenschaft **Eindeutig** den Wert **Ja** erhält. Damit stellen Sie sicher, dass jeder Lieferant jedem Artikel nur einmal zugeordnet werden kann.

Nun müssen Sie noch festlegen, dass die beiden Felder nur die Primärschlüsselwerte der Tabellen **tblArtikel** und **tblLieferanten** annehmen können. Dazu öffnen Sie das Beziehungen-Fenster von Access und fügen die drei Tabellen **tblArtikel**, **tblLieferanten** und **tblArtikelLieferanten** hinzu. Ziehen Sie das Feld **ArtikelID** der Tabelle **tblArtikel** auf das gleichnamige Feld der Tabelle **tblArtikelLieferanten**. Erledigen Sie das Gleiche für das Feld **LieferantID**.

Anschließend legen Sie die Beziehungseigenschaften fest (siehe Bild 2). Aktivieren Sie die Optionen **Mit Referentieller Integrität** und **Löschweitergabe an verwandte Datensätze**. Damit stellen Sie erstens sicher, dass die Fremdschlüsselfelder **ArtikelID** und **LieferantID** nur Werte aufnehmen können, die im Primärschlüsselfeld der jeweils verknüpften Tabelle enthalten sind. Außerdem werden Datensätze in der Tabelle **tblArtikelLieferanten** automatisch gelöscht, wenn ein verknüpfter Datensatz der Tabellen **tblArtikel** oder **tblLieferanten** entfernt wird.

Aber ist das überhaupt in Ordnung? Was geschieht, wenn ich alle Datensätze der Tabelle **tblLieferanten** lösche, die einem Artikel zugeordnet sind – diese besitzt dann ja keinen Lieferanten mehr? Kein Problem: Das Fremdschlüsselfeld **LieferantID** der Tabelle **tblArtikel** legt ja den standardmäßig zu verwendenden Lieferanten fest.

Die entsprechende Verknüpfung ist nicht mit Löschweitergabe ausgestattet. Der Versuch, einen Lieferanten zu löschen, der noch über das Fremdschlüsselfeld der Tabelle **tblArtikel** mit einem Artikel verknüpft ist, führt zu einem Fehler.

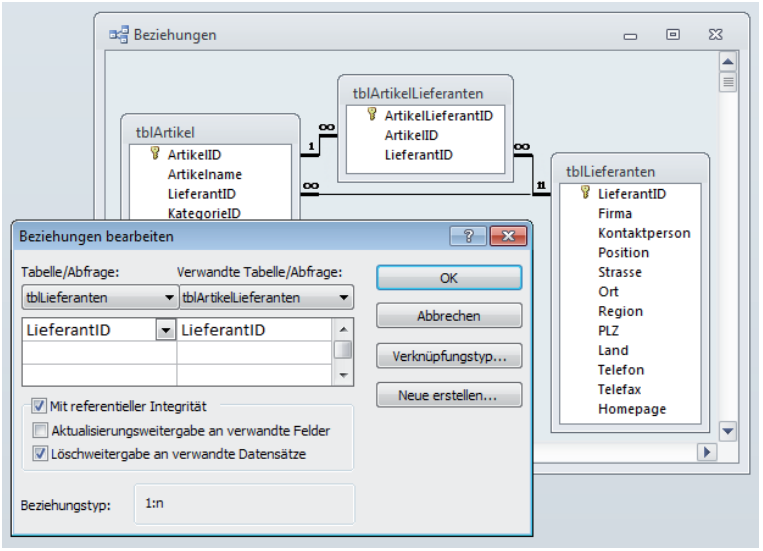


Bild 2: Festlegen der Beziehungen zwischen den Tabellen der m:n-Verknüpfung

Erste Daten hinzufügen

Die Tabelle **tblArtikelLieferanten** soll alle möglichen Lieferanten eines Artikels anzeigen. In der Tabelle **tblArtikel** gibt es bereits ein Fremdschlüsselfeld **LieferantID**, welches den aktuellen Lieferanten festlegen. Natürlich soll dieser Lieferant auch in der Tabelle **tblArtikelLieferanten** auftauchen.

Wie erledigen wir diese Aufgabe? Alle entsprechenden Kombinationen aus **ArtikelID** und **LieferantID** manuell in die Tabelle übertragen? Natürlich nicht! Diese Aufgabe erledigen wir mit einem Einzeiler über den Direktbereich. Dieser sieht wie folgt aus:

```
CurrentDb.Execute "INSERT INTO tblArtikelLieferanten
(ArtikelID, LieferantID) SELECT ArtikelID,
LieferantID FROM tblArtikel", dbFailOnError
```

Sie lösen hiermit eine Aktionsabfrage aus, die für jeden Datensatz der Tabelle **tblArtikel** die Inhalte der Felder **ArtikelID** und **LieferantID** in die entsprechenden Felder der Tabelle **tblArtikelLieferanten** einträgt. Das Resultat sieht später wie in Bild 3 aus.

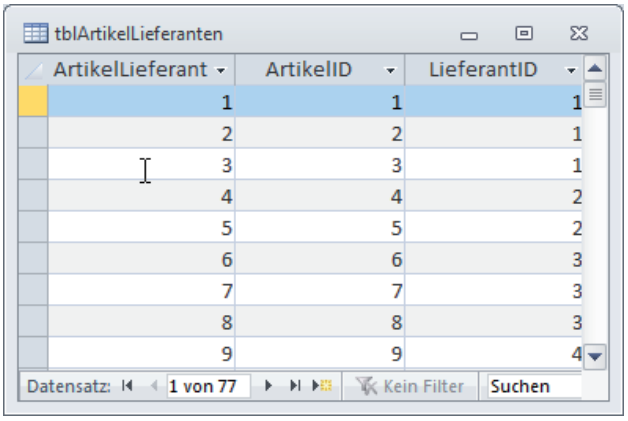


Bild 3: Die Tabelle **tblArtikelLieferanten** mit den Standardlieferanten eines jeden Artikels

Formular aufbauen

Das Formular soll zunächst jeweils einen Datensatz der Tabelle **tblArtikel** anzeigen. Dazu erstellen Sie ein neues Formular in der Entwurfsansicht und stellen die Eigenschaft **Datenherkunft** auf die Tabelle **tblArtikel** ein. Anschließend ziehen Sie die beiden Felder **ArtikelID** und **Artikelname** sowie **LieferantID** in den Detailbereich des Formulars und speichern es unter dem Namen **frmArtikelLieferanten** (siehe Bild 4).

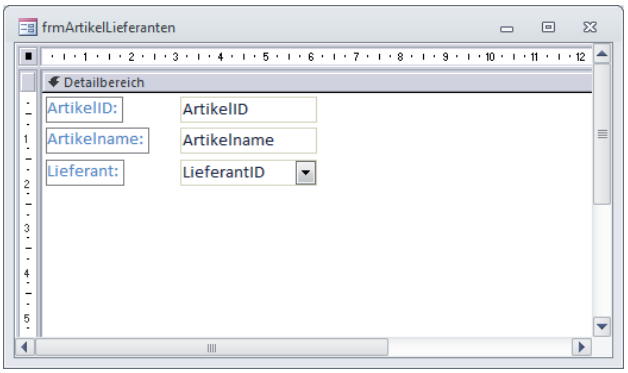


Bild 4: Das Formular zeigt einige Daten der Tabelle **tblArtikel** an.

Danach legen Sie zwei ListView-Steuerelemente namens **lvwLieferantenZugeordnet** und **lvwLieferantenNichtZugeordnet** an (siehe Bild 5). Die Einstellung der Eigenschaften dieser beiden Steuerelemente nehmen wir wiederum per VBA beim Laden des Formulars vor.

Die dazu notwendige Prozedur wird durch das Ereignis **Beim Laden** ausgelöst. Hinterlegen Sie für die entsprechende Eigenschaft den Wert **[Ereignisprozedur]** und klicken Sie auf die Schaltfläche mit den drei Punkten neben der Eigenschaft, um die Ereignisprozedur anzulegen. Ergänzen Sie diese im nun erscheinenden VBA-Editor wie in Listing 1.

Im Kopf des Moduls **Form_frmArtikelLieferanten** deklarieren Sie zunächst zwei Objektvariablen für die beiden ListView-Steuerelemente mit dem Schlüsselwort **WithEvents**. Dies ist nötig, damit wir auf die Ereignisse dieser Nicht-Standard-Steuerelemente von Access zugreifen können.

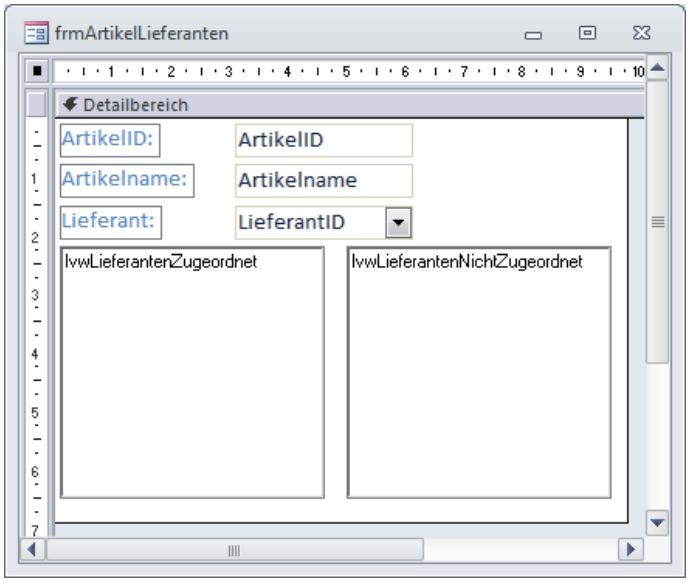


Bild 5: Formular mit den beiden ListView-Steuerelementen

Die folgende Prozedur weist diesen Variablen die entsprechenden Steuerelemente zu. Danach stellt sie einige Eigenschaften ein, die das Aussehen beeinflussen. Außerdem fügt sie jedem Steuerelement eine Überschrift hinzu und stellt die Breite so ein, dass die einzige Spalte die komplette Breite des Steuerelements ausfüllt.

Datenherkunft für die ListView-Steuerelemente

Anschließend benötigen wir zwei Abfragen, welche die Daten für die beiden ListView-Steuerelemente liefern. Die Abfrage für das ListView-Steuerelement **lvwLieferantenZugeordnet** ist die einfachere der beiden Abfragen. Sie enthält die beiden Tabellen **tblArtikelLieferanten** und **tblLieferanten**. Die Tabelle **tblLieferanten** steuert das Feld **Firma** bei, die Tabelle **tblArtikelLieferanten** die beiden Felder **LieferantID** und **ArtikelID**. Das Feld **LieferantID** dient später zum Füllen der **Key**-Eigenschaft der **ListView**-Elemente, das Feld **Firma** liefert den in der einzigen Spalte anzuzeigenden Wert und das Feld **ArtikelID** hilft dabei, die zum aktuell im Formular angezeigten Artikel gehörenden Datensätze dieser Abfrage zu filtern (siehe Bild 6).

```
Dim WithEvents objLvwLieferantenZugeordnet As MSComctlLib.ListView
Dim WithEvents objLvwLieferantenNichtZugeordnet As MSComctlLib.ListView

Private Sub Form_Load()
    Set objLvwLieferantenZugeordnet = Me!lvwLieferantenZugeordnet.Object
    Set objLvwLieferantenNichtZugeordnet = Me!lvwLieferantenNichtZugeordnet.Object
    With objLvwLieferantenZugeordnet
        .View = lvwReport
        .Appearance = ccFlat
        .BorderStyle = ccNone
        .FlatScrollBar = True
        .ColumnHeaders.Clear
        .ColumnHeaders.Add , , "Lieferant zugeordnet", Me!lvwLieferantenZugeordnet.Width
    End With
    With objLvwLieferantenNichtZugeordnet
        .View = lvwReport
        .Appearance = ccFlat
        .BorderStyle = ccNone
        .FlatScrollBar = True
        .ColumnHeaders.Clear
        .ColumnHeaders.Add , , "Lieferant nicht zugeordnet", Me!lvwLieferantenNichtZugeordnet.Width
    End With
End Sub
```

Listing 1: Einrichten der beiden ListView-Steuerelemente

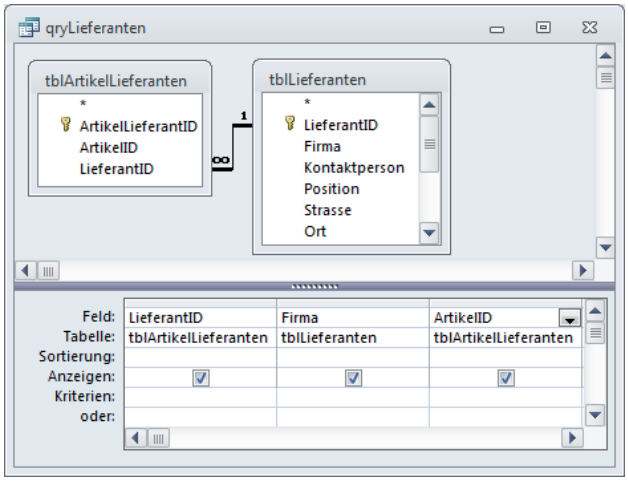


Bild 6: Datenherkunft für das erste **ListView**-Steuerelement zur Anzeige der zugeordneten Lieferanten

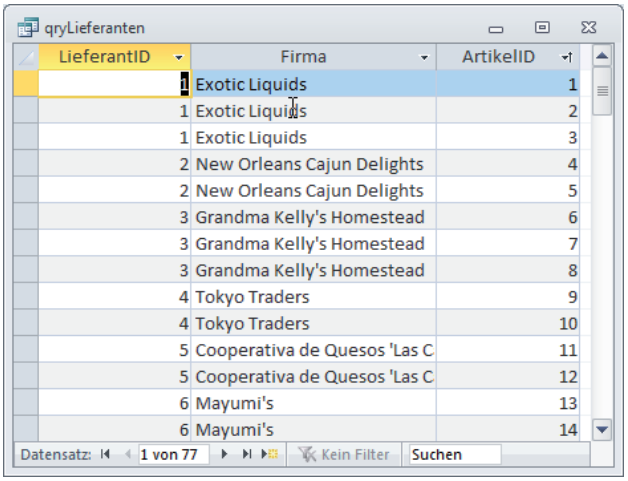


Bild 7: Daten für das erste **ListView**-Steuerelement, allerdings noch nicht nach **ArtikelID** gefiltert

Diese Abfrage liefert ungefiltert die Werte aus Bild 7. Diese müssen vor der Anzeige im ersten **ListView**-Steuerelement noch gefiltert werden.

Wie aber übermitteln wir die **ArtikelID**, deren Lieferanten angezeigt werden sollen? Dazu fügen Sie der Abfrage einen Parameter als Kriterium hinzu. Tragen Sie dazu den Ausdruck **[prmArtikelID]** in der Zeile Kriterium der Spalte **ArtikelID** ein. Wenn Sie die Abfrage nun aufrufen, fragt Access in einem kleinen Dialog nach dem Wert, der als Kriterium verwendet werden sollen. Dies müssen Sie natürlich später nicht von Hand erledigen, das Kriterium wird per VBA übermittelt. Damit das Formular im ersten **ListView**-Steuerelement die hier ermittelten Datensätze ausgibt, verwenden wir eine kleine Prozedur, die eine weitere Prozedur gleich zweimal hintereinander aufruft. Diese Prozedur soll durch das Ereignis **Beim Anzeigen** des Formulars ausgelöst werden, welches beim Anzeigen eines jeden Datensatzes feuert.

Dabei übergibt sie beim ersten Aufruf den Verweis auf das linke **ListView**-Steuerelement, beim zweiten das rechte. Der zweite Parameter ist der Name der Abfrage, die als Datenherkunft beim Füllen der **ListView**-Steuerelemente zum Einsatz kommen soll. Die erste kennen Sie bereits – es ist die Abfrage **qryLieferanten**:

```
Private Sub Form_Current()
    ListeAktualisieren objLvwZugeordnet, _
        "qryLieferanten"
    ListeAktualisieren objLvwNichtZugeordnet, _
        "qryNichtLieferanten"
End Sub
```

```
Private Sub ListeAktualisieren(objListView As ListView, strAbfrage As String)
    Dim db As DAO.Database
    Dim qdf As DAO.QueryDef
    Dim prm As DAO.Parameter
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set qdf = db.QueryDefs(strAbfrage)
    Set prm = qdf.Parameters("prmArtikelID")
    prm.Value = Me!ArtikelID
    Set rst = qdf.OpenRecordset
    objListView.ListItems.Clear
    Do While Not rst.EOF
        objListView.ListItems.Add , "a" & rst!LieferantID, rst!Firma
        rst.MoveNext
    Loop
    rst.Close
    Set rst = Nothing
    Set db = Nothing
End Sub
```

Listing 2: Prozedur zum Füllen eines ListView-Steuerelements auf Basis der angegebenen Abfrage

Programmieren mit Arrays

Dass Sie unter Access Daten in Tabellen speichern und gezielt darauf zugreifen können, wissen Sie als Access [basics]-Leser schon längst. Aber was, wenn Sie nur ein paar gleichartige Daten zwischenspeichern und wieder darauf zugreifen möchten? Legen Sie dann im VBA-Code für jedes Element eine eigene Variable an? Nein! Hier kommen zum Beispiel die Arrays, auch Datenfelder genannt, zum Einsatz. Der vorliegende Artikel zeigt, wie Sie Array erstellen, mit Daten füllen und die Daten wieder auslesen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1301_Arrays.mdb**.

Arrays

Array sind prinzipiell eine Art Tabelle, wobei Sie nicht über Datensätze und Feldnamen auf den in einem Elemend des Array gespeicherten Wert zugreifen, sondern über Index-Werte.

Zusätzlich kann ein Array eine, zwei oder auch mehr Dimensionen enthalten und diese mit Elementen füllen.

Ein Array mit einer Dimension würde beispielsweise einer Tabelle mit einem einzigen Feld entsprechen und ein Array mit zwei Dimensionen einer Tabelle mit einem bis n Feldern. Arrays mit mehr als zwei Dimensionen lassen sich schlecht auf Tabellen übertragen.

Eindimensionale Arrays

Mit einem eindimensionalen Array lassen sich einfache Wertereihen speichern – beispielsweise die Monatsnamen. Um die Vereinfachung durch den Einsatz von Array zu verdeutlichen, schauen wir uns zunächst an, wie Sie Monatsnamen auf umständliche Weise in Variablen speichern können. Dazu legen Sie in einem Modul namens **mdlArrays** (VBA-Editor mit **Alt + F11**) öffnen, dann **Einfügen|Modul** auswählen) zunächst die öffentlich deklarierten String-Variablen an (dieses und die folgenden Beispiele finden Sie im Modul **mdlArrays**):

```
Public strMonat01 As String
Public strMonat02 As String
Public strMonat03 As String
...
Public strMonat12 As String
```

In einer Prozedur namens **MonateInVariablen** füllen Sie diese mit den entsprechenden Werten:

```
Public Sub MonateInVariablen()
    strMonat01 = "Januar"
    strMonat02 = "Februar"
    strMonat03 = "März"
    ...
    strMonat12 = "Dezember"
End Sub
```

Führen Sie diese Prozedur aus, indem Sie die Einfügemarke in der Prozedur platzieren und auf **F5** drücken. Danach können Sie die Werte der einzelnen Variablen etwa im Direktfenster (**Strg + G**) wie folgt abfragen:

```
Debug.Print strMonat03
März
```

Wenn Sie per VBA auf eine der Variablen zugreifen möchten, um den Namen des Monats zu ermitteln, müssen Sie immer den genauen Variablen verwenden. Es gibt keine Möglichkeit, die zwölf Variablen beispielsweise innerhalb einer Schleife schnell auszugeben.

Dies ist ein tolles Betätigungsfeld für Array. Ein Array, das eine bestimmte Menge Daten gleichen Datentyps aufnehmen soll (dies ist eine Einschränkung, die Sie beachten müssen), deklarieren Sie fast genauso wie eine normale Variable dieses Datentyps. Der einzige Unterschied ist, dass Sie den größten zu erwartenden Wert für den Index hinter dem Variablennamen in Klammern angeben:

```
Public strMonate(11) As String
```

Im Gegensatz zu den weiter unten besprochenen dynamischen Arrays handelt es sich hierbei um ein statisches Array.

Sicher werden Sie sich fragen, warum hier die Zahl **11** und nicht die der Anzahl der Monate entsprechende Zahl **12** angegeben wird. Der Grund ist, dass die Indexwerte standardmäßig immer mit **0** beginnen. Das bedeutet, das die Variable strMonate Werte für die Indexwerte 0 bis 11 aufnehmen kann. Der erste

Eintrag des Arrays **strMonate** wird also mit dem Indexwert **0**, der zweite mit **1** und so weiter festgelegt:

```
Public Sub MonateInArray()
    strMonate(0) = "Januar"
    strMonate(1) = "Februar"
    strMonate(2) = "März"
    ...
    strMonate(11) = "Dezember"
End Sub
```

Dementsprechend liefert etwa die Abfrage des Wertes mit dem Index **3** den vierten Eintrag des Arrays, hier **April**:

```
Debug.Print strMonate(3)
April
```

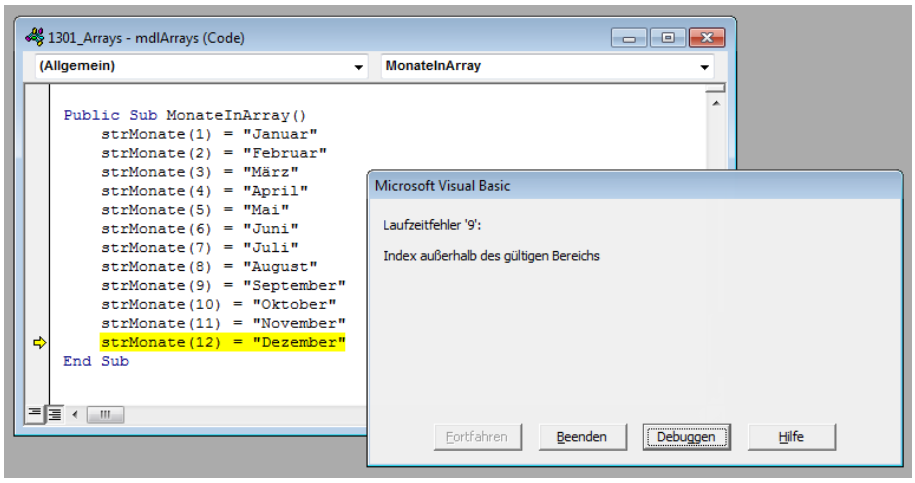


Bild 1: Versuch, ein Array mit einem zu großen Indexwert zu füllen

Nun scheint es unkomfortabel zu sein, nicht die dem Monat entsprechende Zahl für den Zugriff auf das Array zu verwenden. Dies lässt sich jedoch leicht ändern. Wenn Sie nur einen Wert für die Anzahl der Elemente angeben, geht VBA davon aus, dass der Index 0-basiert verwaltet wird. Sie können den Wertebereich für den Indexbereich jedoch detailliert festlegen. In diesem Fall möchten wir gern die Indexwerte 1 bis 12 verwenden. Dazu deklarieren Sie die Array-Variable wie folgt:

```
Public strMonate(1 To 12) As String
```

Die Monatsnamen weisen Sie dementsprechend nun mit den Indexwerten 1 bis 12 zu:

```
Public Sub MonateInArray()
    strMonate(1) = "Januar"
    strMonate(2) = "Februar"
    strMonate(3) = "März"
    ...
    strMonate(12) = "Dezember"
End Sub
```

Somit liefert das Array nun auch bei der Ausgabe die gewünschten Werte:

```
Debug.Print strMonate(3)
März
```

Was aber geschieht, wenn Sie das Array nullbasiert deklarieren, aber dann nicht die Indexwerte **0** bis **11**, sondern **1** bis **12** verwenden? Dann tritt der Fehler aus Bild 1 auf.

Das Array wurde nur für Indexwerte bis maximal **11** deklariert, also ist der Indexwert **12** nicht zulässig. Das gilt andersherum genauso – wenn Sie also ein Array für die Indexwerte **1** bis **12** deklarieren und das Element mit dem Indexwert **0** füllen wollen, lösen Sie ebenfalls einen Fehler aus.

Alle Werte auslesen

Kommen wir gleich zu einem wesentlichen Vorteil von Arrays gegenüber einzelnen, durchnummerierten Variablen: Sie können diese mit wenigen Zeilen per Code durchlaufen.

Wenn Sie wissen, welche für welche Indexwerte Sie die Elementinhalte ausgeben möchten, benötigen Sie nur die folgende Prozedur dazu:

```
Public Sub MonateDurchlaufen()
    Dim i As Integer
    Call MonateInArray
    For i = 1 To 12
        Debug.Print strMonate(i)
    Next i
End Sub
```

Die Prozedur deklariert eine Laufvariable namens **i** und füllt das Array **strMonate()** mit der oben erläuterten Prozedur **MonateInArray**. Danach durchläuft es eine Schleife und füllt dabei die Laufvariable **i** nacheinander mit den Werten von **1** bis **12**. Die einzige Anweisung innerhalb der Schleife gibt den jeweiligen Wert des Arrays **strMonate()** für den Index **i** aus.

Reihenfolge individuell einstellen

In der Regel sorgen Sortierungen nach dem Inhalt bestimmter Felder für die richtige Reihenfolge bei der Anzeige von Daten. Es gibt jedoch Ausnahmen, bei denen Sie ein individuelles Kriterium für die richtige Sortierung benötigen. Dazu fügen Sie einer Tabelle ein eigenes Feld hinzu und füllen dieses mit entsprechenden Zahlenwerten. Wie aber ändert man die Reihenfolge komfortabel? Die Antwort liefert der vorliegende Artikel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1301_ReihenfolgeIndividuellEinstellen.mdb**.

Individuelle Reihenfolge

Wenn Sie beispielsweise Kunden in einer Kundenliste nach der Beliebtheit sortieren möchten, können Sie diese Sortierung über entsprechende Werte in einem Feld namens **ReihenfolgeID** vorgeben. Um die Datensätze eindeutig zu sortieren, darf dieses Feld nur eindeutige Werte aufnehmen – also beispielsweise die Zahlen von **1** bis **n**. Dazu stellen Sie die Eigenschaft **Indiziert** des Feldes **ReihenfolgeID** auf den Wert **Ja (Ohne Duplikate)** ein (siehe Bild 1).

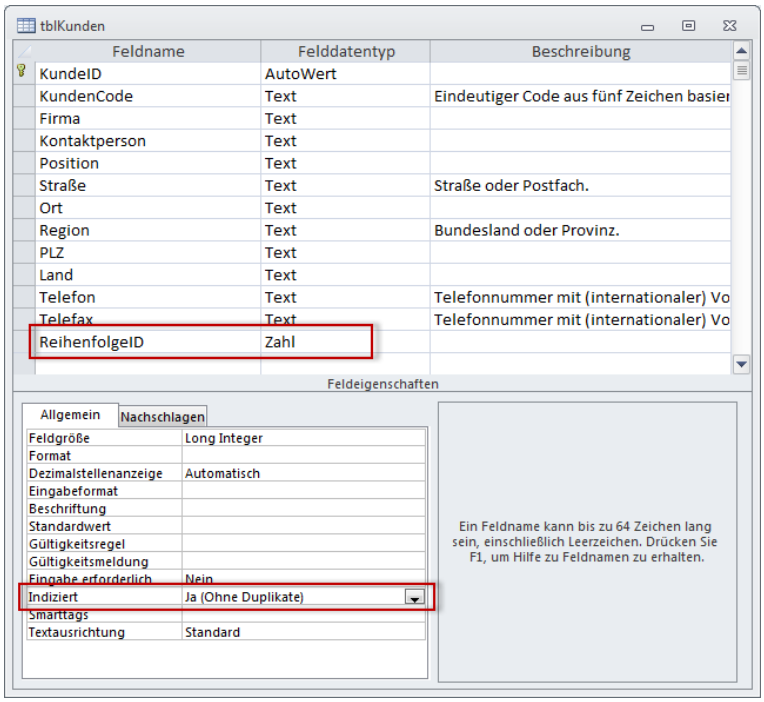


Bild 1: Hinzufügen eines Feldes für die Definition einer individuellen Reihenfolge

Nun weisen Sie allen Datensätzen einen der Beliebtheit entsprechenden Zahlenwert zu und sortieren diese nach dem Feld **ReihenfolgeID**. Das Feld **ReihenfolgeID** erhält einen numerischen Datentyp und sollte den gleichen Wertebereich wie das Primärschlüsselfeld der Tabelle enthalten. Das ist grundsätzlich die Lösung für eine individuelle Sortierung. Allerdings ist diese nicht besonders komfortabel: Wer will schon manuell die Zahlenwerte für die Reihenfolge von Datensätzen eintragen?

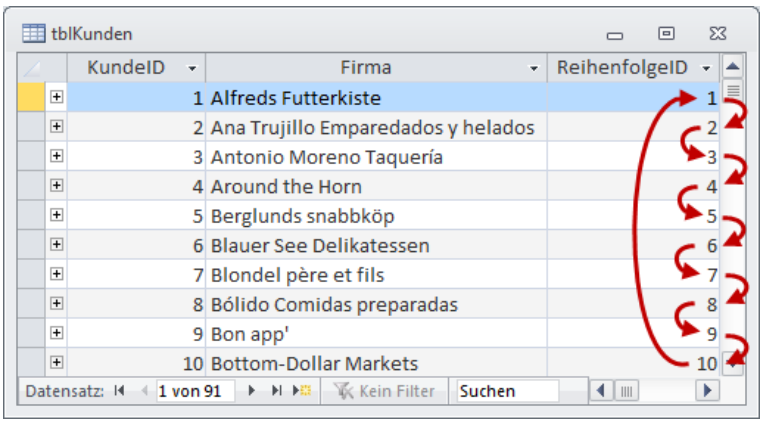


Bild 2: Änderungen im Feld **ReihenfolgeID** beim Verschieben des zehnten Eintrags an die erste Position

Wenn Sie beispielsweise den zehnten Eintrag einer Liste an die erste Stelle verschieben möchten, legen Sie für das Feld **Reihenfolge** des zu verschiebenden Datensatzes den Wert **1** ein. Der Datensatz, der bislang den Wert **1** im Feld **Reihenfolge** enthielt, bekommt die **2**, der mit der **2** die **3** und so weiter.

Wenn Sie einen eindeutigen Index für das Feld **ReihenfolgeID** festgelegt haben, wird es noch komplizierter – Sie müssen dann für das zu verschiebende Element temporär einen neuen Wert festlegen (zum Beispiel **999** oder einfach einen Wert, der um eins größer als der bislang größte Wert ist), dann dem neunten Datensatz den Wert **10**, dem achten den Wert **9** und so weiter zuweisen, bevor der zehnte Datensatz den Wert **1** im Feld **ReihenfolgeID** erhält (siehe Bild 2).

Was aber ist nun der Unterschied zwischen dem Reihenfolge-Feld und anderen Feldern, nach denen Sie Datensätze sortieren – beispielsweise Preis, Gewicht et cetera? Ganz klar: Das Reihenfolge-Feld hat, genau wie in vielen Fällen das Primärschlüsselfeld einer Tabelle, nur eine einzige Aufgabe – und die dient nicht dem Speichern von Geschäftsdaten. Sondern – in diesem Fall – dem Festhalten einer allein vom Benutzer definierten Reihenfolge. Im Gegensatz dazu hat das Primärschlüsselfeld meist einzig und allein die Aufgabe, einen Datensatz eindeutig zu kennzeichnen. Manchmal hat der Primärschlüsselwert auch noch andere Aufgaben und dient beispielsweise als Kundennummer oder Artikelnummer.

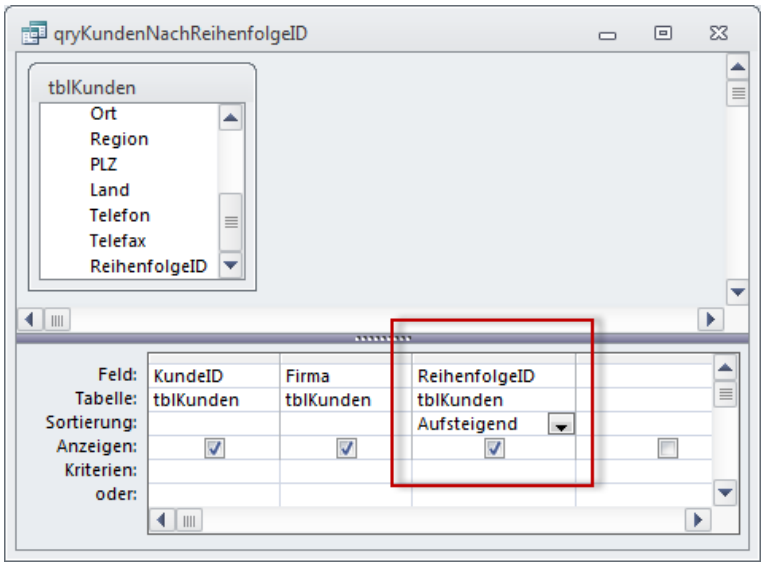


Bild 3: Abfrage zur Sortierung der Kunden nach dem Wert des Feldes **ReihenfolgeID**

Sortieren der Datensätze nach der Reihenfolge

Zum Sortieren der Datensätze nach den Werten des Feldes **ReihenfolgeID** benötigen Sie eine Abfrage. Diese heißt **qryKundenNachReihenfolgeID** und enthält zu Beispielzwecken nur einige wenige Felder der als Datenherkunft dienenden Tabelle – darunter natürlich das Feld **ReihenfolgeID** und, zur fixen eindeutigen Identifizierung, das Feld **KundeID**. Für das Feld **ReihenfolgeID** legen Sie die aufsteigende Sortierung fest (siehe Bild 3). Auf diese Weise sortiert die Tabelle das Ergebnis der Abfrage allein nach dem Wert des Feldes **ReihenfolgeID**.

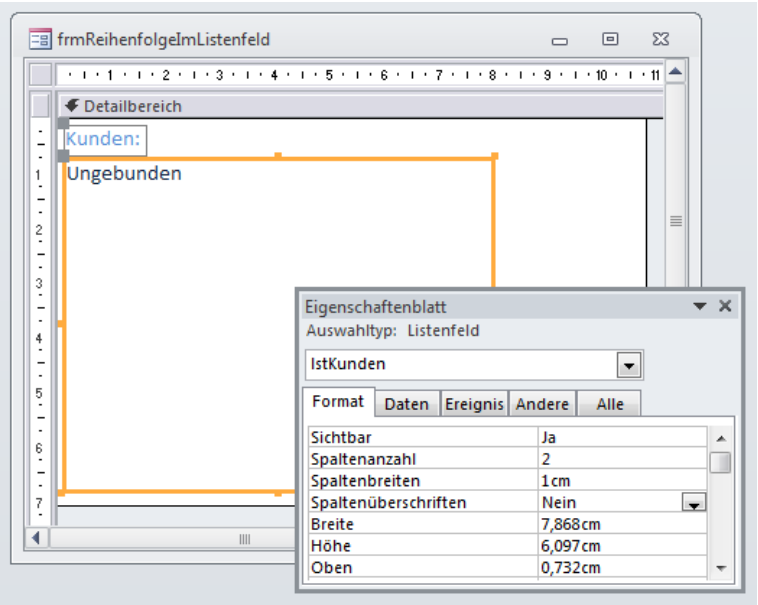


Bild 4: Listenfeld mit den nach dem Feld **ReihenfolgeID** sortierten Datensätzen in der Entwurfsansicht

Reihenfolge komfortabel einstellen

Wie bereits erwähnt, werden Sie die Reihenfolge nicht durch manuelles Anpassen der Werte des Feldes **ReihenfolgeID** einstellen wollen. Deshalb erstellen Sie nun ein Formular, das die Datensätze der Tabelle **tblKunden** in einem Listenfeld anzeigt und einige Steuerelemente enthält, um die Position des aktuell markierten Datensatzes anzupassen. Später zeigen wir noch die Anpassung der Reihenfolge für die in einem Unterformular in der Datenblattansicht angezeigten Daten an.

Das Listenfeld im Formular **frmReihenfolgeImListenfeld** enthält die Abfrage **qryKundenNachReihenfolgeID** als Datenherkunft. Listenfelder sollen normalerweise nicht die Werte des Primärschlüsselwertes anzeigen, aber in diesem Fall schon – dies erleichtert die Prüfung, ob die nachfolgend hinzugefügten Schaltflächen zum Anpassen der Reihenfolge funktionieren. Geben Sie dazu den Wert **2** für die Eigenschaft **Spaltenanzahl** und **1cm** für die Eigenschaft **Spaltenbreiten** ein (siehe Bild 4). Dadurch wird die erste Spalte mit einer Breite von einem Zentimeter angezeigt, die zweite Spalte nimmt die restliche Breite ein. Wie ändern wir nun die Reihenfolge? Es gibt mehrere Möglichkeiten. Wenn Sie für andere Benutzer programmieren, sollten Sie eine offen-

sichtliche Variante verwenden – also beispielsweise entsprechende Schaltflächen. Das Betätigen dieser Schaltflächen wirkt sich immer auf den jeweils markierten Datensatz aus. Die Schaltflächen sollen die folgenden Funktionen ermöglichen:

- Eintrag um eine Position nach oben verschieben
- Eintrag um eine Position nach unten verschieben
- Eintrag ganz nach oben verschieben
- Eintrag ganz nach unten verschieben

Später soll das Formular wie in Bild 5 aussehen. Unten rechts sehen Sie noch eine Schaltfläche mit einer wichtigen Funktion – dem initialen Festlegen der ReihenfolgeID oder zum Zurücksetzen dieses Feldes.

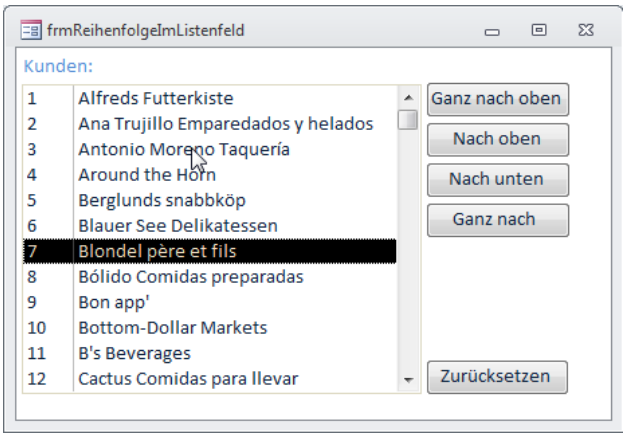


Bild 5: Formular mit den Schaltflächen zum individuellen Einstellen der Reihenfolge

Eintragen der ReihenfolgeID

Wenn Sie den Daten erst nachträglich das Feld **ReihenfolgeID** hinzugefügt haben, müssen Sie dieses noch füllen. Dies soll die Schaltfläche **cmdZuruecksetzen** erledigen, wenn der Benutzer sie anklickt. Dazu legen Sie eine entsprechende Ereignisprozedur an (Schaltfläche markieren, Eigenschaft **Beim Klicken** auf **[Ereignisprozedur]** einstellen und auf die Schaltfläche mit den drei Punkten klicken) und ergänzen diese wie folgt:

```
Private Sub cmdZuruecksetzen_Click()
    ReihenfolgeZuruecksetzen
End Sub
```

Die durch die einzige Anweisung dieser Prozedur aufgerufene Routine sieht wie in Listing 1 aus. Die Prozedur deklariert ein **Database**- und ein **Recordset**-Objekt. Das **Database**-Objekt **db** wird mit einem Verweis auf die aktuelle Datenbank gefüllt, **rst** nimmt ein Recordset mit allen Datensätzen der Abfrage **qryKundenNachReihenfolgeID** auf. Anschließend durchläuft eine **Do While**-Schleife alle Datensätze dieses Recordsets. Dabei wird der jeweils aktuelle Datensatz zum Bearbeiten vorbereitet, der Wert des Feldes **ReihenfolgeID** angepasst und der Datensatz gespeichert. Welchen Wert erhält das Feld **ReihenfolgeID**? Den Wert der Eigenschaft **AbsolutePosition**, also der Position des Datensatzzeigers, plus eins – weil AbsolutePosition immer mit **0** beginnt, wir aber eine Nummerierung von **1** beginnend brauchen.

Datensätze verschieben

Wir schauen uns nun die für die zweite Schaltfläche notwendige Funktion an. Ein Klick auf diese Schaltfläche soll den aktuellen Datensatz um eine Position nach oben verschieben. Dazu führt die Prozedur folgende Schritte durch – bezogen auf Bild 6, wo der Datensatz mit dem Wert **7** im Feld **KundeID** (**ReihenfolgeID** ist auch **7**) mit dem darüber liegenden Datensatz vertauscht werden soll:

```
Public Sub ReihenfolgeZuruecksetzen()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM qryKundenNachReihenfolgeID ORDER BY KundeID", dbOpenDynaset)
    Do While Not rst.EOF
        rst.Edit
        rst!ReihenfolgeID = rst.AbsolutePosition + 1
        rst.Update
        rst.MoveNext
    Loop
    Set db = Nothing
End Sub
```

Listing 1: Zurücksetzen des Feldes **ReihenfolgeID**

- Ermitteln des Primärschlüsselwertes des zu verschiebenden Datensatzes (hier **6** im Feld **KundeID**)
- Ermitteln der **ReihenfolgeID** des zu verschiebenden Datensatzes (ebenfalls 7)
- Ermitteln der **ReihenfolgeID** des Zieldatensatzes (**6**)
- Ermitteln eines noch nicht verwendeten Wertes für das Feld **ReihenfolgeID** (hier **108**)
- Einstellen der **ReihenfolgeID** für den zu verschiebenden Datensatz auf den temporären ReihenfolgeID-Wert (**92**)
- Einstellen der **ReihenfolgeID** des Ziel-Datensatzes auf den ursprünglichen Wert des zu verschiebenden Datensatzes (**7**)
- Einstellen der **ReihenfolgeID** des zu verschiebenden Datensatzes auf die ursprüngliche **ReihenfolgeID** des Zieldatensatzes (**6**)
- Aktualisieren des Listenfeldes

Die erste Frage beim Ändern der Reihenfolge eines Datensatzes lautet: Welcher Datensatz soll überhaupt verschoben werden? Dies ist beim Listenfeld einfach zu beantworten – zumindest, wenn die erste Spalte die Werte des Primärschlüsselfeldes der Datensatzherkunft enthält. Dann liefert die **Value**-Eigenschaft des Listenfeldes, hier **lstKunden**, den gewünschten Wert. Diesen prüft man optimalerweise direkt zu Beginn der durch den Mausklick ausgelösten Prozedur und führt die Änderung der Reihenfolge nur durch, wenn auch ein Datensatz markiert ist.

Die Prozedur sieht wie in Listing 2 aus, dort befinden sich alle notwendigen Schritte in einer entsprechenden **If...Then**-Bedingung, die nur wahr ist, wenn ein Listenfeldeintrag markiert ist.

In diesem Fall speichert die Prozedur zunächst den Wert des Feldes **KundeID** in der Variablen **lngZuVerschiebenID**. Danach ermittelt die Prozedur die aktuelle **ReihenfolgeID** des zu verschiebenden Datensatzes, und zwar mit **DLookup**. Dieser Aufruf holt den Wert des Feldes **ReihenfolgeID** für den Datensatz der Tabelle **tblKunden** mit dem zuvor ermittelten Wert im Feld **KundeID**. Nun benötigen Sie die **ReihenfolgeID** des Zieldatensatzes, in diesem Fall der

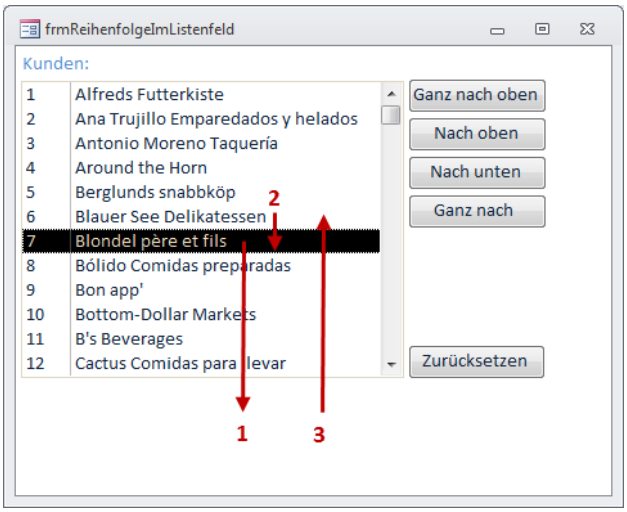


Bild 6: Drei Schritte zum Ändern der Reihenfolge zweier Elemente

in der Reihenfolge direkt vor dem zu verschiebenden Datensatz liegende. Dazu verwenden Sie die **DMax**-Funktion, die den größten Wert des Feldes **ReihenfolgeID** liefert, der kleiner als die **ReihenfolgeID** des zu verschiebenden Datensatzes ist.

Schließlich fehlt noch ein temporärer **ReihenfolgeID**-Wert, den Sie dem zu verschiebenden Datensatz zuweisen können, während der Zieldatensatz mit der **ReihenfolgeID** des zu verschiebenden Datensatzes gefüllt wird. Zur Erinnerung: Das Feld **ReihenfolgeID** ist mit einem eindeutigen Index ausgestattet, Sie können also nicht einfach dem zu verschiebenden Datensatz die Ziel-**ReihenfolgeID** zuweisen oder umgekehrt – diese käme dann zweifach vor.

Die temporäre Reihenfolge holt die Prozedur mit der **DMax**-Funktion, wobei der größte Wert des Feldes **ReihenfolgeID** in der Tabelle **tblKunden** eingelesen und um eins erhöht wird.

Nun folgen genau drei **UPDATE**-Aktionsabfragen. Die erste stellt den Wert des Feldes **ReihenfolgeID** des zu verschiebenden Datensatzes (mit der Bedingung **WHERE KundeID = 7** definiert) auf den temporären Wert ein, in diesem Beispiel **92**.

Der vorherige Wert **7** ist nun nicht mehr vergeben. Somit kann die zweite **UPDATE**-Anweisung diesen dem Feld **ReihenfolgeID** des Zieldatensatzes zuweisen. Dieser wird über den vorherigen Wert im Feld **ReihenfolgeID** gekennzeichnet (**WHERE ReihenfolgeID = 6**).

Schließlich verschiebt die dritte **UPDATE**-Anweisung den nach hinten geschobene Datensatz an seine Ziel-

Feiertage verwalten, Teil 1

Wenn Sie in einer Datenbank jegliche Datumsangaben oder Zeiträume verwalten, die irgendwie durch die Anwesenheit von Feiertagen beeinflusst werden, sollten Sie diese Feiertage in einer Tabelle bereithalten oder diese schnell per VBA berechnen können. Dabei kann es sich etwa um die Berechnung von Fristen handel (Rückgabe innerhalb von 7 Werktagen) oder auch um die Ermittlung der Urlaubstage für den durch zwei Datumsangaben festgelegten Zeitraum. Diese Artikelreihe zeigt, wie Sie die Basisdaten verwalten und daraus die tatsächlichen Feiertage in einem vorgegebenen Zeitraum ermitteln.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1301_FeiertageErmitteln.mdb**.

Feiertage

Es gibt verschiedene Arten von Feiertagen:

- solche, die immer auf das gleiche Datum fallen (Weihnachten, Silvester, Tag der Arbeit, Tag der deutschen Einheit) und
- solche, die immer auf einen bestimmten Wochentag fallen, aber kein festes Datum haben. Diese teilen sich wiederum in zwei Gruppen auf: Die ersten haben einen bestimmten Abstand zum Ostersonntag (Osterfeiertage, Christi Himmelfahrt, Fronleichnam), die zweiten zum vierten Advent (Weihnachten, Buß- und Bettag).

Außerdem werden einige Feiertage nicht in allen, sondern nur in bestimmten Bundesländern gefeiert.

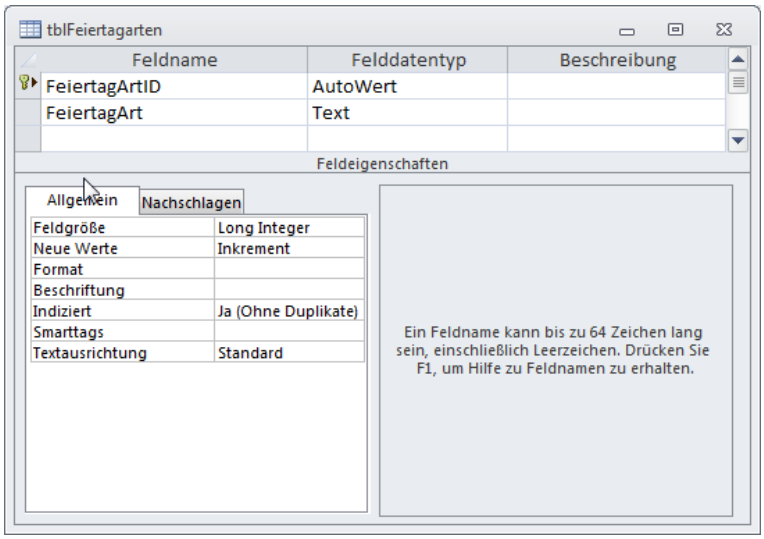


Bild 1: Tabelle der Feiertagsarten in der Entwurfsansicht

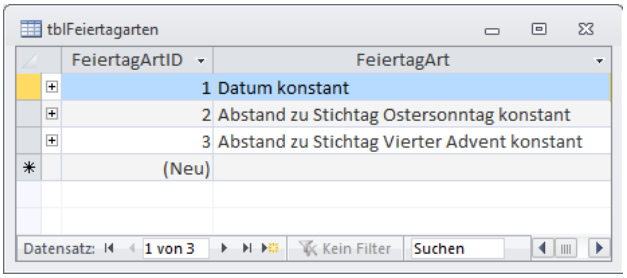


Bild 2: Tabelle der Feiertagsarten mit allen Werten

Tabelle oder dynamisch?

Die Ermittlung von Feiertagen kann auf zwei Arten erfolgen – dynamisch zur Laufzeit, also mit einer geeigneten VBA-Prozedur, oder durch das Speichern der Feiertage in einer Tabelle und anschließendes Ermitteln der Feiertage. Wenn wir schon mit Access arbeiten, nutzen wir natürlich die Variante mit der Tabelle. Mit dieser Tabelle lässt sich anschließend leicht ermitteln, ob und wieviele Feiertage sich in einem bestimmten Zeitraum befinden.

Basistabellen

Wir wissen nun, dass Feiertage bestimmte Eigenschaften haben. Um die Feiertagsdaten für einen Zeitraum zu ermitteln, benötigen wir einen Satz grundlegender Informationen, die wir – der Natur von Access entsprechend – in Tabellen speichern. Wir wissen, dass Feiertage teils ein festes Datum haben und teils von anderen Feiertagen abhängen. Deshalb speichern wir die grundlegenden Informationen in einem Satz von Tabellen. Die erste legt fest, welche Arten von Feiertagen es gibt – mit fixem Datum, von Ostern abhängige oder vom vierten Advent abhängige. Diese drei Varianten speichern wir in einer Tabelle namens **tblFeiertagsarten**, die im Entwurf wie in Bild 1 aussieht.

Gefüllt sieht die Tabelle wie in Bild 2 aus. Was machen wir mit dieser Tabelle? Wir verwenden sie als Lookup-Tabelle für eine weitere Tabelle mit den eigentlichen Basisdaten der einzelnen Feiertage. Die-

se heißt **tblFeiertageBasis** und sieht im Entwurf wie in Bild 3 aus.

In dieser Tabelle finden Sie Daten wie in Bild 4. Zu jedem Feiertag speichert die Tabelle

- die Bezeichnung,
- den Tag und den Monat (für fixe Feiertage, sonst den Wert **0**),
- die Anzahl der Tage bis zum Stichtag, sollte es sich nicht um einen fixen Feiertag handeln und
- die Art des Feiertags.

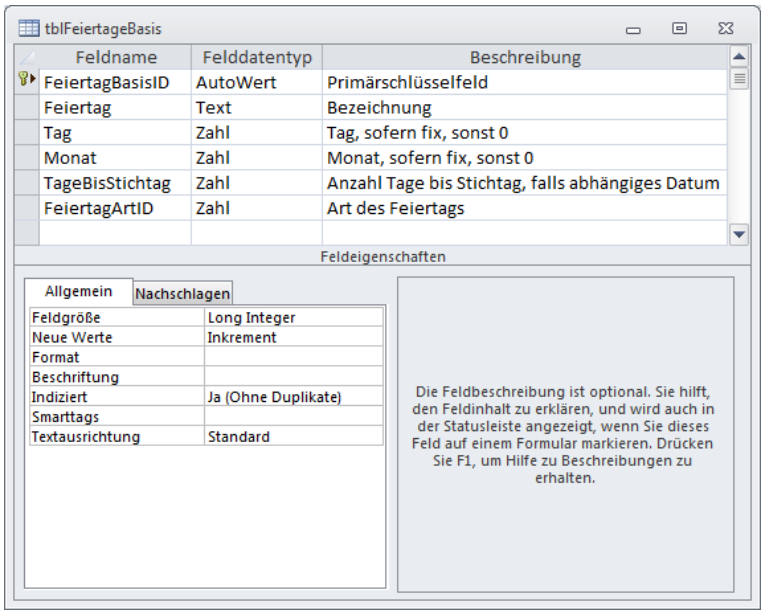


Bild 3: Tabelle der Feiertag-Basisdaten in der Entwurfsansicht

Was fehlt? Die Zuordnung der Feiertage zu den Bundesländern. Wie speichern wir diese Information in Tabellen? Es gibt **n** Feiertage und **m** Bundesländer, von denen jeder Feiertag einem oder mehreren Bundesländern zugeordnet werden soll. Das hört sich nach einer klassischen m:n-Beziehung an – und so ist es auch. Die eine Tabelle der m:n-Beziehung haben wir schon – die Tabelle **tblFeiertageBasis**. Fehlt noch die Tabelle mit den Bundesländern und die Tabelle für die Verknüpfung der Bundesländer mit den Feiertagen.

Die Tabelle zum Speichern der Bundesländer besteht lediglich aus den beiden Feldern **BundeslandID** und **Bundesland**. Interessanter ist die Tabelle zum Herstellen der Verknüpfung zwischen Feiertagen und Bundesländern. Sie heißt **tblFeiertageBundeslaender** und enthält drei Felder: das Primärschlüsselfeld **FeiertagBundeslandID**, das Fremdschlüsselfeld **FeiertagID** und das Fremdschlüsselfeld **BundeslandID**.

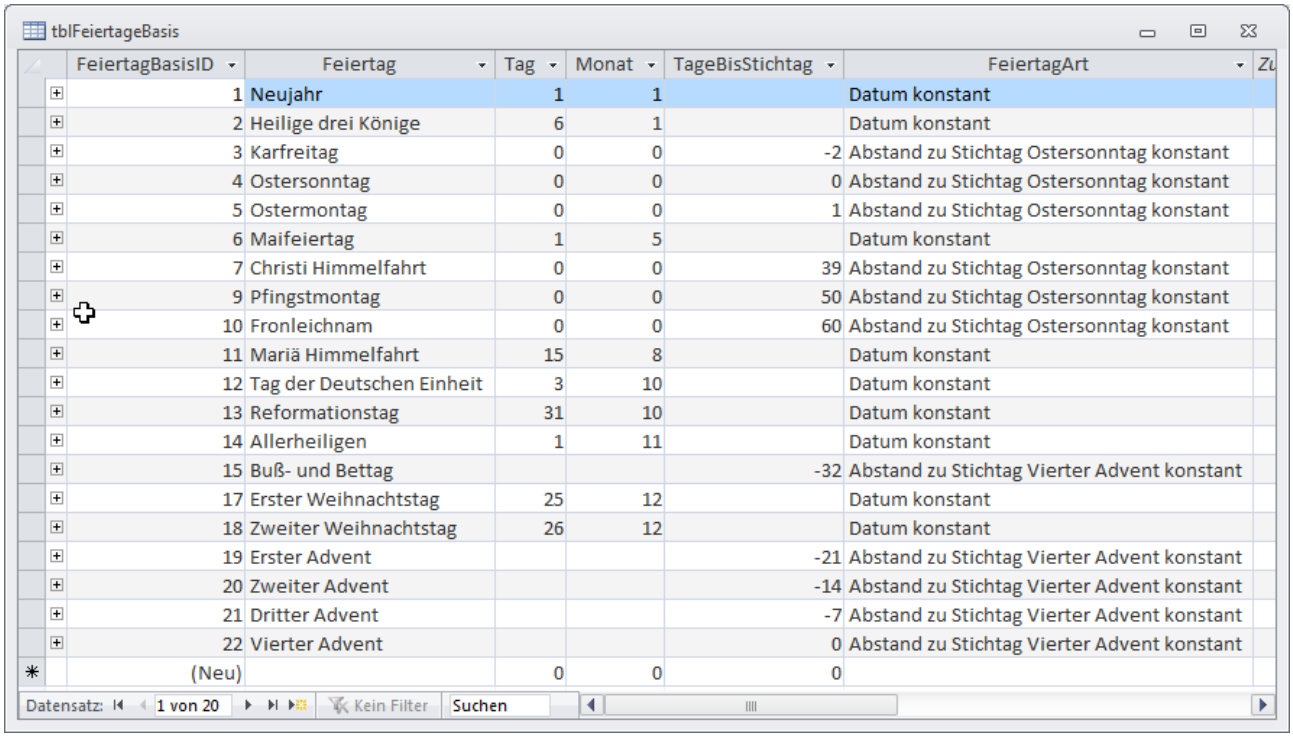


Bild 4: Tabelle der Feiertag-Basisdaten in der Entwurfsansicht

Beide Felder legen Sie als Nachschlagefelder aus. Das Feld **FeiertagID** bezieht die Werte des Nachschlagefeldes aus den Feldern **FeiertagBasisID** und **Feiertag** der Tabelle **tblFeiertageBasis**, die Schlüsselspalte **FeiertagBasisID** soll ausgeblendet werden. Auf diese Weise zeigt das Nachschlagefeld nur die Bezeichnungen der Feiertage an.

Das Feld **BundeslandID** ist ähnlich aufgebaut: Es verwendet die Felder **BundeslandID** und **Bundesland** der Tabelle **tblBundeslaender** zum Füllen des Nachschlagefeldes. Auch hier wird die Schlüsselspalte nicht angezeigt.

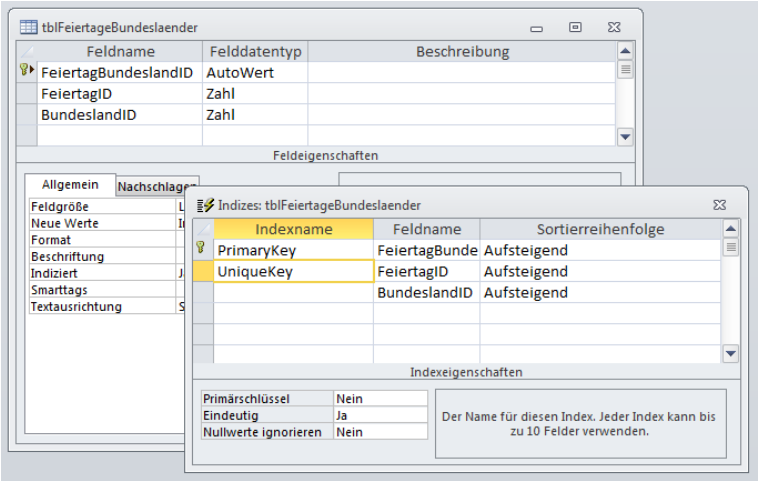


Bild 5: Verknüpfungstabelle für Feiertage und Bundesländer

Schließlich wollen wir noch sicherstellen, dass die Tabelle keine Kombination auf Feiertag und Bundesland mehrfach aufnimmt. Dazu definieren Sie einen eindeutigen Index, der aus den beiden Feldern **FeiertagID** und **BundeslandID** zusammengesetzt wird. Diesen Index legen Sie im Entwurf der Tabelle mithilfe des **Indizes**-Dialogs an. Dort befindet sich zumindest bereits der Index für das Primärschlüsselfeld der Tabelle. Fügen Sie einen neuen Index namens UniqueKey hinzu und wählen Sie untereinander die beiden Felder **FeiertagID** und **BundeslandID** aus. Markieren Sie dann wieder die erste Zeile, in der Sie auch den Indexnamen eingetragen haben, und stellen Sie die Eigenschaft **Eindeutig** im unteren Bereich auf **Ja** ein (siehe Bild 5).

Nachdem die beiden Tabellen **tblFeiertageBasis** und **tblBundeslaender** mit Daten gefüllt wurden, können Sie die gewünschten Verknüpfungen zwischen Feiertagen und Bundesländern anlegen. Bild 6 verdeutlicht, dass die Beziehung eigentlich über in der Tabelle **tblFeiertageBundeslaender** enthaltene Zahlenwerte hergestellt wird – hier haben wir die Feldeigenschaften so eingestellt, dass die Felder nicht die Werte der verknüpften Tabelle, sondern die tatsächlich in den Feldern **FeiertagID** und **BundeslandID** gespeicherten Werte anzeigen.

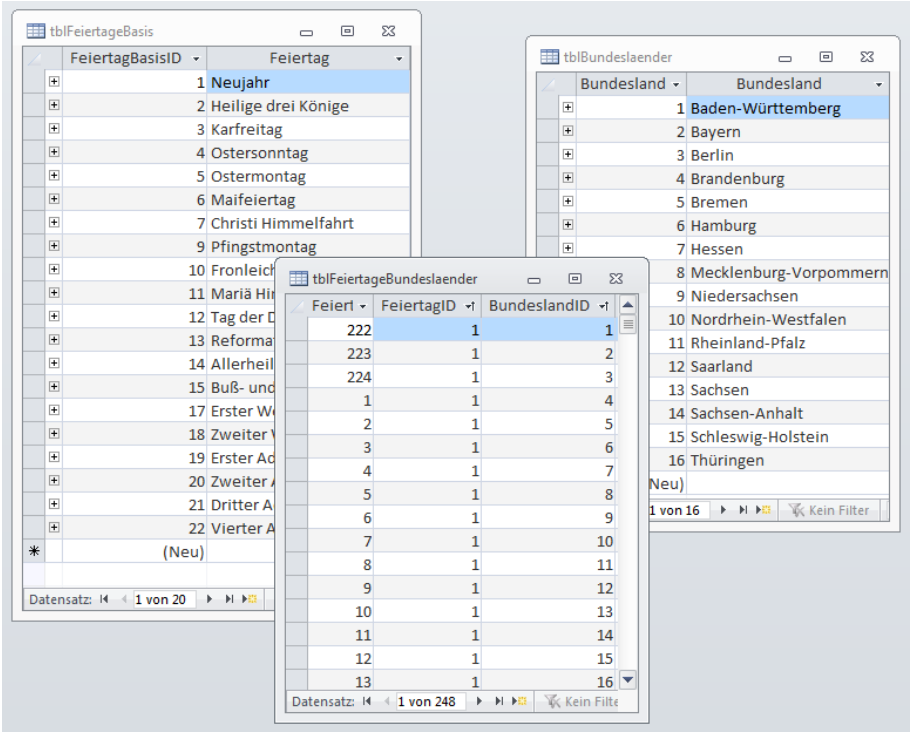


Bild 6: Verknüpfung von Feiertagen und Bundesländern

Feiertage und Bundesländer verwalten

Nun benötigen wir ein Formular, mit dem wir die Feiertage verwalten können. Die Bundesländer geben wir einfach direkt in die Tabelle ein – aktuell ist ja nicht absehbar, dass neue Bundesländer hinzukommen oder Bundesländer wegfallen. Daher soll kein eigenes Formular zur Verwaltung der Bundesländer vorgesehen werden.

Das Formular heißt **frmFeiertage** und verwendet die

Tabelle **tblFeiertageBasis** als Datenherkunft. Ziehen Sie alle Felder dieser Tabelle in den Formularentwurf und ordnen Sie diese etwa wie in Bild 7 an. Je nach dem Wert im Kombinationsfeld **FeiertagArtID** sollen entweder die beiden Felder **Tag** und **Monat** oder das Feld **TageBisStichtag** aktiviert sein.

Die Steuerelementnamen der vier Textfelder und des Kombinationsfeldes haben wir um die Präfixe **txt** beziehungsweise **cbo** erweitert.

Eingabefelder aktivieren und deaktivieren

Je nachdem, welche Feiertagsart ausgewählt ist, soll das Formular die Textfelder für die Eingabe des fixen Datums oder der Anzahl der Tage vor oder nach dem Stichtag aktivieren oder deaktivieren.

Dies soll also entweder beim Anzeigen eines Datensatzes oder nach der Änderung des Inhalts des Kombinationsfeldes **cboFeiertagArtID** geschehen. In beiden Fällen sollen die gleichen Schritte ausgeführt werden, also legen wir für diese Anweisungen eine eigene Prozedur an, die von zwei Ereignisprozeduren aus ausgelöst wird.

Damit diese Prozedur zu den passenden Gelegenheiten aufgerufen wird, legen Sie zwei Ereignisprozeduren an. Die erste soll beim Anzeigen eines jeden Datensatzes ausgelöst werden. Dazu stellen Sie für die Eigenschaft **Beim Anzeigen** des Formulars in der Entwurfsansicht den Wert **[Ereignisprozedur]** ein und klicken auf die Schaltfläche mit den drei Punkten. Die nun im VBA-Editor erscheinende Prozedur ergänzen Sie wie folgt:

```
Private Sub Form_Current()
    Call TextfelderAktivieren
End Sub
```

Die einzige Anweisung dieser Prozedur ruft die Routine **TextfelderAktivieren** auf. Genauso sieht es bei der Prozedur aus, die Sie für die Eigenschaft **Nach Aktualisierung** des Kombinationsfeldes **cboFeiertagArtID** hinterlegen:

```
Private Sub cboFeiertagArtID_AfterUpdate()
    Call TextfelderAktivieren
End Sub
```

Die Prozedur **TextfelderAktivieren** sieht wie folgt aus und kann im gleichen Klassenmodul wie die beiden zuvor erzeugten Prozeduren eingefügt werden:

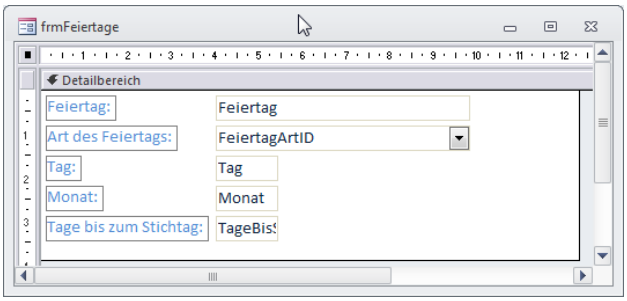


Bild 7: Entwurf des Formulars **frmFeiertage**

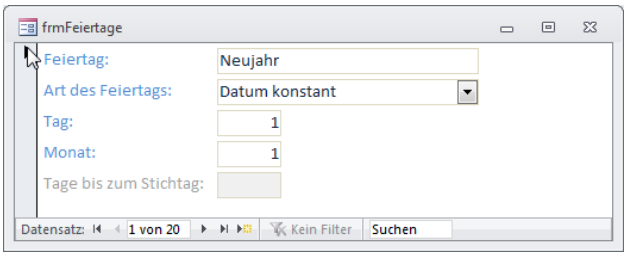


Bild 8: Das Formular **frmFeiertage** mit aktivierten und deaktivierten Feldern.

```
Private Sub TextfelderAktivieren()
    Select Case Me!cboFeiertagArtID
        Case 1
            Me!txtMonat.Enabled = True
            Me!txtTag.Enabled = True
            Me!txtTageBisStichtag.Enabled = False
        Case 2, 3
            Me!txtMonat.Enabled = False
            Me!txtTag.Enabled = False
            Me!txtTageBisStichtag.Enabled = True
    End Select
End Sub
```

Die Prozedur prüft in einer **Select Case**-Bedingung, welchen Wert das Kombinationsfeld **cboFeiertagArtID** aufweist. Für den Wert **1** (für fixe Feiertagdaten) sollen die beiden Textfelder **txtTag** und **txtMonat** aktiviert und das Textfeld **txtTageBisStichtag** deaktiviert werden, für die Werte **2** und **3** (Abhängigkeit vom Ostersonntag beziehungsweise vom vierten Advent) ist es umgekehrt (siehe Bild 8).

Bundesländer zu Feiertagen zuweisen

Nun kommt der kompliziertere Teil: Wir benötigen eine Möglichkeit, den Feiertagen die Bundesländer zuzuweisen, in denen diese gefeiert werden.

Dazu legen Sie zwei Listenfelder namens **lstIstFeiertagIn** und **lstIstKeinFeiertagIn** an. Diese fügen Sie wie in Bild 9 in das Formular ein. Die Schaltflächen zwischen den Listenfeldern heißen **cmdAlleHinzu-**

Debugging im VBA-Editor, Teil 1

Wer mit VBA arbeitet, hat im Gegensatz zu Makros einige Möglichkeiten, den Programmablauf zu beobachten und auf diese Weise Fehler aufzudecken. Auch wenn der VBA-Editor bereits einige Jährchen auf dem Buckel hat und seit Jahren weder VBA noch der VBA-Editor erweitert wurde, bietet er doch einige Möglichkeiten zum Debugging. Dieser Artikel zeigt, wie Debugging funktioniert und welche Werkzeuge der VBA-Editor dazu anbietet.

Was ist Debugging?

Unter Debugging versteht man das Entfernen von Bugs. Bugs? Damit sind Programmfehler gemeint. Während diese Bezeichnung für Fehler wohl unter Ingenieuren schon länger gängig war, wurde er von Grace Hopper salonfähig gemacht: Dieser fand nämlich 1947 bei den Arbeiten am Mark II tatsächlich eine Motte als Auslöser eines Fehlers vor (siehe Wikipedia).

Unter VBA versteht man unter Debugging vor allem die Untersuchung des Programmablaufs, wobei Prozeduren mit geeigneten Menübefehlen oder Tastenkombinationen in mehr oder weniger großen Schritten durchlaufen werden. Bei jedem Schritt lassen sich die aktuellen Werte von Variablen ermitteln. Falsch oder nicht gefüllte beziehungsweise gesetzte Variablen sind der Hauptgrund für logische Fehler im Programmablauf.

Statt die Prozeduren schrittweise zu durchlaufen, können Sie auch gleich festlegen, an welcher Stelle diese angehalten werden soll. Tritt hingegen ein Laufzeitfehler auf, bietet der VBA-Editor automatisch die Möglichkeit, die fehlerhafte Stelle zu untersuchen. Doch genug der warmen Worte – schauen wir uns doch gleich die passenden Werkzeuge an.

Debugging im Fehlerfall

Wer sich noch nicht aktiv um das Thema Debugging gekümmert hat, wird wohl am ehesten durch einen Fehler damit in Verbindung gekommen sein. Nehmen wir folgende Funktion zum Dividieren als Beispiel:

```
Public Function Dividieren(intDividend As Integer, _
        intDivisor As Integer) As Single
    Dim sngErgebnis As Single
    sngErgebnis = intDividend / intDivisor
    Dividieren = sngErgebnis
End Function
```

Wenn Sie diese Funktion mit den Werten **3** und **0** über das Direktfenster aufrufen, erscheint die Feh-

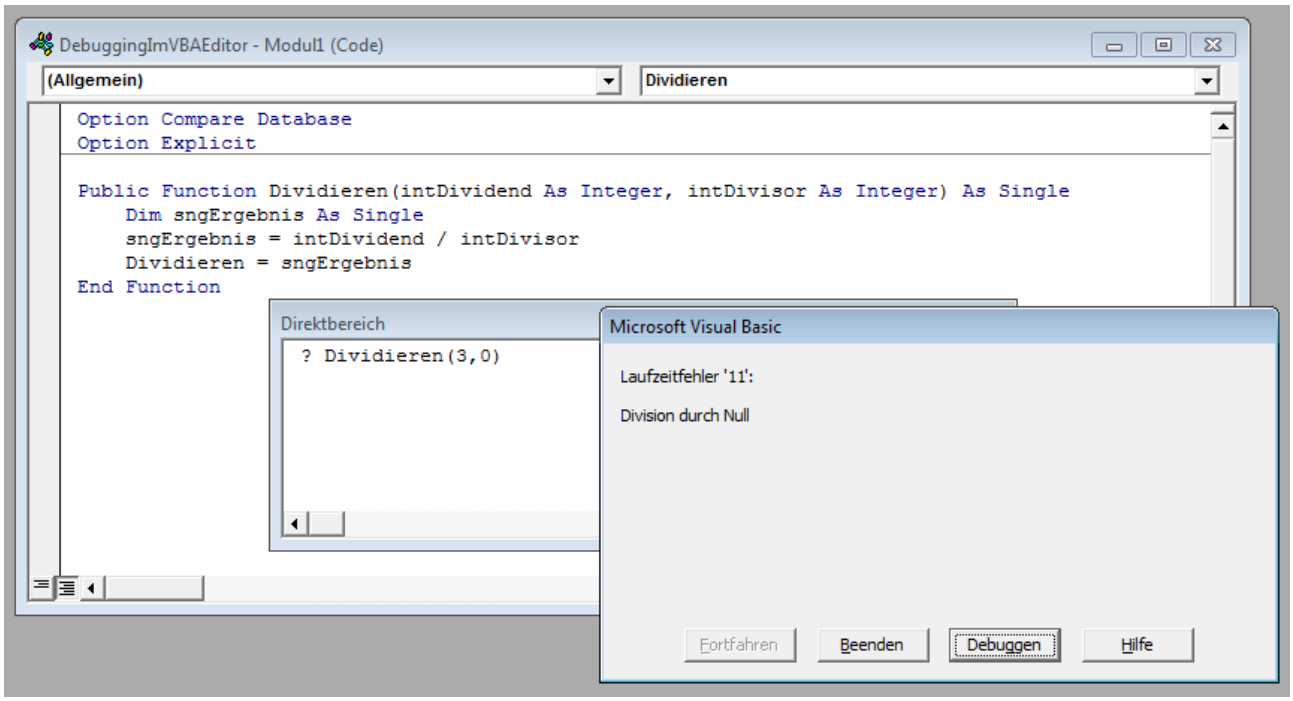


Bild 1: Laufzeitfehler mit der Möglichkeit, zu debuggen

lermeldung aus Bild 1. Dort finden Sie unter anderem eine Schaltfläche mit der Beschriftung **Debuggen**, mit der Sie in den Debugging-Modus einsteigen können.

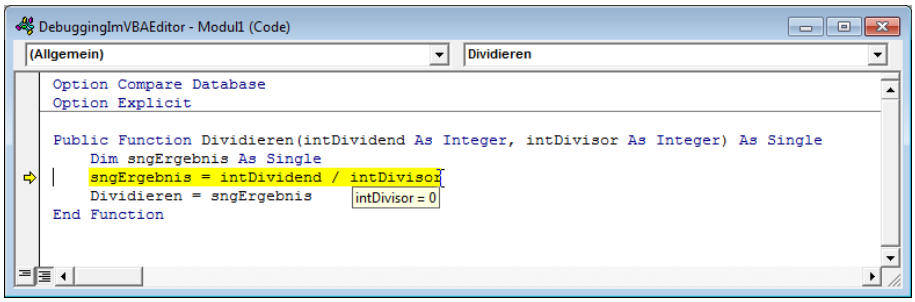


Bild 2: Debugging eines Laufzeitfehlers

Ein Klick auf diese Schaltfläche schließt die Fehlermeldung und hebt die den Fehler auslösende Zeile im VBA-Editor farblich hervor (siehe Bild 2). Sie können nun beispielsweise mit der Maus über die in der Zeile enthaltenen Variablen fahren, um ihre Werte anzuzeigen.

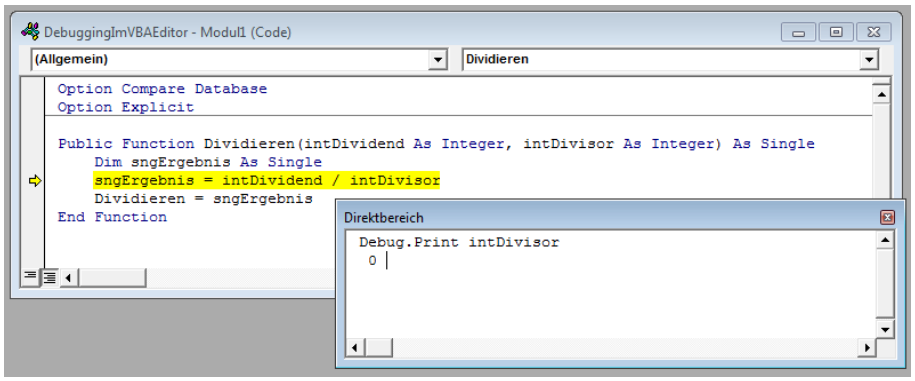


Bild 3: Variablenwerte über das Direktfenster ermitteln

Dies gelingt übrigens auch auf andere Weise (manchmal – vor allem bei komplizierteren Ausdrücken – zeigt sich die direkte Anzeige im Codefenster etwas störrisch): Dabei nehmen Sie das Direktfenster zuhilfe und lassen sich dort mit der **Debug.Print**-Anweisung den Inhalt der zu untersuchenden Variablen ausgeben. Dabei geben Sie den Namen der Variablen als Parameter an (siehe Bild 3). Hier erkennen Sie direkt, dass der Divisor einen unzulässigen Werte angenommen hat (vermutlich unerwarteterweise) und können eine entsprechende Prüfung einbauen.

Variableninhalt ändern

Wenn Sie eine umfangreichere Abfolge von Routinen testen, möchten Sie an dieser Stelle möglicherweise nicht von vorn beginnen.

Dann gibt es mehrere Möglichkeiten (die alle natürlich nicht mit einem Neustart mit korrigiertem Code gleichzusetzen sind) – zum Beispiel diese:

- Sie ändern den unerwünschten Variablenwert und fahren mit der Prozedur fort.
- Sie ändern das Ergebnis und fahren dann mit der Prozedur fort.

Im ersten Fall weisen Sie der Variablen **intDivisor** über den Direktbereich einen neuen Wert zu:

```
intDivisor = 1
```

Danach führen Sie die Prozedur mit einem Klick auf die Taste **F5** oder durch das Aufrufen des Menübefehls **Ausführen|Fortsetzen** fort. Die Zeile löst keinen Fehler mehr aus und die Funktion liefert das entsprechende Ergebnis. Wenn Sie lieber gleich den vollständigen Rückgabewert der Funktion ändern möchten, verschieben Sie den gelben Pfeil, der die aktuell auszuführende Zeile markiert, einfach auf die Zeile **End Function** und weisen Sie dem Rückgabewert **Dividieren** der Funktion den gewünschten Wert zu – beispielsweise so (siehe Bild 4):

```
Dividieren = 3
```

Auf diese Weise können Sie auch alle anderen Zeilen anspringen oder Variablenwerte ändern.

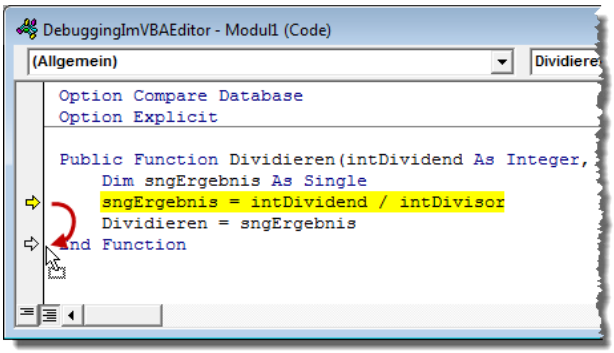


Bild 4: Aktuelle Zeile ändern

ListView-Steuerelement: Drag and Drop, Teil II

ListView-Steuerelemente unterstützen im Gegensatz zum Listenfeld sogar den Einsatz von Drag and Drop. Damit können Sie nicht nur Einträge von einem ListView-Steuerelement zum nächsten ziehen, sondern auch die Reihenfolge der Einträge eines ListView-Steuerelements einstellen. Im zweiten Teil dieser Mini-Artikelreihe zeigen wir, wie Sie per Drag and Drop die Reihenfolge von ListView-Einträgen anpassen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1302_ListView.mdb**.

Ändern der Reihenfolge per Drag and Drop

In der Datenblattansicht, der Endlosansicht oder in Listenfeldern lässt sich mit viel Trickserei ein Drag-and-Drop-Vorgang nachbauen. Das ist allerdings mit viel Aufwand verbunden. Viel einfacher wird es mit dem **ListView**-Steuerelement. Dieses liefert von Haus aus grundlegende Drag-and-Drop-Funktionen mit, die Sie nur noch ausprogrammieren müssen.

tblKunden

Feldname	Felddatentyp	Beschreibung
KundeID	AutoWert	
KundenCode	Text	Eindeutiger Code aus fünf Zeichen basierend
Firma	Text	
Kontaktperson	Text	
Position	Text	
Straße	Text	Straße oder Postfach.
Ort	Text	
Region	Text	Bundesland oder Provinz.
PLZ	Text	
Land	Text	
Telefon	Text	Telefonnummer mit (internationaler) Vorwal
Telefax	Text	Telefonnummer mit (internationaler) Vorwal
ReihenfolgeID	Zahl	

Bild 1: Vorbereitung der Tabelle **tblKunden**

Und damit lassen sich natürlich nicht nur Einträge von einem **ListView**-Steuerelement zum nächsten bewegen, wie im ersten Teil dieser Artikelserie beschrieben, sondern Sie können damit auch die Reihenfolge der Elemente eines einzelnen **ListView**-Steuerelements einstellen.

Beispieldaten

Das **ListView**-Steuerelement soll das Sortieren der Datensätze der Tabelle **tblKunden** nach dem Wert eines neuen Feldes namens **ReihenfolgeID** ermöglichen. Dieses Feld fügen wir der entsprechenden Tabelle der Beispieldatenbank hinzu; es erhält den Datentyp **Zahl** (siehe Bild 1).

Dieses Feld müssen wir noch initial füllen, was wir gleich beim Öffnen des Formulars erledigen. Damit ist auch sichergestellt, dass zwischenzeitlich hinzugefügte Felder oder entfernte Felder berücksichtigt werden.

Beispielformular

Das Beispielformular heißt **frmListViewDragDropReihenfolge**. Fügen Sie diesem zunächst ein einfaches **ListView**-Steuerelement hinzu (**Steuerelemente|ActiveX-Steuerelemente**, dort **Microsoft ListView Control 6.0 (SP6)** auswählen). Benennen Sie dieses Steuerelement mit **lvwKunden**.

Nun sorgen Sie dafür, dass das **ListView**-Steuerelement beim Laden des Formulars mit den gewünschten Daten gefüllt wird – allerdings nicht, ohne den Wert des Feldes **ReihenfolgeID** zuvor anzupassen.

Doch zunächst zu der Prozedur, die beim Laden des Formulars ausgelöst wird. Diese legen Sie durch Auswählen des Eintrags **[Ereignisprozedur]** für die Eigenschaft **Beim Laden** des Formulars und anschließendes Anklicken der Schaltfläche mit den drei Punkten (**...**) an.

Die nun erscheinende Prozedur, die gleichzeitig mit dem Klassenmodul **Form_frmListViewDragDropReihenfolge** angelegt wird, füllen Sie wie in Listing 1 abgebildet. Damit die dort verwendete Variable **objLvwKunden**, mit der die Prozedur auf das **ListView**-Steuerelement verweist, bekannt ist, deklarieren Sie diese in einer eigenen Zeile im Kopf des Moduls – also gleich unter den beiden **Option ...**-Zeilen:

```
Dim WithEvents objLvwKunden As MSComctlLib.ListView
```

Das Schlüsselwort **WithEvents** verwenden wir, damit wir im Klassenmodul des Formulars ganz einfach Ereignisprozeduren anlegen können, die durch das

```
Private Sub Form_Load()
    Call ReihenfolgeAktualisieren
    Set objLvwKunden = Me!lvwKunden.Object
    With objLvwKunden
        .View = lvwReport
        .Appearance = ccFlat
        .BorderStyle = ccNone
        .FlatScrollBar = False
        .ColumnHeaders.Clear
        .ColumnHeaders.Add , , "Kunde", Me!lvwKunden.Width
        .OLEDragMode = ccOLEDragAutomatic
        .OLEDropMode = ccOLEDropManual
    End With
    KundenFuellen
End Sub
```

Listing 1: Einrichten und Füllen des ListView-Steuerelements

ListView-Steuerelement ausgelöst werden. Dies ist für die Programmierung der Drag-and-Drop-Funktionalität wichtig.

Die Prozedur ruft zunächst die Prozedur **ReihenfolgeAktualisieren** auf, die sich um das Füllen des Feldes **ReihenfolgeID** kümmert – mehr dazu weiter unten.

Danach füllt sie die Variable **objLvwKunden** mit einem Verweis auf das entsprechende Steuerelement. Anschließend stellt es einige Eigenschaften dieses Steuerelements ein. Dies könnten Sie auch über den Eigenschaften-Dialog des Steuerelements durchführen, aber wir erledigen das per Code.

Neben einigen optischen Einstellungen sorgt die Methode **Add** der **ColumnHeaders**-Auflistung für das Anlegen einer einzigen Spalte, welche später die Werte des Feldes **Firma** der Kundendatensätze aufnehmen soll. Da dies die einzige Spalte ist, stellt die Prozedur die Spaltenbreite auf die Breite des Steuerelements ein. Schließlich folgen zwei Einstellungen, die für die Programmierbarkeit von Drag and Drop wichtig sind: **OLEDragMode** und **OLEDropMode**.

Den Abschluss macht ein Aufruf der Prozedur **KundenFuellen**, die schließlich die Kundennamen in das **ListView**-Steuerelement füllt.

Reihenfolge aktualisieren

Die Prozedur **ReihenfolgeAktualisieren** aus Listing 2 verwendet die Abfrage **qryKundenNachReihenfolgeID** als Basis eines Recordsets. Die Prozedur durchläuft alle Datensätze dieser Abfrage in einer **Do While**-Schleife und bearbeitet dabei den jeweils aktuelle Datensatz, indem sie den Wert des Feldes **ReihenfolgeID** auf einen Wert einstellt, welcher der Eigenschaft **AbsolutePosition** des aktuellen Recordsets plus eins entspricht. Der Wert **1** wird deshalb addiert, weil **AbsolutePosition** für den ersten Datensatz den Wert **0** liefert, die Werte des Feldes **ReihenfolgeID** jedoch mit **1** beginnen sollen.

Zum Einstellen dieses Wertes versetzt die Prozedur den Datensatz mit der **Edit**-Methode in den Bearbeitungsmodus, ändert den Wert des Feldes **ReihenfolgeID** und speichert die Änderung mit der **Update**-Methode. Die **MoveNext**-Methode verschiebt den Datensatzzeiger auf den nächsten Datensatz. Dieser Vorgang wird solange wiederholt, bis die Eigenschaft **EOF** des **Recordset**-Objekts den Wert **True** annimmt – der Datensatzzeiger befindet sich somit hinter dem letzten Datensatz.

Interessant ist die Abfrage **qryKundenNachReihenfolgeID**, welche festlegt, in welcher Reihenfolge die Kundendatensätze tatsächlich nummeriert werden.

```
Public Sub ReihenfolgeAktualisieren()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM qryKundenNachReihenfolgeID", dbOpenDynaset)
    Do While Not rst.EOF
        rst.Edit
        rst!ReihenfolgeID = rst.AbsolutePosition + 1
        rst.Update
        rst.MoveNext
    Loop
    Set db = Nothing
End Sub
```

Listing 2: Einstellen der ReihenfolgeID der enthaltenen Datensätze

Die Abfrage finden Sie in Bild 2. Sie basiert auf der Tabelle **tblKunden** und enthält drei Felder, von denen nur das Feld **ReihenfolgeID** angezeigt wird. Das zweite und das dritte Feld dienen der korrekten Sortierung vor dem Aktualisieren der Reihenfolge.

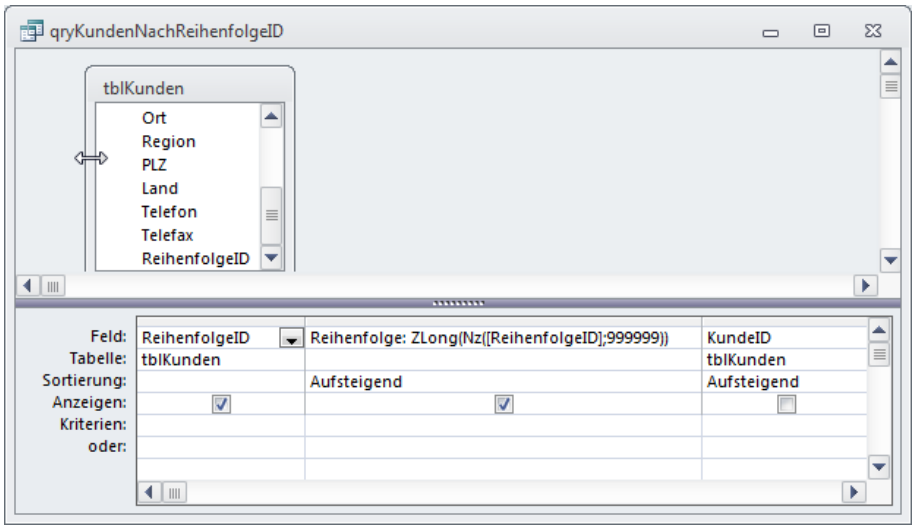


Bild 2: Abfrage zum Einstellen der **ReihenfolgeID** der Kundendatensätze

Wichtig ist, dass diese Abfrage die Reihenfolge der vorhandenen Kundendatensätze nicht durcheinander bringt und dass neue Datensätze hinten eingereiht werden.

Daher sortiert die Abfrage die Datensätze zunächst nach dem Feld **ReihenfolgeID** und erst danach nach dem Feld **KundeID**. Würde man dies so belassen, würde die Abfrage jedoch zuerst die neu hinzugefügten Datensätze liefern. Diese weisen nämlich den Wert **Null** im Feld **ReihenfolgeID** auf und sind in der Reihenfolge vor den mit Zahlenwerten gefüllten Feldern zu finden.

Also müssen wir dafür sorgen, dass der Wert **Null** durch einen Wert ersetzt wird, der größer als die bisher vergebenen Werte im Feld **ReihenfolgeID** ist. Da die Sortierung per Drag and Drop ohnehin nur für eine überschaubare Anzahl Datensätze sinnvoll ist, gehen wir davon aus, dass der Wert **999.999** ausreichend ist. Der folgende Ausdruck prüft, ob ReihenfolgeID leer ist (also den Wert **Null** enthält) und fügt in diesem Fall den Wert **999.999** ein:

```
Reihenfolge: ZLong(Nz([ReihenfolgeID];999999))
```

Das Ergebnis der **Nz**-Funktion wird in der Abfrage aber dummerweise als String interpretiert. Sie können sich dies in der Abfrage **qryKundenNachReihenfolgeIDFalsch** ansehen, die das Ergebnis aus Bild 3 liefert.

Zumindest erfolgt dort die Sortierung so, als ob es sich bei dem Feld um ein Textfeld handelt – **10** landet so beispielsweise vor **2**.

Dies lässt sich allerdings beheben, indem wir das Ergebnis der **Nz**-Funktion noch mit der **CLng**-Funktion in einen **Long**-Wert konvertieren.

Eintragen der ListView-Werte

Zuletzt ruft die Prozedur **Form_Load** die Routine **KundenFuellen** auf (siehe Listing 3). Diese erzeugt ein Recordset auf Basis der Tabelle **tblKunden**, wobei die Reihenfolge durch das soeben frisch gefüllte Feld **ReihenfolgeID** festgelegt wird.

Die **Clear**-Methode der **ListItems**-Auflistung säubert das **ListView**-Steuerelement von eventuell vorhandenen Einträgen. Danach durchläuft eine **Do While**-Schleife alle Datensätze des Recordsets und legt mit der **Add**-Methode der gleichen Auflistung jeweils ein neues Element je Datensatz an. Dabei vergibt die Methode als **Key** den Buchstaben **a** plus dem Primärschlüsselwert und als anzuzeigenden **Text** den Namen der Firma des aktuellen Datensatzes.

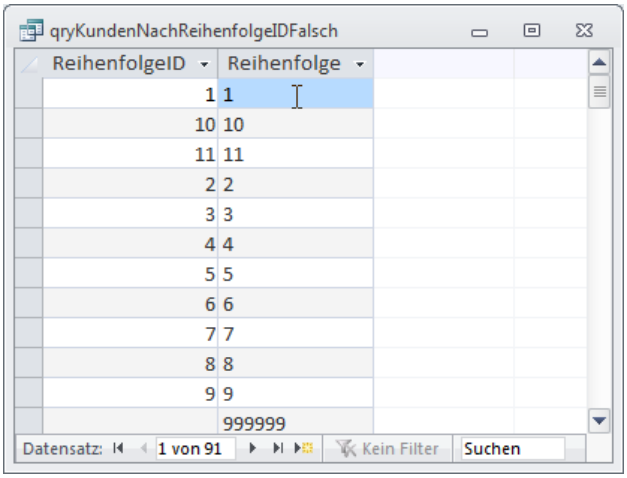


Bild 3: Fehlerhafte Interpretation der Ergebnisse von Nz

```
Private Sub KundenFuellen()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM tblKunden ORDER BY ReihenfolgeID", dbOpenDynaset)
    objLvwKunden.ListItems.Clear
    Do While Not rst.EOF
        objLvwKunden.ListItems.Add , "a" & rst!KundeID, rst!Firma
        rst.MoveNext
    Loop
    rst.Close
    Set rst = Nothing
    Set db = Nothing
End Sub
```

Listing 3: Einlesen der Datensätze und Einfügen in das **ListView**-Steuerelement

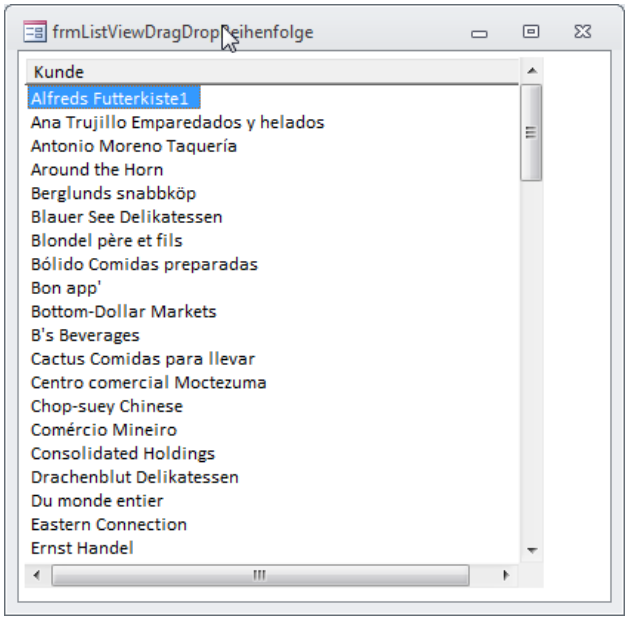


Bild 4: Das gefüllte **ListView**-Steuerelement

Fertig – das Formular sieht nach dem Öffnen wie in Bild 4 aus.

Drag and Drop

Nun benötigen wir einige Ereignisprozeduren, die durch das Ziehen eines Eintrags mit der Maus ausgelöst werden. Zuvor überlegen wir uns jedoch noch, wie sich das Ziehen und Fallenlassen eines Eintrags überhaupt auswirken soll. Immerhin gibt es da eine Kleinigkeit zu beachten. So können Sie einen Eintrag immer nur auf einen anderen Eintrag ziehen, aber nicht zwischen zwei Einträge. Dementsprechend müssen wir festlegen, wie das **ListView**-Steuerelement verfahren soll, wenn Sie einen Eintrag auf einen anderen ziehen.

Soll der gezogene Eintrag dann vor dem Zieleintrag eingefügt werden oder hinter diesem Eintrag? Und was geschieht, wenn der Benutzer einen Eintrag aus der Mitte auf den ersten oder den letzten Eintrag zieht? Einigen wir uns auf die folgende Vorgehensweise: Wenn der Eintrag auf einen über ihm liegenden Eintrag gezogen wird, nimmt er die Position über diesem Eintrag ein und wenn der Eintrag auf einen unter ihm liegenden Eintrag gezogen wird, landet er darunter. So lassen sich auch einfach Einträge an die erste und die letzte Position verschieben.

Aktionen beim Drag and Drop

Wenn der Benutzer ein Element auf ein anderes zieht, sollen folgende Aktionen ausgeführt werden:

- Das gezogene Element soll die Position des Zielelements einnehmen.
- Wenn das Zielelement über dem gezogenen Element liegt, rutschen das Zielelement und alle darunter liegenden Elemente bis zum vorherigen Platz des gezogenen Elements um eine Position nach unten. Wenn das Zielelement unterhalb des gezogenen Elements liegt, nimmt das gezogene Element ebenfalls den Platz des Zielelements ein. Das Zielelement und alle darüberliegenden Elemente bis zur vorherigen Position des vorherigen Elements rücken um eine Position nach oben.
- In der Tabelle **tblKunden** müssen die Werte des Feldes **ReihenfolgeID** für die betroffenen Elemente angepasst werden.

Start des Drag-and-Drop-Vorgangs

Wenn der Benutzer auf ein Element klickt und die Maus bei gedrückter linker Maustaste bewegt, löst dies das Ereignis **OLEStartDrag** des Steuerelements

Feiertage verwalten, Teil 2

Im ersten Teil dieser Artikelreihe haben Sie die Tabelle zum Speichern der Basisdaten der Feiertage sowie die Formulare zum Verwalten der Feiertage sowie für die Zuordnung zu den Bundesländern kennengelernt. Im vorliegenden zweiten Teil schauen wir uns an, wie die flexiblen Feiertage ermittelt werden und wie Sie aus den ermittelten Informationen eine Liste der Feiertage für einen bestimmten Zeitraum erstellen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1302_FeiertageErmitteln.mdb**.

Fixe und flexible Feiertage

Die fixen Feiertage finden immer am gleichen Datum statt: Neujahr am 1. Januar, Heilige drei Könige am 6. Januar, der Maifeiertag am 1. Mai et cetera. Diese vermerken wir in der Tabelle **tblFeiertageBasis**, indem wir den Namen des Feiertags eintragen, den Tag und den Monat festlegen, das Feld **TageBisStichtag** leer lassen und das Feld **FeiertagArt** mit **Datum konstant** füllen (siehe Bild 1).

Für die flexiblen Feiertage, von denen es zwei Varianten gibt, tragen wir ebenfalls den Feiertagnamen ein. Für den **Tag** und den **Monat** legen Sie allerdings jeweils den Wert **0** fest. Dafür tragen Sie im Feld **TageBisStichtag** einen Zahlenwert ein, der angibt, wieviele Tage der Feiertag vor oder hinter dem jeweiligen fixen Tag liegt. Um festzulegen, um welche Variante der flexiblen Feiertage es sich handelt, tragen Sie unter **Feiertagsart** entweder **Abstand zu Stichtag Ostersonntag konstant** oder **Abstand zu Stichtag Vierter Advent konstant** ein.

Stichtage für flexible Feiertage

Nun benötigen wir noch die Stichtage für die beiden flexiblen Feiertage, und zwar für Ostersonntag und für den vierten Advent. Im Falle des vierten Advents ist dies einfach: Es handelt sich dabei um den letzten Sonntag vor dem ersten Weihnachtsfeiertag. Um diesen Stichtag zu berechnen, verwenden Sie die Funktion aus Listing 1. Die Prozedur erwartet die Zahl des zu untersuchenden Jahres als Parameter mit dem Datentyp **Integer** (für Werte bis 32.767) und liefert einen Wert des Typs **Date** zurück (siehe Bild 2).

Die Funktion füllt eine temporäre Variable namens **datTemp** mit einem Datum, dass dem 24. Dezember des angegebenen Jahres entspricht. Dazu setzt

tblFeiertageBasis

FeiertagBasisID	Feiertag	Tag	Monat	TageBisStichtag	FeiertagArt
1	Neujahr	1	1		Datum konstant
2	Heilige drei Könige	6	1		Datum konstant
3	Karfreitag	0	0	-2	Abstand zu Stichtag Ostersonntag konstant
4	Ostersonntag	0	0	0	Abstand zu Stichtag Ostersonntag konstant
5	Ostermontag	0	0	1	Abstand zu Stichtag Ostersonntag konstant
6	Maifeiertag	1	5		Datum konstant
7	Christi Himmelfahrt	0	0	39	Abstand zu Stichtag Ostersonntag konstant
9	Pfingstmontag	0	0	50	Abstand zu Stichtag Ostersonntag konstant
10	Fronleichnam	0	0	60	Abstand zu Stichtag Ostersonntag konstant
11	Mariä Himmelfahrt	15	8		Datum konstant
12	Tag der Deutschen Einheit	3	10		Datum konstant
13	Reformationstag	31	10		Datum konstant
14	Allerheiligen	1	11		Datum konstant
15	Buß- und Bettag			-32	Abstand zu Stichtag Vierter Advent konstant
17	Erster Weihnachtstag	25	12		Datum konstant
18	Zweiter Weihnachtstag	26	12		Datum konstant
19	Erster Advent			-21	Abstand zu Stichtag Vierter Advent konstant
20	Zweiter Advent			-14	Abstand zu Stichtag Vierter Advent konstant
21	Dritter Advent			-7	Abstand zu Stichtag Vierter Advent konstant
22	Vierter Advent			0	Abstand zu Stichtag Vierter Advent konstant
(Neu)		0	0	0	

Datensatz: 1 von 20 | Kein Filter | Suchen

Bild 1: Basisdaten zu den Feiertagen

```
Public Function VierterAdvent(intJahr As Integer) As Date
    Dim datTemp As Date
    datTemp = CDate("24.12." & intJahr)
    Do While Not Weekday(datTemp) = vbSunday
        datTemp = datTemp - 1
    Loop
    VierterAdvent = datTemp
End Function
```

Listing 1: Berechnen des Datums des vierten Advents

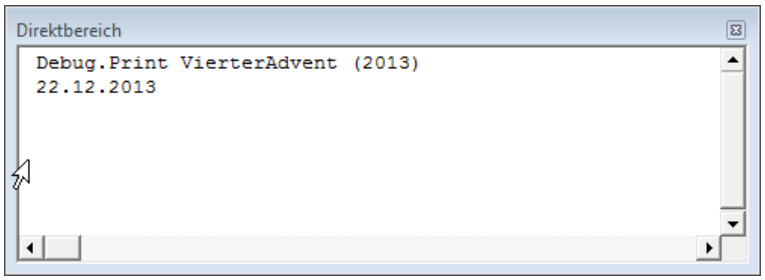


Bild 2: Test der Funktion zum Ermitteln des vierten Advents

es die Zeichenkette **24.12.** und die mit der Variablen **lngJahr** übergebene Jahreszahl zusammen und konvertiert den Ausdruck mit der Funktion **CDate** zu einem Datumswert zusammen.

Dies könnten Sie theoretisch auch mit der **DateSerial**-Funktion erledigen. Die entsprechende Zeile sieht dann wie folgt aus:

```
datTemp = DateSerial(intJahr, 12, 24)
```

Nun startet die Prozedur eine **Do While**-Schleife. Diese wird solange durchlaufen, bis der Wochentag des in **datTemp** gespeicherten Datums dem Sonntag entspricht. Innerhalb der Schleife zieht die Funktion mit jedem Durchlauf einen Tag vom Datum ab.

Im Jahr 2012 etwa fiel Heiligabend auf einen Montag, der vierte Advent dementsprechend auf den Sonntag davor. Die Funktion prüft beim ersten Aufruf der **Do While**-Zeile, ob der Wochentag des Datums den Wert **vbSunday** zurückliefert. Das ist im ersten Durchlauf des Beispiels für 2012 nicht der Fall, stattdessen liefert die **Weekday(datTemp)** für den **24.12.2012** den Wert **2** (**vbMonday**). Also ist die Abbruchbedingung nicht erfüllt und der in der Schleife enthaltene Befehl wird durchlaufen. Dieser subtrahiert vom in **datTemp** gespeicherten Wert die Zahl **1** – dies entspricht genau einem Tag (Tage sind ganze Zahlen, Uhrzeiten der Bereich hinter dem Komma). Im nächsten Durchlauf hat **datTemp** den Wert **23.12.2012** und **Weekday** liefert den gewünschten Wert **1** (**vbSunday**). Die Abbruchbedingung ist erfüllt und die Schleife wird kein weiteres Mal durchlaufen.

Schließlich stellt die Funktion **datTemp** als Wert der Rückgabevariablen **VierterAdvent** ein (gleichzeitig Funktionsname).

Berechnung des Datums des Ostersonntags

Die Berechnung des Ostersonntags ist etwas komplizierter als die des vierten Advents. Um ehrlich zu sein: Wir sollten die Formel zur Berechnung, die durch die Funktion aus Listing 2 abgebildet wird, einfach hinnehmen.

Allein eine in der Routine enthaltene VBA-Funktion wollen wir kurz inspizieren – dabei handelt es sich um die **Mod**-Funktion. Diese ist eigentlich eher ein Operator als eine Funktion, denn die beiden Parameter werden durch das Schlüsselwort **Mod** verbunden. **Mod** liefert den Rest der Division des ersten und des zweiten genannten Zahlenwertes. **3 Mod 2** würde also entsprechend den Wert **1** zurückgeben. Desweiteren liefert die Funktion genau das Datum des Ostersonntags für das im Parameter angegebene Jahr zurück.

```
Public Function Ostersonntag(intJahr As Integer) As Date
    Dim a As Integer
    Dim b As Integer
    Dim c As Integer
    Dim d As Integer
    Dim e As Integer
    Dim intTag As Integer
    Dim intMonat As Integer
    a = intJahr Mod 19
    b = intJahr Mod 4
    c = intJahr Mod 7
    d = (19 * a + 24) Mod 30
    e = (2 * b + 4 * c + 6 * d + 5) Mod 7
    intTag = 22 + d + e
    intMonat = 3
    If intTag > 31 Then
        intTag = d + e - 9
        intMonat = 4
    End If
    Ostersonntag = intTag & "." & intMonat & "." & intJahr
End Function
```

Listing 2: Berechnung des Datums des Ostersonntags

fen, um den VBA-Editor zu öffnen und direkt die gewünschte Ereignisprozedur zu erhalten.

Monatsnamen und Wochentage

Wenn Sie einmal per Funktion auf den Namen eines Monats oder eines Wochentages zugreifen möchten, verwenden Sie die beiden Funktionen **MonthName** oder **WeekdayName**.

MonthName erwartet die Nummer des gewünschten Monats, also beispielsweise **1** für **Januar**. Im Direktfenster funktioniert die Funktion wie folgt:

```
? MonthName(1)
Januar
```

Wenn Sie nur die Abkürzung des Monatsnamens benötigen, stellen Sie den zweiten Parameter der Funktion auf den Wert **True** ein:

```
? MonthName(1, True)
Jan
```

WeekdayName erwartet eine Zahl zwischen 1 und 12. Auch hier ein Beispiel für das Direktfenster:

```
? WeekdayName(1)
Montag
```

Welcher Wochentag der Zahl **1** entspricht, legen Sie mit dem dritten Parameter fest. Wenn dies beispielsweise der Sonntag sein soll und die Funktion die Abkürzung ausgeben soll, verwenden Sie folgenden Aufruf:

```
? WeekdayName(1, True, vbSunday)
So
```

Diese Funktionen können Sie sowohl unter VBA als auch in Ausdrücken in Abfragen oder Steuerelementen verwenden.

Dort werden die Funktionsnamen dann allerdings in deutscher Sprache ausgegeben.

Texte in leeren Feldern vorblenden

Wenn Sie ein Formular mit Textfeldern verwenden und den Benutzer über noch nicht ausgefüllte Felder informieren möchten (die gegebenenfalls auch Pflichtfelder sein können), tragen Sie einfach nur einen bestimmten Wert für die Eigenschaft **Format** des Textfelds ein.

Wenn das Textfeld **Vorname** beispielsweise in roter Schrift den Text **[Vorname]** anzeigen soll, solange das Feld leer ist, verwenden Sie den folgenden Ausdruck:

```
@;[Rot]"[Vorname]"
```

Bild 4 zeigt, wie Sie die Eigenschaft mit dem entsprechenden Ausdruck füllen und wie das Ergebnis aussieht.

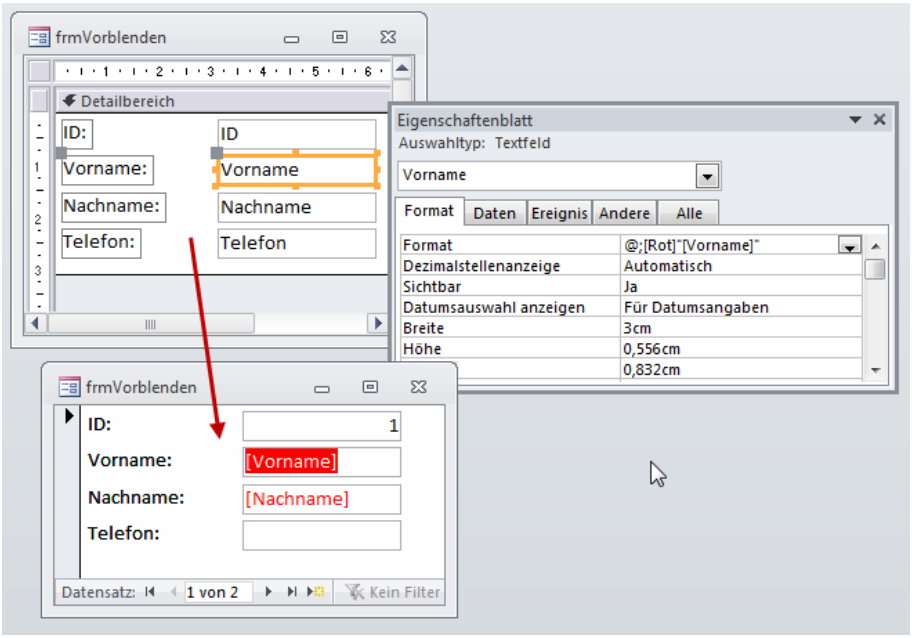


Bild 4: Einstellung zum Vorblenden eines Wertes in leeren Feldern

Wenn der Benutzer einen Wert eingibt, geschieht dies noch in roter Schrift. Nach dem Aktualisieren des Inhalts erscheint dieser jedoch in schwarzer Schrift.

Dies funktioniert nicht nur mit gebundenen Feldern, also solchen, deren Eigenschaft **Steuerelementinhalt** den Namen eines Feldes der Datenherkunft enthält, sondern auch mit ungebundenen Feldern.

Steuerelementlayouts mit Access 2007 und jünger

Wenn Steuerelemente nebeneinander oder untereinander an einer Linie ausgerichtet werden sollen, ihre Größe anzugleichen ist oder die Reihenfolge angepasst werden soll, zog dies bis Access 2003 eine wahre Klickorgie nach sich. Mit Access 2007 hat Microsoft sogenannte Layouts eingeführt, mit denen Sie Steuerelemente zu Gruppen zusammenfassen und diese einfacher anpassen können. Dieser Artikel zeigt, welche Möglichkeiten diese Technik bietet.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1303_Steuerelementlayouts.accdb**.

Layouts – wo einsetzen?

Layouts setzen Sie in Formularen oder Berichten ein, wobei die Auswirkungen auf die Datenblattansicht nur marginal sind – dazu später mehr.

Ein Layout ist prinzipiell eine Schablone, die Sie über die gewünschten Steuerelemente stülpen und die Sie damit vereinfacht anordnen können. Layouts sind also immer sinnvoll, wenn Sie entweder die Größe, die Ausrichtung oder die Anordnung einer Gruppe von Steuerelementen anpassen möchten.

Es gibt zwei Typen von Layouts – tabellarische oder gestapelte. Beim tabellarischen Layout werden die Steuerelemente so angeordnet, dass die Bezeichnungsfelder in einer oberen Reihe landen und die Steuerelemente mit den eigentlichen Inhalten in einer unteren Reihe. Die obere Reihe befindet sich dann in der Regel im Formularkopf beziehungsweise in einem Kopfbereich eines Berichts, die untere Reihe im Detailbereich.

Beim gestapelten Layout landen die Bezeichnungsfelder links und die gebundenen Steuerelemente rechts (wir arbeiten in den folgenden Beispielen immer mit gebundenen Steuerelementen und ihren Bezeichnungsfeldern – Sie können aber auch andere Steuerelemente in ein Layout einbeziehen).

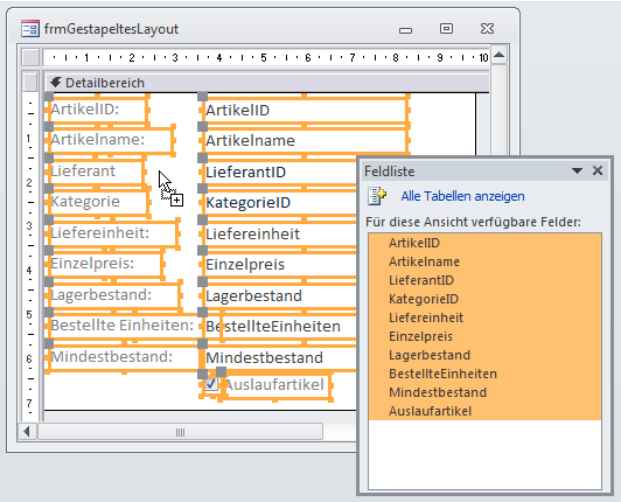


Bild 1: Einfügen der Felder in das Formular

Gestapeltes Layout

Aber schauen wir uns doch einfach an, wie Layouts funktionieren. Für ein erstes gestapeltes Layout erstellen Sie ein neues Formular namens **frmArtikelGestapelt** und fügen seiner Eigenschaft Datenherkunft die Tabelle **tblArtikel** hinzu. Ziehen Sie dann alle Felder aus der Feldliste in den Detailbereich der Entwurfsansicht dieses Formulars (siehe Bild 1).

Wechseln Sie nun im Ribbon zum Tab **Anordnen**. Dort finden Sie alle für die folgenden Schritte notwendigen Befehle (siehe Bild 2).

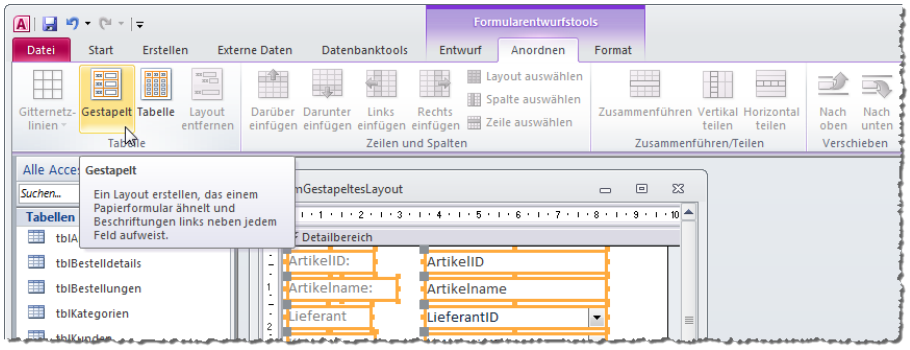


Bild 2: Einfügen der Felder in das Formular

Wir markieren zunächst alle Steuerelemente, die in das Layout **Gestapelt** überführt werden sollen und klicken dann auf die Schaltfläche **Gestapelt** im Ribbon-Tab **Anordnen**. Das Ergebnis sieht wie in Bild 3 aus: Die Steuerelemente wurden neu ange-

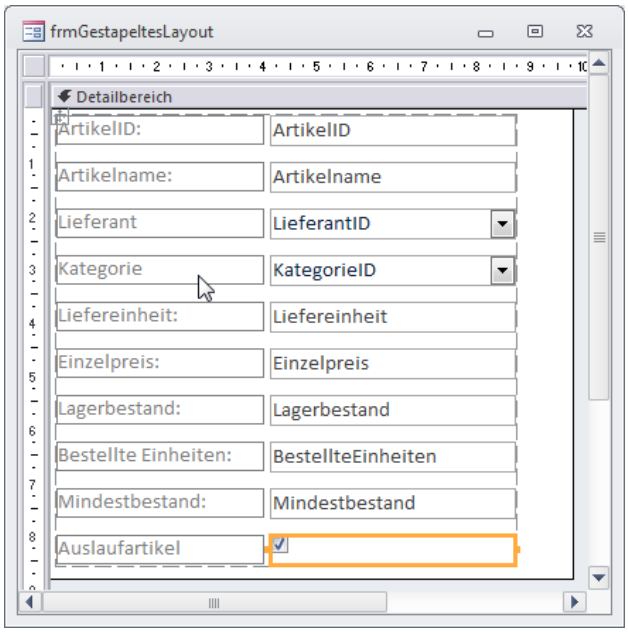


Bild 3: Die Steuerelemente nach dem Hinzufügen des Layouts **Gestapelt**.

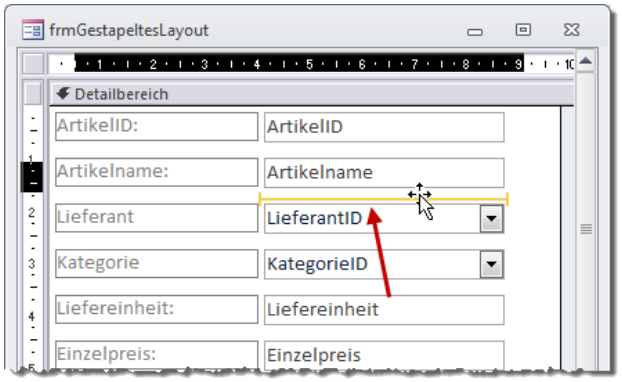


Bild 4: Verschieben eines Steuerelements per Drag and Drop

ordnet und befinden sich nun innerhalb eines gestrichelten Rahmens.

Positionen tauschen

Durch das Anlegen des Layouts allein haben Sie noch nicht viel gewonnen – mit Ausnahme dessen, dass Sie nun die vielen Features des Layouts nutzen können. Die am wenigsten offensichtlichen offenbaren sich typischerweise, wenn Sie einfach mal mit der Maus im Entwurf herumhantieren und etwa das Feld **Liefereinheit** per Drag and Drop nach oben verschieben – so wie in Bild 4.

Das Ergebnis ist beeindruckend – zumindest wenn man bedenkt, was für ein Aufwand es in früheren Access-Versionen war, die Reihenfolge einiger Steuerelemente zu ändern: Das Steuerelement wird mit Bezeichnungsfeld an die gewünschte Position verschoben und die übrigen Steuerelemente rücken um eine Position nach unten (siehe Bild 5).

Wenn Sie entscheiden, dass ein vorhandenes Feld nicht benötigt wird, gibt es mehrere Möglichkeiten. Die erste ist, allen das Bezeichnungsfeld oder das gebundene Steuerelement zu markieren und dieses zu löschen. Wenn Sie das Bezeichnungsfeld löschen, wird dieses entfernt und hinterlässt eine leere Zelle im Layout. Wenn Sie das gebundene Steuerelement löschen, werden Bezeichnungsfeld und Steuerelement gelöscht und hinterlassen eine komplette leere Zeile im Layout. Wenn Sie jedoch beide markieren, entfernt Access die komplette Zeile aus dem Layout und die darunter liegenden Steuerelemente werden um eine Position nach oben verschoben.

Das Entfernen allein des Steuerelements ist sinnvoll, wenn Sie dort anschließend ein anderes Steuerelement aus der Feldliste einfügen möchten. Dann ziehen Sie dieses einfach aus der Feldliste dort hinein.

Anderenfalls können Sie auch Einträge der Feldliste zwischen bereits vorhandene Einträge einfügen – die darunter liegenden Zeilen werden dann einfach nach unten verschoben.

Theoretisch hatten wir uns vorgestellt, dass sich nach dem markieren einer Zeile auch die beiden Ribbon-Einträge **Verschieben|Nach oben** und **Verschieben|Nach unten** im Tab **Anordnen** zum Tauschen der vertikalen Position nutzen lassen. Allerdings lassen sich diese beiden Schaltfläche nicht aktivieren.

Größe ändern

Wenn Sie nun entscheiden, dass die Breite der Bezeichnungsfelder zu groß ist, können Sie diese für

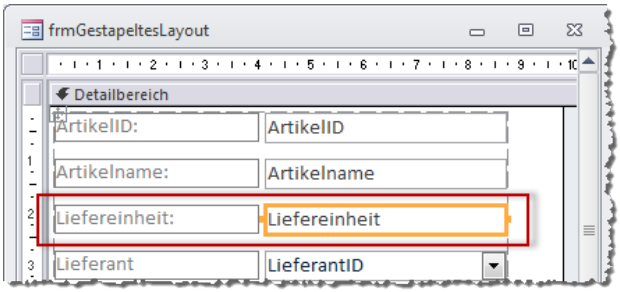


Bild 5: Drag and Drop verschiebt das Bezeichnungsfeld gleich mit und platziert die unter dem Ziel liegenden Steuerelemente um eine Position nach unten.

alle Bezeichnungsfelder gleichzeitig einstellen. Sie brauchen nicht erst alle betroffenen Steuerelemente zu markieren, sondern nur eines – und dessen Breite stellen Sie dann durch Verschieben des rechten Randes ein (siehe Bild 6). Das Schöne ist: Die Breite der übrigen Bezeichnungsfelder wird direkt mit angepasst.

Die Höhe der Steuerelemente können Sie indes nicht auf diese Weise ändern.

Elemente auswählen

Um verschiedene Aktionen mit den Elementen des Layouts durchzuführen, müssen Sie die betroffenen Elemente markieren. Dies gelingt auf verschiedene Arten, zum Beispiel mit der Maus.

Wenn Sie einmal mit der Maus auf ein Element oder auch den Formular-/Berichtshintergrund außerhalb des Layouts klicken, wird die aktuelle Markierung des Layouts aufgehoben und die enthaltenen Steuerelemente erscheinen wie herkömmliche Steuerelemente. Wenn Sie nun auf ein beliebiges Element klicken, wird dieses markiert und der gestrichelte Rahmen des Layouts erscheint.

Wenn Sie mehrere Elemente markieren möchten, klicken Sie diese bei gedrückter **Strg**- oder **Umschalt**-Taste an. Wenn Sie eine komplette Spalte markieren möchten, haben Sie mehrere Möglichkeiten: Zum Beispiel bewegen Sie den Mauszeiger zwischen zwei Elemente einer Spalte, bis ein schwarzer Pfeil nach unten erscheint. Wenn Sie nun die linke Maustaste betätigen, markieren Sie die komplette Spalte (siehe Bild 7). Sie können auch ein Element der Spalte anklicken und dann den Ribbon-Eintrag **Zeilen und Spalten|Spalte auswählen** betätigen.

Auf die gleiche Weise markieren Sie alle Elemente einer Zeile: Bewegen Sie den Mauszeiger zwischen zwei Elemente (oder vor das erste Element), bis ein schwarzer Pfeil nach rechts erscheint und klicken Sie dann mit der linken Maustaste. Oder Sie markieren zuerst ein Element der Zeile und wählen dann den Ribbon-Eintrag **Zeilen und Spalten|Zeile auswählen** aus.

Wenn Sie alle Elemente markieren möchten, aktivieren Sie das Layout durch einen Mausklick auf eine beliebige Stelle innerhalb des Layouts und klicken Sie dann auf das kleine graue Kästchen oben links (siehe Bild 8). Oder Sie markieren zuerst ein Element

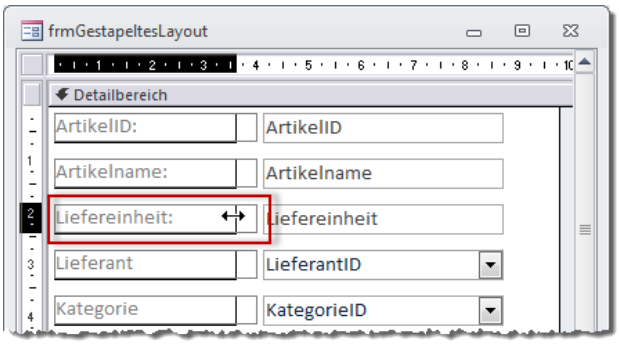


Bild 6: Das Verändern der Breite eines Steuerelements einer Spalte ändert die übrigen Steuerelemente gleich mit.

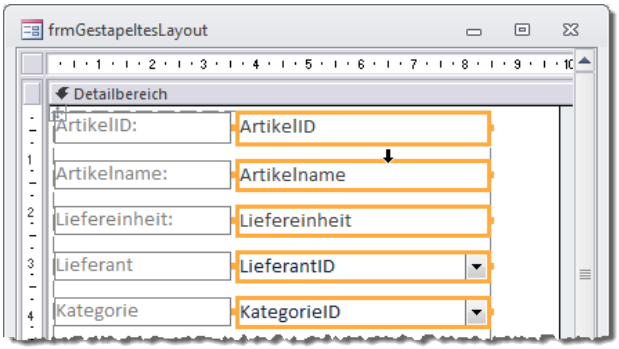


Bild 7: Markieren einer Spalte im Layout

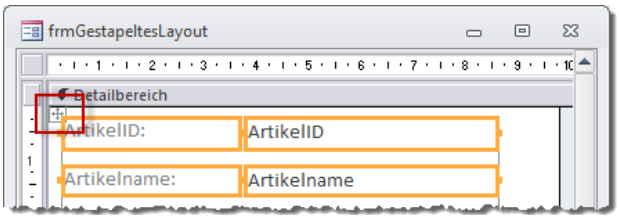


Bild 8: Markieren des kompletten Layouts

und wählen dann den Ribbon-Eintrag **Zeilen und Spalten|Layout auswählen** aus.

Gitternetzlinien

Durch das Anwenden des Layouts hat Access ein Raster angelegt, in dem sich die Steuerelemente befinden. Nachdem Sie wissen, wie Sie die enthaltenen Elemente markieren, können Sie mit diesem Raster eine ganze Menge anstellen. Zum Beispiel können Sie Gitternetzlinien zum Raster hinzufügen. Die notwendigen Befehle finden Sie im Ribbon unter dem Menüpunkt **Gitternetzlinien**.

Auf welche Zellen sich die über diesen Menüpunkt ausgewählten Befehle auswirken, legen Sie durch das vorherige Markieren der Zellen fest. Wenn Sie beispielsweise das komplette Layout markieren und dann den Eintrag **Gitternetzlinien|Beide**

Balkendiagramme mit Bordmitteln

Zur Darstellung von Daten mit Balkendiagrammen gibt es verschiedene Varianten. Wenn man beachtet, dass ein Balkendiagramm eine sehr einfache Visualisierung zum Vergleichen von Zahlenwerten ist, sollte man sich fragen, wie man dies mit einfachsten Mitteln hinbekommt. In diesem Fall ist es einfach: Die Länge des Balkens soll ja schlicht und einfach einem Zahlenwert entsprechen. Warum also nicht einfach für jeden Datensatz ein beliebiges Zeichen entsprechend diesem Zahlenwert anzeigen? Wie dies gelingt, zeigt der vorliegende Artikel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1303_Abfragebalkendiagramm.mdb**.

Einfaches Balkendiagramm

Der einfachste Weg, eine Art Balken auf Basis eines Zahlenwertes zu erzeugen, ist die Darstellung eines geeigneten Zeichens wie etwa dem Pipe-Zeichen (**|**) in entsprechender Anzahl. Dies sieht beispielsweise wie in der Abfrage **qryBalkendiagrammEinfach** aus Bild 1 aus.

Stellt sich die Frage, wie man die Zeichen in entsprechender Anzahl ausgibt – und die ist leicht zu beantworten, wenn man sich einmal die gängigen VBA-Funktionen angesehen hat. Dort gibt es auch eine Funktion namens **String()**. Diese erwartet zwei Parameter: die Anzahl der anzuzeigenden Zeichen und das anzuzeigende Zeichen. Sie können diese Funktion ganz einfach im Direktbereich des VBA-Editors testen. Geben Sie dort beispielsweise den folgenden Ausdruck ein, erhalten Sie zehn Mal das Pipe-Zeichen:

```
? String(10, "|")
||||||||||
```

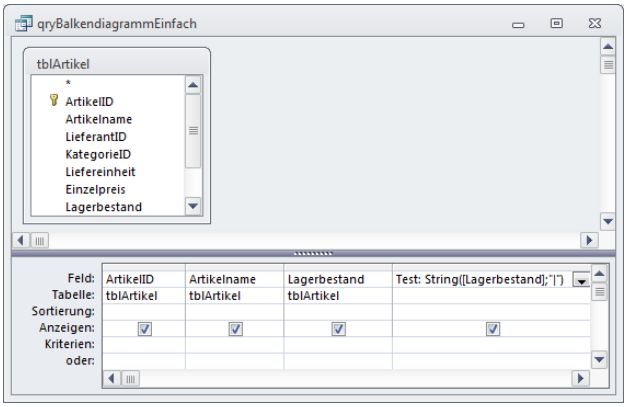


Bild 2: Entwurfsansicht einer Abfrage zur Darstellung eines einfachen Balkendiagramms

Das ist auch schon das komplette Geheimnis hinter der Abfrage **qryBalkendiagrammEinfach**. Deren Entwurf finden Sie in Bild 2. Das Feld **Balken** verwendet dabei den folgenden Ausdruck, um den gewünschten Balken auszugeben:

```
Test: String([Lagerbestand];"|")
```

Diese Darstellung sieht allerdings sehr rudimentär aus, sodass wir noch etwas weiterforschen. Eine Verbesserung wäre es etwa, wenn wir tatsächlich einen Balken erhalten, bei dem man nicht die einzelnen Zeichen erkennt. Auch dies ist möglich: Man muss

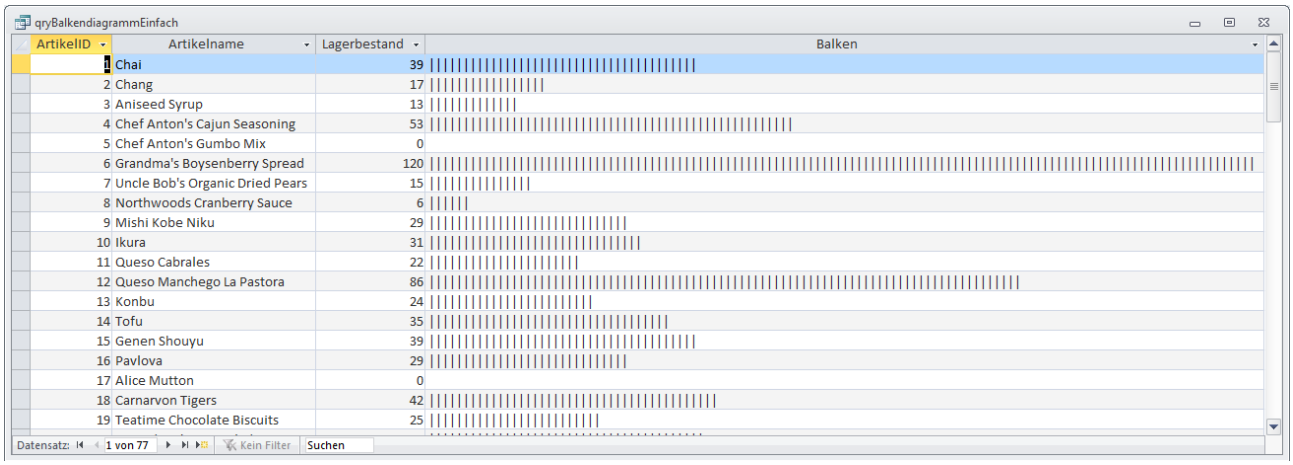


Bild 1: Einfaches Balkendiagramm direkt in der Abfrage

lediglich ein Zeichen finden, dass keine Lücke zwischen zwei Zeichen liefert.

Balkendiagramm mit Sonderzeichen

Nun wollen wir für das Balkendiagramm ein geeigneters Zeichen finden – eines, mit dem sich ein durchgängiger Balken erzeugen lässt. Dazu erstellen Sie für das nächste Beispiel zunächst eine Kopie der Abfrage **qryBalkendiagrammEinfach** und nennen Sie diese **qryBalkendiagrammMitSonderzeichen**.

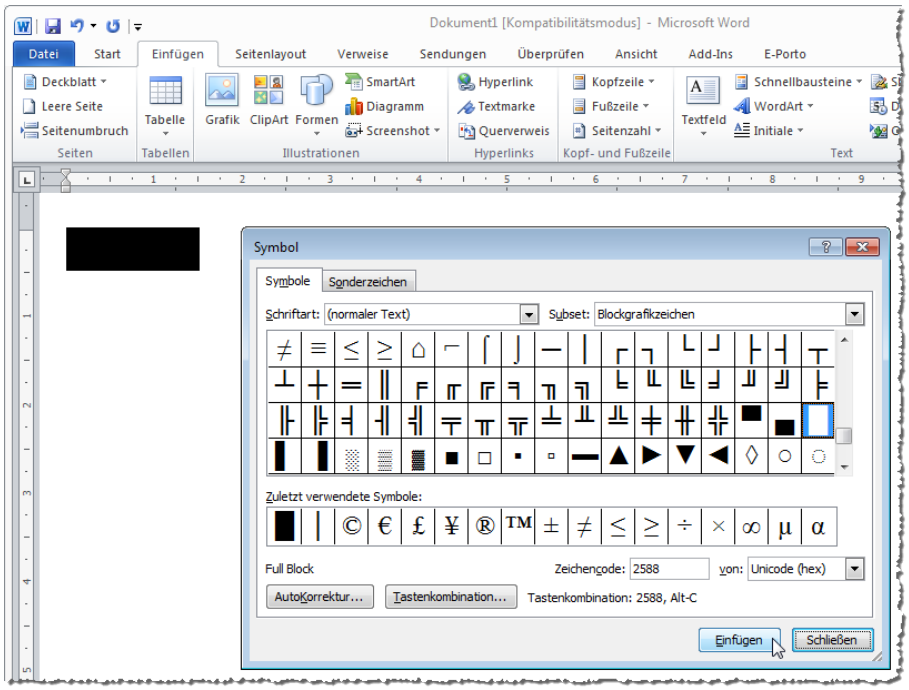


Bild 3: Sonderzeichen suchen mit Word

Sonderzeichen finden

Bevor wir uns nun um die Anpassung der Abfrage und um das Formular kümmern, benötigen wir zunächst einmal ein geeignetes Sonderzeichenn in der entsprechenden Schriftart. Wie aber finden wir ein solches Sonderzeichen? Ganz einfach: mithilfe von Word.

Dort gibt es einen Dialog namens **Symbol**, der alle Zeichen einer Schriftart liefert. Nach einer kleinen Suche findet sich beispielsweise ein Zeichen wie in Bild 3, das sich hervorragend zur Abbildung von Balkendiagrammen eignet.

Theoretisch brauchten sich nun nur den ASCII-Code für dieses Zeichen zu merken (hier **2588**). Wichtig ist, dass es sich hierbei um ein Unicode-Zeichen handelt.

Dieses sollte sich mit der **ChrW()**-Funktion (nicht mit der **Chr()**-Funktion, die liefert nur die Zeichen für Codes von **0** bis **255**) ausgeben lassen. Außerdem müssen Sie noch herausfinden, zu welcher Schriftart das Zeichen gehört.

Dies ist aus dem Dialog nicht direkt ersichtlich, sodass Sie das Zeichen am einfachsten einmal in das aktuelle Word-Dokument einfügen und im Ribbon-Bereich **Start|Schriftart** den Namen der Schriftart ermitteln. Das Zeichen mit dem Code 2588 wird allerdings in allen Schriftarten ähnlich abgebildet.

Nun fügen wir der Abfrage **qryBalkendiagrammMitSonderzeichen** in der vierten Spalte den folgenden Ausdruck hinzu:

```
Balken: String([Lagerbestand];ChrW(2588))
```

Das Ergebnis ist nicht besonders ermutigend (siehe Bild 4) – anscheinend interpretiert Access die Sonderzeichen anders.

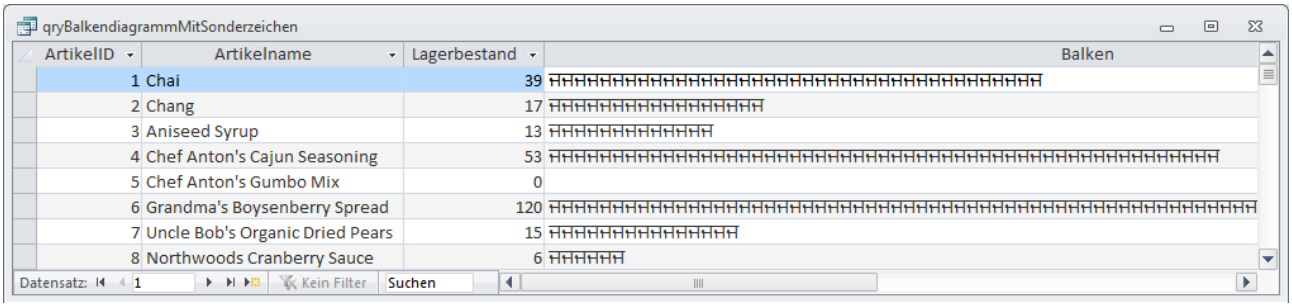


Bild 4: Die Abrage liefert das falsche Sonderzeichen

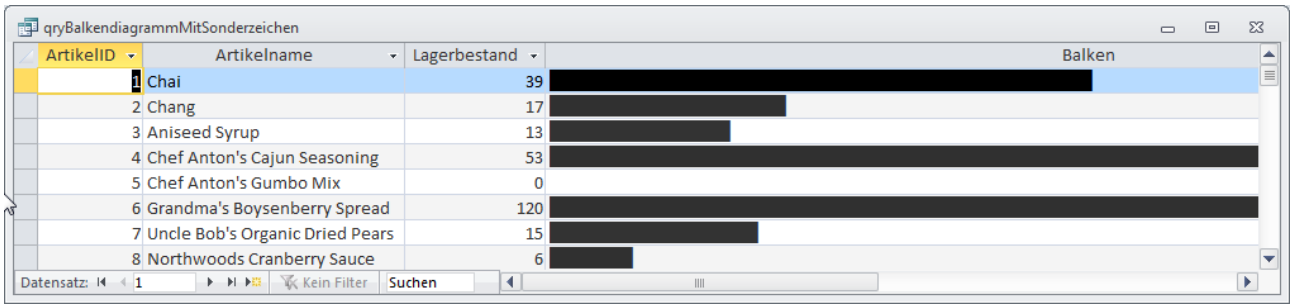


Bild 5: Mit dem manuell ermitteln Unicode liefert die Abfrage das richtige Zeichen.

Also versuchen wir, den Zeichencode auf die harte Tour zu ermitteln: Sie kopieren das Zeichen aus dem Word-Dokument, erstellen in einem leeren Formular in Access ein Bezeichnungsfeld und fügen das kopierte Zeichen aus der Zwischenablage als Beschriftung des Bezeichnungsfeldes ein.

Speichern Sie das Formular unter **frmTest** und wechseln Sie in die Formularansicht. Aktivieren Sie dann den Direktbereich des VBA-Editors (mit **Strg + G**) und geben Sie dort den folgenden Befehl ein:

```
? AscW(Forms!frmTest.Controls(0).Caption)
```

Dieser liefert einen ganz anderen Code für das angegebene Zeichen, nämlich **9608**. Nach der Änderung der vierten Spalte des Abfrageentwurfs liefert die Datenblattansicht schließlich das richtige Zeichen (siehe Bild 5).

Balken im Formular

Die Anzeige der Balken in der Datenblattansicht der Abfrage ist bereits hilfreich. Allerdings soll der Benutzer ja nicht mit Tabellen oder Abfragen in der Datenblattansicht arbeiten, sondern die Daten in Formularen oder Berichten präsentiert bekommen

Also erstellen wir nun ein Formular, dass die Daten in der Endlosansicht anzeigt.

Legen Sie dazu ein neues Formular in der Entwurfsansicht an und weisen Sie der Eigenschaft **Datenherkunft** die Abfrage **qryBalkendiagrammMitSonderzeichen** zu.

Ziehen Sie dann alle Felder der Abfrage aus der Feldliste in den Detailbereich des Formulars (siehe Bild 6).

Von dort aus ist es ab Access 2007 ganz einfach, die Steuerelemente passend für die Darstellung in einem Endlosformular anzuordnen – Sie fügen die Steuerelemente einfach einem tabellarischen Layout hinzu und nehmen dann noch ein paar kleinere Anpassungen vor, um das Ergebnis aus Bild 7 zu erhalten.

Bild 6: Ausgangsposition des Formulars in der Endlosansicht

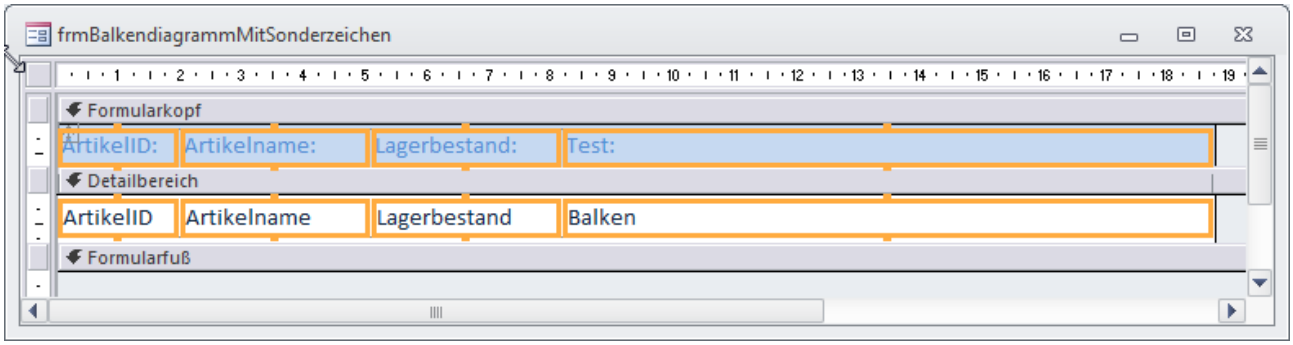


Bild 7: Das Formular mit der Steuerelementanordnung für die tabellarische Ansicht

Haupt- und Unterformular synchron

Es macht wenig Spaß, sich in der Formularansicht durch die Datensätze zu wühlen, bis man endlich zum gesuchten Datensatz gelangt. Genausowenig Freude macht das Bearbeiten von Daten in der Datenblattansicht – zumindest, wenn das Formular nicht alle Felder gleichzeitig anzeigt und man hin- und herscrollen muss. Warum aber nicht beides verbinden und einfach zum Detailformular mit jeweils einem Datensatz noch ein Unterformular hinzufügen, dass alle Datensätze anzeigt und zum Blättern einlädt? Dieser Artikel zeigt, wie dies funktioniert.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1303_HauptUndUnterformularSynchron.mdb**.

Hauptformular erstellen

Das Hauptformular **frmArtikelSynchron** soll zunächst an die Tabelle mit den anzuzeigenden Daten gebunden werden, in diesem Fall **tblArtikel**.

Dazu stellen Sie die Eigenschaft **Datenherkunft** auf diese Tabelle ein. Dann ziehen Sie alle Felder der Tabelle aus der Feldliste in den Detailbereich des Formulars (siehe Bild 1).

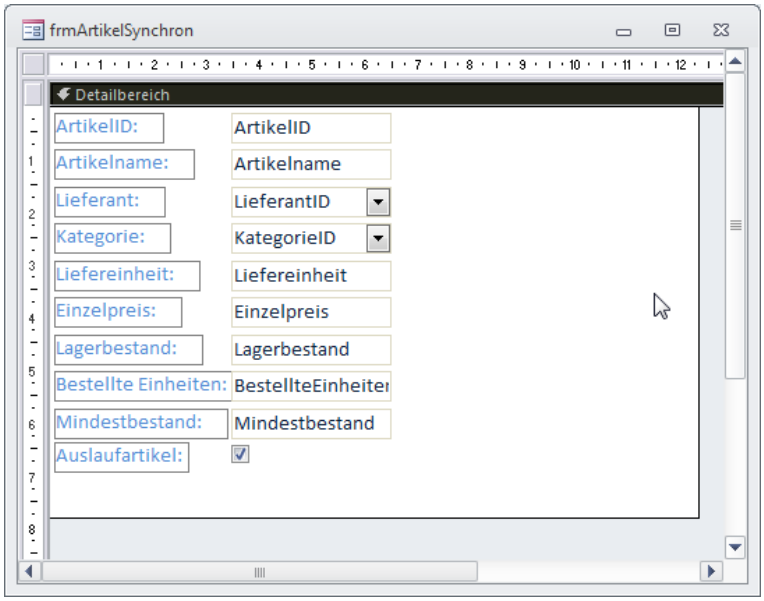


Bild 1: Hauptformular mit Artikeldetails

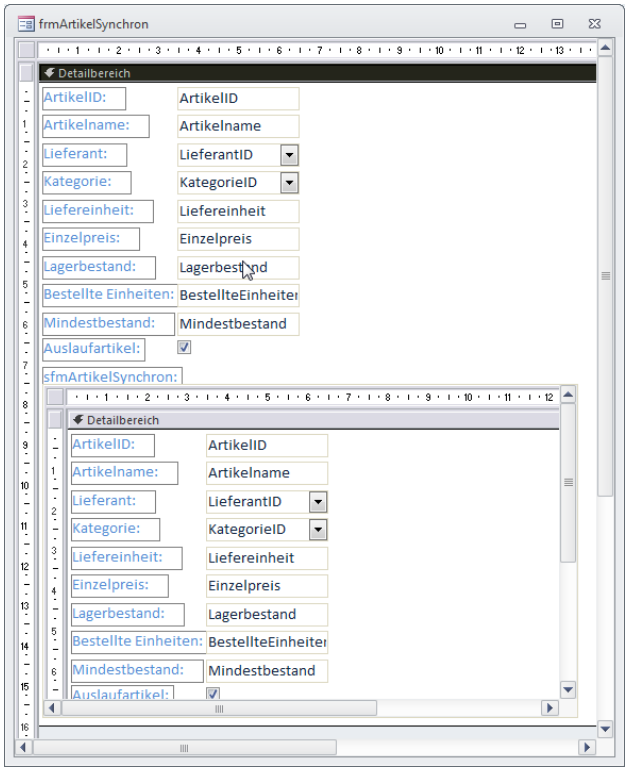


Bild 2: Haupt- und Unterformular

Unterformular erstellen

Das Unterformular soll die gleichen Daten wie das Hauptformular enthalten. Was liegt also näher als das bereits erstellte Hauptformular einfach zu kopieren?

Schließen Sie dieses also, markieren Sie den Eintrag **frmArtikelSynchron** im Datenbankfenster beziehungsweise Navigationsbereich und betätigen Sie die Tastenkombinationen **Strg + C** und **Strg + V**. Access fragt nun nach einen neuen Namen für das zu kopierende Formular, geben Sie hier einfach **sfmArtikelSynchron** an.

Dieses Formular benötigt nur eine Änderung: Öffnen Sie es im Entwurf und stellen Sie die Eigenschaft **Standardansicht** auf den Wert **Datenblatt** ein. Dann schließen Sie das Unterformular wieder.

Nun öffnen Sie das Hauptformular erneut in der Entwurfsansicht und ziehen das Unterformular **sfmArtikelSynchron** in den Detailbereich des Formularentwurfs.

Debugging im VBA-Editor, Teil 2

Nachdem Sie im ersten Teil dieser Artikelreihe einige codebasierten Debugging-Techniken kennengelernt haben, schauen wir uns in diesem Teil weitere VBA-gesteuerte Features an. Außerdem werfen wir einen Blick auf die Tools, die der VBA-Editor uns zum Debuggen bietet. Dazu gehören beispielsweise das Lokalfenster und die Überwachungen.

Anhalten bei bestimmtem Wert

Wenn Sie eine Stelle im Code zur Laufzeit untersuchen möchten, fügen Sie dort entweder einen Haltepunkt oder die Stop-Anweisung ein. Beides hält den Code bei jedem Erreichen der entsprechenden Stelle an.

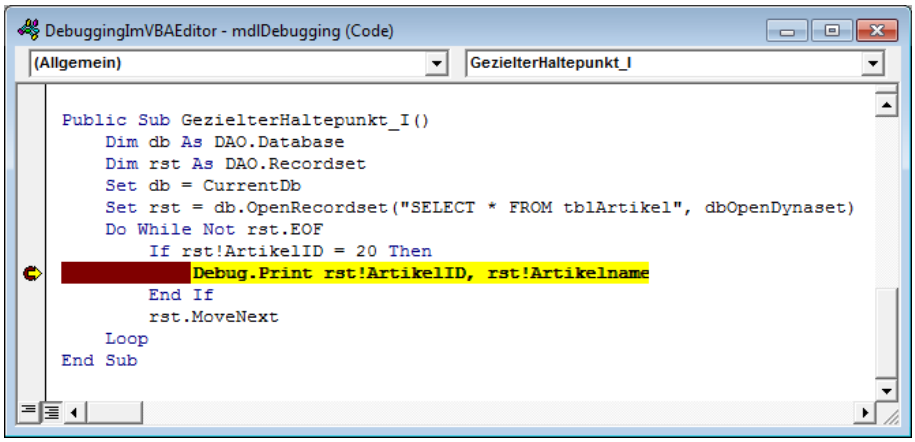


Bild 1: Anhalten des Codes bei Erreichen eines bestimmten Wertes

Manchmal wird die betroffene Codestelle jedoch sehr oft durchlaufen, sodass der Code entsprechend oft angehalten und beispielsweise die Variablen untersucht werden müssen. In vielen Fällen wissen Sie jedoch, dass ein Problem nur unter bestimmten Bedingungen auftaucht. So könnte es sein, dass Sie in einer Schleife alle Datensätze der Tabelle **tblArtikel** durchlaufen, aber schon wissen, dass ein Problem nur bei einem bestimmten Datensatz auftaucht.

Wenn Sie wissen, dass ein Problem immer in Zusammenhang mit dem Datensatz mit dem Wert **20** im Feld **ArtikelID** auftaucht, können Sie gezielt einen Haltepunkt setzen. Damit dieser auch nur für den betroffenen Datensatz erreicht wird, fassen Sie diesen in eine entsprechende **If...Then**-Bedingung ein. Dies sieht beispielsweise wie in Bild 1 aus, wobei die Bedingung **rst!ArtikelID = 20** lautet. Nur wenn diese Bedingung erfüllt ist, wird die innerhalb der If...Then-Bedingung enthaltene und mit einem Haltepunkt versehene Anweisung erreicht. Nun können Sie beispielsweise weitere Werte des aktuellen Datensatzes im Direktfenster ausgeben – etwa so:

```
Debug.Print rst!Artikelbestand
```

Haltepunkt per Assert

Die **Debug**-Klasse bietet neben der **Print**-Methode noch eine weitere, relativ unbekannte Methode. Diese heißt **Assert** und erwartet einen Ausdruck als Parameter, der entweder den Wert **True** oder **False** zurückliefert. Wenn der Ausdruck den Wert **False** zurückgibt, hält die Prozedur automatisch in der Zeile mit der **Debug.Assert**-Anweisung an.

Im Gegensatz zum vorherigen Beispiel, wo die **Debug.Print** mit dem Haltepunkt genau beim Eintreten der gewünschten Bedingung ausgelöst wird, müssen Sie bei **Debug.Assert** also genau dem umgekehrten Fall formulieren.

Auf jeden Fall brauchen Sie mit **Debug.Assert** nur eine statt drei Zeilen, um den Code beim Eintreten der gewünschten Bedingung anzuhalten:

```
Public Sub GezielterHaltepunkt_II()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset( _
        "SELECT * FROM tblArtikel", dbOpenDynaset)
    Do While Not rst.EOF
        Debug.Assert Not (rst!ArtikelID = 20)
        rst.MoveNext
    Loop
End Sub
```

Sie müssen lediglich beachten, dass Sie den Ausdruck durch Voranstellen des Schlüsselworts **Not** ins Gegenteil umkehren.

Das Lokal-Fenster

Wenn Sie eine Prozedur per Einzelschritt durchlaufen oder diese an der gewünschten Stelle auf eine der oben angegebenen Arten angehalten haben, wissen Sie in vielen Fällen noch nicht, wonach Sie suchen müssen. Da hilft es oft, sich einfach einmal einen Überblick über die aktuellen Werte aller derzeit verwendeten Variablen zu verschaffen.

Dabei ist das Lokal-Fenster eine große Hilfe. Sie blenden es mit dem Menübefehl **Ansicht|Lokal-Fenster** ein. Wenn aktuell kein Code durchlaufen wird, ist das Fenster leer. Deshalb schauen wir uns nun an, was das Lokalfenster beim Durchlaufen der Prozedur **GezielterHaltepunkt_II** tut.

Lassen wir die Prozedur ruhig genau an der mit Debug.Assert festgelegten Position anhalten, indem wir die Einfügemarke an beliebiger Stelle innerhalb der Prozedur platzieren und dann auf **F5** drücken.

Wie Bild 2 zeigt, geschieht in der Prozedur doch mehr als erwartet – es gibt mehr Variablen als gedacht. Das liegt daran, dass die beiden in der Prozedur verwendeten Objektvariablen db und rst jeweils auf Objekte verweisen, die eine ganze Reihe von Eigenschaften enthalten. Im Lokal-Fenster können Sie all diese Werte zum aktuellen Zeitpunkt prüfen. Sie brauchen also nicht jede Eigenschaft mit **Debug.Print** im Direktbereich auszugeben, sondern können die Eigenschaften im Lokal-Fenster verfolgen.

Die Stärke des Lokal-Fensters liegt dabei darin, immer alle aktuell verfügbaren Eigenschaften abzubilden. Der Nachteil ist, dass dies mitunter recht viele werden können – und darunter leidet natürlich die Übersicht.

Bevor wir uns die Alternative anschauen, werfen wir noch einen genaueren Blick auf das Lokal-Fenster. Dieses zeigt im oberen Bereich den Namen der aktuellen Prozedur an sowie das Modul und das Objekt, aus dem die Prozedur stammt.

Darunter folgen die eigentlich interessanten Informationen, aufgeteilt in drei Spalten:

- **Ausdruck**: In dieser Spalte finden Sie die Namen von Objekten und ihren Eigenschaften beziehungsweise untergeordneten Auflistungen oder Objekten. Einfachen Variablen etwa vom Typ **String** oder **Long**, die keine Objekte sind, werden immer direkt unterhalb des Standard- oder Klassenmoduls angezeigt, in dem sie sich befinden – hier wäre dies **mdlDebugging**. Ein Objekt wie das

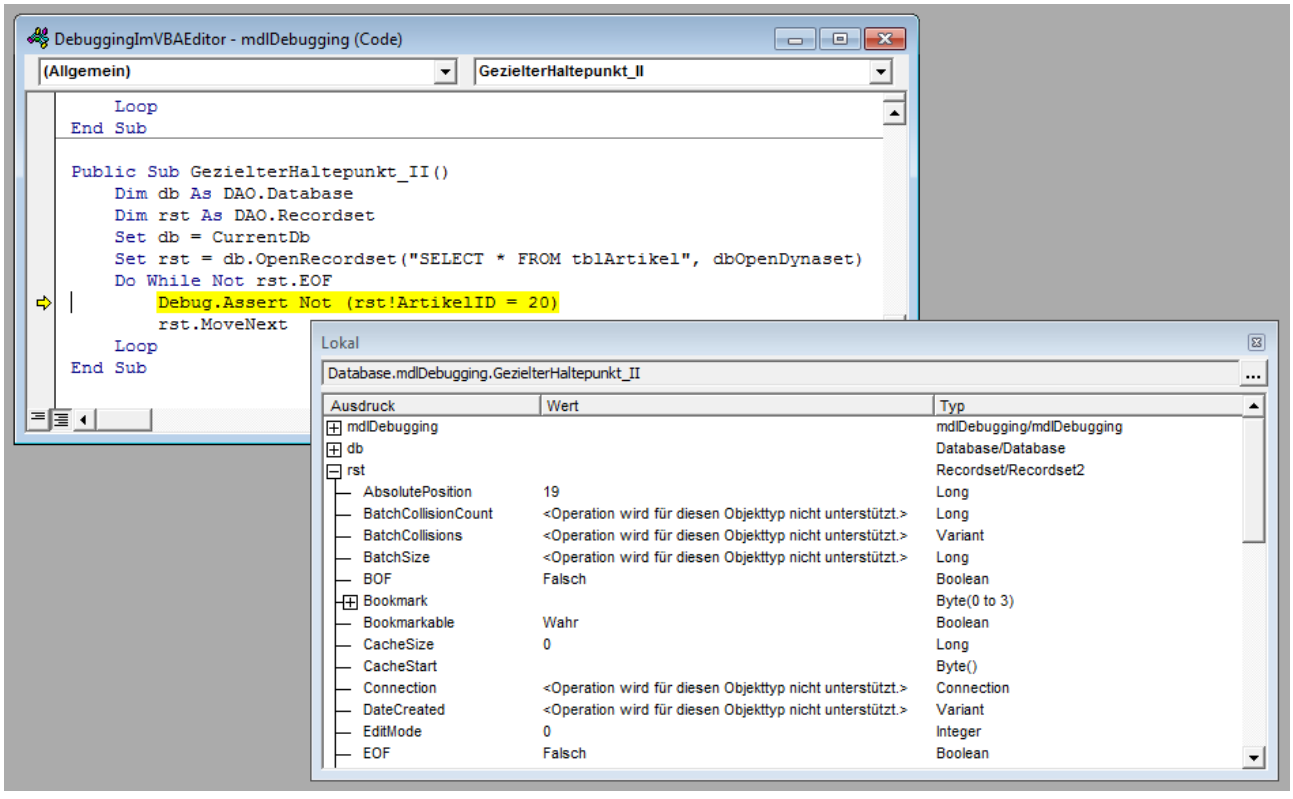


Bild 2: Das Lokal-Fenster liefert eine Reihe Informationen.

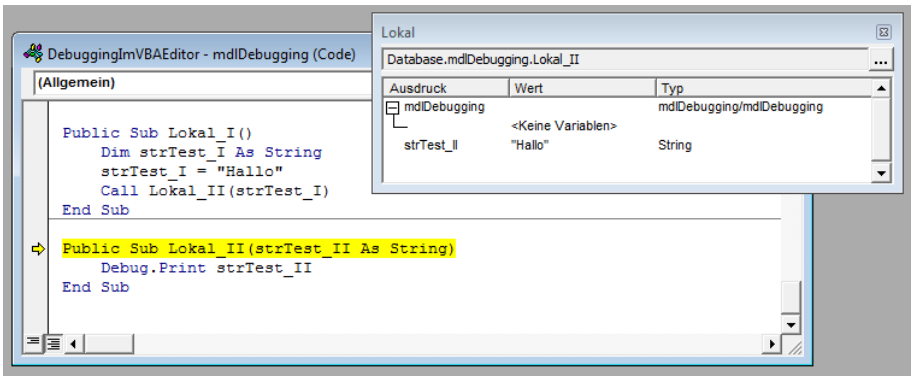


Bild 3: Lokal heißt lokal

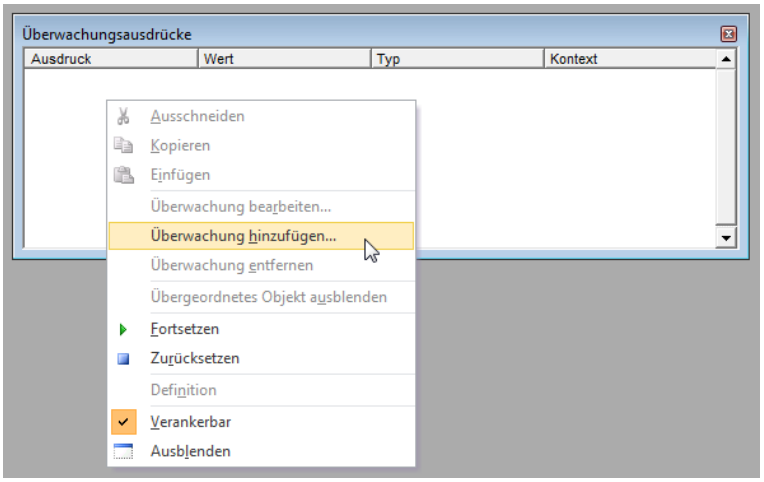


Bild 4: Hinzufügen einer Überwachung

Recordset-Objekt **rst** ist schon recht komplex und enthält nicht nur eigene Eigenschaften, sondern auch noch Verweise auf weitere Objekte, die wiederum Eigenschaften und Objekte enthalten können.

- **Wert**: Diese Spalte enthält den aktuellen Wert von Eigenschaften. Das heißt, dass der Wert dieser nächsten Spalte leer ist, wenn die aktuelle Zeile ein Objekt und keine Eigenschaft enthält.

- **Typ**: Diese Spalte liefert den Datentyp der in der aktuellen Zeile abgebildeten Eigenschaft beziehungsweise des Objekts.

Die Bezeichnung Lokal ist übrigens wörtlicher zu nehmen als man vielleicht zunächst annimmt: Das Lokal-Fenster zeigt tatsächlich nur die Variablen der aktuellen Routine an. Dies zeigt Bild 3: Dort ruft deklariert und füllt eine erste Prozedur eine Variable und übergibt diese dann an die zweite Prozedur. Dort angelangt, zeigt das Lokal-Fenster auch nur noch den Wert der in der zweiten Prozedur per Parameter gefüllten Variablen an.

Zusammenfassend lässt sich das Lokal-Fenster praktisch einsetzen, wenn man noch nicht genau weiß, welche Werte es zu beobachten gilt. Andererseits ist die Beobachtung hiermit auf die aktuell ausgeführte Prozedur eingeschränkt.

Überwachungsfenster

Und damit kommen wir zu einem weiteren Werkzeug – dem Überwachungsfenster. Dieses blenden Sie mit dem Menübefehl **Ansicht|Überwachungsfenster** ein. Im Gegensatz zum Lokal-Fenster zeigt es, auch während des Ablaufs einer Prozedur, standardmäßig erst einmal gar nichts an. Der Unterschied zum Lokal-Fenster ist, dass Sie zuerst festlegen müssen, welche Objekte oder Ausdrücke im Überwachungsfenster beobachtet werden sollen.

Um dies zu erledigen, bietet das Überwachungsfenster den Kontextmenüeintrag **Überwachung hinzufügen...** an (siehe Bild 4).

Hier können Sie sowohl alle als Variable deklarierten Objekte angeben als auch beliebige weitere Ausdrücke wie etwa **Forms!frmTest.Name**. Der wesentliche Unterschied zum Lokal-Fenster ist dabei, dass Sie immer auch die aktuellen Werte von Variablen erhalten, die nicht zur aktuell durchlaufenen Routine gehören.

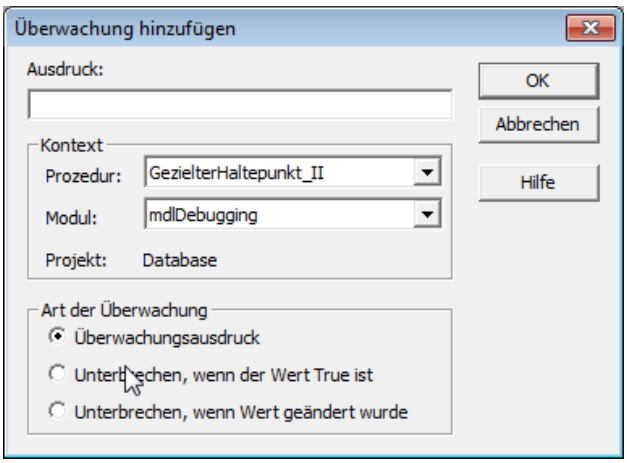


Bild 5: Definieren des Überwachungsausdrucks

Filterkriterien für Formulare, Teil I

Was helfen Artikel-, Kunden-, Bestell- und sonstige Daten, wenn Sie diese zwar schön in einem Formular darstellen, aber diese nicht entsprechend filtern können? In diesem Artikel zeigen wir Ihnen, wie Sie verschiedene individuelle Filterkriterien mithilfe entsprechender Steuerelemente im Formular festlegen können. Damit filtern Sie dann die in einem Unterformular enthaltenen Daten – in diesem Fall Artikelstammdaten.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1304_FilterkriterienFuerFormulare.mdb**.

Basisaufbau

Die Grundlage für die nachfolgend beschriebenen Beispiele zum Filtern von Daten besteht aus einem Haupt- und einem Unterformular. Im Hauptformular befinden sich die Steuerelemente, mit denen Sie die Filterkriterien eingeben, das Unterformular zeigt die zu durchsuchenden Daten an. Alternativ können Sie statt eines Unterformulars auch ein Listenfeld oder ein ListView-Steuerelement verwenden – beides ist mit geringen Änderungen verbunden, die wir an anderer Stelle erläutern.

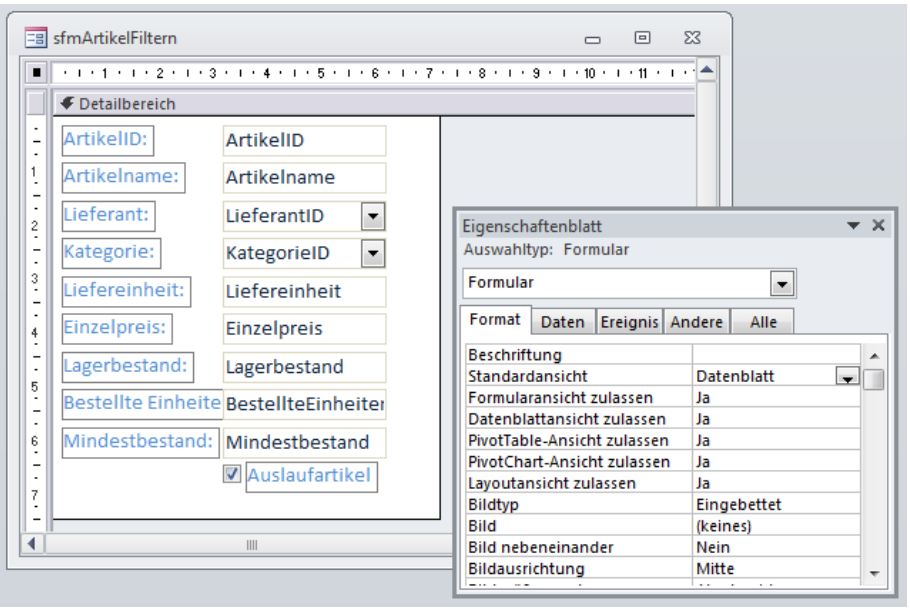


Bild 1: Entwurf des Unterformulars mit den zu filternden Daten

Das Unterformular der Beispieldatenbank heißt **frmArtikelFiltern**. Stellen Sie für die Eigenschaft **Datenherkunft** dieses Formulars die Tabelle **tblArtikel** ein. Ziehen Sie dann alle Felder der Datenherkunft aus der Feldliste in den Detailbereich der Entwurfsansicht des Formulars. Sie brauchen diese nicht ordentlich anzuordnen, denn das Unterformular soll seine Daten ohnehin in der Datenblattansicht anzeigen. Dazu stellen Sie die Eigenschaft **Standardansicht** des Formulars auf den Wert **Datenblatt** ein (siehe Bild 1).

Nun müssen Sie das Formular nur noch speichern und schließen, den Rest erledigen wir vom Hauptformular auf.

Das Hauptformular der Beispieldatenbank heißt **frmArtikelFiltern** und nimmt zunächst nur das Unterformular auf. Wenn Sie das Hauptformular in der Entwurfsansicht geöffnet haben, ziehen Sie das Unterformular aus dem Datenbankfenster beziehungsweise aus dem Navigationsbereich in den Detailbereich des Formulars.

Es darf den Detailbereich komplett ausfüllen, die Steuerelemente zum Einstellen der Filterkriterien platzieren wir später im Formularkopf. Sollten Sie Access 2007 oder neuer verwenden (was ich hoffe), stellen Sie außerdem die Eigenschaften **Horizontaler Anker** und **Vertikaler Anker** auf **Beide** ein. Damit stellen Sie sicher, dass das Unterformular beim Vergrößern des Hauptformulars ebenfalls vergrößert wird.

Da das Hauptformular nicht selbst Daten anzeigt, benötigen wir die entsprechenden Elemente nicht – stellen Sie daher die Eigenschaften **Trennlinien**, **Datensatzmarkierer** und **Navigationsschaltflächen** auf **Nein** ein. Außerdem legen Sie für die Eigenschaft **Automatisch zentrieren** den Wert **Ja** und für **Bildlaufleisten** den Wert **Nein** fest.

Das Formular sieht nun wie in Bild 2 aus.

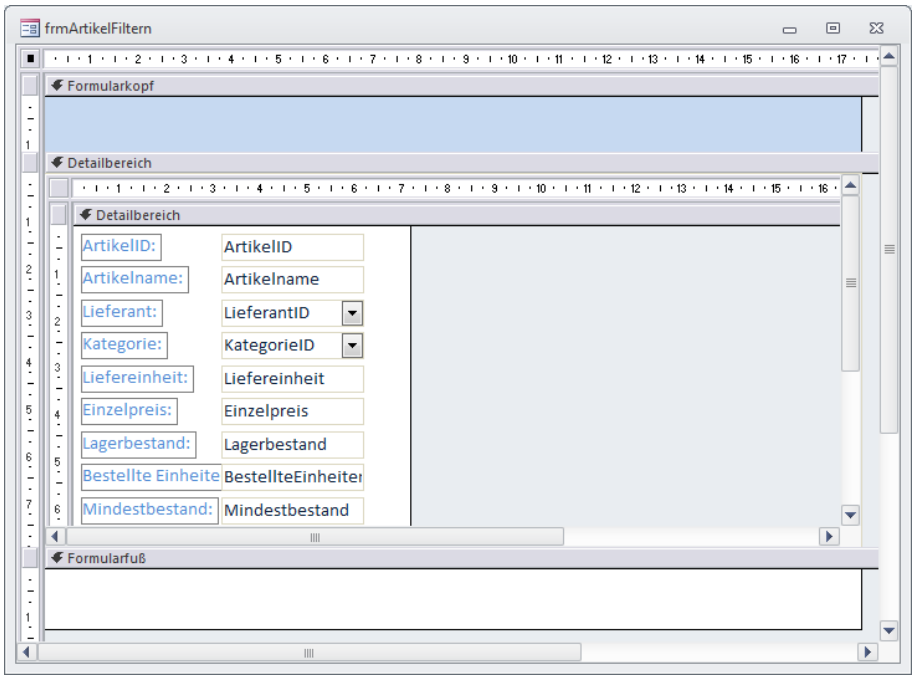


Bild 2: Haupt- und Unterformular in der Entwurfsansicht

Steuerelemente für Filterkriterien

Filterkriterien können Sie mit den verschiedensten Steuerelementen eingeben. In den meisten Fällen ist die Art des Steuerelements bereits vorgegeben. Wenn Sie beispielsweise nach dem Artikelnamen filtern möchten, verwenden Sie dazu ein Textfeld. In diesem können Sie nach Wunsch Platzhalter vordefinieren, damit nach allen Ausdrücken gesucht wird, die mit der angegebenen Zeichenkette beginnen oder enden oder die diese einfach an irgendeiner Stelle enthalten. Oder Sie überlassen dem Benutzer, die notwendigen Platzhalter Sternchen (*****) oder Fragezeichen (**?**) zu platzieren.

Wenn Sie nach einem Datum filtern, setzen Sie zwei Textfelder ein, um ein Start- und gegebenenfalls noch ein Enddatum für ein Datumsbereich eingeben zu können.

Wenn Sie nach einem Feld filtern möchten, dessen Werte mit einem Nachschlagefeld ausgewählt wurden, wird es bereits interessant. Dann enthält das Nachschlagefeld in der Regel nicht den angezeigten Ausdruck als Wert, sondern den Wert des Primärschlüsselfeldes der Tabelle, aus der die Daten des Nachschlagefeldes stammen. Sie können dann auf zwei Arten nach den Daten dieses Feldes filtern: erstens nach dem Wert der gebundenen Spalte, also dem tatsächlich gespeicherten Wert, oder nach dem angezeigten Wert.

Bei der ersten Variante stellen Sie zur Eingabe des Filterkriteriums ein Kombinationsfeld zur Verfügung, das einfach alle Einträge anbietet, die auch im zu durchsuchenden Feld vorliegen. Wenn Sie also alle Artikel suchen, die zur Kategorie **Getränke** gehören, würden Sie im Kombinationsfeld zum Filtern den Wert **Getränke** auswählen.

Es kann natürlich auch sein, dass Sie alle Artikel ermitteln möchten, deren Lieferant mit dem Buchstaben **A** beginnt. Dies ist etwas aufwendiger, weil wir hier nicht mehr nur die Daten der Tabelle **tblArtikel** benötigen, sondern auch die Daten der Tabelle **tblLieferanten**, welche die im Kombinationsfeld angezeigten Daten liefert.

Schließlich gibt es noch die **Ja/Nein**-Felder. Nach **Ja/Nein**-Feldern zu filtern, ist komplizierter, als es scheint: Naiverweise mag man davon ausgehen, dass ein einfaches Kontrollkästchen ausreicht, um nach solchen Werten zu filtern. Damit können Sie allerdings normalerweise nur den Wert **Ja** oder **Nein** beziehungsweise **Wahr** oder **Falsch** einstellen.

Was aber, wenn die zu durchsuchende Datenherkunft alle Datensätze liefern soll, also sowohl solche mit dem Wert **Ja** als auch die mit dem Wert **Nein**?

Kontrollkästchen besitzen zwar den sogenannten Dreifachstatus, aber der ist eher weniger intuitiv zu bedienen (mit drei Mausklicks durchlaufen Sie dabei die drei Werte **-1** (**Ja**), **0** (**Nein**) und **Null**, wobei man **Null** mit **Alle anzeigen** belegen könnte.

Alternativ könnte man die drei Werte **Ja**, **Nein** und **Alle** auch mit einem Kombinationsfeld oder mit einer Optionsgruppe abdecken.

Bei Zahlenwerten gibt es schließlich auch verschiedene Möglichkeiten: Entweder Sie prüfen nur auf exakte Übereinstimmungen oder Sie verwenden auch hier, genau wie beim Datum, zwei Textfelder zur Eingabe von Wertebereichen.

Natürlich kann es auch sein, dass ein beliebiges Feld mit Texten, Zahlen oder Datumsangaben viele gleichartige Werte enthält.

In diesem Fall ist es vielleicht sinnvoll, statt eines Textfeldes ein Kombinationsfeld zum Filtern einzusetzen.

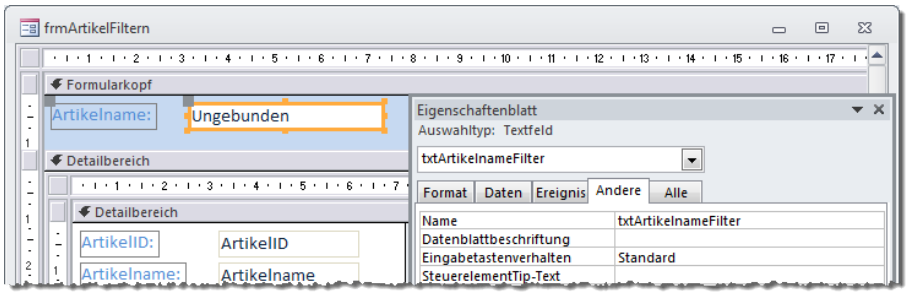


Bild 3: Textfeld zur Eingabe eines Filterkriteriums für den Artikelnamen

Diese könnte dann alle möglichen Werte in einer sinnvollen Reihenfolge zur Auswahl bereitstellen, um schnell die betroffenen Datensätze zu ermitteln – so sparen Sie dem Benutzer sogar noch die Eingabe des Suchbegriffs.

Filtern nach Textfeldern

Beginnen wir mit dem Filtern der Datensätze nach dem Inhalt eines Textfeldes, in diesem Fall **Artikelname**.

Zur Eingabe des Vergleichswertes legen Sie im Formularkopf ein Textfeld namens **txtArtikelnameFilter** an (siehe Bild 3).

Es gibt nun ein paar Fragen zu klären. Die erste ist: Soll der Benutzer selbst Platzhalter eingeben können oder möchten Sie entsprechende Elemente zur Benutzeroberfläche hinzufügen?

Eine weitere Möglichkeit wäre, standardmäßig bestimmte Platzhalter festzulegen, um beispielsweise alle Einträge zu filtern, deren Artikelname mit dem eingegebenen Ausdruck beginnt oder deren Artikelnamen den Ausdruck an beliebiger Stelle enthält.

Die zweite Frage ist: Wann soll gefiltert werden – nach der Eingabe eines jeden Zeichens oder erst nach Abschluss der Eingabe etwa durch Betätigen der Eingabetaste? Wir schauen uns beide Varianten an.

Einfacher Filter

Am einfachsten ist es, den Benutzer mit einem Textfeld zu konfrontieren, in das er den Vergleichswert für das zu filternde Feld einträgt. Wenn er die Eingabetaste betätigt, soll eine Ereignisprozedur ausgelöst werden, welche das Recordset des Unterformulars mit einem entsprechenden Filter versieht.

Ein passendes Ereignis für diesen Fall ist **Nach Aktualisierung**. Wechseln Sie also in den Entwurf des Formulars, markieren Sie das Textfeld **txtArtikelnameFilter** und wählen Sie für die Eigenschaft **Nach Aktualisierung** den Wert **[Ereignisprozedur]** aus.

Klicken Sie dann auf die Schaltfläche mit den drei Punkten (**...**), damit Access die Prozedur **txtArtikelnameFilter_AfterUpdate** anlegt. Diese ergänzen Sie wie in Bild 4 mit den folgenden Zeilen:

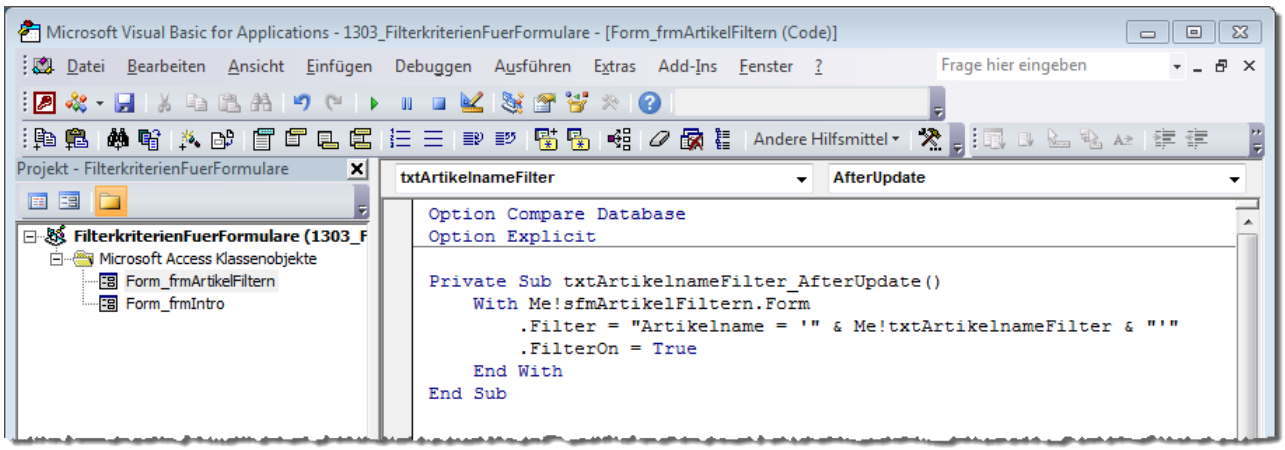


Bild 4: Diese Prozedur wird nach dem Aktualisieren des Inhalts des Textfeldes ausgelöst.

```
With Me!sfmArtikelFiltern.Form
    .Filter = "Artikelname = '" 7
                & Me!txtArtikelnameFilter & "'"
    .FilterOn = True
End With
```

Beim Definieren des Filterkriteriums, dass direkt der **Filter**-Eigenschaft des Unterformulars zugewiesen wird, gibt es nur wenig zu beachten: Es besteht aus dem Feldnamen, nach dem gefiltert werden soll, dem Vergleichsoperator (hier dem Gleichheitszeichen) und dem Vergleichswert. Bei Vergleichen mit Feldern, die Texte enthalten, müssen Sie den Vergleichswert in Anführungszeichen setzen. Dann setzt sich der Vergleichsausdruck so zusammen:

```
"Artikelname = '" & Me!txtArtikelnameFilter & "'"
```

Der Vergleichsausdruck muss insgesamt in Anführungszeichen eingefasst werden, da er als Literal an die **Filter**-Eigenschaft übergeben wird. Da der Vergleichsausdruck seinerseits in Anführungszeichen eingefasst werden muss, haben wir ein kleines Problem. Sie können es nämlich nicht einfach in die bereits in Anführungszeichen eingefassten Teile des Vergleichsausdrucks integrieren:

```
"Artikelname = "" & Me!txtArtikelnameFilter & """
```

Wenn Sie auf der Verwendung von Anführungszeichen bestehen, müssen Sie diese innerhalb von Anführungszeichen verdoppeln, also so:

```
"Artikelname = """ & Me!txtArtikelnameFilter & """"
```

In den meisten Fällen ist dies die einzige Möglichkeit. In diesem Fall wird der Vergleichsausdruck jedoch von der Jet-Engine verarbeitet, die etwa für die Bereitstellung von Daten aus Tabellen und Abfragen zuständig ist.

Und da die **Filter**-Eigenschaft letztlich ein SQL-Ausdruck ist, der von der Jet-Engine zur Bereitstellung der Daten verwendet wird, können Sie dort die unter Access-SQL zulässigen Zeichen verwenden.

Für das Anführungszeichen bedeutet dies, dass es auch durch ein Hochkomma ersetzt werden kann – dies erhöht in vielen Fällen die Übersicht und spart die Eingabe von teilweise bis zu sechs aufeinanderfolgenden Anführungszeichen.

Bild 5: Der einfache Vergleich akzeptiert keine Platzhalter.

Nach dem Auswerten des Ausdrucks entsteht etwa für den Vergleichswert **Chai** der folgende Vergleichsausdruck:

```
Artikelname = 'Chai'
```

Dies liefert beispielsweise genau den Datensatz zurück, dessen Feld **Artikelname** den Wert **Chai** enthält.

Platzhalter erlauben

Ein Wechsel in die Formularansicht und die Eingabe einiger Filterkriterien mit Platzhaltern zeigt, dass das Unterformular tatsächlich nur Datensätze für exakt übereinstimmende Vergleichsausdrücke liefert.

Das bedeutet, dass beispielsweise Platzhalter wirkungslos bleiben – siehe Bild 5.

Es wäre aber schon praktisch, wenn der Benutzer Platzhalter wie das Sternchen (*) einsetzen könnte. Dies lässt sich durch eine kleine Änderung erreichen.

Wir legen ein neues Textfeld namens **txtArtikelnameMitPlatzhalter** im Formularkopf an und erstellen auch für dieses eine Ereignisprozedur, die durch das Ereignis **Nach Aktualisierung** ausgelöst wird.

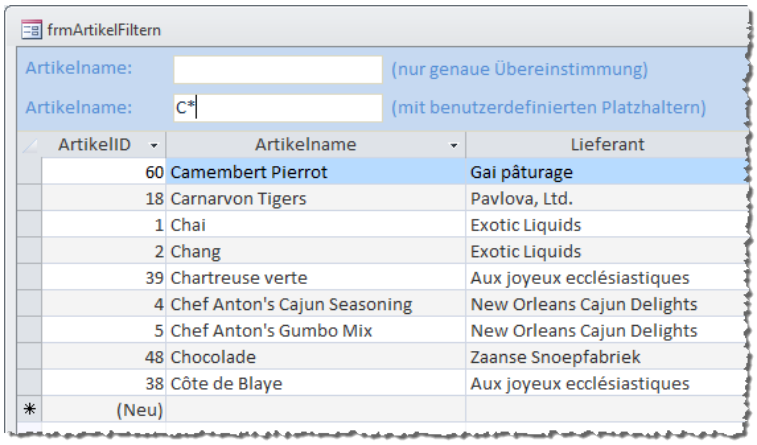


Bild 6: Mit Platzhaltern macht das Suchfeld mehr Sinn.

Textfelder: Value, OldValue und Text

Textfelder sind das Hauptwerkzeug zur Eingabe von Daten. Sie können diese an die Felder der Datenherkunft eines Formulars binden oder diese ungebunden einsetzen. So intuitiv Textfelder zu benutzen sind, soviele Möglichkeiten und Fallstricke bieten sie auch. Dieser Artikel zeigt, wie Sie zu verschiedenen Zeitpunkten auf den vorherigen Wert, den aktuellen Wert und den aktuell angezeigten Text eines gebundenen Steuerelements im Formular zugreifen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1304_TextfelderImGriff.mdb**.

Gebundene Textfelder

Wenn Sie ein Formular an eine Tabelle oder Abfrage binden, können Sie Textfeldern ganz einfach den Inhalt eines der Felder der Datenherkunft als Inhalt zuweisen. Dazu brauchen Sie lediglich die Eigenschaft Steuerelementinhalt auf den Namen des entsprechenden Feldes einzustellen.

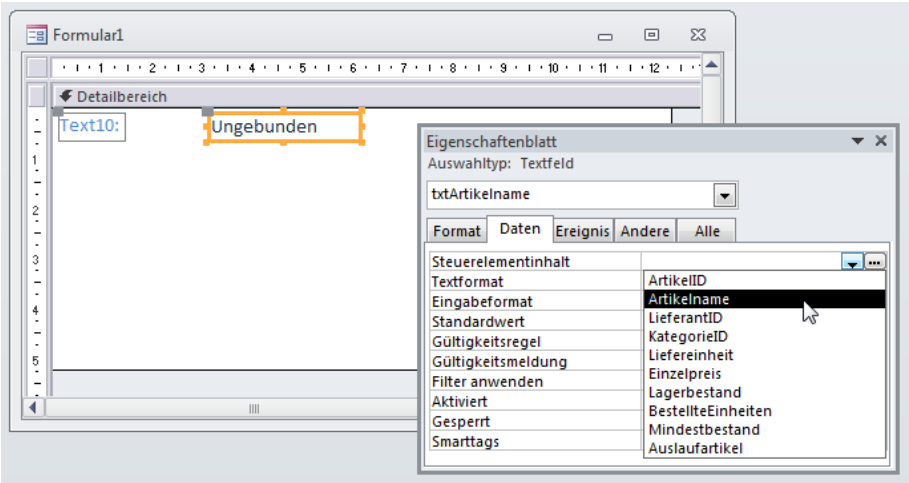


Bild 1: Zuweisen eines Feldes als Steuerelementinhalt

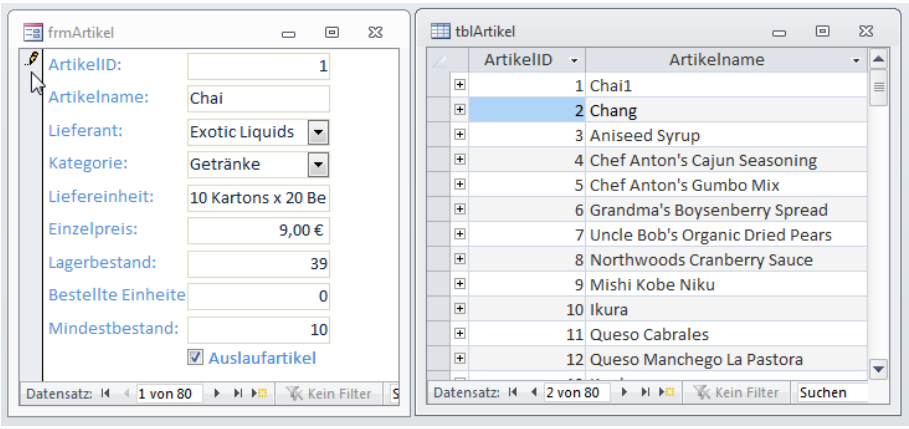


Bild 2: Wann wird der geänderte Wert gespeichert?

Wenn Sie der Eigenschaft **Datenherkunft** eines Formulars etwa die Tabelle **tblArtikel** zuweisen, können Sie ein Textfeld in den Detailbereich des Formularentwurfs ziehen und dann aus einer Liste eines der Felder der Datenherkunft auswählen (siehe Bild 1). Viel schneller geht dies natürlich, wenn Sie das entsprechende Feld direkt aus der Feldliste in den Entwurf des Formulars ziehen.

Was erreichen wir nun eigentlich durch die Bindung von Formular und Steuerelement an Tabelle/Abfrage und Feld? Zunächst einmal zeigt das Textfeld den Wert des Feldes für den aktuellen Datensatz der Datenherkunft des Formulars an. Die Bindung sorgt aber auch dafür, dass Änderungen am Inhalt zu einem bestimmten Zeitpunkt in die Tabelle übertragen werden. Zwischendurch geschehen jedoch noch eine Reihe anderer Dinge, die wir im Folgenden betrachten werden.

Speicherzeitpunkt

Interessant ist zum Beispiel die Frage, wann Access überhaupt die Änderung am Inhalt eines Textfeldes in der zugrunde liegenden Tabelle speichert. Dies lösen wir mithilfe der nebeneinander platzierten Objekte aus Bild 2 auf. Öffnen Sie das Formular **frmArtikel** und die Tabelle **tblArtikel** der Beispieldatenbank und ordnen Sie diese wie in der Abbildung an. Ändern Sie dann den Wert des Feldes **Artikelname**, tut sich nichts. Auch beim Verschieben des Fokus auf ein anderes Textfeld geschieht nichts. Erst wenn Sie den Datensatz speichern (beispielsweise durch einen Mausklick auf den Datensatzmarkierer mit dem Be-

arbeiten-Symbol oder durch die Auswahl eines anderen Datensatzes), wird die Änderung in die Tabelle übertragen.

Wo aber speichert Access die geänderten Daten des Textfeldes? Hier gibt es mehrere Eigenschaften, die Sie per VBA abfragen können:

- **Value**: Enthält den aktuellen Wert des Textfeldes.
- **Text**: Enthält den aktuell angezeigten Inhalt des Textfeldes.
- **OldValue**: Enthält den vorherigen Wert des Textfeldes.

Diese Informationen helfen noch nicht weiter. Was ist zum Beispiel der Unterschied zwischen dem aktuellen Wert (**Value**) und dem aktuell angezeigten Wert (**Text**)? Angenommen, das Feld **Artikelname** enthält beim Anzeigen des aktuellen Datensatzes den Wert **Chai**, dann liefern sowohl **Value** als auch **Text** den Wert **Chai**.

Wenn der Benutzer den Inhalt nun ändert, indem er diesen etwa auf **Chai (Tee)** erweitert, dann enthält **Text** jeweils den angezeigten Inhalt.

Solange der Fokus in diesem Textfeld verbleibt und der Datensatz auch nicht beispielsweise mit der Tastenkombination **Strg + S** gespeichert wird, bleibt in der Eigenschaft **Value** jedoch der Text **Chai** erhalten.

OldValue enthält übrigens auch den ursprünglichen Wert.

Sobald der Benutzer jedoch den Fokus auf ein anderes Steuerelement verschiebt, ohne den Datensatz etwa durch einen Datensatzwechsel zu speichern, ändern sich die Werte der Eigenschaften: **Chai (Tee)** wird nun in die Eigenschaft **Value** eingetragen. Text enthält den gleichen Wert, und **OldValue** liefert den zuvor gültigen Wert **Chai**.

Achtung: Auch wenn Sie den Inhalt des Textfeldes mehrfach ändern, ohne den Datensatz zu speichern, enthält **OldValue** immer den zuletzt gespeicherten Wert, nicht den Wert vor der letzten Änderung.

Wenn Sie dies zu verschiedenen Zeitpunkten prüfen möchten, legen Sie im VBA-Editor ein neues Standardmodul etwa namens **mdlBeispiele** an und fügen Sie die folgende Prozedur hinzu:

```
Public Sub Textfeldeigenschaften()
    With Forms!frmArtikel!Artikelname
        Debug.Print "Value:    " & .Value
        Debug.Print "Text:     " & .Text
        Debug.Print "OldValue: " & .OldValue
    End With
End Sub
```

Diese gibt die jeweils aktuellen Werte aus. Aber Vorsicht: Wenn Sie zu einem anderen Steuerelement wechseln, löst die Prozedur einen Fehler aus – die **Text**-Eigenschaft können Sie nur abfragen, wenn das zu prüfende Steuerelement den Fokus besitzt.

Wenn Sie den Datensatz speichern (durch den Wechsel zu einem anderen Datensatz, mit **Strg + S** oder durch das Schließen des Formulars), werden alle drei Eigenschaften mit dem in der Tabelle befindlichen Wert aktualisiert.

Value als Standardeigenschaft

Die **Value**-Eigenschaft ist die Standardeigenschaft von Textfeldern und kann somit auch ohne Angabe der Eigenschaft abgefragt werden.

Die folgenden beiden Zeilen liefern, abgesetzt im Direktfenster des VBA-Editors, folglich die gleichen Werte:

```
? Forms!frmArtikel!Artikelname.Value
Chai (Tee)

? Forms!frmArtikel!Artikelname
Chai (Tee)
```

Value, OldValue und Text im zeitlichen Ablauf

Nun schauen wir uns noch anhand eines praktischen Beispiels an, wie die drei Eigenschaften sich in Abhängigkeit der verschiedenen Zustände von Steuerelement und Formular verhalten.

Dazu haben wir das Formular **frmArtikel** etwas erweitert. Das Textfeld **Artikelname** wurde in **txtArtikelname** umbenannt, außerdem haben wir die drei Textfelder **txtOldValue**, **txtValue** und **txtText** hinzugefügt. Diese sollen, nach dem Eintreten bestimmter Ereignisse, den aktuellen Wert der drei Eigenschaften ausgeben.

Diese Ereignisse sind die folgenden:

Änderungen protokollieren, Teil I

Textfelder sind das Hauptwerkzeug zur Eingabe von Daten. Sie können diese an die Felder der Datenherkunft eines Formulars binden oder diese ungebunden einsetzen. So intuitiv Textfelder zu benutzen sind, soviele Möglichkeiten und Fallstricke bieten sie auch. Dieser Artikel zeigt, was alles bei der Eingabe von Daten mit Textfeldern geschieht und wann welche Texte wo gespeichert werden.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1304_AenderungenProtokollieren.mdb**.

Warum protokollieren?

Änderungen an Datensätzen sind in der Regel unumkehrbar, wenn der Datensatz einmal gespeichert ist. Solange dies noch nicht geschehen ist und Sie einfach nur die Werte eines oder mehrerer Felder aktualisiert haben, lässt sich dies noch rückgängig machen – beispielsweise durch Betätigen der **Escape**-Taste.

Anderenfalls können Sie nach einer Fehleingabe nur hoffen, dass Sie den vorherigen Zustand des Datensatzes wiederherstellen können – oder dass eine Sicherungskopie noch den gewünschten Stand enthält. Wenn Sie dem vorbeugen möchten, können Sie die Änderungen an Datensätzen auch protokollieren. Dabei gibt es zwei Varianten:

- Sie speichern die Änderung eines jeden Feldes als eigenen Datensatz in einer Tabelle, welche den Namen der betroffenen Tabelle und des Feldes, den alten und den neuen Wert, den betroffenen Datensatz sowie das Änderungsdatum und gegebenenfalls den ausführenden Benutzer speichert.

- Sie legen für jede Tabelle, für die Sie Änderungen protokollieren möchten, eine Kopie an. Vor einer Änderung übertragen Sie den vorherigen Zustand in diese Tabelle – auch hier wieder mit dem Datum der Änderung und dem Namen der ausführenden Person.

In diesem Artikel schauen wir uns beide Varianten an.

Formularereignisse als Ausgangspunkt

Während es etwa unter dem SQL Server Trigger gibt, denen Sie Code zum Protokollieren von Änderungen zuweisen können, müssen Sie in reinen Access-Datenbanken (also solchen, die ihre Daten wiederum in einer Access-Datenbank speichern – egal, ob in einer einzigen Datei oder aufgeteilt in Frontend und Backend) entsprechende Ereignisprozeduren zu den Formularen hinzufügen, in welchen die Änderungen durchgeführt werden.

In den meisten Fällen werden Sie die Änderungen beim Speichern des Datensatzes protokollieren. Der richtige Zeitpunkt ist das Ereignis **Vor Aktualisierung** des Formulars. Es wird unmittelbar vor dem Speichern ausgelöst – und vor dem Ereignis **Nach Aktualisierung**.

Der Vorteil des Ereignisses **Vor Aktualisierung** ist: Sie können zu diesem Zeitpunkt sowohl auf den vorherigen als auch auf den aktuellen Wert eines Feldes der Datenherkunft zugreifen – und zwar über die Eigenschaften **OldValue** und **Value** (weitere Informationen hierzu finden Sie im Artikel **Textfelder: Value, OldValue und Text**.

Was soll protokolliert werden?

Die Protokollierung soll so erfolgen, dass alle zur Kontrolle und sogar zur Wiederherstellung der geänderten Daten notwendigen Informationen gespeichert weden. Dazu legen Sie eine Tabelle namens **tblAenderungen** an, die im Entwurf wie in Bild 1 aussieht.

Dabei kommen die folgenden Felder zum Einsatz:

- **AenderungID**: Primärschlüsselfeld der Tabelle

- **Tabelle**: Name der Tabelle, deren Datensatz geändert wurde

- **Aktion**: Wir wollen nicht nur Änderungen, sondern auch das Anlegen oder Löschen von Datensätzen dokumentieren. Dieses Feld soll die Werte **New**, **Edit** und **Delete** aufnehmen.

- **Feld**: Name des geänderten Feldes

- **PKName**: Name des Primärschlüsselfeldes der betroffenen Tabelle
- **PKWert**: Wert des Primärschlüsselfeldes des geänderten Datenatzes
- **Zeit**: Datum und Uhrzeit der Änderung
- **Benutzer**: Aktueller Benutzer zum Zeitpunkt der Änderung
- **AlterWert**: Wert des Feldes vor der Änderung
- **NeuerWert**: Wert des Feldes nach der Änderung

Diese Art der Protokollierung erzeugt natürlich eine Menge Daten: Es wird tatsächlich zu jedem geänderten Feld ein eigener Datensatz in der Tabelle **tblAenderungen** angelegt.

Wenn wir nun im Formular **frmArtikel** beispielsweise den Wert des Feldes **Artikelname** von **Chai** in **Chai (Tee)** ändern, soll die Tabelle einen Datensatz wie in Bild 2 erhalten.

Nun benötigen wir nur noch eine Ereignisprozedur, die genau dies erledigt.

Änderung in Tabelle schreiben

Wir gehen zunächst vereinfachend davon aus, dass wir nur Änderungen am Feld **Artikelname** in der Tabelle speichern möchten. Dazu legen Sie für das Ereignis **Vor Aktualisierung** des Formulars die folgende Ereignisprozedur an:

```
Private Sub Form_BeforeUpdate(Cancel As Integer)
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM ↵
        tblAenderungen WHERE 1 = 2", dbOpenDynaset)
    If Not Me!txtArtikelname.Value = _
            Me!txtArtikelname.OldValue Then
```

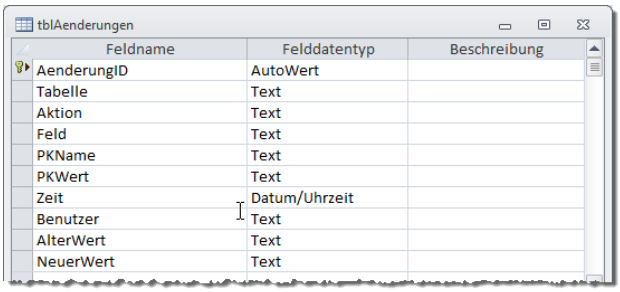


Bild 1: Die Tabelle **tblAenderungen** in der Entwurfsansicht

```
        rst.AddNew
        rst!Tabelle = "tblArtikel"
        rst!Aktion = "Edit"
        rst!Feld = "Artikelname"
        rst!PKName = "ArtikelID"
        rst!PKWert = Me!ArtikelID
        rst!Zeit = Now
        rst!Benutzer = CurrentUser
        rst!AlterWert = Me!txtArtikelname.OldValue
        rst!NeuerWert = Me!txtArtikelname.Value
        rst.Update
    End If
    Set db = Nothing
End Sub
```

Diese Ereignisprozedur erstellt ein Recordset auf Basis der Tabelle **tblAenderungen**, das allerdings wegen des Kriteriums (**1=2**) keinen einzigen Datensatz enthält. Dies ist auch nicht nötig, denn wir wollen ja nur einen neuen Datensatz hinzufügen.

Der Übersichtlichkeit halber legen wir diesen Datensatz zunächst mit der **AddNew**-Methode an und weisen alle Feldwerte einzeln zu. Die meisten Zuweisungen sind selbsterklärend. Die Zeit wird mit der Funktion **Now** ermittelt, der aktuelle Access-Benutzer mit **CurrentUser** (hier könnte man noch den aktuellen Windows-Benutzer referenzieren – zumindest da Access seit der Version 2007 das Sicherheitssystem nicht mehr unterstützt). Den alten und den neuen Wert des gebundenen Steuerelements erhalten wir mit den Eigenschaften **OldValue** und **Value**.

Bild 2: Die Tabelle **tblAenderungen** mit einer ersten Änderung

Berichte per E-Mail verschicken, Teil I

Wenn Sie Berichte mit Inhalten wie Rechnungen, Lieferscheinen et cetera per E-Mail versenden möchten, brauchen Sie zweierlei: erstens die Umwandlung des Berichts in ein Format, das jeder Benutzer lesen kann – hier kommt eigentlich nur PDF in Frage. Und zweitens einen Automatismus, der diesen Bericht im PDF-Format an eine neue E-Mail anhängt, diese mit den Daten wie Empfänger, Betreff und Inhalt versieht und diese dann auch noch absendet. Dieser Artikel zeigt, wie es funktioniert.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1304_BerichtePerEMail.mdb**.

Vorbereitungen

Wir nutzen in diesem Artikel die ab Access 2007 angebotenen Techniken zum Erstellen von PDF-Dokumenten. Wenn Sie die Lösung für Access 2003 oder älter anpassen möchten, finden Sie die entsprechenden Informationen im Artikel **Bericht als PDF speichern** (siehe **Access [basics]** 3/2012).

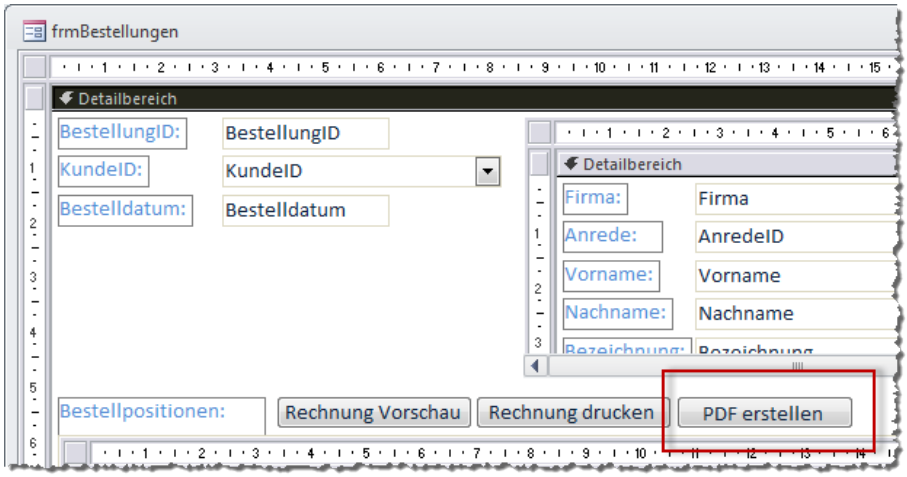


Bild 1: Neue Schaltfläche zum Erstellen eines PDF-Dokuments auf Basis der Rechnung

Die Beispieldatenbank baut auf der Datenbank zum Artikel **Rechnungsbericht** aus Ausgabe 6/2012 auf.

Rechnung als PDF speichern

Ausgangspunkt für die Erstellung eines PDF-Dokuments ist in unserem Beispiel das Formular **frmBestellung**, und hier speziell die neu angelegte Schaltfläche **cmdPDFErstellen** (siehe Bild 1). Diese Schaltfläche soll genau die Rechnung für die aktuell im Formular angezeigte Bestellung erzeugen.

Für die Schaltfläche hinterlegen Sie eine Ereignisprozedur, die durch das Ereignis Beim Klicken ausgelöst wird. Wählen Sie dazu im Eigenschaftsfenster für die Eigenschaft **Beim Klicken** den Eintrag **[Ereignisprozedur]** aus und klicken Sie auf die Schaltfläche mit den drei Punkten (...). Die nun im VBA-Editor erscheinende Prozedur ergänzen Sie wie in Listing 1.

Die Prozedur erledigt die folgenden Schritte:

- Zusammenstellen eines Dateinamens nach dem Schema **Rechnung_<Nummer>.pdf**
- Öffnen der Rechnung für den aktuellen Bestelldatensatz. Dazu wird als **WhereCondition** ein Ausdruck wie **BestellungID = <Nummer>** übergeben.
- Die nun geöffnete Rechnung wird mit der **OutputTo**-Methode des **DoCmd**-Objekts als PDF-Dokument gespeichert.
- Schließlich wird der Bericht wieder geschlossen.

```
Private Sub cmdPDFErstellen_Click()
    Dim strRechnung As String
    strRechnung = "Rechnung_" & Me.BestellungID & ".pdf"
    DoCmd.OpenReport "rptRechnung", WhereCondition:="BestellungID = " & Me!BestellungID, View:=acViewPreview
    DoCmd.OutputTo acOutputReport, "rptRechnung", acFormatPDF, CurrentProject.Path & "\" & strRechnung
    DoCmd.Close acReport, "rptRechnung"
End Sub
```

Listing 1: Erstellen eines PDF-Dokuments per Mausklick

Die **DoCmd.OutputTo**-Methode ist einen genaueren Blick wert. Sie erwartet die folgenden Parameter:

- **ObjectType**: Art des zu exportierenden Objekts, hier **acOutputReport** für einen Bericht
- **ObjectName**: Name des zu exportierenden Objekts, hier der Name des Berichts (**rptRechnung**).
- **OutputFormat**: Format, in dem das Objekt gespeichert werden soll – hier **acFormatPDF**. Die verschiedenen Konstanten können Sie im Objektkatalog einsehen (einfach im VBA-Editor auf **F2** drücken – Ergebnis siehe Bild 2).
- **OutputFile**: Pfad der zu erstellenden PDF-Datei. Wird hier aus dem aktuellen Datenbankverzeichnis (**CurrentProject.Path**) und dem in **strRechnung** enthaltenen Dateinamen zusammengesetzt.

PDF-Dokument nach dem Erstellen öffnen

Wenn Sie das PDF-Dokument gleich nach der Erzeugung in Ihrem PDF-Reader anzeigen möchten, fügen Sie der Prozedur noch die folgende Anweisung hinzu:

```
Application.FollowHyperlink CurrentProject.Path &
"\" & strRechnung
```

Das PDF-Dokument wird dann wie in Bild 3 angezeigt – hier im **Adobe Reader**.

Speicherort und -zeit sichern

Wenn Sie eine Rechnung einmal erstellt haben, sollten Sie die der Rechnung zugrunde liegenden Daten nicht mehr ändern. Dazu legen wir in der Tabelle **tblBestellungen** neben dem Feld **Rechnungsdatum** noch ein Feld namens **Rechnungsdatei** an (siehe Bild 4).

Den Code der Prozedur **cmdPDFErstellen_Click** erweitern Sie unten um die folgenden beiden Zeilen:

```
Me!Rechnungsdatum = Now
Me!Rechnungsdatei = CurrentProject.Path & "\" _
    & strRechnung
```

Diese beiden Zeilen tragen den aktuellen Zeitpunkt und den Dateinamen in die Felder **Rechnungsdatum** und **Rechnungsdatei** des aktuell im Formular angezeigten Datensatzes ein (siehe Bild 5).

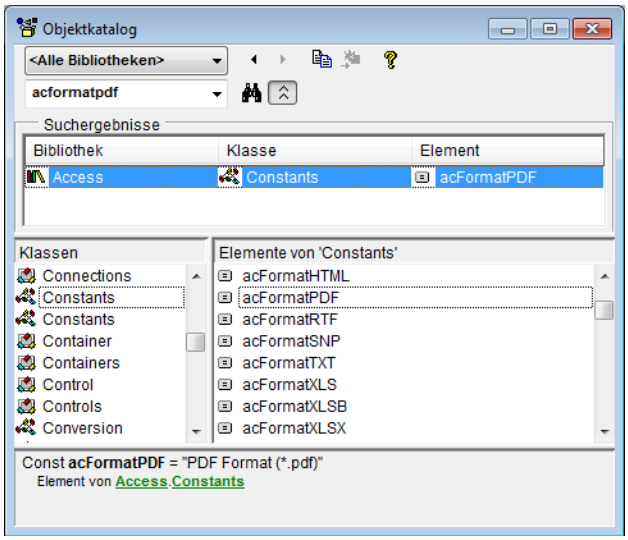


Bild 2: Neue Schaltfläche zum Erstellen eines PDF-Dokuments auf Basis der Rechnung

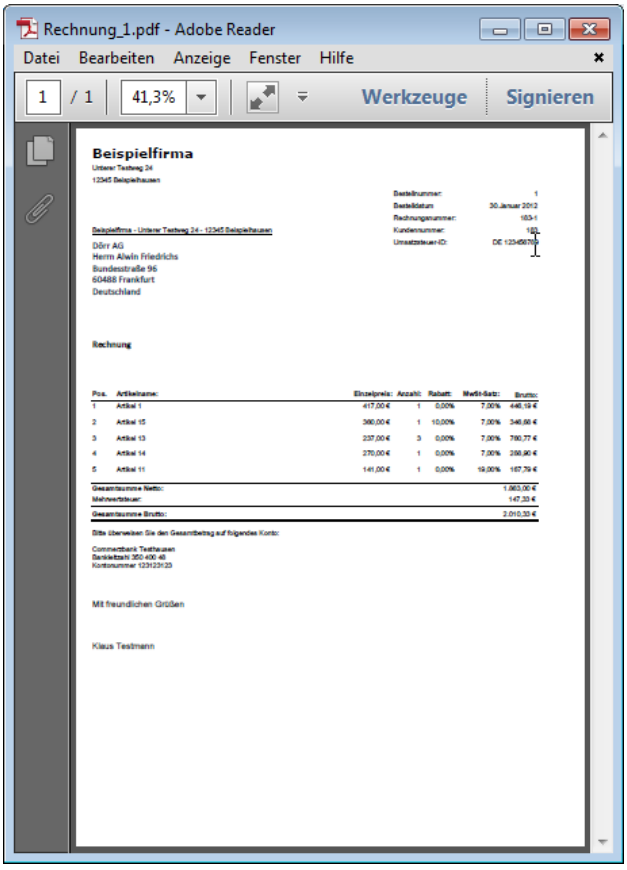


Bild 3: Ein Access-Bericht im PDF-Format

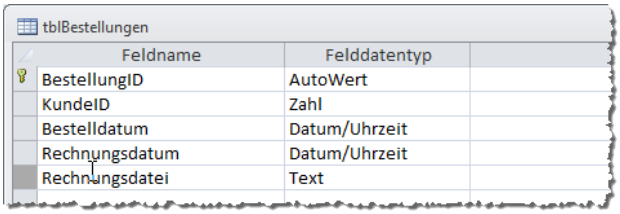


Bild 4: Feld zum Speichern der Rechnungsdatei

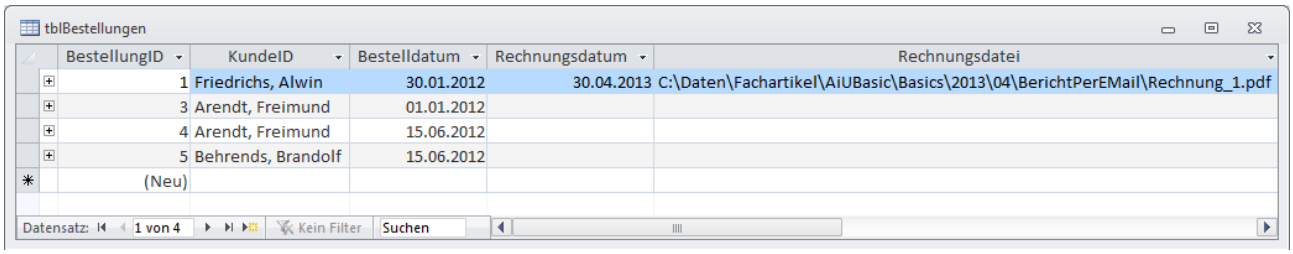


Bild 5: Bestellung mit vollständigen Rechnungsdaten

Vorhandene Rechnungen anzeigen

Mit einer weiteren Schaltfläche sollen bereits gespeicherte Rechnungen im PDF-Format angezeigt werden können. Diese Schaltfläche heißt **cmdPDFAnzeigen** und löst beim Klicken die folgende Prozedur aus:

```
Private Sub cmdPDFAnzeigen_Click()
    Application.FollowHyperlink Me!rechnungsdatei
End Sub
```

Schaltflächen ein- und ausblenden

Beim Anzeigen einer Rechnung soll das Formular die beiden Schaltflächen **cmdPDFErstellen und cmdPDFAnzeigen** aktivieren beziehungsweise deaktivieren – je nachdem, ob die PDF-Datei bereits erstellt wurde.

Da wir diesen Vorgang gleich noch von einer anderen Stelle aus anstoßen müssen, gliedern wir die notwendigen Anweisungen in eine eigene Prozedur aus:

```
Private Sub SchaltflaechenAktivieren()
    If Len(Nz(Me!Rechnungsdatei, "")) > 0 Then
        Me!cmdPDFAnzeigen.Enabled = True
        Me!cmdPDFErstellen.Enabled = False
    Else
        Me!cmdPDFAnzeigen.Enabled = False
        Me!cmdPDFErstellen.Enabled = True
    End If
End Sub
```

Die Prozedur prüft, ob das Feld **Rechnungsdatei** bereits einen Dateinamen enthält und aktiviert die Schaltflächen entsprechend durch Einstellen der Eigenschaft **Enabled** auf die Werte **True** oder **False**.

Diese Prozedur soll beispielsweise beim Anzeigen eines jeden Datensatzes ausgelöst werden. Dazu erweitern Sie die bereits vorhandene Ereignisprozedur **Form_Current** (im Eigenschaftsfenster Beim Anzeigen) um eine Zeile:

```
Private Sub Form_Current()
    ...
    SchaltflaechenAktivieren
End Sub
```

Wann sollen die Schaltflächen noch angepasst werden? Natürlich direkt beim Speichern des Berichts als PDF-Dokument. Also fügen Sie auch der Prozedur **cmdPDFErstellen** einen entsprechenden Aufruf hinzu:

```
Private Sub cmdPDFErstellen_Click()
    ...
    SchaltflaechenAktivieren
End Sub
```

Bearbeiten nach Rechnungserstellung verhindern

Wenn die Rechnung einmal im PDF-Format erstellt wurde, soll die Bearbeitung der Bestelldaten nicht mehr möglich sein. Dies können Sie ebenfalls in der Prozedur **SchaltflaechenAktivieren** erledigen – und zwar durch Einstellen der Eigenschaft **AllowEdits** auf den Wert **True** beziehungsweise **False**.

Und dies möglichst nicht nur für das aktuelle Formular (**Me**), sondern auch für die Unterformulare (**Me!sfmKundendaten.Form** und **Me!sfmBestellpositionen.Form**):

```
Private Sub SchaltflaechenAktivieren()
    If Len(Nz(Me!Rechnungsdatei, "")) > 0 Then
        Me!cmdPDFAnzeigen.Enabled = True
        Me!cmdPDFErstellen.Enabled = False
        Me.AllowEdits = False
        Me!sfmBestellpositionen.Form.AllowEdits = False
        Me!sfmKundendaten.Form.AllowEdits = False
    Else
        Me!cmdPDFAnzeigen.Enabled = False
        Me!cmdPDFErstellen.Enabled = True
        Me.AllowEdits = True
        Me!sfmBestellpositionen.Form.AllowEdits = True
        Me!sfmKundendaten.Form.AllowEdits = True
```

Filterkriterien für Formulare, Teil II

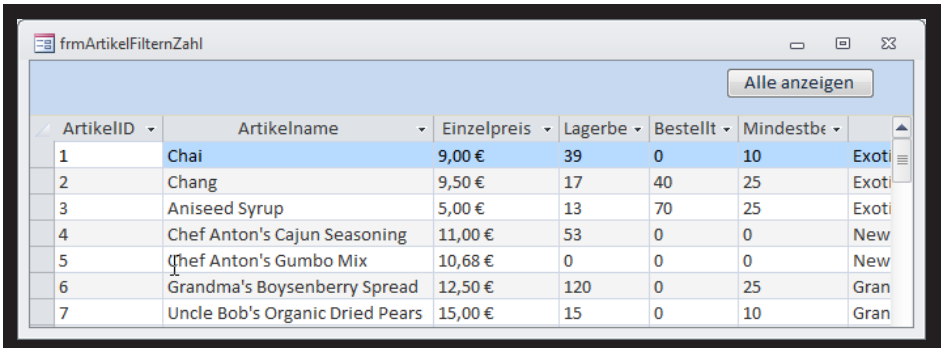
Was helfen Artikel-, Kunden-, Bestell- und sonstige Daten, wenn Sie diese zwar schön in einem Formular darstellen, aber diese nicht entsprechend filtern können? In diesem Artikel zeigen wir Ihnen, wie Sie verschiedene individuelle Filterkriterien mithilfe entsprechender Steuerelemente im Formular festlegen können. Damit filtern Sie dann die in einem Unterformular enthaltenen Daten – in diesem Fall Artikelstammdaten.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1305_FilterkriterienFuerFormulare.mdb**.

Filtern nach Zahlenfeldern

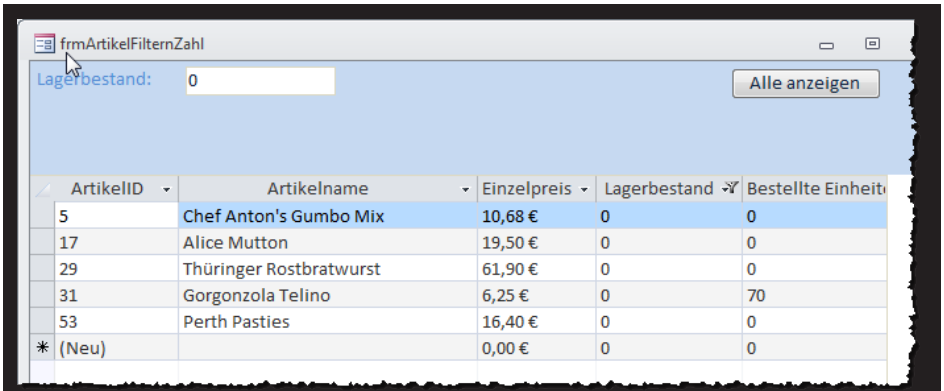
Für das Filtern nach Zahlen haben wir der Übersichtlichkeit halber ein neues Formular samt Unterformular angelegt. Diese heißen **frmArtikelFilternZahl** und **sfmArtikelFilternZahl**.



ArtikelID	Artikelname	Einzelpreis	Lagerbe	Bestellt	Mindestbe	
1	Chai	9,00 €	39	0	10	Exoti
2	Chang	9,50 €	17	40	25	Exoti
3	Aniseed Syrup	5,00 €	13	70	25	Exoti
4	Chef Anton's Cajun Seasoning	11,00 €	53	0	0	New
5	Chef Anton's Gumbo Mix	10,68 €	0	0	0	New
6	Grandma's Boysenberry Spread	12,50 €	120	0	25	Gran
7	Uncle Bob's Organic Dried Pears	15,00 €	15	0	10	Gran

Bild 1: Neues Formular für die Beispiele zum Filtern von Datensätzen nach Zahlenwerten

Die Schaltfläche zum Leeren des Filters haben wir gegenüber dem vorherigen Beispiel beibehalten (siehe Bild 1). Lediglich die dadurch ausgelöste Prozedur referenziert hier ein anderes Unterformular (siehe Listing 1).



ArtikelID	Artikelname	Einzelpreis	Lagerbestand	Bestellte Einheit
5	Chef Anton's Gumbo Mix	10,68 €	0	0
17	Alice Mutton	19,50 €	0	0
29	Thüringer Rostbratwurst	61,90 €	0	0
31	Gorgonzola Telino	6,25 €	0	70
53	Perth Pasties	16,40 €	0	0
* (Neu)		0,00 €	0	0

Bild 2: Filtern nach dem Lagerbestand

Das erste Suchfeld soll den Inhalt des Feldes **Lagerbestand** untersuchen und genau passende Datensätze zurückliefern. Wenn der Benutzer also beispielsweise den Wert **0** einträgt und die Eingabetaste betätigt, sollen nur die aufgebrauchten Artikel erscheinen (siehe **Bild 2**). Dies erledigen Sie mit der Ereignisprozedur aus Listing 2.

```
Private Sub cmdAlleAnzeigen_Click()
    Me!sfmArtikelFilternZahl.Form.Filter = „“
End Sub
```

Listing 1: Filter leeren

Zahlenbereiche untersuchen

Im zweiten Beispiel wollen wir alle Artikel finden, deren Einzelpreis in einem bestimmten Bereich liegt – also etwa über 10 Euro, unter 20 Euro oder zwischen 5 und 15 Euro.

Dazu richten Sie zwei Textfelder namens **txtPreisMin** und **txtPreisMax** ein.

Außerdem soll der Benutzer die Suche durch einen Mausklick auf eine Schaltfläche starten können –

```
Private Sub txtLagerbestandSuche_AfterUpdate()
    Me!sfmArtikelFilternZahl.Form.Filter = _
        "Lagerbestand = " & Me!txtLagerbestandSuche
    Me!sfmArtikelFilternZahl.Form.FilterOn = True
End Sub
```

Listing 2: Nach einem bestimmten Lagerbestand suchen

diese heißt **cmdPreisbereiche** (siehe Bild 3).

Die Schaltfläche erhält eine Ereignisprozedur, die zunächst nur vollständige Angaben verarbeiten soll, also die Angabe sowohl des minimalen als auch des maximalen Preises.

Die Ereignisprozedur sieht wie in Listing 3 aus. Sie setzt zunächst ein Filterkriterium zusammen, dass für die Angaben aus Bild 4 etwa so aussieht:

```
Einzelpreis BETWEEN 10 AND 12
```

Am Ergebnis können Sie gleich ablesen, dass **BETWEEN 10 AND 12** auch alle Datensätze liefert, deren Wert im Feld **Einzelpreis** genau **10** oder **12** beträgt.

Nun wäre es noch praktisch, wenn der Benutzer nur das erste Textfeld füllen könnte, wenn er alle Artikel mit Einzelpreisen über dem angegebenen Wert erhalten möchte – ohne Obergrenze. Oder er nutzt das zweite Textfeld, um nur die Obergrenze zu definieren. Wenn Sie dies mit der vorhandenen Prozedur ausprobieren, erhalten Sie die Fehlermeldung aus Bild 5.

Dem schaffen wir durch die Änderung aus Listing 4 bei der Zusammenstellung des Filterkriteriums Abhilfe.

Die erste **If...Then**-Bedingung prüft, ob **txtPreisMin** einen Wert enthält. In diesem Fall trägt Sie einen Ausdruck wie **AND Einzelpreis >= 10** in die Variable **strFilter** ein.

Beim Textfeld **txtPreisMax** läuft es ähnlich. Der Ausdruck (etwa **AND Einzelpreis <= 20**) wird allerdings an den bereits bestehenden Ausdruck in der Variablen **strFilter** angehängt. Auf diese Weise kann **strFilter** nun

```
Private Sub cmdPreisbereiche_Click()
    Dim strFilter As String
    strFilter = "Einzelpreis BETWEEN " & Me!txtPreisMin & " AND " _
        & Me!txtPreisMax
    Me!sfmArtikelFilternZahl.Form.Filter = strFilter
    Me!sfmArtikelFilternZahl.Form.FilterOn = True
End Sub
```

Listing 3: Nach einem Preisbereich filtern

```
Private Sub cmdPreisbereiche_Click()
    Dim strFilter As String
    If Not IsNull(Me!txtPreisMin) Then
        strFilter = " AND Einzelpreis >= " & Me!txtPreisMin
    End If
    If Not IsNull(Me!txtPreisMax) Then
        strFilter = strFilter _
            & " AND Einzelpreis <= " & Me!txtPreisMax
    End If
    If Len(strFilter) > 0 Then
        strFilter = Mid(strFilter, 5)
    End If
    Me!sfmArtikelFilternZahl.Form.Filter = strFilter
    Me!sfmArtikelFilternZahl.Form.FilterOn = True
End Sub
```

Listing 4: Nach Preisbereichen filtern, detaillierte Fassung

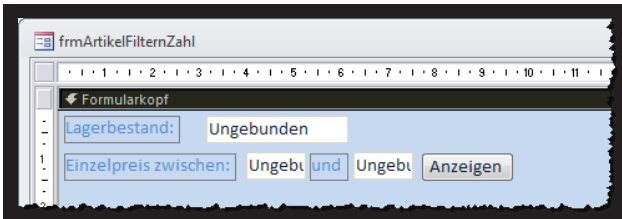


Bild 3: Filtern nach Mindest- und Höchstpreis

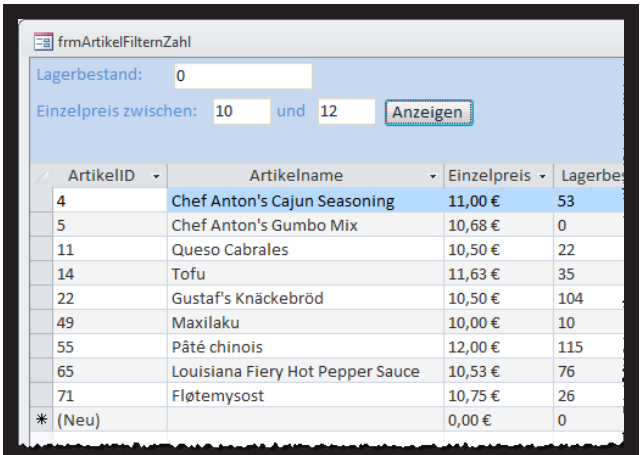


Bild 4: Preisfilter im Einsatz

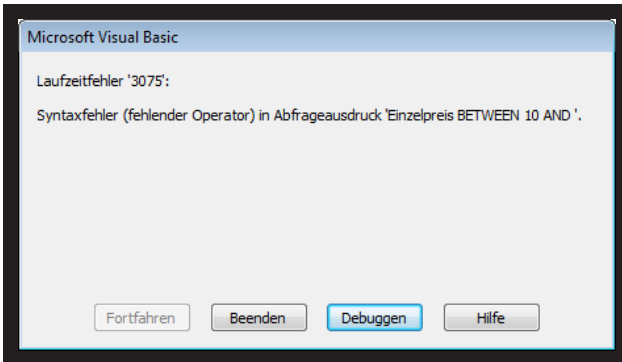


Bild 5: Fehler beim Auslassen eines Parameters

beispielsweise folgende Ausdrücke enthalten:

- Nur **txtPreisMin** gefüllt: **AND Einzelpreis >= 10**
- Nur **txtPreisMax** gefüllt: **AND Einzelpreis <= 12**
- Beide gefüllt: **AND Einzelpreis >= 10 AND Einzelpreis <= 12**

Der Ausdruck beginnt also immer mit dem Schlüsselwort **AND**, was bei direkter Zuweisung einen Fehler auslösen würde. Nur wenn der Benutzer keines der beiden Felder füllt, bleibt die Variable **strFilter** komplett leer.

Enthält **strFilter** jedoch einen Wert, soll das führende **AND** entfernt werden.

Dazu prüft die Prozedur, ob die Länge von **strFilter** größer als **0** ist und übernimmt den Inhalt von **strFilter** ab dem fünften Zeichen:

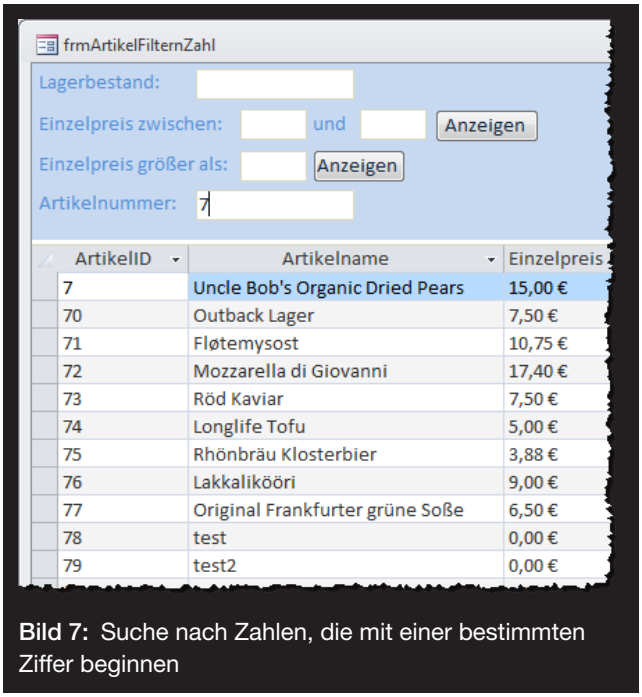


Bild 7: Suche nach Zahlen, die mit einer bestimmten Ziffer beginnen

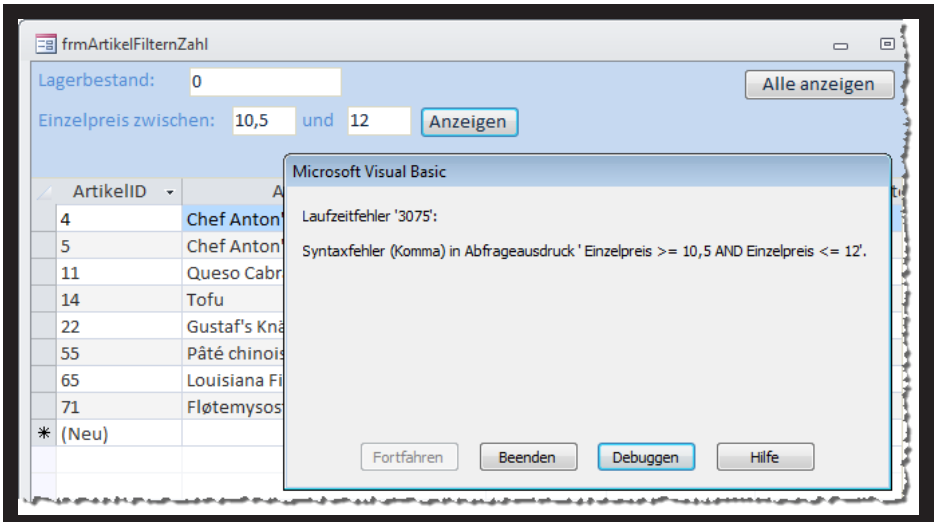


Bild 6: Fehler bei falschem Dezimaltrennzeichen

```
If Len(strFilter) > 0 Then
    strFilter = Mid(strFilter, 5)
End If
```

Der resultierende Filterausdruck sieht dann etwa so aus:

```
Einzelpreis >= 10 AND Einzelpreis <= 12
```

Dies entspricht prinzipiell dem weiter oben verwendeten **IN...BETWEEN...**-Ausdruck.

Behandlung von Dezimalzahlen

Wenn der Benutzer eine Zahl mit dem in Deutschland üblichen Dezimaltrennzeichen eingibt – also dem Komma –, liefert unsere Prozedur einen Fehler (siehe Bild 6). Dies liegt daran, dass ein Ausdruck wie **10,5** direkt in das SQL-Filterkriterium übertragen wird.

Dieses sieht dann wie folgt aus:

```
Einzelpreis >= 10,5
```

In SQL-Ausdrücken wird das Komma jedoch als Aufzählungszeicheninterpretiert, was den entsprechenden Fehler auslöst.

Damit der Ausdruck funktioniert, müssen wir das Komma durch einen Punkt ersetzen. Dazu verwenden wir die **Replace**-Funktion wie in der Prozedur aus Listing 5.

```
Private Sub cmdPreisMitKomma_Click()
    Dim strFilter As String
    strFilter = "Einzelpreis >= " _
        & Replace(Me!txtPreisMitKomma, ",", ".")
    Me!sfmArtikelFilternZahl.Form.Filter = strFilter
    Me!sfmArtikelFilternZahl.Form.FilterOn = True
End Sub
```

Listing 5: Nach größeren oder gleichen Preisen suchen

Die **Replace**-Funktion nimmt sich den Inhalt des Textfeldes vor und ersetzt alle Vorkommen des Komma-Zeichens durch einen Punkt.

Zahlenfelder wie Texte filtern

Gelegentlich möchte man in Zahlen genauso wie in Texten suchen – beispielsweise um alle Artikelnummern zu ermitteln, die mit einer bestimmten Zahl beginnen. Dies sieht im Ergebnis beispielsweise wie in Bild 7 aus.

Gibt der Benutzer etwa den Suchausdruck **7** ein, soll das Datenblatt alle Artikel anzeigen, deren Artikelnummer mit **7** beginnen.

Dies erreichen mit dem Textfeld **txtArtikelnummer** und dem Ereignis **Bei Änderung**, für das Sie die Ereignisprozedur aus Listing 6 hinterlegen.

Die Prozedur ermittelt bei jeder Änderung des Inhalts mit der Eigenschaft **Text** den aktuell im Textfeld angezeigten Ausdruck und stellt ein entsprechendes Kriterium zusammen.

Damit alle Artikel ausgegeben werden, deren Artikelnummer mit dem Suchkriterium beginnt, enthält der Suchausdruck noch das Sternchen (*) als Platzhalter. Der Suchausdruck sieht beispielsweise so aus:

```
ArtikelID LIKE '7*'
```

Filtern nach Datumsfeldern

Wenn Sie ein Formular nach dem Wert eines Datumsfeldes filtern möchten, müssen Sie vor allem die Speicherung von Datumsfeldern in Access kennen. Datumsangaben werden hier nämlich intern als Double-Werte gespeichert.

Die Zahl vor dem Komma entspricht dabei der Anzahl der Tage seit dem 31.12.1899, die Zahl hinter dem Komma der Uhrzeit.

Dabei entspricht etwa 0 der Uhrzeit 0 Uhr, 0,5 entspricht 12 Uhr und 0,75 der Uhrzeit 18 Uhr.

Wenn Sie nun eine Datumsangabe wie 1.1.2013, 12:00 Uhr mit dem Datum 1.1.2013 vergleichen, liefert dies keine Übereinstimmung, denn: Der 1.1.2013 um 12:00 hat den Wert 41275,5, der 1.1.2013 ohne Angabe einer Uhrzeit hat den Wert 41275.

Wenn Sie also alle Datumsangaben erhalten möchten, die auf einen bestimmten Tag fallen, muss dieser Wert größer gleich dem Datum ohne Uhrzeit und kleiner als das Datum des folgenden Tages sein – zum Beispiel so:

```
1.1.2013 <= x < 2.1.2013
```

x entspricht dann in jedem Fall dem Datum 1.1.2013 – egal, ob es noch eine Uhrzeit beziehungsweise eine oder mehrere Nachkommastellen enthält oder nicht.

```
Private Sub txtArtikelnummer_Change()
    Dim strFilter As String
    strFilter = "ArtikelID LIKE '" _
        & Me!txtArtikelnummer.Text & "*'"
    Me!sfmArtikelFilternZahl.Form.Filter = strFilter
    Me!sfmArtikelFilternZahl.Form.FilterOn = True
End Sub
```

Listing 6: Filter nach Ändern des Vergleichswertes anpassen

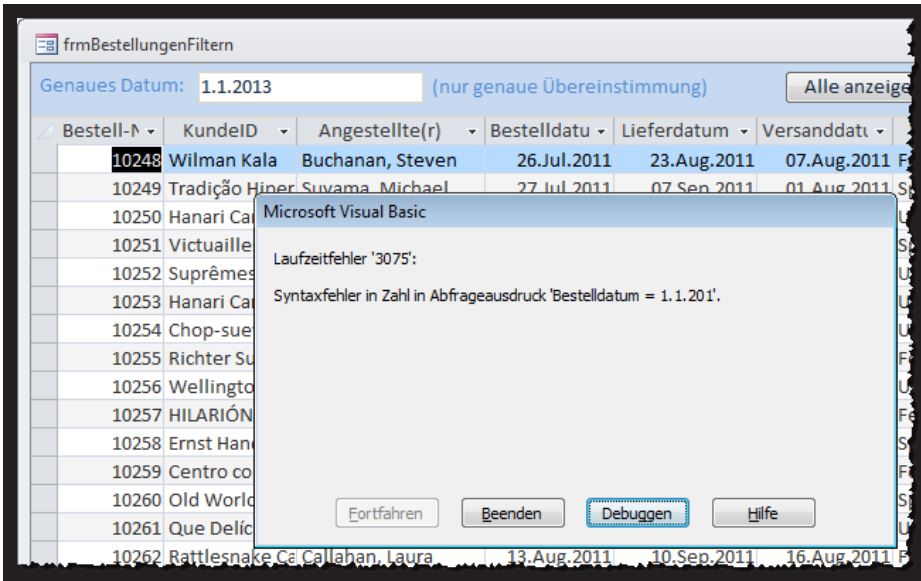


Bild 8: Syntaxfehler bei der einfachen Datumsübergabe

E-Mails mit Outlook versenden

Microsoft Outlook ist für viele Access-Nutzer das Mittel der Wahl, um E-Mails zu versenden. Kein Wunder: Dank VBA-Schnittstelle lässt es sich ganz einfach von Access aus ansprechen. So können Sie im einfachsten Fall per Mausklick eine leere E-Mail öffnen, um diese auszufüllen und zu versenden. Oder Sie erstellen gleich die komplette E-Mail inklusive Empfänger, Betreff, Inhalt und gegebenenfalls auch den Anhang und versenden diese, ohne dass sie überhaupt angezeigt wird. Dieser Artikel zeigt die grundlegenden Techniken für die Programmierung von Outlook.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1305_MailMitOutlook.mdb**.

Vorbereitungen

Access stellt im VBA-Editor standardmäßig die Objekte, Methoden, Eigenschaften und Ereignisse der vier Bibliotheken **Visual Basic for Applications**, **Microsoft Access x.0 Object Library**, **OLE Automation** und **Microsoft DAO 3.6 Object Library** (ab Access 2007 **Microsoft x.0 Access Database Engine Object**) bereit. Diese enthalten die wesentlichen für die Arbeit mit Access notwendigen Elemente. Wenn Sie Ihre Arbeit auf andere Office-Anwendungen wie beispielsweise Outlook, Word oder Excel ausdehnen möchten, benötigen Sie entweder einen entsprechenden Verweis auf die jeweilige Bibliothek oder Sie binden das gewünschte Objekt erst zur Laufzeit ein.

Weitere Informationen zu diesen beiden Techniken, die Early Binding und Late Binding genannt werden, finden Sie im Artikel **Early Binding und Late Binding**.

In diesem Artikel verwenden wir Early Binding für den Zugriff auf das Objektmodell von Outlook. Das bedeutet, dass Sie im VBA-Editor mit dem Menüeintrag **Extras|Verweise** den Eintrag **Microsoft Outlook x.0 Object Library** hinzufügen (siehe Bild 1).

Wenn Sie nun mit der Taste **F2** den Objektkatalog öffnen und dort im Kombinationsfeld oben links den Eintrag **Outlook** auswählen, finden Sie unten eine Liste aller Objekte der Outlook-Bibliothek vor – eines davon ist das **MailItem**-Objekt, mit dem wir gleich zu tun haben werden (siehe Bild 2).

Formular zum Steuern der Beispiele

Die Beispiele dieses Artikels steuern wir von einem Formular aus. Das Formular heißt **frmMails** und erhält zunächst eine Schaltfläche namens **cmdMailErzeugen**.

Hinterlegen Sie für die Ereigniseigenschaft **Beim Klicken** dieser Schaltflä-

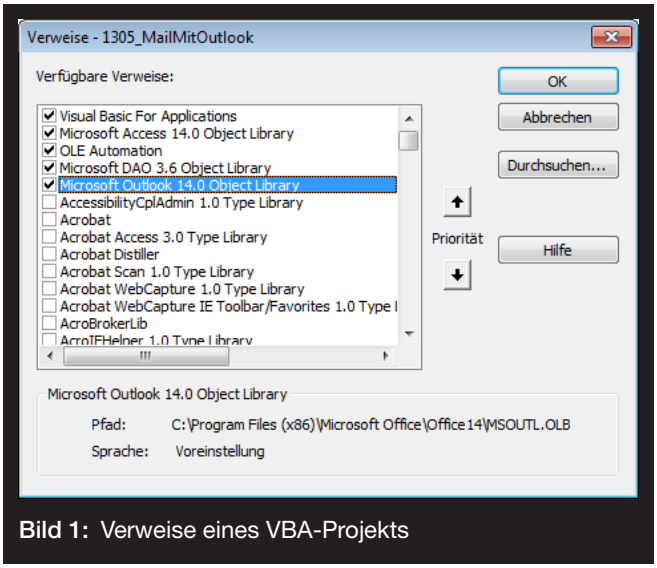


Bild 1: Verweise eines VBA-Projekts

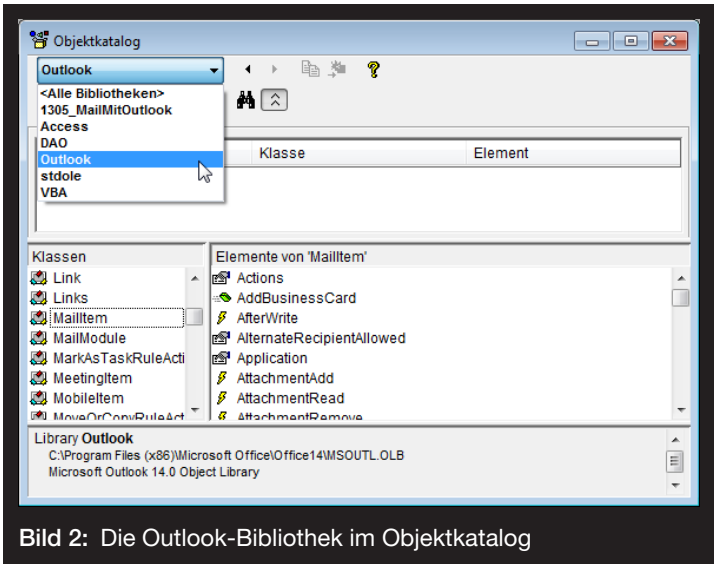


Bild 2: Die Outlook-Bibliothek im Objektkatalog

che den Wert **[Ereignisprozedur]** und klicken Sie dann auf die Schaltfläche rechts neben dem Eigenschaftswert.

Es öffnet sich der VBA-Editor mit der leeren Prozedur. Dort ergänzen Sie zunächst die Zeilen aus Listing 1.

```
Private Sub cmdMailErzeugen_Click()
    Dim objOutlook As Outlook.Application
    Set objOutlook = New Outlook.Application
    MsgBox objOutlook.Name
    ' ...
End Sub
```

Listing 1: Outlook starten

Diese Anweisungen tun nichts anderes, als eine Outlook-Instanz zu erstellen beziehungsweise eine bestehende Instanz mit der Objektvariablen **objOutlook** zu referenzieren und den Wert der Eigenschaft **Name** der mit der Objektvariablen referenzierten Outlook-Instanz auszugeben.

Um dies zu reproduzieren, rufen Sie die Prozedur mit den folgenden Voraussetzungen auf und betrachten, was geschieht:

- Wenn Outlook bereits geöffnet ist: In diesem Fall referenziert die Prozedur die Outlook-Instanz mit der Objektvariablen **objOutlook** und gibt deren Namen aus. Das Schlüsselwort **New** ist hier eigentlich irreführend, aber Sie wissen ja nun, was es bewirkt, wenn Outlook bereits geöffnet ist. Da die Objektvariable objOutlook innerhalb der Ereignisprozedur deklariert ist, wird diese anschließend automatisch wieder geleert.
- Wenn Outlook noch nicht geöffnet ist: Dann erzeugt die Anweisung mit dem **New**-Schlüsselwort eine neue Instanz von Outlook und referenziert diese mit **objOutlook**. Nach der Ausgabe von **objOutlook.Name** wird die Prozedur wieder beendet. Da **objOutlook** die Outlook-Instanz somit nicht mehr referenziert, wird auch die Outlook-Instanz wieder beendet. Sie können dies im Windows Tast-Manager reproduzieren. Vor dem Aufruf sollte dort kein Eintrag namens **OUTLOOK.EXE** vorhanden sein. Während das Meldungsfenster angezeigt wird, finden Sie diesen Eintrag jedoch im Task-Manager vor. Nach dem Schließen des Meldungsfensters und dem Beenden der Prozedur verschwindet **OUTLOOK.EXE** wieder aus der Liste der Tasks.

Eine wichtige Information ist an dieser Stelle, dass Outlook durch das einfache Instanzieren nicht angezeigt wird – dies ist auch nur mit einem anderen Befehl möglich. Allerdings ist dies auch nicht das Ziel dieses Artikels – wir möchten schließlich eine E-Mail verschicken.

```
Private Sub cmdMailErzeugen_Click()
    Dim objOutlook As Outlook.Application
    Dim objMail As Outlook.MailItem
    Set objOutlook = New Outlook.Application
    Set objMail = objOutlook.CreateItem(olMailItem)
    With objMail
        .Display
    End With
End Sub
```

Listing 2: Erstellen und Anzeigen einer ersten E-Mail

Leere E-Mail erstellen

Um eine neue, leere E-Mail zu erstellen, sind nur noch wenige Ergänzungen nötig. Die erste ist die Deklaration eines Objekts vom Typ **Outlook.MailItem**. Dieses referenziert die nachfolgend erstellte E-Mail. Die **CreateItem**-Methode mit dem Wert **olMailItem** für den ersten Parameter erzeugt ein neues E-Mail-Objekt, dass wir gleich mit der Objektvariablen **objMail** referenzieren.

Statt **olMailItem** könnten Sie auch eine andere Variable verwenden – beispielsweise um einen Termin oder einen Kontakt zu erzeugen. Dies besprechen wir jedoch in einem späteren Artikel.

Das **MailItem**-Objekt bietet einige Methoden an, zum Beispiel **Display** (siehe Listing 2). Damit zeigen Sie die frisch erstellt E-Mail nun auch noch an (siehe Bild 3). Die E-Mail wird mit dem standardmäßig gewählten Absender und gegebenenfalls mit einer Signatur angezeigt.

Sie können die E-Mail nun ausfüllen und durch Anklicken der Senden-Schaltfläche abschicken.

Die **Display**-Methode bietet noch einen Parameter namens **Modal** an, der standardmäßig mit dem Wert **False** belegt ist. Stellen Sie diesen auf den Wert **True** ein, wird der aufrufende Programmcode solange an der **Display**-Anweisung angehalten, bis

der Benutzer die E-Mail versendet oder anderweitig schließt.

Sobald die E-Mail versendet wurde, können Sie jedoch nicht mehr über die Objektvariable **objMail** auf die Eigenschaften der E-Mail zugreifen – Sie können also nicht ohne Weiteres etwa auf den Betreff oder den Inhalt zugreifen, um diesen auszulesen und in der Datenbank zu archivieren.

Mail nach dem Versand speichern

Dabei wäre gerade dies interessant: Wenn Sie etwa von einer Kundendatenbank aus eine E-Mail an den aktuell im Kundenformular angezeigten Kunden senden, wäre es doch praktisch, die Korrespondenz anschließend direkt in Zusammenhang mit dem Kundendatensatz zu speichern. Dazu sind jedoch fortgeschrittene Techniken nötig, die wir in einem weiteren Artikel besprechen.

E-Mail vollautomatisch füllen und senden

In vielen Fällen möchten Sie aber vermutlich direkt die komplette E-Mail per VBA-Code fertigstellen und versenden. Dazu finden Sie nachfolgend die grundlengenden Techniken.

Die wichtigsten Einstellungen sind dabei der Empfänger, der Betreff und der Inhalt der E-Mail. Diese Informationen wollen wir per Formular eingeben und direkt in einer Tabelle namens **tblMails** speichern, die zunächst wie in Bild 4 aussieht (später kommen noch weitere Felder hinzu).

Die Tabelle brauchen wir hier eigentlich nicht, aber der Autor ist zu faul, die Testwerte immer wieder neu in das Formular einzutippen ...

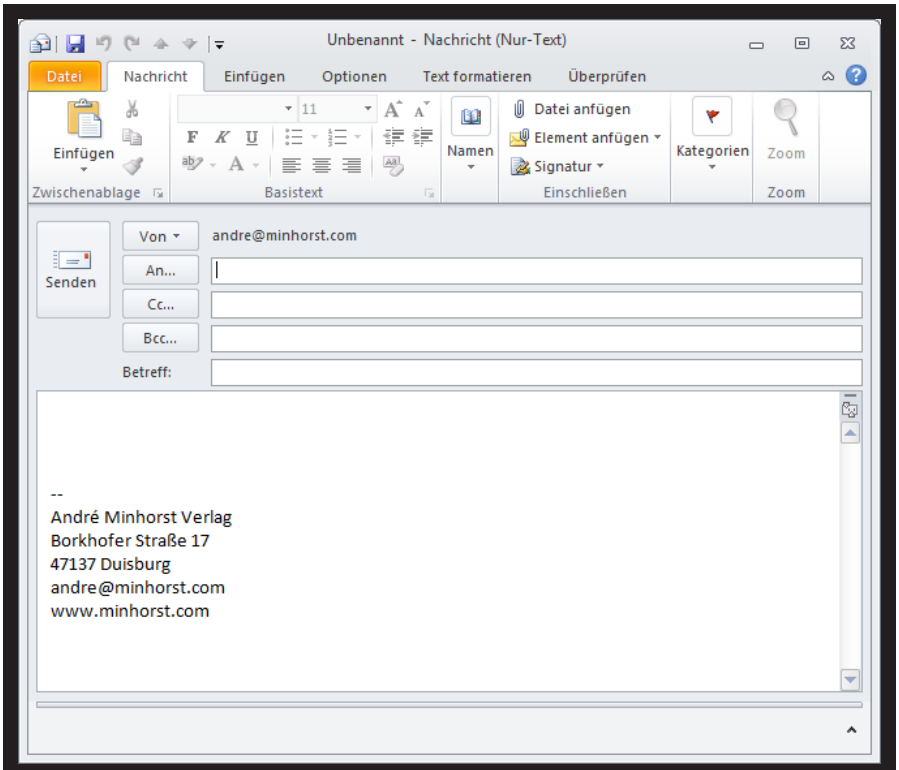
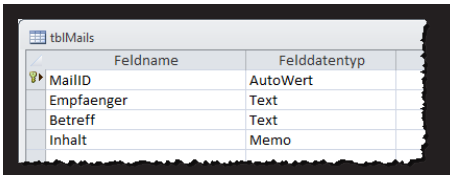


Bild 3: Eine per VBA erzeugte E-Mail

Die Tabelle enthält zunächst vier Felder namens **MailID** (Primärschlüsselfeld), **Empfaenger** (E-Mail-Adresse des Empfängers), **Betreff** und **Inhalt** (als Memofeld, da gelegentlich länger als 255 Zeichen).

Das Formular **frmMail** binden wir über die Eigenschaft **Datenherkunft** an die Tabelle **tblMails** und ziehen die vorhandenen Felder in den Detailbereich des Formularentwurfs. Fügen Sie außerdem eine Schaltfläche namens **cmdMailSenden** hinzu (siehe Bild 5).

Damit die in die drei Felder eingetragenen Daten nun direkt in einer zu erstellenden E-Mail landen, hinterlegen Sie für die Schaltfläche **cmdMailSenden** etwa die Ereignisprozedur aus Listing 3.


tblMails

Feldname	Felddatentyp
MailID	AutoWert
Empfaenger	Text
Betreff	Text
Inhalt	Memo

Bild 4: Tabelle zum Speichern von E-Mails

Die Prozedur erstellt wiederum eine Referenz auf eine bestehende oder neue Outlook-Instanz, erzeugt ein neues **MailItem**-Objekt und füllt dann die Eigenschaften mit den entsprechenden Werten.

Für **Subject** (**Betreff**) und **Body** (**Inhalt**) ist dies direkt über die Zuweisung der jeweiligen Feldinhalt möglich.

Berichte per E-Mail versenden, Teil II

Im ersten Teil dieser Artikelreihe haben Sie erfahren, wie Sie ein PDF-Dokument auf Basis eines Access-Berichts erstellen – zum Beispiel zum Erzeugen einer Rechnung. Im zweiten Teil schauen wir uns nun an, wie Sie diesen Bericht per E-Mail an den Empfänger versenden. Und diese Technik sollen Sie nicht nur für einzelne Empfänger, sondern gleich für ganze Listen von Rechnungsempfängern erledigen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1305_Bericht-PerEMail.mdb**.

Absendemöglichkeit hinzufügen

Als Grundlage zum Versenden von E-Mails mit einer Rechnungsdatei im Anhang verwenden wir das Formular **frmBestellungen** der Beispieldatenbank. Dort können Sie gleich nach der Bestellung die Rechnung im PDF-Format erzeugen. Ein weiterer Mausklick soll die Rechnung bereits versenden.

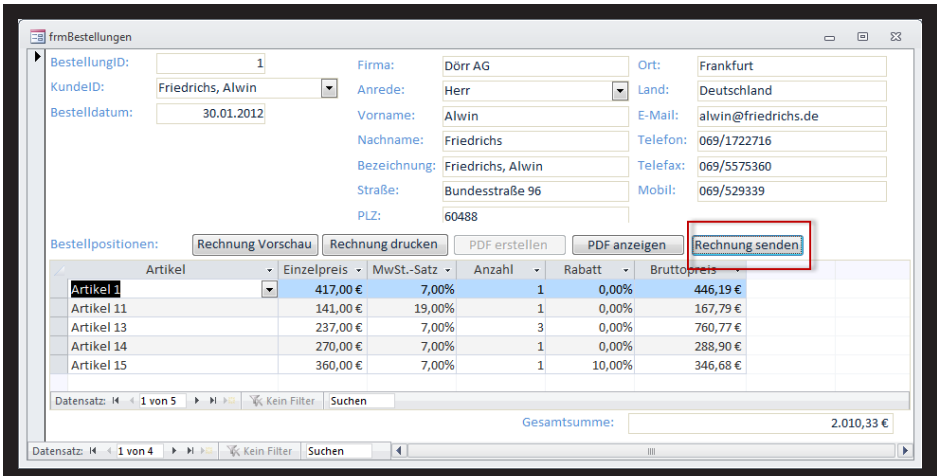


Bild 1: Hinzufügen einer Schaltfläche zum Versenden der Rechnung per E-Mail

Dazu fügen Sie zum Formular eine eine neue Schaltfläche namens **cmdRechnungSenden** hinzu (siehe Bild 1). Bevor wir diese mit Leben füllen, sorgen wir noch für eine entsprechende Aktivierung beziehungsweise Deaktivierung der Schaltfläche. Sie soll nur dann anklickbar sein, wenn der Benutzer die PDF-Rechnung bereits erzeugt hat. Das Aktivieren und Deaktivieren der Schaltflächen übernimmt die bereits im ersten Teil der Artikelserie beschriebene Prozedur **SchaltflaechenAktivieren**.

Dieser fügen Sie die Anweisung aus Listing 1 hinzu. Sie stellt die Eigenschaft **Enabled** der Schaltfläche **cmdRechnungSenden** auf den Wert True ein, wenn das Feld **Rechnungsdatei** der Datenherkunft des Formulars bereits den Namen der erstellten Rechnung im PDF-Format enthält.

```
Private Sub SchaltflaechenAktivieren()
    Dim bol As Boolean
    bol = Len(Nz(Me!Rechnungsdatei, „“)) > 0
    ...
    Me!cmdRechnungSenden.Enabled = bol
    ...
End Sub
```

Listing 1: Aktivieren und Deaktivieren der Schaltfläche zum Versenden einer Rechnung per E-Mail

Rechnung senden per Mausklick

Den eigentlichen Rechnungsversand löst der Benutzer durch einen Klick auf die neu hinzugefügte Schaltfläche aus. Damit dies geschieht, stellen Sie die Eigenschaft **Beim Klicken** der Schaltfläche **cmdRechnungSenden** auf den Wert **[Ereignisprozedur]** ein, klicken auf die Schaltfläche mit den drei Punkten rechts von der Eigenschaft und ergänzen die nun im VBA-Editor erscheinende Ereignisprozedur wie in Listing 2.

Die Prozedur deklariert ein **Outlook**-Objekt und ein **MailItem**-Objekt sowie eine String-Variable, in welcher der Inhalt der E-Mail zusammengestellt wird. Damit die beiden Objekttypen zur Verfügung stehen, benötigen Sie einen entsprechenden Verweis auf die Outlook-Bibliothek – mehr dazu erfahren Sie im Artikel **E-Mails mit Outlook versenden**.

```
Private Sub cmdRechnungSenden_Click()
    Dim objOutlook As Outlook.Application
    Dim objMail As Outlook.MailItem
    Dim strBody As String
    Set objOutlook = New Outlook.Application
    Set objMail = objOutlook.CreateItem(olMailItem)
    With objMail
        .Recipients.Add Me!sfmKundendaten.Form!EMail
        .Subject = "Rechnung zur Bestellung " & Me!BestellungID & " vom " & Me!Bestelldatum
        strBody = "Hallo!" & vbCrLf & vbCrLf
        strBody = strBody & "Hiermit erhalten Sie die Rechnung zu Ihrer Bestellung mit der Nummer " _
            & Me!BestellungID & " vom " & Me!Bestelldatum & "." & vbCrLf & vbCrLf
        strBody = strBody & "Mit freundlichen Grüßen" & vbCrLf
        strBody = strBody & "Ihr Hans Muster"
        .Body = strBody
        .Attachments.Add CStr(Me!Rechnungsdatei)
        .Importance = olImportanceHigh
        .ReadReceiptRequested = True
        .Display
        '.Send
    End With
End Sub
```

Listing 2: Zusammenstellen und Anzeigen der E-Mail mit der Rechnung als Anhang

Danach füllt die Prozedur die Objektvariable **objOutlook** mit einem Verweis auf das mit **New** erhaltene **Outlook.Application**-Objekt und erzeugt mit der **CreateItem**-Methode dieses Objekts ein neues Objekt des Typs **MailItem**.

Anschließend muss die E-Mail mit den entsprechenden Daten gefüllt werden:

- Empfänger
- Betreff
- Inhalt
- Anlage

Die Empfänger-Adresse liefert das Feld **EMail** des Unterformulars mit den Kundendaten zur aktuell angezeigten Bestellung. Den Betreff stellen wir aus einem Text, der Bestellnummer und dem Bestelldatum zusammen, sodass dieser etwa **Rechnung zur Bestellung 1234 vom 1.2.2013** lautet.

Den Inhalt der E-Mail fügen wir der Übersicht halber zeilenweise zusammen. Auch hier integrieren wir wieder die Bestellnummer und das Bestelldatum. Der Inhalt von **strBody** landet schließlich in der Eigenschaft **Body** des **MailItem**-Objekts.

Rechnungsbericht hinzufügen

Fehlt noch der Rechnungsbericht: Diesen fügt die Prozedur mit der **Add**-Methode der **Attachments**-Auflistung hinzu.

Den Pfad der Rechnungsdatei entnimmt sie dem Feld **Rechnungsdatei** der Datenherkunft des Formulars.

Zu guter Letzt stellen wir die Priorität der E-Mail mit der Rechnung auf Hoch ein und fordern vom Empfänger eine Empfangsbestätigung an – beides ist optional.

Wenn Sie eine solche Funktion neu zu einer Anwendung hinzufügen, wollen Sie die E-Mails vermutlich zunächst per Sichtkontrolle prüfen.

Daher zeigt die Prozedur die E-Mails zunächst mit der **Display**-Methode an, der Benutzer kann diese dann mit einem Klick auf die Senden-Schaltfläche der E-Mail absenden.

Wenn Sie sich später vom ordnungsgemäßen Zusammenstellen der E-Mails überzeugt haben, ersetzen Sie die **Display**-Methode durch die **Send**-Methode – die E-Mails werden dann direkt verschickt.

Das Ergebnis sieht schließlich wie in Bild 2 aus: Eine sendebereite E-Mail mit allen benötigten Daten.

Hier noch der Hinweis, dass zu einer gewerblich versendeten E-Mail noch eine entsprechende Signatur gehört.

Rechnungsversand notieren

Wenn Sie die Rechnung versenden, sollten Sie noch das Versanddatum notieren. Dazu fügen Sie der Tabelle **tblBestellungen** ein Feld namens **Rechnungsversand**, welches das Versanddatum der Rechnung speichert (siehe Bild 3).

Mehrere Rechnungen auf einmal senden

Mit steigender Anzahl der zu versendenden Rechnungen möchten Sie diese Aufgabe vielleicht automatisieren und gleich mehrere Rechnungs-E-Mails auf einen Schlag versenden.

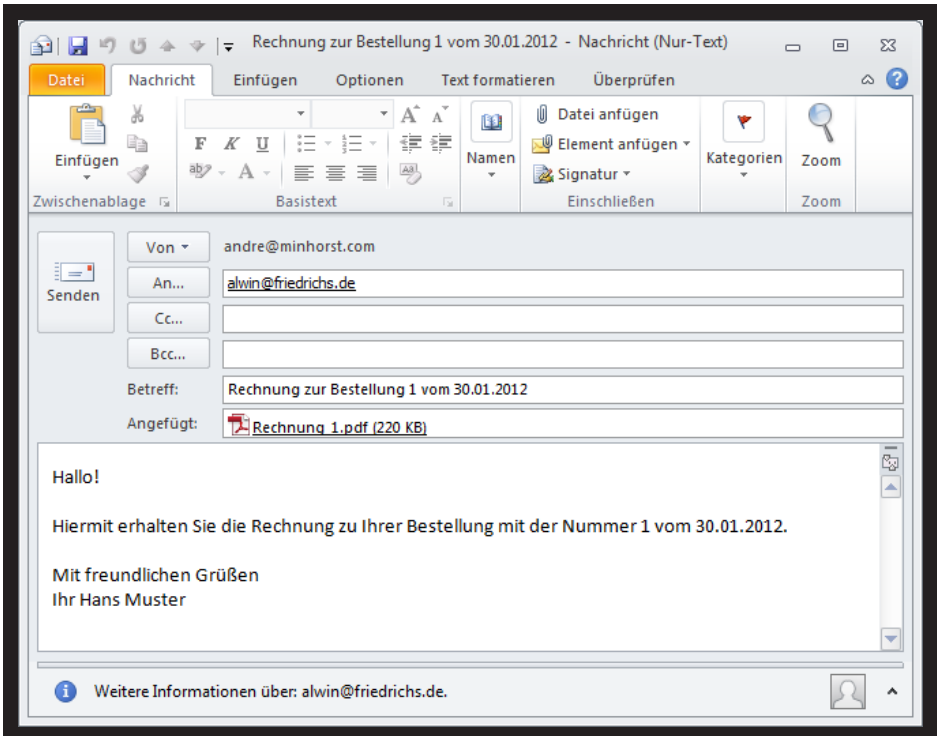


Bild 2: Anzeige einer E-Mail mit Rechnung vor dem Versand

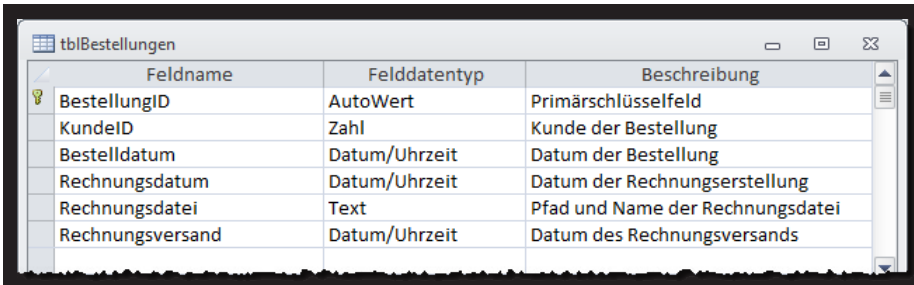


Bild 3: Entwurfsansicht der Tabelle **tblBestellungen**

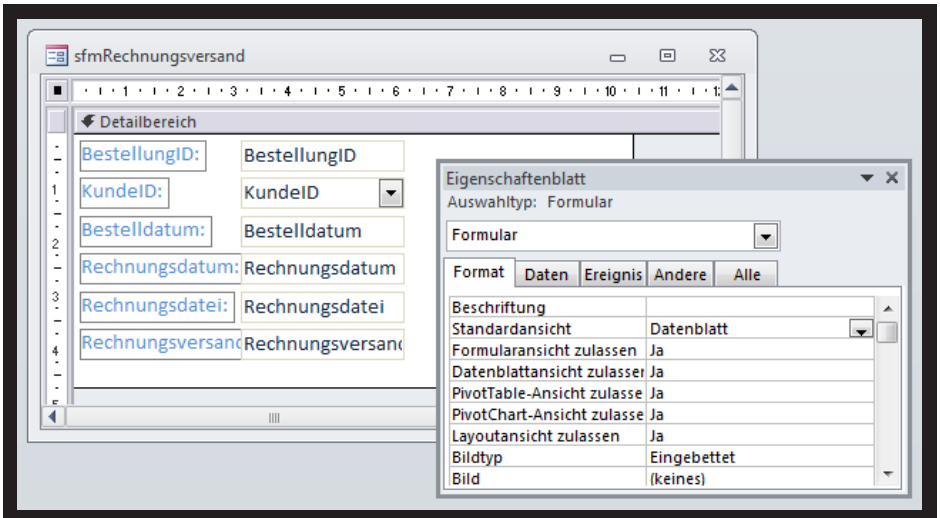


Bild 4: Entwurf des Unterformulars **sfmRechnungsversand**, das alle Bestellungen ohne versendete Rechnung anzeigen soll

Kein Problem: Wir erstellen ein neues Formular, das alle Bestellungen anzeigt, deren Rechnung noch nicht versendet wurde.

Wir beginnen mit dem Unterformular **sfmRechnungsversand**, dass eine Abfrage mit allen Felder der Tabelle **tblBestellungen** enthält und diese mit der Standardansicht **Datenblatt** anzeigen soll (siehe Bild 4).

Die Datenherkunft soll allerdings nicht alle Datensätze anzeigen, sondern nur solche, deren Feld Rechnungsversand noch leer ist – das heißt, dass die Rechnung noch nicht versendet wurde.

Die Entwurfsansicht der dazu verwendeten Abfrage namens **qrysfmRechnungsversand** finden Sie in Bild 5.

Das Hauptformular heißt **frmRechnungsversand** und hat keine eigene Datenherkunft. Dafür nimmt es das Unterformular **sfmRechnungsversand** im unteren Bereich auf.

Darüber soll das Formular zwei Schaltflächen enthalten: Eines zum Erstellen und Versenden der Rechnungen zu allen noch offenen Bestellungen und eines zum Erstellen und Versenden einer einzigen Rechnung – nämlich der zum aktuell im Unterformular markierten Datensatz.

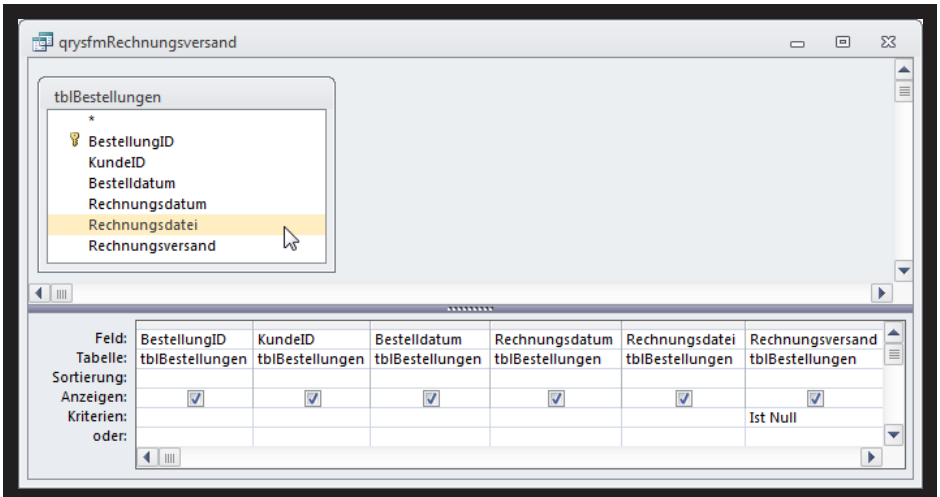


Bild 5: Datenherkunft des Unterformulars **sfmRechnungsversand**

Moment: Erstellen und Versenden? Ja, diese beiden Vorgänge sollen für Rechnungen, die weder erstellt noch versendet wurden, zusammengefasst werden.

Bei Bestellungen mit bereits vorhandenen Rechnungen soll ein Klick auf eine der Schaltflächen lediglich den Versand des Rechnungsformulars erledigen.

Beide Schaltflächen führen prinzipiell die gleichen Schritte durch – gegebenenfalls das Anlegen des Rechnungsdokuments im PDF-Format und das anschließende Versenden. Der Unterschied, dass die eine Schaltfläche dies für eine einzige Bestellung durchführt und die andere für alle aktuell angezeigten Bestellungen.

Daher werden wir die eigentliche Funktion in eine Prozedur auslagern, die dann von den durch die beiden Schaltflächen ausgelösten Ereignisprozeduren ausgelöst wird.

Da das Formular keine Datenherkunft besitzt, blenden Sie die sonst zum Blättern in den Datensätzen sinnvollen Elemente durch Einstellen der Eigenschaften **Navigationsschaltflächen**, **Datensatzmarkierer**, **Trennlinien** und **Bildlaufleisten** auf den Wert **Nein** aus.

Das Formular mit den beiden Schaltflächen **cmdMarkierteRechnung** und **cmdAlleRechnungen** sieht nun wie in Bild 6 aus.

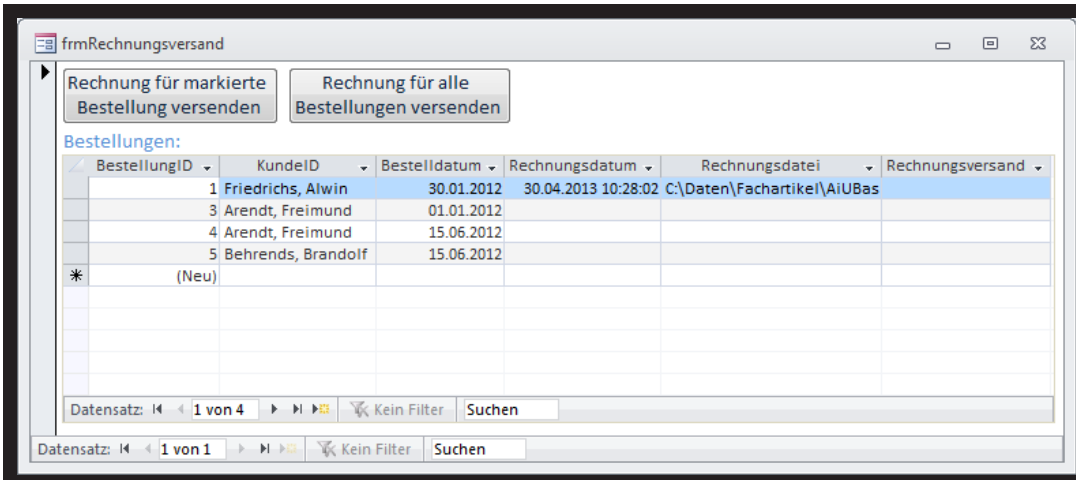


Bild 6: Das Formular **frmRechnungsversand** mit den beiden Schaltflächen zum Versenden einer oder aller noch nicht versendeten Rechnungen

Eine E-Mail versenden

Die Prozedur zum Versenden einer einzigen Rechnungs-E-Mail aus Listing 3 ermittelt zunächst den Wert des Feldes **BestellungID** für den aktuell ausgewählten Datensatz.

Hat der Benutzer aktuell keinen Datensatz im Unterformular ausgewählt, erhält die Variable **lngBestellungID** statt

E-Mails sicher versenden

Wenn Sie eine E-Mail über Outlook mit VBA versenden, können Sie dies auf zwei Arten erledigen – durch direktes Versenden oder durch Anzeigen der Mail und anschließendes Anklicken der Senden-Schaltfläche. In beiden Fällen gibt es keine von VBA auswertbare Eigenschaft, die Informationen über den Versandstatus der E-Mail liefert. Und wenn Sie die per VBA gefüllte E-Mail vor dem Versand noch zur Kontrolle anzeigen, kann der Benutzer noch Änderungen am Inhalt vornehmen, die nicht in der Datenbank gespeichert werden. Dieser Artikel zeigt, wie Sie sich den Versand bestätigen lassen und gleichzeitig den versendeten Inhalt erfassen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1306_MailSicherVersenden.mdb**.

Voraussetzungen

Die hier vorgestellten Techniken funktionieren nur mit neueren Outlook-Versionen (ab Outlook 2007).

Ausgangssituation

Im Artikel **Berichte per E-Mail versenden** aus Ausgabe 5/2013 haben Sie erfahren, wie Sie Outlook-Mails per VBA erstellen und anzeigen, bevor Sie diese verschicken (siehe Bild 1).

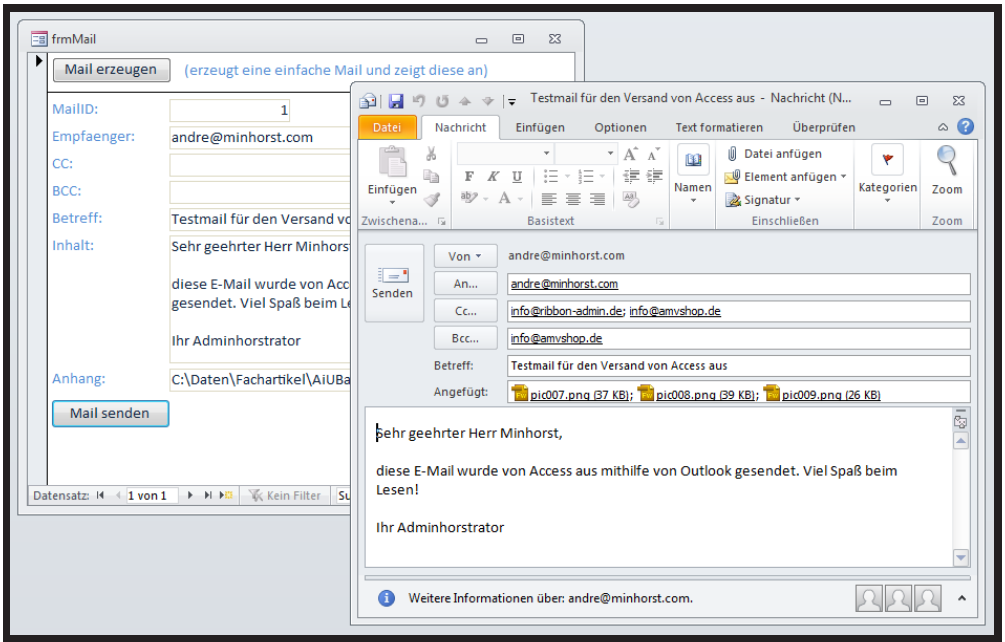


Bild 1: Erstellen einer E-Mail per VBA und Anzeige in der Outlook-Vorschau

Betreff und Inhalt der E-Mail sowie weitere Eigenschaften wurden bereits in einem Access-Formular eingegeben, die Anzeige der Outlook-Mail vor dem Versenden soll lediglich der Kontrolle dienen.

Es sind jedoch auch noch andere Szenarien denkbar: Beispielsweise könnten Sie aus einem Kundenformular heraus eine E-Mail an den aktuell angezeigten Kunden erstellen. Sie würden dann direkt die E-Mail mit Outlook anzeigen und nicht erst ein Access-Formular erstellen, dass prinzipiell die gleichen Felder wie Outlook anzeigt. Der Benutzer trägt dann Betreff und Inhalt der E-Mail ein und versendet diese.

Im Gegensatz zur vorherigen Variante, wo wir die E-Mail bereits in einem Access-Formular erstellt und dabei gleichzeitig die Inhalt in der an das Formular gebundenen Tabelle gespeichert haben, werden Betreff, Inhalt et cetera nun nicht in der Datenbank gespeichert.

Aber auch bei der Variante, bei der die Mail im Formular erstellt und in der Tabelle gespeichert wird, kann der Benutzer ja noch Änderungen an der E-Mail vornehmen, wenn diese vor dem Versenden noch in Outlook angezeigt wird. Das heißt, dass die Tabelle mit den E-Mails unter Umständen gar nicht den tatsächlich gesendeten Inhalt enthält.

Es wäre also für beide Fälle interessant, vor dem Versenden der E-Mail über die **Senden**-Schaltfläche der

E-Mail in Outlook noch einmal den Inhalt einzulesen und in die Datenbank zu übertragen.

Wie aber können wir dies bewerkstelligen? Hilfe bietet das Objektmodell von Outlook ab der Version 2007. Genau wie die Access-Objekte wie Formulare, Steuerelemente oder Berichte bieten auch die Outlook-Objekte Ereignisse an, für die Sie Ereignisprozeduren implementieren können. Diese werden durch Ereignisse wie **Open**, **Send** et cetera ausgelöst. Eine Übersicht etwa aller Ereignisse des **MailItem**-Objekts liefert der Objektkatalog, den Sie mit **F2** einblenden.

Dort wählen Sie die Bibliothek **Outlook** aus und selektieren in der Liste links das zu untersuchende Objekt. Auf der rechten Seite finden Sie dann alle Eigenschaften, Methoden und Ereignisse des jeweiligen Objekts (siehe Bild 2). Wenn Sie dort im Kontextmenü den Eintrag **Elemente gruppieren** aktivieren, gruppiert der Dialog die Einträge nach Eigenschaften, Methoden und Ereignissen.

Ereignis beim Senden einer E-Mail

Wenn wir, wie oben beschrieben, eine E-Mail vor dem Senden noch auslesen und Informationen wie den Betreff oder den Inhalt in einer Tabelle der Datenbank speichern möchten, müssen wir das Ereignis **Send** des **MailItem**-Objekts implementieren.

Die Beispiele zu den folgenden Abschnitten finden Sie übrigens im Formular **frmMail** und im dazugehörigen Klassenmodul **Form_frmMail**.

Grundlagen zum Implementieren von Ereignissen aus anderen Objekten wie Formularen, Steuerelementen oder auch externen Objekten wie hier von Outlook finden Sie im Artikel **Ereignisse abgreifen** in dieser Ausgabe von **Access [basics]**.

Wir möchten das **Send**-Ereignis dazu nutzen, die gegebenenfalls vom Nutzer in der Outlook-Mail geänderten Daten wie den Betreff oder den Inhalt in die entsprechenden Textfelder des Formulars zu übertragen, in dem die Mailinhalte ursprünglich zusammengestellt wurden.

Dazu deklarieren Sie im Klassenmodul des Formulars zunächst eine Objektvariable für das Objekt, dessen Ereignisse wir implementieren möchten – also im Klassenmodul **Form_frmMail**.

Die Deklaration sieht wie folgt aus:

```
Dim WithEvents objMail As Outlook.MailItem
```

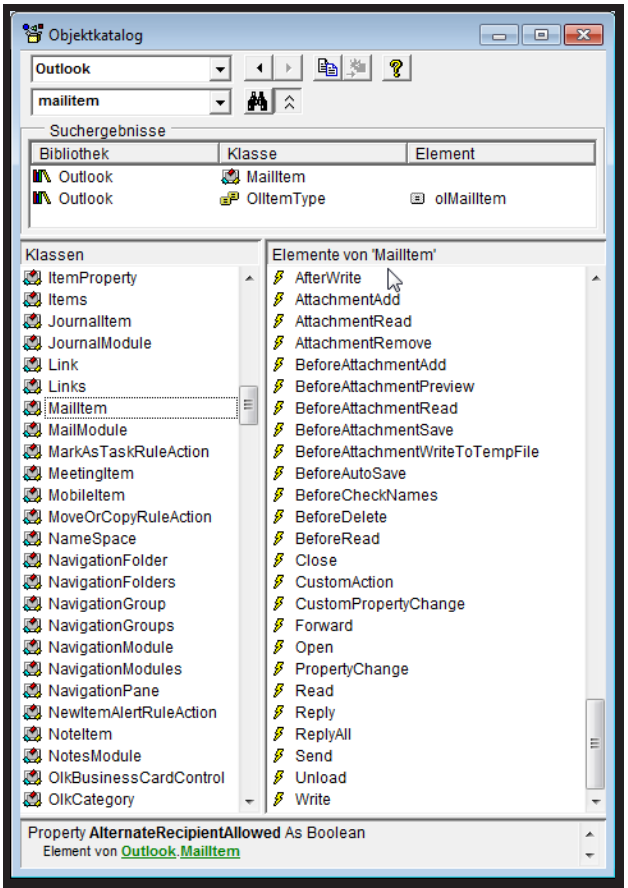


Bild 2: Ereignisse des **MailItem**-Objekts in Outlook

```
Private Sub cmdMailSenden_Click()
    Dim objOutlook As Outlook.Application
    'nächste Zeile auskommentieren
    'Dim objMail As Outlook.MailItem
    Dim i As Integer
    Dim strAnhaenge() As String
    Set objOutlook = New Outlook.Application
    Set objMail = objOutlook.CreateItem(olMailItem)
    With objMail
        '...
        .Display
        '.Send
    End With
End Sub
```

Listing 1: Mail füllen und vor dem Versenden anzeigen

Wichtig ist hier das Schlüsselwort **WithEvents**. Dieses gibt an, dass im aktuelllen Klassenmodul Ereignisprozeduren für das mit dieser Variablen referenzierte Objekt vorhanden sein könnten.

Ereignisse abgreifen

Normalerweise implementieren Sie die Ereignisprozeduren von Formularen und Steuerelementen jeweils im Klassenmodul des jeweiligen Formulars. Aber wussten Sie, dass Sie die Ereignisse auch in anderen Klassenmodulen implementieren können? Falls nicht: Es funktioniert! Und für diese Technik gibt es auch noch sinnvolle Einsatzzwecke. Wie dies gelingt und welche Einsatzzwecke es dafür gibt, erfahren Sie im vorliegenden Artikel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1306_EreignisseAbgreifen.mdb**.

Ereignisse implementieren

Bestimmte Prozeduren werden durch Ereignisse ausgelöst. Einige haben Sie in **Access [basics]** bereits kennengelernt – zum Beispiel das Ereignis **Beim Klicken** einer Schaltfläche oder Formularereignisse wie **Beim Öffnen**, **Beim Anzeigen**, **Beim Laden** oder **Beim Entladen** eines Formulars.

Diese können Sie nutzen, indem Sie erstens für die Ereigniseigenschaft festlegen, dass das Ereignis eine Ereignisprozedur auslösen soll. Dies erledigen Sie durch Eintragen des Wertes **[Ereignisprozedur]** für die entsprechende Ereigniseigenschaft.

Zusätzlich benötigen Sie eine Ereignisprozedur, deren erste Zeile eine spezielle, vom Namen des Objekts (also des Formulars oder Steuerelements) und vom Ereignis abhängige Syntax aufweisen muss. Diese sieht etwa für das Laden eines Formulars wie folgt aus:

```
Private Sub Form_Load()
```

Die erste Zeile mancher Ereignisprozeduren enthält noch Parameter, die verschiedene Informationen liefern, die beim Durchführen der Prozedur hilfreich sein können. In manchen Fällen nehmen die Parameter auch Rückgabewerte entgegen, mit denen sich das weitere Verhalten von Access steuern lässt.

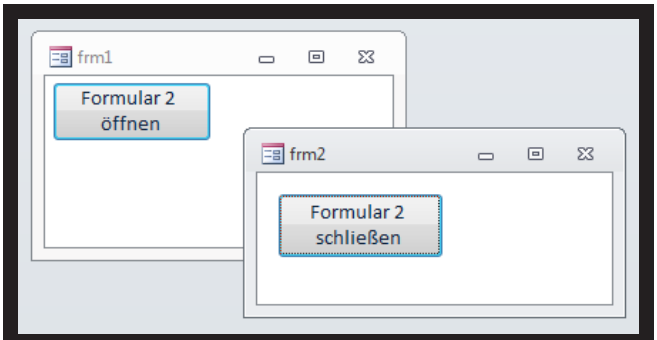


Bild 1: Formular 1 öffnet Formular 2, Formular 2 wird wieder geschlossen

Einfache Beispiele

Die Konstellation aus Bild 1 liefert ein paar Beispiele für Ereignisprozeduren. Die Schaltfläche **cmdForm2Oeffnen** löst die folgende Prozedur aus und öffnet damit das zweite Formular:

```
Private Sub cmdForm2Oeffnen_Click()
    DoCmd.OpenForm "frm2"
End Sub
```

Die Schaltfläche **cmdForm2Schliessen** im zweiten Formular schließt das Formular wieder und gibt eine Meldung aus, dass Formular 2 geschlossen wird:

```
Private Sub cmdForm2Schliessen_Click()
    DoCmd.Close acForm, Me.Name
    MsgBox "Formular 2 wird geschlossen."
End Sub
```

Formular 1 beim Schließen von Formular 2 informieren

Das ist alles kein Problem, wenn die Formulare unabhängig voneinander sind. Wenn Formular 1 aber nun Werte anzeigt (beispielsweise in einem Unterformular in der Datenblattansicht), die in Formular 2 bearbeitet werden, soll das Unterformular von Formular 1 nach dem Schließen von Formular 2 tunlichst aktualisiert werden, damit es den aktuellen Stand des bearbeiteten Datensatzes anzeigt.

Formular 1 muss also irgendwie informiert werden, wenn der Benutzer Formular 2 schließt. Dies können Sie auf folgende Art erledigen: Dabei öffnet Formular 1 das Formular 2 mit dem Wert **acDialog** für den Parameter **WindowMode**. Dies bewirkt, dass Access den aufrufenden Code erst fortsetzt, wenn Formular 2 den Fokus

verliert. Dies kann durch zwei Ereignisse geschehen – wenn der Benutzer das Formular schließt oder wenn es nicht mehr sichtbar ist.

Schauen wir uns dies für die Beispielformulare **frm1_1** und **frm2_1** an. Die Schaltfläche in **frm1_1** löst die Ereignisprozedur aus Listing 1 aus. Formular 2 wird dadurch als modaler Dialog geöffnet, die zweite Anweisung der Prozedur wird noch nicht ausgeführt.

Erst wenn der Benutzer Formular 2 schließt, zeigt die Prozedur die Meldung mit dem Text **Formular 2 ist geschlossen** an.

Der Nachteil bei dieser Vorgehensweise ist, dass der Benutzer erst nach dem Schließen von Formular 2 wieder auf Formular 1 oder andere Formulare zugreifen kann. Das kann aber auch zum Vorteil werden – nämlich dann, wenn das aufrufende Formular die im aufgerufenen Formulare eingegebenen Daten oder Aktionen erwartet.

Ereignisse abgreifen

Es geht aber noch etwas eleganter: Beim folgenden Beispiel ruft die Schaltfläche in Formular 1 wiederum Formular 2 auf, diesmal allerdings nicht im modalen Modus. Als Beispiele dienen die Formulare **frm1_2** und **frm2_2**.

In diesem Beispiel wird auch eine Meldung ausgegeben, wenn das zweite Formulare geschlossen wird – allerdings nicht innerhalb einer Prozedur im Klassenmodul des zweiten Formulars und auch nicht in der Prozedur, die das Formular öffnet.

```
Private Sub cmdForm2Oeffnen_Click()
    DoCmd.OpenForm „frm2_1", WindowMode:=acDialog
    MsgBox „Formular 2 ist geschlossen oder unsichtbar"
End Sub
```

Listing 1: Formular modal öffnen

Stattdessen legen wir eine Ereignisprozedur an, die zwar durch das Ereignis **Beim Schließen** des zweiten Formulars ausgelöst wird, die sich aber nicht im Klassenmodul des zweiten Formulars befindet – sondern im Klassenmodul des ersten Formulars!

Dazu legen Sie zuerst fest, dass im Klassenmodul des ersten Formulars die Ereignisse des zweiten Formulars abgefangen werden sollen. Dies erledigen Sie durch die Deklaration einer Objektvariablen des Typs **Form**:

```
Dim WithEvents frm As Form
```

Sie erkennen sicher schon die Besonderheit gegenüber anderen Deklarationszeilen. Die Zeile enthält nämlich das Schlüsselwort **WithEvents**. Dieses gibt an, dass Access beim Feuern von Ereignissen in dem mit der Objektvariablen **frm** referenzierten Formular das aktuelle Klassenmodul nach eventuellen Implementierungen der Ereignisse in Form von Ereignisprozeduren suchen soll.

An welcher Stelle aber füllen wir diese Objektvariable mit dem Verweis auf das Formular **frm2_2**? Die Voraussetzung dafür ist, dass dieses Formular bereits geöffnet ist. Der beste Zeitpunkt ist also der direkt nach dem Öffnen des Formulars. Da wir dies per VBA-Code erledigen, ist der richtige Code zum Referenzieren des Formulars also schnell gefunden:

```
Private Sub cmdForm2Oeffnen_Click()
    DoCmd.OpenForm "frm2_2"
    Set frm = Forms!frm2_2
    ...
End Sub
```

Was erledigen wir nun mit dieser Variablen? Schauen wir doch einmal, was das Codefenster im VBA-Editor bietet.

Das linke Kombinationsfeld offeriert alle Objekte, die im Klassenmodul zur Verfügung stehen und Ereignisse anbieten, die Sie in Form entsprechender Ereignisprozeduren implementieren können. Wählen Sie dort einen Eintrag aus, zeigt das linke Kombinationsfeld alle möglichen Ereignisse dieses Objekts an (siehe Bild 2).

In unserem Fall wählen wir die Objektvariable **frm** aus, welche sich ja genau genommen gar nicht konkret auf das Formular **frm2_2** bezieht, sondern alle Ereignisse des Objekttyps **Form** bereitstellt.

Was haben wir vor? Wir möchten das Schließen des Formulars abfangen und eine Aktion durchführen – in diesem Fall die Ausgabe einer einfachen Meldung. Also wählen wir einfach im linken Kombinationsfeld den Eintrag **frm** und im rechten **OnClose** aus.

Der VBA-Editor legt daraufhin eine leere Ereignisprozedur an, die Sie wie folgt füllen:

Filterkriterien für Formulare, Teil III: Kombinationsfelder

In den ersten beiden Teilen dieser Artikelreihe haben Sie erfahren, wie Sie Felder der verschiedenen Datentypen filtern. Nun geht es ans Eingemachte: Wir filtern nach dem Inhalt von Kombinationsfeldern. Eigentlich könnte dies ganz leicht sein: Der Benutzer wählt einfach eine Wert des Kombinationsfeldes aus und die Datenherkunft wird nach dem Wert im entsprechenden Feld gefiltert. Allerdings kann es ja auch geschehen, dass man nicht nach dem Wert des gebundenen Feldes im Kombinationsfeld, sondern nach dem angezeigten Wert filtern möchte. Wir schauen uns beide Fälle an.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1306_FilterkriterienFuerFormulare.mdb**.

Beispielformular

Das Beispielformular sieht wie in Bild 1 aus. Es heißt **frmFilternNachKombinationsfeld** und enthält im Formularkopf ein Kombinationsfeld namens **cboLieferantNachID** und ein Textfeld namens **txtLieferantNachName**.

Im Detailbereich haben wir ein Unterformular namens **frmFilternNachKombinationsfeld** eingerichtet, das die Daten der Tabelle **tblArtikel** anzeigt.

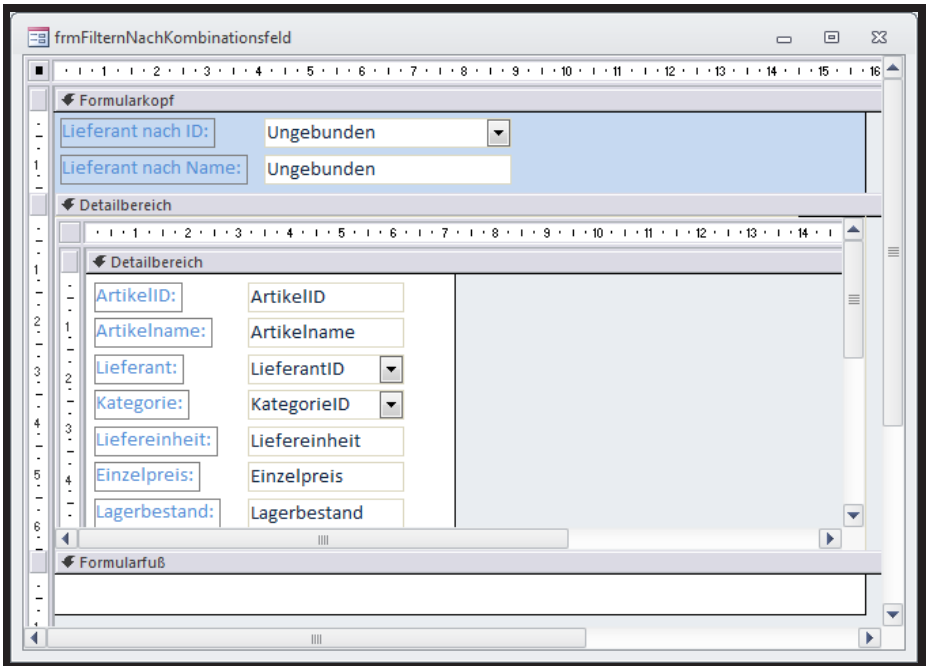


Bild 1: Formular zum Filtern nach Kombinationsfeldern und ihren Inhalten

Filtern nach der gebundenen Spalte

Das Kombinationsfeld **cboLieferantNachID** soll nach der Auswahl eines Eintrags direkt diejenigen Artikel im Unterformular anzeigen, die vom angegebenen Lieferanten geliefert werden. Dazu muss das Kombinationsfeld allerdings erst einmal alle Lieferanten anzeigen. Diese können Sie auf zwei Arten beziehen:

- Erstens über die Tabelle **tblLieferanten**. Dies würde alle Lieferanten liefern, die überhaupt vorhanden sind.
- Zweitens über eine Abfrage, welche die Tabellen **tblArtikel** und **tblLieferanten** verknüpft. Auf diese Weise würde das Kombinationsfeld nur diejenigen Lieferanten anzeigen, die aktuell einem Artikel zugeordnet sind.

Da beide Varianten interessant sind, schauen wir uns beide an. Im ersten Fall weisen Sie der Eigenschaft **Datensatzherkunft** des Kombinationfeldes die folgende SQL-Anweisung zu:

```
SELECT LieferantID, Firma
FROM tblLieferanten ORDER BY Firma;
```

Die Datensätze der Tabelle **tblLieferanten** werden durch die Klausel **ORDER BY Firma** direkt nach dem Firmennamen des Lieferanten sortiert.

Damit das Kombinationsfeld nur die Werte des zweiten Feldes anzeigt, aber den Wert aus dem Feld **LieferantID** aus dem ersten Feld als gebundene Spalte verwendet, stellen

Sie noch die beiden folgenden Eigenschaften ein:

- **Spaltenanzahl: 2**
- **Spaltenbreiten: 0cm**

Das Filtern selbst ist relativ leicht – Sie brauchen nur eine Ereignisprozedur für die Ereigniseigenschaft **Nach Aktualisierung** des Formulars zu hinterlegen, welche den Filterausdruck des Unterformulars einstellt. Diese Ereignisprozedur finden Sie in Listing 1.

```
Private Sub cboLieferantNachID_AfterUpdate()
    With Me!sfmFilternNachKombinationsfeld.Form
        .Filter = "LieferantID = " & Me!cboLieferantNachID
        .FilterOn = True
    End With
End Sub
```

Listing 1: Lieferanten nach ID filtern

Nach der Auswahl eines der Einträge aus dem Kombinationsfeld zeigt das Unterformular nun bereits alle verfügbaren Einträge an (siehe Bild 2).

Alle Artikel anzeigen

Was aber geschieht, wenn Sie wieder alle Datensätze anzeigen möchten? Das Leeren des Kombinationsfeldes hilft an dieser Stelle nicht weiter. Genau genommen löst dies nach der Aktualisierung des Kombinationfeldes etwa durch Betätigen der Eingabetaste sogar einen Fehler aus – und zwar Fehler **3075**, **Syntaxfehler (fehlender Operator) in Abfrageausdruck 'LieferantID = '**. Da **cboLieferantNachID** leer ist, stellt die Prozedur also einen ungültigen Ausdruck zusammen. Dies können Sie mit einer einfachen Prüfung umgehen: Sie fragen einfach vor dem Setzen des Filters den Wert des Kombinationfseldes ab. Ist dieser **Null**, wird der Inhalt durch **0** ersetzt. In diesem Fall leert die neue Version der Prozedur aus Listing 2 den Filter und das Unterformular zeigt wieder alle Datensätze an. Anderenfalls führt das Formular den Filter wie gewohnt aus.

Eine weitere Alternative ist ein spezieller Eintrag im Kombinationsfeld, der die Bezeichnung **<Alle anzeigen>** aufweist und ebenfalls zum Leeren des Filters führt.

Dieser Wert sollte außerdem als erster Eintrag des Kombinationsfeldes angezeigt werden. Dazu ist ein kleiner Trick nötig, der den Einsatz des **UNION**-Schlüsselworts erforderlich macht.

Solche Abfragen können Sie nicht über den Abfrageentwurf, sondern nur über die SQL-Ansicht eingeben.

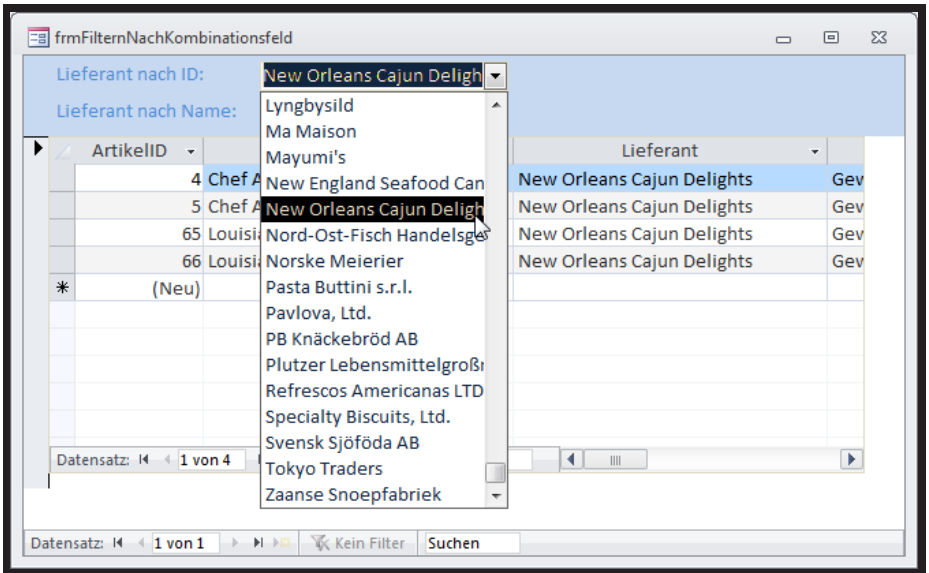


Bild 2: Filtern von Datensätzen nach der gebundenen Spalte des Nachschlagefeldes

```
Private Sub cboLieferantNachID_AfterUpdate()
    With Me!sfmFilternNachKombinationsfeld.Form
        If Not Nz(Me!cboLieferantNachID, 0) = 0 Then
            .Filter = "LieferantID = " & Me!cboLieferantNachID
            .FilterOn = True
        Else
            .Filter = ""
        End If
    End With
End Sub
```

Listing 2: Berücksichtigung eines geleerten Kombinationsfeldes zur Auswahl der Artikel

Also gehen Sie folgendermaßen vor (im Beispielformular unter **cboLieferantNachIDUndAlle** zu finden):

- Markieren Sie die Eigenschaft **Datensatzherkunft** des Kombinationsfeldes und klicken Sie auf die Schaltfläche mit den drei Punkten.
- Es erscheint die Entwurfsansicht der Abfrage. Aktivieren Sie nun die SQL-Ansicht (unter Access 2010 beispielsweise wie in Bild 3).
- Ändern den Sie den SQL-Ausdruck wie in der Abbildung.

Der Ausdruck sieht wie folgt aus:

```
SELECT 0 AS LieferantID,
"<Alle anzeigen>" AS Firma
FROM tblLieferanten
UNION
SELECT LieferantID, Firma
FROM tblLieferanten
ORDER BY Firma;
```

Dieser Ausdruck zeigt die Einträge immer noch nach dem Wert des Feldes **Firma** sortiert an. Da das Zeichen **<** vor dem ersten Buchstaben liegt, erscheint der Eintrag **<Alle anzeigen>** als erster Eintrag.

Hier kann es jedoch zu Problemen kommen, die unerfahrene Entwickler möglicherweise nicht lösen können: Wenn Sie beispielsweise den von der Abfrageentwurfsansicht erstellten SQL-Ausdruck einsetzen, werden alle Felder (auch die als Sortierkriterium verwendeten) noch mit vorangestelltem Tabellennamen versehen.

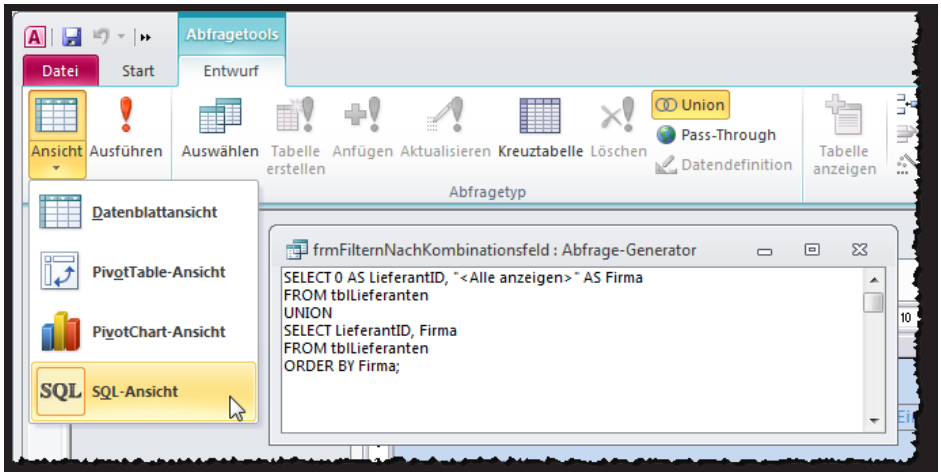


Bild 3: Abfrage zur Anzeige eines Eintrags mit dem Text **<Alle anzeigen>**

Der folgende Ausdruck führt beispielsweise zu einem Fehler:

```
SELECT 0 AS LieferantID,
"<Alle anzeigen>" AS Firma
FROM tblLieferanten
UNION
SELECT tblLieferanten.LieferantID,
tblLieferanten.Firma
FROM tblLieferanten
ORDER BY tblLieferanten.Firma;
```

Die Fehlermeldung lautet: Der **ORDER BY**-Ausdruck (**tblLieferanten.Firma**) enthält Felder, die nicht durch die Abfrage ausgewählt wurden.

Nur die Felder, die in der ersten Abfrage angegeben sind, können in dem **ORDER BY**-Ausdruck verwendet werden.

Das heißt, dass die Abfrage in der ersten der durch das **UNION**-Schlüsselwort verbundenen Abfragen ein Feld namens **tblLieferanten.Firma** erwartet. Also versuchen wir, es Access recht zu machen und ändern den Feldnamen in der ersten Abfrage:

```
SELECT 0 AS LieferantID,
"<Alle anzeigen>"
AS tblLieferanten.Firma ...
```

Dies führt wiederum zu einem Fehler, da es sich bei dem Feldnamen ja um einen Alias-Namen handelt – und der darf nur aus alphanumerischen Zeichen und dem Unterstrich bestehen. Also weichen wir auf die **ORDER BY**-Klausel aus und entfernen dort den Tabellennamen aus der Feldbezeichnung:

```
SELECT
...
ORDER BY Firma;
```

Nun funktioniert es. Nun fehlen noch zwei Dinge: Erstens soll das Kombinationsfeld gleich beim Öffnen des Formulars den ersten Eintrag anzeigen und zweitens benötigen wir auch für dieses Kombinationsfeld eine Ereignisprozedur, die den Filtervorgang durchführt.

Die erste Aufgabe lösen wir mit einem Einzeiler, der gleich beim Laden des Formulars ausgeführt werden

Textfelder im Griff, Teil II: Markierungen

Textfelder sind eines der wichtigsten Steuerelemente zur Eingabe von Daten. Dementsprechend sollten Sie maximalen Komfort beim Umgang mit Textfeldern gewährleisten. Wie das aussieht? Zum Beispiel so, dass der Benutzer so wenig Arbeit wie möglich hat – etwa durch das richtige Positionieren der Einfügemarke beziehungsweise Markierung. Dieser Artikel zeigt, welche Möglichkeiten die Markierung grundsätzlich bietet und wie Sie diese per VBA und mit anderen Einstellungen anpassen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1306_TextfelderMarkieren.mdb**.

Textfelder markieren

Sie werden es bereits festgestellt haben: Beim Verschieben des Fokus auf ein Textfeld kann es zu verschiedenen Aktionen kommen. Entweder Access markiert den kompletten Inhalt vom ersten bis zu letzten Zeichen oder platziert die Einfügemarke ganz vorn oder ganz hinten im Textfeld. Ist das Textfeld leer, landet die Einfügemarke natürlich immer an der gleichen Stelle.

Wussten Sie, dass Sie dieses Verhalten beeinflussen können? Es gelingt ganz einfach: Sie brauchen nur eine Einstellung in den Access-Optionen vorzunehmen. Diese finden Sie etwa unter Access 2010, wenn Sie den Befehl **Optionen** im **Datei**-Menü auswählen und dann im **Optionen**-Dialog zum Bereich **Clienteinstellungen** wechseln.

Dort finden Sie die Option **Cursorverhalten bei Eintritt in Feld** mit den folgenden Einstellungen (siehe Bild 2):

- **Ganzes Feld markieren**
- **Zum Anfang des Feldes gehen**
- **Zum Ende des Feldes gehen**

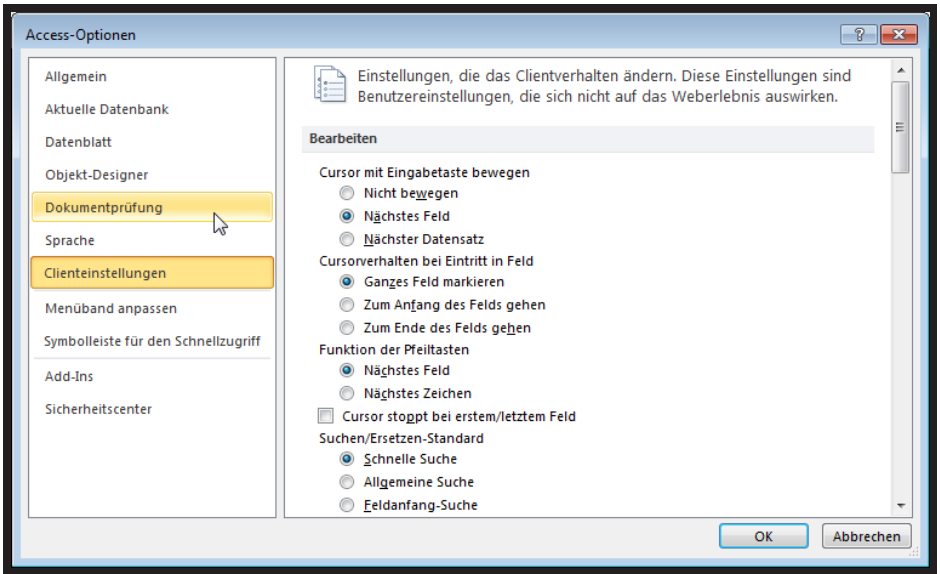


Bild 1: Einstellungen für das Verhalten beim Eintritt in ein Feld

Beim Wechsel zum Textfeld aus Bild 1 war beispielsweise die Option **Ganzes Feld markieren** eingestellt.

Damit kommen wir nun gleich zu einem wichtigen Punkt: Die Einstellung, die Sie in den Access-Optionen vornehmen, gilt für die aktuelle Sitzung und für alle von nun an begonnenen Sitzungen. Access erweckt zwar den Eindruck, dass man die Anwendung neu starten müsste, damit die Änderung wirkt, aber dies ist nicht der Fall – wenn Sie etwa wünschen, dass die Einfügemarke beim Eintreten in ein Feld gleich am Ende des Inhalts landet, dann geschieht dies nach dem Einstellen der Option auf den Wert **Zum Ende des Feldes gehen** auch. Sollten Sie jedoch parallel weitere Access-Instanzen geöffnet haben, wirkt sich die Änderung der Option nicht darauf aus.

Bild 2: Beim Wechsel in ein Textfeld markiert Access den kompletten Inhalt.

Eintrittsverhalten per VBA

Sie können diese Eigenschaft auch per VBA einstellen. Um dies zu veranschaulichen, haben wir die notwendigen Codezeilen im Formular **frmTextfeldmarkierungen** untergebracht. Die Optionsgruppe **ogrCursorverhalten** aus Bild 3 hat drei Optionen mit den Werten **0**, **1** und **2**.

Die meisten Optionen von Access können Sie mit dem VBA-Befehl **GetOption** einlesen und mit **SetOption** schreiben – so auch in diesem Fall. Dies nutzen wir zunächst beim Öffnen des Formulars, um den Wert der Optionsgruppe entsprechend einzustellen. Dies erledigt die Ereignisprozedur, die durch das Ereignis **Beim Laden** ausgelöst wird (Eigenschaft auf **[Ereignisprozedur]** einstellen, dann auf die Schaltfläche mit den drei Punkten klicken und die im VBA-Editor erscheinende Prozedur ergänzen) – siehe Listing 1.

Wenn Sie die Option nun ändern, soll diese gleich in die entsprechende Access-Option übertragen werden. Dazu benötigen wir eine weitere Prozedur, die diesmal durch das Ereignis **Nach Aktualisierung** der Optionsgruppe **ogrCursorverhalten** ausgelöst wird (siehe Listing 2).

Beim Einstellen der Option bleibt immerhin der Hinweis von Access aus, dass die Anwendung erst neu gestartet werden muss, damit die Option wirksam wird.

Nicht optimal

Ein Problem bleibt jedoch bestehen: Die Änderung bezieht sich jeweils auf den aktuellen Rechner, und sie wirkt sich auch auf alle im Anschluss geöffneten Access-Anwendungen aus. Am besten wäre es daher, man würde gleich zu Beginn prüfen, welchen Wert die Option aktuell besitzt, den eigenen, gewünschten Wert per VBA einstellen und diesen anschließend auf den vorgefundenen Wert zurücksetzen.

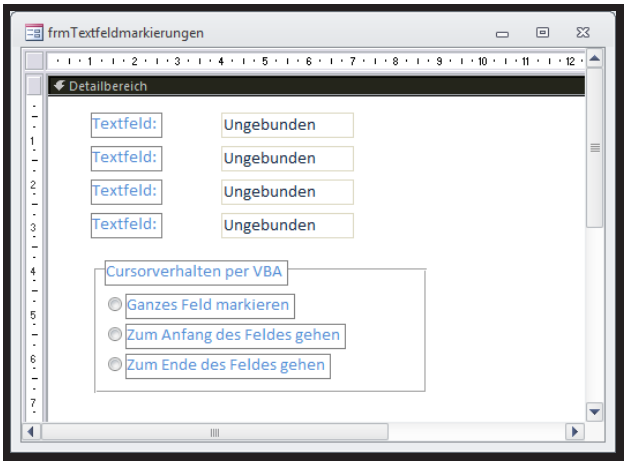


Bild 3: Einstellen des Cursorverhaltens per Formular/VBA

Das hört sich wirklich kompliziert an und ist in vielen Fällen auch gar nicht nötig. Otto Normalverbraucher ist in der Regel mit den Standardeinstellungen zufrieden, und für Sonderfälle gibt es wesentlich elegantere Lösungen. Und damit können Sie das Verhalten der Einfügemarke in Textfeldern sogar noch viel genauer steuern!

Handbetrieb

Falls Sie es noch nicht wussten: Es gibt natürlich auch für den Benutzer genügend Möglichkeiten, die Markierung beziehungsweise die Position der Einfügemarke anzupassen:

- Ein Doppelklick in den Text markiert das aktuelle Wort komplett.
- Ein einfacher Klick auf das komplett markierte Textfeld hebt die Markierung auf und positioniert die Einfügemarke an der angeklickten Stelle.
- **F2** hebt die aktuelle Markierung auf und platziert die Einfügemarke am Ende des Textes oder markiert den kompletten Text, sofern dieser aktuell nicht komplett markiert ist.
- Bei gedrückter Umschalttaste und den Tasten **Nach links** und **Nach rechts** erweitern Sie die aktuelle Markierung entsprechend.
- Bei gleichzeitig gedrückter Strg-Taste wird die Markierung jeweils um ein Wort erweitert (also bis zum nächsten Leerzeichen be-

```
Private Sub Form_Load()
    Me!ogrCursorverhalten = GetOption("Behavior Entering Field")
End Sub
```

Listing 1: Cursorverhalten abfragen und darstellen

```
Private Sub ogrCursorverhalten_AfterUpdate()
    SetOption "Behavior Entering Field", Me!ogrCursorverhalten
End Sub
```

Listing 2: Cursorverhalten per VBA einstellen

ziehungsweise bis zum Ende des Textes).

Lokale Einstellung

In der Tat können Sie das Verhalten für jedes einzelne Textfeld detailliert festlegen. Auch hier ist natürlich der Einsatz von VBA vonnöten – aber eine Access-Anwendung lässt sich nun einmal nicht nur mit Assistenten und ein paar Mausklicks zusammenstellen.

Eigenschaften für die Markierung

Zum Definieren der Markierung in einem Textfeld hat Microsoft uns die Eigenschaften **SelLength**, **SelStart** und **SelText** markiert. Diese haben die folgende Bedeutung:

- **SelStart**: Legt die Position des Beginns der Markierung fest.
- **SelLength**: Legt die Anzahl der markierten Zeichen fest.
- **SelText**: Gibt den markierten Text zurück.

Um die Werte dieser Felder zu beobachten, fügen Sie dem Beispielformular vier Textfelder mit den folgenden Namen hinzu (siehe Bild 4):

- **txtBeispiel**
- **txtSelStart**
- **txtSelLength**
- **txtSelText**

Das erste Feld soll Beispieltexte aufnehmen und verschiedene Markierungen erlauben. Die übrigen Felder sollen die jeweiligen Werte der Eigenschaften **SelStart**, **SelLength** und **SelText** ausgeben. Fehlt nur noch ein Mechanismus, der bei Änderungen des Textes beziehungsweise der Markierung im Textfeld **txtBeispiel** die Inhalte der Felder zur Darstellung der Eigenschaftswerte aktualisiert.

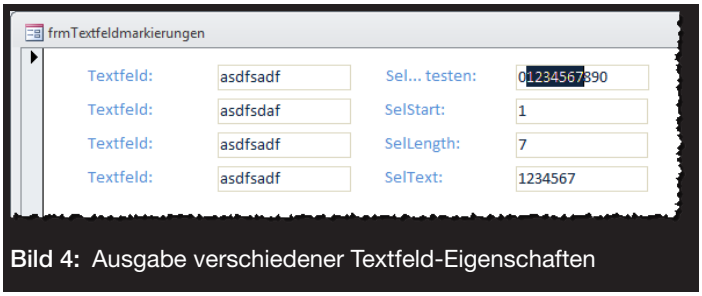


Bild 4: Ausgabe verschiedener Textfeld-Eigenschaften

Dies sollte nicht nur nach der Betätigung der Tastatur, sondern auch nach dem Platzieren der Einfügemarke oder dem Markieren von Textabschnitten mit der Maus geschehen.

Dazu bieten sich zwei Ereignisse an: **Bei Maustaste auf** und **Bei Taste auf** des Textfeldes **txtBeispiel**. Für das Ereignis **Bei Taste auf** legen Sie die Ereignisprozedur aus Listing 3 an.

Die Prozedur liest schlicht die Eigenschaftswerte der drei Eigenschaften **SelStart**, **SelLength** und **SelText** ein und schreibt sie in die drei dafür vorgesehenen Textfelder. Für die Prozedur, die durch das Ereignis **Bei Mauszeiger auf** ausgelöst wird, legen Sie übrigens genau die gleichen Befehle an.

```
Private Sub txtBeispiel_KeyUp(KeyCode As Integer, Shift As Integer)
    Me!txtSelStart = Me!txtBeispiel.SelStart
    Me!txtSelLength = Me!txtBeispiel.SelLength
    Me!txtSelText = Me!txtBeispiel.SelText
End Sub
```

Listing 3: Textfeldeigenschaften nach Tastaturbetätigung abfragen

Nun sind Sie an der Reihe: Schreiben Sie einen Text in das Textfeld, bewegen Sie die Einfügemarke mit den Cursortasten nach links oder rechts oder markieren Sie mit den Cursortasten oder der Maus Bereiche des Textes. Die drei Textfelder liefern akkurat die aktuellen Werte der entsprechenden Eigenschaften.

Dabei lernen wir beispielsweise, dass **SelStart** den Wert **0** liefert, wenn sich die Einfügemarke an der ganz linken Position befindet oder dass die Länge des Textes bei fehlender Markierung ebenfalls **0** ist.

Markierungen einstellen

Die drei Eigenschaften **SelStart**, **SelLength** und **SelText** können Sie übrigens auch selbst einstellen. Und das ist gar nicht einmal unpraktisch, wie die folgenden Beispiele zeigen werden.

Um dies zu demonstrieren, haben wir zunächst drei Textfelder mit den Namen **txtCursorLinks**, **txtCursorRechts** und **txtAllesMarkieren** angelegt (siehe Bild 5). Das erste Feld soll nun beim Fokuserhalt die Einfügemarke ganz links anzeigen, das zweite ganz rechts und das dritte soll den kompletten enthaltenen Text markieren.

Rekursion mit VBA

Manche Abläufe erfordern den Einsatz von Schleifen mit einer bestimmten Anzahl von Durchläufen oder einer vordefinierten Abbruchbedingung. In speziellen Fällen reichen Schleifen jedoch nicht aus, um zum Ziel zu kommen: Dann müssen rekursiv definierte Funktionen her. Dies sind solche Funktionen, die sich selbst aufrufen. Der vorliegende Artikel erklärt, wie solche Funktionen arbeiten und liefert einige Beispiele.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1307_RekursionMitVBA.mdb**.

Beispiel Fakultät

Bei der rekursiven Programmierung ruft sich eine Funktion selbst auf. Für einen einfachen Einstieg nehmen wir ein Beispiel, dass vielerorts zum Einsatz kommt: Die Berechnung der Fakultät einer Zahl. Die Fakultät einer Zahl **n** wird so definiert:

```
n! = n x (n-1) x ... x 2 x 1
```

Das bedeutet, dass man alle ganzen Zahlen miteinander multipliziert, die kleiner oder gleich der Zahl n sind. Ein Sonderfall ist die Fakultät von **0** – hierfür ist nämlich der Wert **1** definiert.

Um den Schritt zur Rekursion zu machen, definieren wir die Fakultät etwas anders:

```
n! = n x (n-1)!
```

Die Fakultät ist hier als das Produkt der Zahl selbst mit der Fakultät der nächstkleineren Zahl definiert.

Und diese Fakultät entspricht natürlich wiederum die Zahl multipliziert mit der Fakultät der nächstkleineren Zahl. Es gibt hier auch eine Ausnahme: Wenn **n** den Wert **1** annimmt, ist die Fakultät von **n** gleich **1**.

Fakultät per VBA

Wenn wir die Fakultät mit einer Funktion berechnen möchten, die sich selbst aufruft, müssen wir dieser zunächst einmal die Zahl übergeben, für welche die Fakultät berechnet werden soll. Der Funktionskopf sieht also beispielsweise so aus:

```
Public Function Fakultaet(lngZahl _
                As Long) As Long
```

Beim ersten Aufruf soll die Funktion die Fakultät für die Zahl **lngZahl** ermitteln. Diese entspricht aber wiederum dem Produkt der Zahl und der Fakultät der Zahl minus Eins.

Innerhalb der Funktion rufen wir die Funktion also erneut auf – mit der um Eins verminderten Zahl. Das Ergebnis multiplizieren wir dann mit der Zahl selbst. Die Anweisung sieht also so aus:

```
Fakultaet = lngZahl *
Fakultaet(lngZahl - 1)
```

Insgesamt erhielten wir nun eine Funktion wie die Folgende:

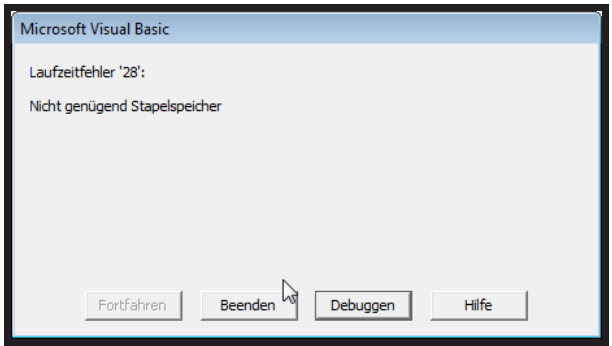


Bild 1: Fehler bei rekursiver Funktion ohne Abbruchbedingung

```
Public Function Fakultaet(lngZahl _
                As Long) As Long
    Fakultaet = lngZahl * _
                Fakultaet(lngZahl - 1)
End Function
```

Wenn Sie die Funktion nun etwa aus dem Direktfenster mit dem Befehl **Debug.Print Fakultaet(4)** aufrufen, sollte das Ergebnis eigentlich 24 lauten (**1 x 2 x 3 x 4**). Allerdings zeigt Access nach kurzer Zeit eine Fehlermeldung an, die wie in Bild 1 aussieht. Was ist hier passiert? Ganz einfach: Wir haben der Funktion nicht mitgeteilt, wann sie anhalten soll. In diesem Fall berechnet sie nacheinander die Fakultät für **4**, **3**, **2**, **1**, **0**, **-1**, **-2** und so weiter.

Der Fehler wird schließlich dadurch ausgelöst, dass die Funktion die Zwischenergebnisse jeweils im Arbeitsspeicher unterbringen muss – und

mit wachsender Anzahl Aufrufe ist der Speicher irgendwann einmal voll.

Also bauen wir die Funktion wie in Listing 1 auf. Die Funktion prüft bei jedem Aufruf zunächst, ob der Wert der mit **lngZahl** gelieferten Zahl gleich Eins ist. Ist dies der Fall, ist auch die Fakultät Eins – die Funktion braucht also kein weiteres Mal aufgerufen zu werden. Der Ablauf zur Berechnung der Fakultät von 4 sieht also so aus:

- Erster Aufruf mit **lngZahl = 4**
- Zweiter Aufruf mit **lngZahl = 3**
- Dritter Aufruf mit **lngZahl = 2**
- Vierter Aufruf mit **lngZahl = 1**: Fakultaet liefert **1** zurück, der vierte Aufruf ist beendet.
- Im dritten Aufruf wird der Funktionswert **Fakultaet** auf **2 x 1 = 2** eingestellt, der dritte Aufruf ist beendet.
- Im zweiten Aufruf wird der Funktionswert **Fakultaet** auf **3 x 2 = 6** eingestellt, der zweite Aufruf ist beendet.
- Im ersten Aufruf wird der Funktionswert **Fakultaet** auf **4 x 6 = 24** eingestellt, der erste Aufruf ist beendet und gibt als Ergebnis **24** zurück.

Hier wird auch gleich deutlich, dass eine rekursiv definierte Funktion immer Aktionen vor dem Aufruf der nächsten Ebnee ausführen kann, aber auch Aktionen nach diesem Aufruf. In diesem Fall tritt die Zuweisung des Ergebnisses immer erst nach dem Aufruf der nächsten Ebene und deren Abarbeitung auf.

```
Public Function Fakultaet(lngZahl As Long) As Double
    If lngZahl = 1 Then
        Fakultaet = 1
    Else
        Fakultaet = lngZahl * Fakultaet(lngZahl - 1)
    End If
End Function
```

Listing 1: Rekursive Berechnung der Fakultät einer Zahl

```
Public Function FakultaetIterativ(lngZahl As Long) As Double
    Dim dblFakultaet As Double
    Dim i As Integer
    FakultaetIterativ = 1
    For i = 1 To lngZahl
        FakultaetIterativ = i * FakultaetIterativ
    Next i
End Function
```

Listing 2: Iterative Berechnung der Fakultät

Iteration statt Rekursion

In vielen Fällen kann man die Rekursion durch eine Iteration ersetzen. In unserem Fall sähe eine entsprechende Funktion wie in Listing 2 aus. Die Funktion **FakultaetIterativ** durchläuft eine **For...Next**-Schleife über die Werte von **1** bis zu der Zahl, deren Fakultät ermittelt werden soll.

Dabei multipliziert sie einfach das jeweils in **FakultaetIterativ** gespeicherte Ergebnis, das zu Beginn auf **1** eingestellt wird, mit dem aktuellen Wert der Laufvariablen **i** – also **1 x 1 = 1** im ersten Durchlauf, **2 x 1 = 2** im zweiten, **3 x 2 = 6** im dritten und **6 x 4 = 24** im vierten Durchlauf.

Die Vorteile dieser Variante liegen im geringeren Speicherbedarf und daraus resultierend in einer besseren Performance.

Allerdings lassen sich viele Probleme, die sich mit einfachen rekursiven Funktionen abbilden lassen, nicht durch Iteration darstellen. Das hier genannte Beispiel der Fakultät ist geradlinig – jeder Funktionsaufruf führt zu maximal einem weiteren Funktionsaufruf. Es gibt jedoch auch Probleme, bei denen innerhalb der rekursiven Funktion mehrfache Aufrufe der Funktion stattfinden – beispielsweise innerhalb einer Schleife.

Dafür gibt es einige Beispiele, zum Beispiel diese:

- Einlesen der Verzeichnisse und Dateien: Hier kann jedes Verzeichnis Unterverzeichnisse enthalten, die wiederum eingelesen werden müssen.
- Hierarchische Daten wie Vorgesetzte/Untergebene oder Stücklisten, bei denen jedes Bauteil aus einem oder mehreren anderen Teilen besteht

Vorgesetze und Untergebene

Das wohl bekannteste Beispiel für den Einsatz von Rekursion unter

Access und VBA ist das der Mitarbeiterhierarchie.

Hier soll es einen oder mehrerer Vorgesetzte einer obersten Ebene geben, die sich dadurch auszeichnen, dass sie keinen Vorgesetzten haben. Alle anderen Mitarbeiter haben einen Vorgesetzten – entweder den der obersten Ebene oder einen darunter angesiedelten Mitarbeiter.

Typischerweise legt man das Datenmodell so an, dass man die sogenannte reflexive Beziehung innerhalb einer einzigen Tabelle abbildet. Dies sieht so aus, dass etwa die Tabelle **tblMitarbeiter** wie in Bild 2 ein Feld namens **VorgesetzterID** aufweist, dass mit dem Primärschlüsselfeld der gleichen Tabelle verknüpft ist.

Im Beziehungsfenster haben wir die Tabelle **tblMitarbeiter** zweimal hinzugefügt, um die Beziehung anlegen zu können. Man kann dann wie in Bild 3 den jeweiligen Vorgesetzten in das Feld **VorgesetzterID** eintragen.

Der Mitarbeiter mit dem Wert **1** im Feld **MitarbeiterID** ist der einzige Mitarbeiter der höchsten Ebene. Die Mitarbeiter mit den Werten **2** und **3** im Feld **MitarbeiterID** sind diesem direkt untergeordnet. Die übrigen Mitarbeiter haben die Mitarbeiter der zweiten Ebene als Vorgesetzte.

Es gibt noch eine zweite, weniger bekannte Variante der Zuordnung von Vorgesetzten und Untergebenen. Dabei enthält die Mitarbeitertabelle selbst kein Feld, um die Zuordnung vorzunehmen.

Stattdessen erledigen Sie dies mit einer weiteren Tabelle, welche die beiden an einer Zuordnung beteiligten Mitarbeiter als Vorgesetzten und Untergebenen definiert. Die für dieses Beispiel verwendete Tabelle **tblMitarbeiterOhne** ist prinzipiell wie die Tabelle **tblMitarbeiter** aufgebaut, es fehlt jedoch das Feld **VorgesetzterID**.

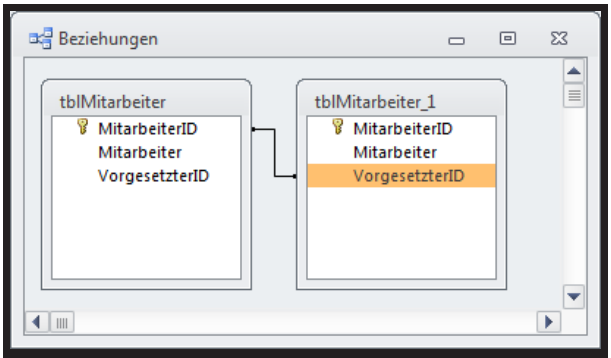


Bild 2: Vorgesetzte, Variante I

Bild 3: Werte der Tabelle **tblMitarbeiter** inklusive der Vorgesetzten

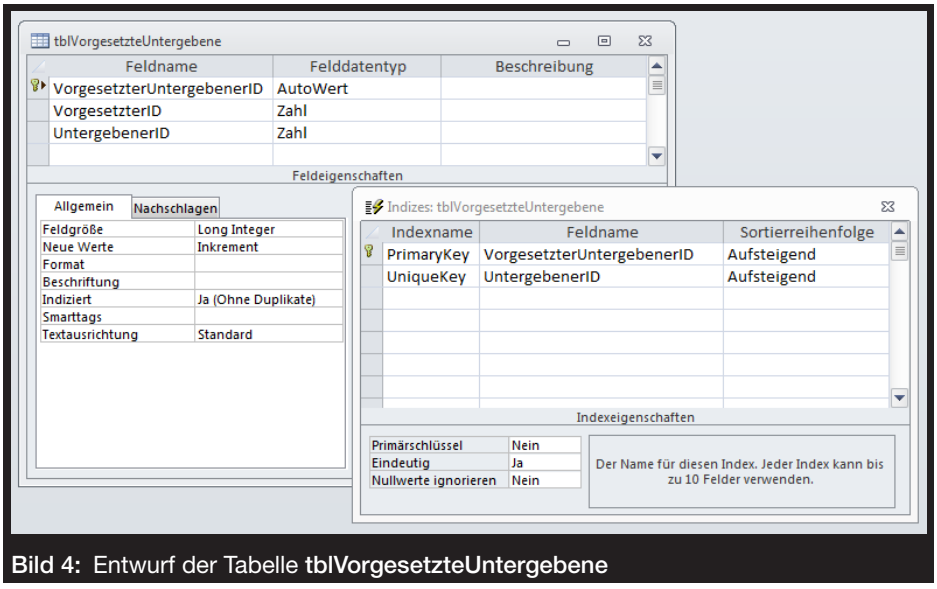


Bild 4: Entwurf der Tabelle **tblVorgesetzteUntergebene**

Dafür gibt es nun eine weitere Tabelle namens **tblVorgesetzteUntergebene**, die neben einem Primärschlüsselfeld zwei Felder namens **VorgesetzterID** und **UntergebenerID** enthält. Die Tabelle ist wie in Bild 4 aufgebaut. Außerdem müssen Sie auch noch einen eindeutigen Schlüssel für das Feld **UntergebenerID** festlegen, damit jeder Untergebene nur einem Vorgesetzten zugeordnet werden kann.

Platzhalter in Texten ersetzen

In Datenbanken kommt es sehr oft vor, dass an bestimmten Stellen Texte einzufügen sind. Meist bildet man dies durch den Einsatz von Textfeldern ab, welche die Daten der entsprechenden Datensätze aufnehmen. Oft sollen jedoch auch Fließtexte flexibel mit Texten gefüllt werden – beispielsweise Texte von E-Mails oder Word-Dokumente. Wir kümmern uns in diesem Artikel um den weniger komplexen Fall der reinen Textdatei, wie er auch in E-Mails zum Einsatz kommt. Eine E-Mail soll mit einem Text gefüllt werden, der zunächst noch Platzhalter enthält, die aber durch entsprechende Einträge einer Tabelle oder Abfrage ersetzt werden sollen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1307_PlatzhalterInTextenErsetzen.mdb**.

Beispielbeschreibung

Im Beispiel sollen ein Text nur einmal getippt, aber mehrfach genutzt werden können – quasi als eine Art Textvorlage. Diese Vorlage soll Platzhalter in einem bestimmten Format enthalten, die später durch Einträge in einer Tabelle der Datenbank ersetzt werden sollen.

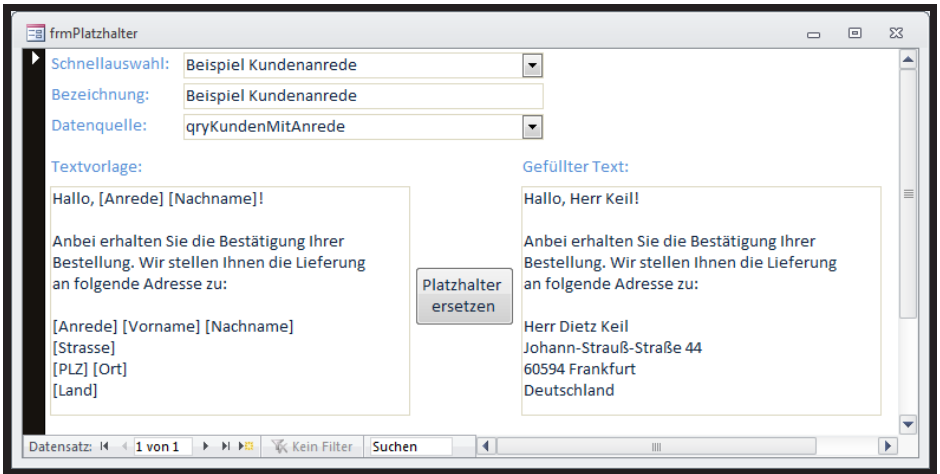


Bild 1: Beispielformular für das Ersetzen von Platzhaltern

Ziel ist es, ein Formular wie das aus Bild 1 zu erstellen. Das Formular speichert die auf der linken Seite dargestellten Daten in einer Tabelle, um Textvorlagen und Quelltabellen wiederverwendbar zu machen. Oben wählen Sie die Textvorlage anhand der Bezeichnung aus.

Dann legen Sie die Datenquelle fest, aus der die für die Platzhalter einzusetzenden Daten stammen. Das Textfeld **Textvorlage** schließlich nimmt den Text mit den Platzhaltern auf. Ein Klick auf die Schaltfläche **Platzhalter ersetzen** soll den Text von links mit konkreten Werten aus der angegebenen Tabelle füllen und den Inhalt rechts abbilden.

Zu Beispielzwecken geschieht das Ersetzen zunächst nur mit dem ersten Datensatz der angegebenen Datenquelle.

Tabelle zum Speichern der Textvorlagen

Die Textvorlagen speichern wir in einer Tabelle namens **tblTextvorlagen**. Diese enthält neben dem Primärschlüsselfeld **TextvorlageID** noch ein Textfeld namens **Bezeichnung** sowie ein Memofeld namens **Textvorlage**, welche den eigentlichen Text aufnimmt. Außerdem bereiten wir ein Feld namens **Datenquelle** vor, dass die Tabelle oder Abfrage aufnimmt, deren Daten für die Platzhalter eingesetzt werden soll.

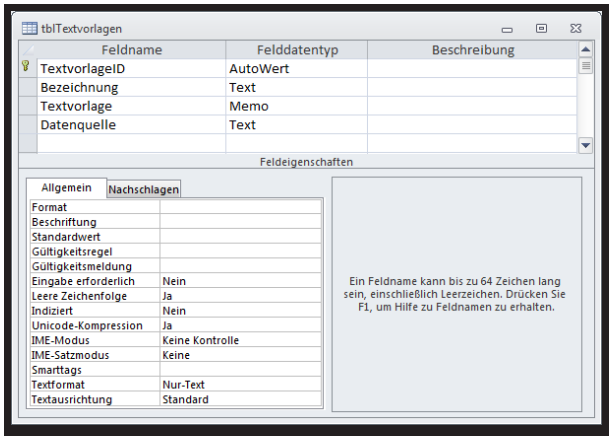


Bild 2: Entwurfsansicht der Tabelle **tblTextvorlagen**

Den Entwurf der Tabelle finden Sie in Bild 2.

Die im Feld **Textvorlage** gespeicherten Texte sollen Platzhalter in der Form **[Platzhalter]** aufnehmen. Diese sollen dann für einen bestimmten Datensatz der dafür vorgesehenen Tabelle oder Abfrage mit den entsprechenden Werten ersetzt werden.

Formulare zum Ersetzen der Platzhalter

Erstellen Sie ein neues Formular und speichern Sie es unter dem Namen **frmPlatzhalter**. Stellen Sie dann die Datenherkunft des Formulars auf die Tabelle **tblTextvorlagen** ein. Ziehen Sie die Felder **Bezeichnung**, **Datenquelle** und **Textvorlage** in den Entwurf des Formulars und platzieren Sie diese etwa so wie in Bild 3.

Ganz oben legen wir ein Kombinationsfeld an, mit dem Sie schnell eine der Textvorlagen auswählen können. Dieses heißt **cboSchnellauswahl** und erhält die folgende Abfrage als Datensatzherkunft:

```
SELECT TextvorlageID, Bezeichnung
FROM tblTextvorlagen;
```

Stellen Sie die Eigenschaften **Spaltenanzahl** und **Spaltenbreiten** auf die Werte **2** und **0cm** ein, damit nur die Bezeichnungen der Textvorlagen im Kombinationsfeld angezeigt werden.

Damit das Formular nach der Auswahl eines der Einträge gleich den entsprechenden Datensatz anzeigt, hinterlegen Sie für die Ereigniseigenschaft **Nach Aktualisierung** die Prozedur aus Listing 1.

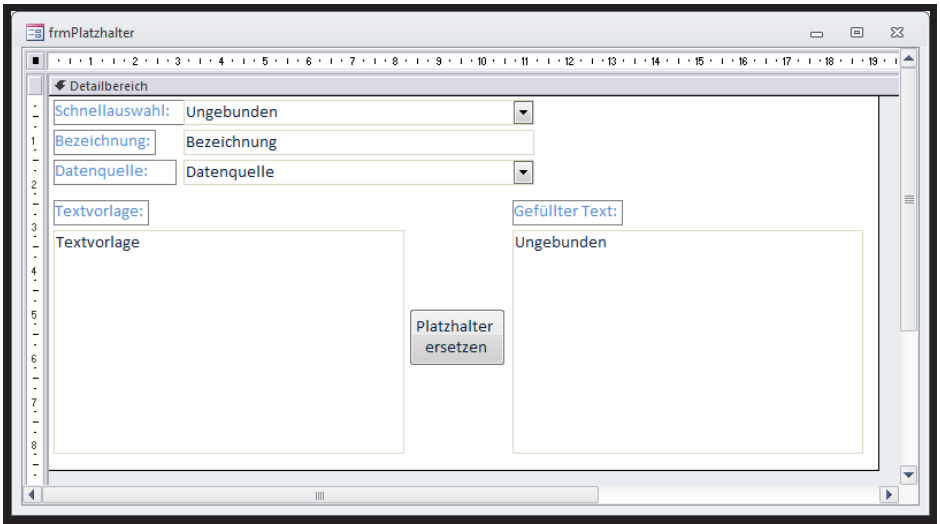


Bild 3: Entwurfsansicht des Formulars **frmPlatzhalter**

```
Private Sub cboSchnellauswahl_AfterUpdate()
    Me.Recordset.FindFirst "TextvorlageID = " & Me!cboSchnellauswahl
End Sub
```

Listing 1: Aktualisieren der Textvorlage

```
Private Sub cmdPlatzhalterErsetzen_Click()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim fld As DAO.Field
    Dim strText As String
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT TOP 1 * FROM " & Me!cboDatenquelle)
    strText = Me!Textvorlage
    For Each fld In rst.Fields
        strText = Replace(strText, "[" & fld.Name & "]", fld.Value)
    Next fld
    Me!txtGefuellterText = strText
End Sub
```

Listing 2: Ersetzen der Platzhalter durch die Inhalte der Felder des ersten Datensatzes der angegebenen Datenquelle

Das Feld **Datenquelle** des Formulars wandeln Sie nach dem Hineinziehen aus der Feldliste von einem Textfeld in ein Kombinationsfeld um. Dies erledigen Sie mit dem Eintrag **Ändern zu|Kombinationsfeld** des Steuerelements in der Entwurfsansicht des Formulars. Damit das Kombinationsfeld alle Tabellen und Abfragen der aktuellen Datenbank anzeigt, stellen sie die Eigenschaft **Datensatzherkunft** des Kombinationfelds auf den folgenden Ausdruck ein:

```
SELECT Name FROM MSysObjects
WHERE Type IN (1,5);
```

Platzhalter ersetzen

Nun fehlt eigentlich nur noch die Ereignisprozedur, die durch das Er-

Berichte per E-Mail verschicken, Teil III

Im dritten und letzten Teil dieser Artikelreihe schauen wir uns an, wie Sie die E-Mails zum Versenden der Berichte noch professioneller befüllen. Dazu verwenden wir eine Funktion, die wir im Artikel **Platzhalter in Texten ersetzen** vorstellen. Fehlt nur noch eine kleine Erweiterung der Benutzeroberfläche unserer Beispielanwendung, um Vorlagen zu verwalten und diese zu verwenden.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1307_BerichtePerEMail.mdb**.

Vorbereitungen

Wie bereits erwähnt, haben wir im Artikel Platzhalter in Texten ersetzen bereits einige Vorarbeiten geleistet. Aus der Beispieldatenbank zu diesem Artikel haben wir nun einige Objekte in die Datenbank zum vorliegenden Artikel gezogen. Dabei handelt es sich um die folgenden Objekte:

- Tabelle **tblTextvorlagen**
- Formular **frmPlatzhalter**
- Modul **mdlPlatzhalter**

Formular frmRechnungsversand anpassen

Als erstes kümmern wir uns um die Anpassung des Formulars **frmRechnungsversand**. Dieses erhält, wie in Bild 1 dargestellt, zwei neue Kombinationfelder namens **cboVorlageBetreff** und **cboVorlageMail**. Immerhin kann es ja sein, dass nicht nur der Inhalt der E-Mail dynamisch gefüllt werden soll – auch der Betreff kann ja durchaus angepasst werden. Ein Beispiel ist die Angabe der Bestellnummer oder Rechnungsnummer, damit der Kunde die entsprechende Mail direkt wiederfindet. Oder Sie

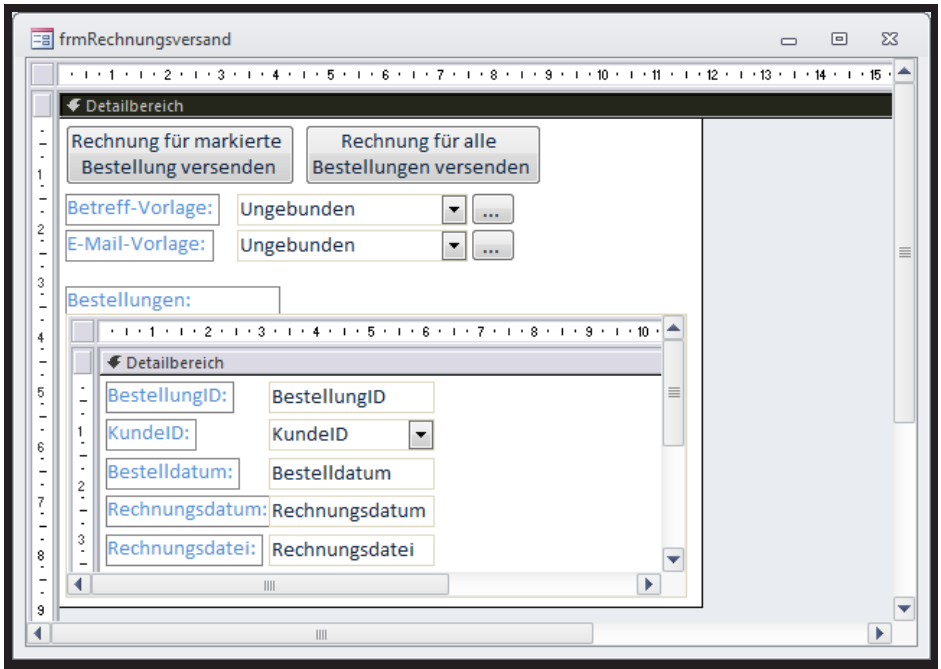


Bild 1: Kombinationsfelder zum Auswählen der Vorlagen für Betreff und E-Mail

versehen eine E-Mail mit dem Namen des Empfängers – manch ein Newsletter-Marketingspezialist schwört auf diese Methode, um Aufmerksamkeit zu gewinnen.

Beide Kombinationsfelder statten wir mit der gleichen Datensatzherkunft aus, und zwar mit der folgenden:

```
SELECT tblTextvorlagen.TextvorlageID,
tblTextvorlagen.Bezeichnung FROM
tblTextvorlagen ORDER BY tblTextvorla-
gen.Bezeichnung;
```

Wenn Sie nun noch die Eigenschaften **Spaltenanzahl** und **Spaltenbreiten** auf die Werte **2** und **0cm** einstellen, zeigen beide Kombinationsfelder die Bezeichnungen der Textvorlagen aus der Tabelle **tblTextvorlagen** an.

Nach der Auswahl eines der Einträge soll nichts geschehen – die Vorlagen werden erst nach dem Anklicken einer der beiden Schaltflächen zum Versenden der aktuell markierten oder aller Rechnungen entsprechend der Textvorlage gefüllt.

Allerdings wollen wir dem Benutzer noch die Möglichkeit geben, die Textvorlagen zu bearbeiten – und zwar über die beiden Schaltflächen rechts neben den Kombinationsfeldern. Die beiden Schaltflächen heißen **cmdVorlagenBetreffAnzeigen** und **cmdVorlagenMailAnzeigen**. Die Ereig-

Das TreeView-Steuerelement: Grundlagen, Teil I

Das TreeView-Steuerelement ist der Alleskönner zur Anzeige hierarchischer Daten. Sie zeigen damit sowohl die Daten aus Tabellen mit reflexiver Verknüpfung an als auch Daten, die aus verschiedenen verknüpften Tabellen stammen – wie etwa Kunden und Projekte. Das Steuerelement kann als ActiveX-Steuerelement im Gegensatz zu den eingebauten Steuerelementen von Access nicht direkt an eine Datenherkunft gebunden werden, also ist ein wenig Handarbeit nötig, um dieses zu füllen. In dieser Artikelreihe lernen Sie die Grundlagen zu diesem Steuerelement kennen und erfahren, wie Sie es mit Daten füllen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1307_TreeView.mdb**.

Hinweis

Unter Access 2013 gibt es ein paar Haken und Ösen, was den Einsatz des **TreeView**-Steuerelements angeht. Daher zeigen wir die Beispiele zu dieser Artikelreihe anhand der Access-Version 2013 vor.

Und noch ein Hinweis: Wenn Sie den Einsatz von ActiveX-Steuerelementen wie **ImageList**, **ListView** oder **TreeView** planen, sollten Sie die 32bit-Variante der jeweiligen Office-Version installieren. Die 64bit-Versionen unterstützen diese Steuerelemente nicht.

TreeView-Steuerelement anlegen

TreeView-Steuerelemente setzt man in Formularen ein. Nachdem Sie ein neues Formular erstellt haben, fügen Sie dort ein neues **TreeView**-Steuerelement ein. Das **TreeView**-Steuerelement finden Sie im Ribbon nicht unter den eingebauten Steuerelementen, sondern in den ActiveX-Steuerelementen.

Die Liste der ActiveX-Steuerelemente zeigen Sie an, indem Sie im Ribbon unter **Entwurf|Steuerelemente** die Liste der Steuerelemente aufklappen und den Eintrag **ActiveX-Steuerelemente** auswählen (siehe Bild 1). Nun erscheint ein Dialog namens **ActiveX-Steuerelement einfügen**, mit dem Sie das **TreeView**-Steuerelement auswählen können (siehe Bild 2). Nach einem Klick auf die **OK**-Schaltfläche zeigt Access das Steuerelement zunächst ziemlich unscheinbar im Formular an (siehe Bild 3).

Vergrößern Sie das **TreeView**-Steuerelement nun auf die gewünschte Größe und legen Sie den Namen des Steuerelements über das Eigenschaftsfenster auf **ctlTreeView** fest.

Benennung

Bei eingebauten Steuerelemente verwenden wir in der Regel dem Steuerelement entsprechende Präfixe wie

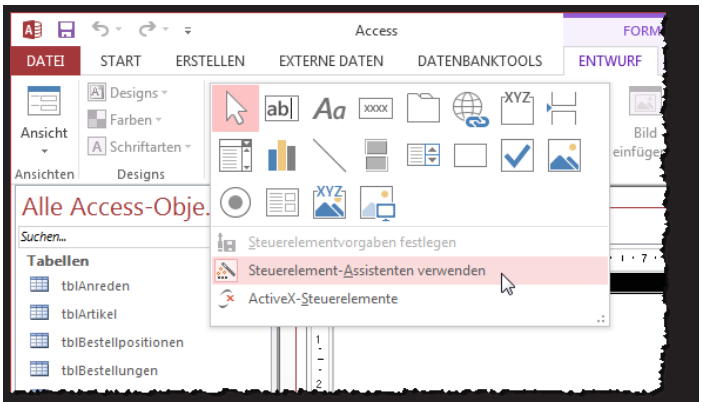


Bild 1: Aufrufen der Liste der ActiveX-Steuerelemente

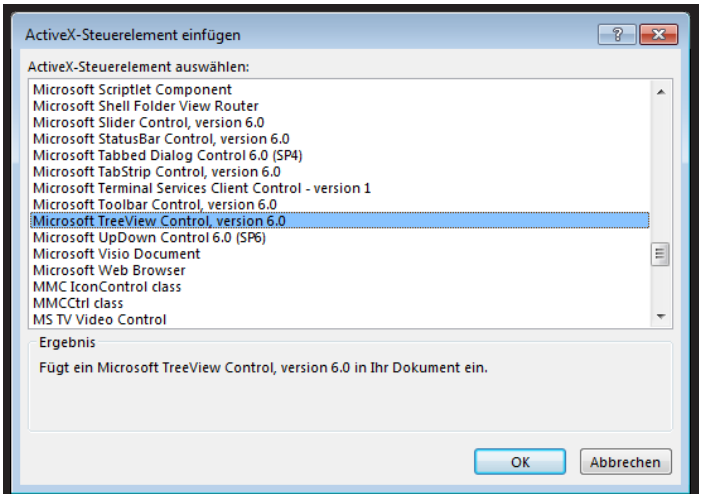


Bild 2: Auswahl des TreeView-Steuerelements

txt, **cmd**, **cbo** und so weiter. Eigentlich sollte man für das **TreeView**-Steuerelement dementsprechend **tvw** oder für das **ListView**-Steuerelement **lvw** verwenden. Allerdings wird sich gleich zeigen, dass wir im VBA-Code nicht direkt auf die Steuerelementnamen zugreifen (zumindest in vielen Fällen), sondern eine Objektvariable erstellen, um das Steuerelement und seine Methoden, Ereignisse und Eigenschaften zu referenzieren. Wir können also im Code nicht immer auf den Steuerelementnamen zugreifen.

Im Laufe meiner Zeit als Access-Entwickler habe ich mir angewöhnt, das TreeView-Steuerelement selbst mit **ctlTreeView** zu benennen (wobei **ctl** für **Control** steht) und im Code die Bezeichnung **objTreeView** zu verwenden – **obj** steht hier für Objekt(variable). In den meisten Fällen ist das kein Problem, da selten mehr als ein **ListView**- oder **TreeView**-Steuerelement pro Formular zum Einsatz kommt. Und sollte dies doch einmal der Fall sein, verwenden Sie eben Bezeichnungen wie **ctlTvwVerzeichnisse** und **objTvwVerzeichnisse**.

TreeView-Eigenschaften

Bevor wir das TreeView-Steuerelement mit den ersten Daten füllen, schauen wir uns seine Eigenschaften an. Dabei stellen wir fest, dass das Eigenschaftsfenster nur einige wenige Standardeigenschaften vorhält, die für alle ActiveX-Steuerelement gleich sind.

Wenn Sie in die Tiefe gehen wollen, müssen Sie das eigentliche Eigenschaftsfenster des **TreeView**-Steuerelements öffnen. Dies erreichen Sie über den Kontextmenü-Eintrag **TreeView-Object|Properties** (siehe Bild 4).

Gegebenenfalls erwartet Sie nun der Dialog aus Bild 5. Im Gegensatz zur funktionierenden Variante sind hier in den meisten Fällen weder Eigenschaftswerte voreingestellt noch können Sie welche auswählen – die Kombinationsfelder zeigen leere Listen an.

TreeView/MSCOMCTL.ocx reparieren

Abhilfe schaffen Sie über die folgende Vorgehensweise (gilt auch für andere Versionen, in denen Probleme auftauchen): Prüfen Sie zunächst, ob sich auf Ihrem Rechner die neueste Version der Datei **MSCOMCTL.ocx** befindet. Diese finden Sie im Verzeichnis **C:\Windows\System32** (Windows 7 32bit, sonst gegebenenfalls **C:\Windows\SYSWOW64**). Die Version ermitteln Sie auf der Registerseite **Details** des Eigenschaften-Dialogs, den Sie über das Kontextmenü der Datei öffnen. Dort sollte für die Eigenschaft **Da-**

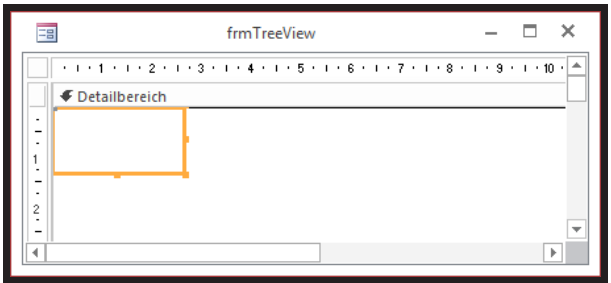


Bild 3: Ein frisches TreeView-Steuerelement im Formularentwurf

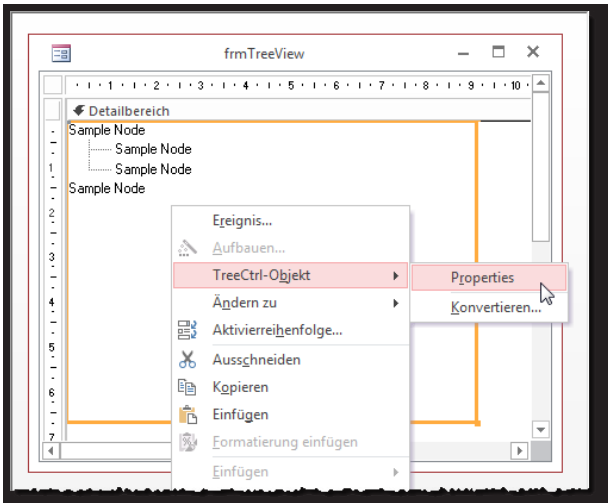


Bild 4: Aufruf des Eigenschaftsfensters eines ActiveX-Steuerelements

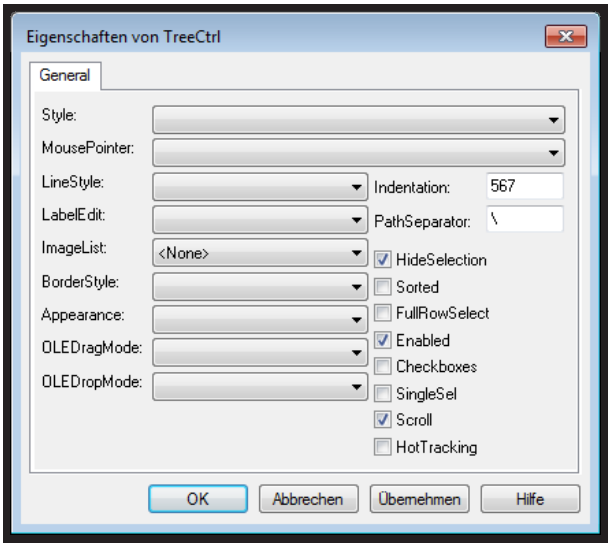


Bild 5: Die Eigenschaften des TreeView-Steuerelements – ohne Werte und Auswahlmöglichkeit

teiversion der Wert **6.1.98.34** eingetragen sein (Stand Mitte 2013).

Ist dies nicht der Fall, laden Sie diese Dateiversion herunter und überschreiben die vorhandene Datei damit.

Nun prüfen Sie den Eintrag für die Datei **MSCOMCTL.ocx** in der Registry. Die Registry starten Sie über das Startmenü von Windows mit dem Befehl **Regedit.exe**. Den passenden Eintrag finden Sie unter **HKEY_CLASSES_ROOT\TypeLib\{831FDD16-0C5C-11D2-A9FC-0000F8754DA1}**. Dort befinden sich entweder Einträge für die Version **2.0**, **2.1** oder beide (siehe Bild 6).

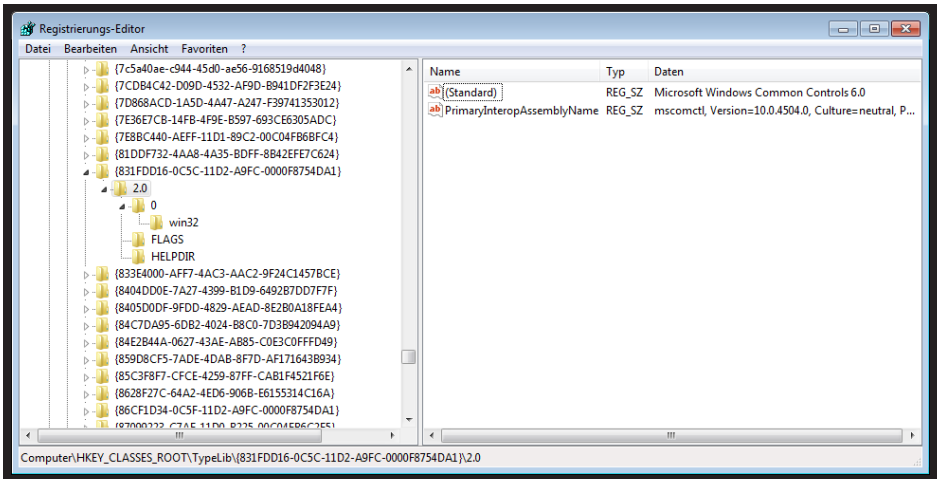


Bild 6: Eintrag für die Datei **MSCOMCTL.ocx** in der Registry

Entfernen Sie die Einträge, wir fügen diese gleich neu hinzu. Dazu starten Sie die Eingabeaufforderung von Windows. Geben Sie **cmd** im Startmenü von Windows ein.

Wenn ein Suchergebnis namens **cmd.exe** erscheint, klicken Sie mit der rechten Maustaste darauf und wählen aus dem Kontextmenü den Eintrag **Als Administrator ausführen** (siehe Bild 7). Es reicht nicht, wenn Sie mit Administratorrechten angemeldet sind!

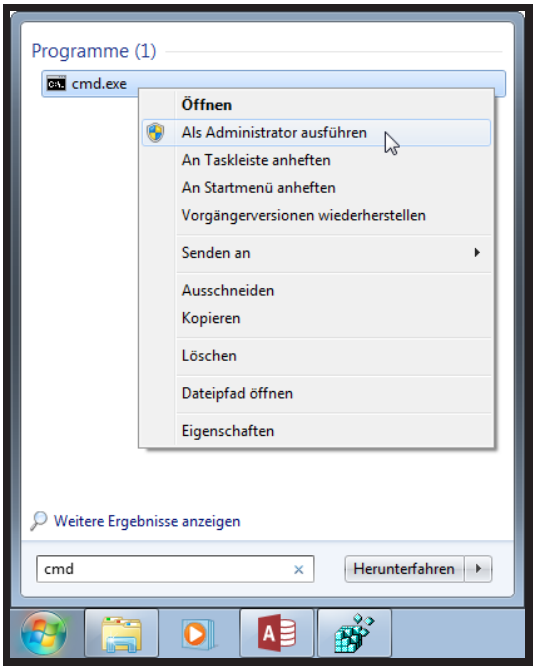


Bild 7: Öffnen der Kommandozeile

Dort geben Sie nun, wenn Sie sich bereits im Verzeichnis **c:\Windows\System32** beziehungsweise **c:\Windows\SYSWOW64** befinden, den folgenden Befehl ein:

```
Regsvr32.exe MSCOMCTL.ocx
```

Windows sollte nun eine Meldung wie die aus Bild 8 anzeigen. Wenn Sie nun erneut einen Blick in die Registry werfen, finden Sie dort unter **HKEY_CLASSES_ROOT\TypeLib\{831FDD16-0C5C-11D2-A9FC-0000F8754DA1}** allein den Eintrag 2.1 mit Untereinträgen vor.

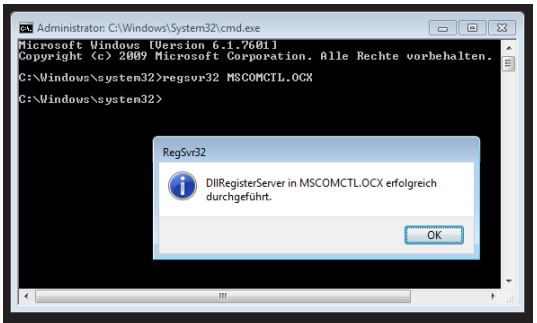


Bild 8: Registrieren der Datei MSCOMCTL.ocx

Wenn Sie nun Access schließen und wieder öffnen und erneut das Eigenschaftsfenster des TreeView-Steuerelements anzeigen, liefert dies das erwartete Bild (siehe Bild 9).

Übrigens treten auch in Zusammenhang mit älteren Access-Versionen, also etwa Access 2010, Probleme mit dem TreeView-Steuerelement auf. Die oben beschriebene Vorgehensweise sollte diese jedoch beheben.

Eigenschaften, zweiter Versuch

Werfen wir nun einen Blick auf die Eigenschaften und ihre Bedeutung:

Das TreeView-Steuerelement: Grundlagen, Teil II

Das TreeView-Steuerelement ist der Alleskönner zur Anzeige hierarchischer Daten. Sie zeigen damit sowohl die Daten aus Tabellen mit reflexiver Verknüpfung an als auch Daten, die aus verschiedenen verknüpften Tabellen stammen – wie etwa Kunden und Projekte. Das Steuerelement kann als ActiveX-Steuerelement im Gegensatz zu den eingebauten Steuerelementen von Access nicht direkt an eine Datenherkunft gebunden werden, also ist ein wenig Handarbeit nötig, um dieses zu füllen. In dieser Artikelreihe lernen Sie die Grundlagen zu diesem Steuerelement kennen und erfahren, wie Sie es mit Daten füllen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1308_TreeView.mdb**.

TreeView-Verlust vorbeugen

Gerade während der Entwicklung und beim Experimentieren mit den Möglichkeiten des TreeViews treten des öfteren Fehler auf. Wird ein Fehler nicht behandelt, führt dies dazu, dass Objektvariablen wie **objTreeView** zurückgesetzt und somit geleert werden.

Das bedeutet, das erstens keine für dieses Steuerelement programmierten Ereignisse mehr ausgelöst werden und dann der Zugriff per Code auf die entsprechende Objektvariable wiederum zu Fehlern führt.

Es gibt jedoch eine Möglichkeit, diese Variable bei Bedarf immer wieder neu mit einem Verweis auf das **TreeView**-Steuerelement zu füllen.

Dazu deklarieren Sie die Variable zum Speichern des Verweises ähnlich wie zuvor, diesmal allerdings unter einem anderen Namen – nämlich **m_TreeView**. Der Grund ist, dass wir den zuvor verwendeten Namen **objTreeView** als Name einer **Property Get**-Prozedur verwenden. Genaueres über solche Prozeduren erfahren Sie in einem späteren Artikel über die Klassenprogrammierung unter VBA – fürs erste reicht es, wenn Sie wissen, dass die **Property Get**-Prozedur ähnlich wie eine Funktion einen Wert oder einen Verweis zurückliefert.

```
Dim WithEvents m_Treeview As MSComctlLib.TreeView

Private Property Get objTreeView() As MSComctlLib.TreeView
    If m_Treeview Is Nothing Then
        Set m_Treeview = Me!ctlTreeView.Object
    End If
    Set objTreeView = m_Treeview
End Property
```

Listing 1: **objTreeView** liefert immer einen Verweis auf das **TreeView**-Steuerelement in **ctlTreeView**.

In diesem Fall prüft diese Prozedur, ob die Variable **m_TreeView** bereits gefüllt ist, und zwar durch den **Is**-Vergleich mit dem Ausdruck **Nothing**. Ist dies wahr, wurde die Variable entweder noch nicht befüllt oder ist durch einen Fehler wieder geleert worden.

In beiden Fällen soll diese jedoch gefüllt werden. Der Inhalt der Variablen wird dann, unabhängig davon, ob er erst jetzt gefüllt wurde oder bereits gefüllt war, mit der **Property Get**-Prozedur **objTreeView** zurückgegeben (siehe Listing 1).

objTreeView muss auf diese Weise nicht mehr in der **Form_Load**-Ereignisprozedur gefüllt werden.

Dieses Beispiel finden Sie im Formular **frmTreeView_mTreeView**.

TreeView-Eigenschaften per Code einstellen

Wenn Sie hier und da **TreeView**-Steurelemente einbauen, möchten Sie vielleicht gern die Eigenschaften wie bei einem **TreeView**-Steuerelement einstellen, mit denen bereits gute Erfahrungen in einer anderen Anwendung gemacht haben.

In meinen Anwendungen verzichte ich daher völlig auf die Einstellung der Eigenschaften über den entsprechenden Dialog des **TreeView**-Steuerelements im Formularentwurf. Ich stelle die Eigenschaften immer gleich beim Laden des Formulars per VBA-

Code ein. Für einige Standardeigenschaften, die sich auf das Aussehen des **TreeView**-Steuerelements beziehen, sieht dies etwa wie folgt aus:

```
Private Sub Form_Load()
    With objTreeView
        .Appearance = ccFlat
        .BorderStyle = ccNone
        .LineStyle = tvwRootLines
        .Style = 7
            tvwTreelinesPlusMinusText
    End With
End Sub
```

In den folgenden Beispielen kommen je nach Bedarf noch weitere Eigenschaften hinzu. Wenn Sie diese Eigenschaften für **TreeView**-Steuerelemente in andere Formulare oder gar andere Anwendungen übernehmen möchten, kopieren Sie einfach die entsprechenden Codezeilen in die **Form_Load**-Prozedur.

TreeView füllen

Nun haben wir die Voraussetzungen dafür geschaffen, dass TreeView-Steuerelement per VBA mit den gewünschten Werten zu füllen.

Das **TreeView**-Steuerelement besitzt eine Auflistung namens **Nodes**, die alle im **TreeView**-Steuerelement angezeigten Element aufnimmt. Ein einzelnes Element im **TreeView** heißt dementsprechend **Node**. Um ein **Node**-Objekt zum TreeView hinzuzufügen, verwenden Sie die **Add**-Methode der **Nodes**-Auflistung.

Wenn Sie einen Node zur obersten Ebene der Hierarchie im **TreeView**-Steuerelement einfügen möchten, sieht der Aufruf etwa wie in Listing 2 aus. Das Element wird hier bei Laden des Formulars hinzugefügt (siehe Bild 1).

```
Private Sub Form_Load()
    objTreeView.Nodes.Add , , "a1", "Erster Knoten"
End Sub
```

Listing 2: Hinzufügen eines ersten Elements zum TreeView-Steuerelement

Die **Add**-Methode erwartet die folgenden Parameter:

- **Relative**: Gibt den **Key**-Wert eines Elements an, zu dem das neue Element in Beziehung stehen soll.
- **Relationship**: Gibt die Beziehung des neuen Elements zu dem in Relative bezeichneten Element an. Die Werte sind **tvwChild**, **tvwFirst**, **tvwLast**, **tvwNext**, **tvwPrevious**.
- **Key**: Ein eindeutiger Schlüssel für jedes Element, der mit einem Buchstaben beginnen muss.
- **Text**: Der durch das Element angezeigte Text.
- **Image**: Name eines Bildes, das als Icon des Elements angezeigt werden soll.
- **SelectedImage**: Name eines Bildes, dass statt des in **Image** festgelegten Bildes angezeigt werden soll, wenn das Element ausgewählt ist.

Zum Anlegen eines ersten Elements reichen die beiden Parameter **Key** und **Text** aus.

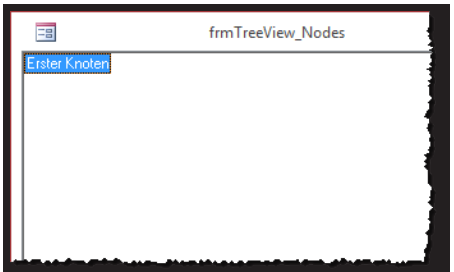


Bild 1: Erstes Element im TreeView-Steuerelement

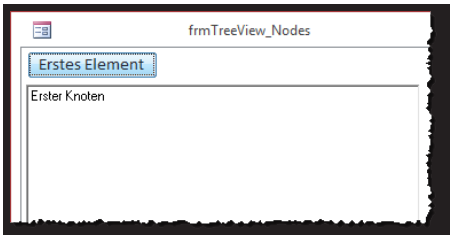


Bild 2: Erstes Element per Schaltfläche

Zu den Eigenschaften **Image** und **SelectedImage** kommen wir in einem späteren Teil der Artikelreihe.

Da es etwas müßig ist, das Formular zum Ausprobieren des Codes nach jeder Änderung immer wieder neu zu schließen und zu öffnen, fügen wir diesem eine Schaltfläche hinzu, das die Prozedur zum Füllen des **TreeView**-Steuerelements auslöst.

Für die folgenden Beispiele erweitern wir das Formular dann entsprechend um weitere Schaltflächen.

Die erste aus Bild 2 löst somit folgende Prozedur aus:

```
Private Sub cmdErstesElement_Click()
    objTreeView.Nodes.Add , , "a1", _
        "Erster Knoten"
End Sub
```

TreeView leeren

Wenn einem **TreeView** mehrfach die gleichen Elemente hinzufügen möchten, müssen Sie die vorhandenen Elemente zuvor entfernen.

Anderenfalls tritt ein Fehler auf, sobald Sie ein Element hinzufügen, dessen **Key**-Wert bereits einem der anderen Elemente im **TreeView**-Steuerelement zugeordnet ist. Es ist allerdings kein Problem, die vorhandenen Elemente zu löschen – dazu verwenden Sie einfach die Clear-Anweisung der Nodes-Auflistung:

```
objTreeView.Nodes.Clear
```

Weitere Elemente in der obersten Ebene

Um der obersten Ebene weitere Elemente hinzuzufügen, rufen Sie einfach die **Add**-Methode mit den gewünschten **Key**- und **Name**-Werten auf – also beispielsweise so:

```
objTreeView.Nodes.Add , , "a1", _
                      "Erster Knoten"
objTreeView.Nodes.Add , , "a2", _
                     "Zweiter Knoten"
objTreeView.Nodes.Add , , "a3", _
                     "Dritter Knoten"
```

Die Elemente werden dann genau in dieser Reihenfolge angelegt.

Verwandte Elemente anlegen

Wenn Sie weitere Elemente anlegen und dabei deren Position einstellen möchten, verwenden Sie die beiden Parameter **Relative** und **Relationship**. Der am häufigsten eingesetzte Fall dürfte das Einfügen eines Kindelements sein. Die Prozedur aus Listing 3 legt einige Eltern- und Kindelemente an.

```
Private Sub cmdRelativeWerte_Click()
    objTreeView.Nodes.Clear
    objTreeView.Nodes.Add , , "a1", "Erster Knoten"
    objTreeView.Nodes.Add "a1", tvwChild, "a11", "Erster Kind-Knoten"
    objTreeView.Nodes.Add , , "a2", "Zweiter Knoten"
    objTreeView.Nodes.Add "a2", tvwChild, "a21", "Zweiter Kind-Knoten"
    objTreeView.Nodes.Add "a2", tvwChild, "a22", "Dritter Kind-Knoten"
End Sub
```

Listing 3: Hinzufügen zweier Elemente und einiger Kindelemente

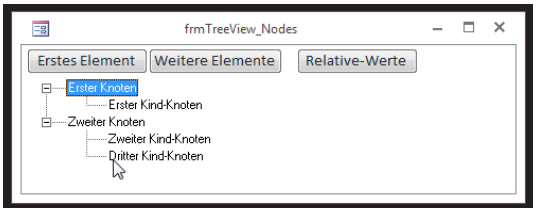


Bild 3: Eltern- und Kindelemente

Das Ergebnis finden Sie in Bild 3.

Dies ist der Normalfall: Die Elemente werden gleich in der gewünschten Reihenfolge angelegt, die beispielsweise durch die Sortierung der zugrunde liegenden Daten vorgegeben wird. Gleiches gilt für das Hinzufügen der untergeordneten Elemente.

Es kann jedoch auch geschehen, dass Sie nachträglich ein Element vor oder hinter ein bestimmtes Element oder an erster oder letzter Stelle der Ebene eines bestimmten Elements einfügen möchten.

Dann kommen neben **tvwChild** weitere Konstanten für den Parameter **Relationship** zum Tragen. Die Prozedur aus Listing 4 ist die erweiterte Version der zuvor beschriebenen Prozedur. Zu den bereits vorhandenen Elementen fügt die Prozedur die folgenden Elemente hinzu:

- das Element **a3** als erstes Element der Ebene, in der sich das Element **a1** befindet,
- das Element **a12** als letztes Element der Ebene des Elements **a11**,
- das Element **a21a** als folgendes Element des Elements **a21** und
- das Element **a21b** als vorheriges Element des Elements **a22**.

```
Private Sub cmdRelativeWerteII_Click()
    ....
    objTreeView.Nodes.Add "a1", tvwFirst, "a3", "Als Erster eingeschoben"
    objTreeView.Nodes.Add "a11", tvwLast, "a12", "Als Letzter eingeschoben"
    objTreeView.Nodes.Add "a21", tvwNext, "a12a", "Nach Element eingeschoben"
    objTreeView.Nodes.Add "a22", tvwPrevious, "a12b", "Vor Element eingeschoben"
End Sub
```

Listing 4: Hinzufügen von Elementen als erstes, letztes, folgendes und vorheriges Element einer Ebene

Das Ergebnis finden Sie in Bild 4.

Elemente beim Anlegen ausklappen

Wenn ein Element noch untergeordnete Elemente besitzt, sollen diese möglicherweise gleich angezeigt werden. Standardmäßig werden die Zweige jedoch nicht aufgeklappt. Dies können Sie allerdings nachholen – und zwar, indem Sie die Eigenschaft **Expanded** des jeweiligen übergeordneten Elements auf **True** einstellen. Das gelingt am einfachsten, wenn Sie das **Node**-Element direkt beim Erstellen einer passenden Objektvariablen zuweisen – mehr dazu weiter unten.

Verweis mit Objektvariable

Wenn Sie einmal einen Verweis auf eines der Elemente des TreeView-Steuerelements in einer Variablen speichern wollen, deklarieren Sie zunächst eine entsprechende Variable des Typs **Node** – zum Beispiel mit dem Namen **objNode**. Diese füllen Sie dann mit der Set-Anweisung mit dem Verweis auf das entsprechende Element. Der Ausdruck **objTreeView.SelectedItem** liefert beispielsweise einen Verweis auf das aktuell markierte Element. Die Prozedur aus Listing 5 zeigt, wie ein Element referenziert wird und wie Sie anschließend über die Objektvariable auf die Eigenschaften des **Node**-Elements zugreifen.

Ein weiterer guter Anlass, die Elemente jeweils in einer **Node**-Variablen zu speichern, ist das anschließende Zuweisen von Eigenschaften an ein Element. Sie brauchen dann nicht für jede Eigenschaft den kompletten Ausdruck zu verwenden, sondern können die Eigenschaften über die Objektvariable zuweisen. Im Beispiel aus Listing 6 stellt die Prozedur zum Beispiel für alle hinzugefügten Elemente gleich die Eigenschaft **Expanded** auf den Wert **True** ein, damit alle Äste ausgeklappt werden.

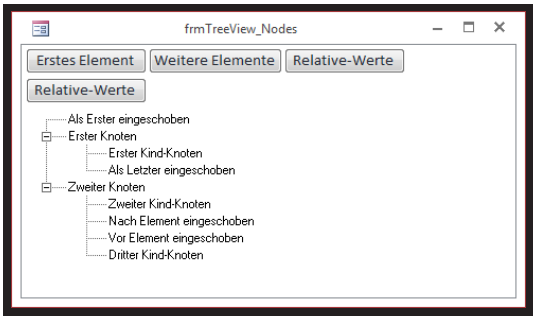


Bild 4: Nachträgliches Einfügen von Elementen in Abhängingkeit anderer Elemente

Anzahl der Elemente ermitteln

Wenn Sie die Anzahl aller Elemente eines **TreeView**-Steuerelements ermitteln möchten, verwenden Sie die **Count**-Eigenschaft der **Nodes**-Auflistung des **TreeView**-Steuerelements. Vom Direktfenster aus könnte dies so aussehen:

```
Debug.Print Forms!frmTreeView_Nodes!ctlTreeView.Object.Nodes.Count
 9
```

Über die durch eine Schaltfläche ausgelöste Ereignisprozedur geben Sie die Anzahl wie folgt auf:

```
Private Sub cmdAnzahlDerNodes_Click()
    MsgBox "Node-Anzahl: " _
        & objTreeView.Nodes.Count
End Sub
```

Elemente referenzieren

Im Verlauf der Programmierung eines **TreeView**-Steuerelements werden Sie sicher auch einmal eines oder mehrere Elemente des **TreeView**-Steuerelements referenzieren müssen – beispielsweise um ein Element zu aktivieren oder seine Eigenschaften zu ändern.

Die erste Variante, dies zu erledigen, ist die direkte Referenzierung über

```
Private Sub cmdAktuellesElement_Click()
    Dim objNode As MSComctlLib.Node
    Set objNode = objTreeView.SelectedItem
    Debug.Print objNode.Key, objNode.Text
End Sub
```

Listing 5: Speichern eines Verweises auf ein Element in einer Variablen

```
Private Sub cmdAlleAusklappen_Click()
    Dim objNode As MSComctlLib.Node
    objTreeView.Nodes.Clear
    Set objNode = objTreeView.Nodes.Add(, , "a1", "Erster Knoten")
    objNode.Expanded = True
    Set objNode = objTreeView.Nodes.Add("a1", tvwChild, "a11", "Erster Kind-Knoten")
    objNode.Expanded = True
    ...
End Sub
```

Listing 6: Referenzieren von Node-Objekten und anschließende Einstellung von Variablen

Bilder in Access, Teil I

Bilder kommen in Access an den verschiedensten Stellen vor: Als Daten der Datenbank etwa zur Darstellung von Personen oder Artikeln, in Menüs, Ribbons und Kontextmenüs, in ListView- und TreeView-Steuerelementen oder als Element einer Schaltfläche oder weiterer Steuerelemente. Es gibt verschiedene Möglichkeiten, Bilder in Access verfügbar zu machen. In dieser Artikelreihe schauen wir uns die unterschiedlichen Varianten an und gehen auf Vor- und Nachteile ein.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1308_Bilder.mdb**.

Bilder in Access

Wenn Sie sich einmal verschiedene Anwendungen ansehen, die sich auf Ihrem Rechner befinden oder Vergleiche zu früheren Anwendungen ziehen, werden Sie feststellen, dass die Darstellung sich stetig verändert und durch den Einsatz von Icons eine hübsche und benutzerfreundliche Oberfläche erstellen lässt.

Dies bezieht sich allein auf die Elemente der Benutzeroberfläche. Eine ganz andere Rolle spielt die Darstellung von Daten in Form von Bildern – etwa durch die Abbildung von Personen oder Artikeln.

Beide Einsatzmöglichkeiten von Bildern haben verschiedene Ziele und es verbergen sich auch verschiedenste technische Ansätze dahinter. In dieser Artikelreihe werden wir uns diese von A bis Z ansehen.

Einige der hier beschriebenen Techniken sind erst ab Access 2007 mit der dort eingeführten neuen Version des Bildsteuerelements verwendbar.

Bilder in OLE-Feldern speichern

Seit ewigen Zeiten gibt es mit dem OLE-Feld eine Möglichkeit, sowohl Bilder als auch Dateien wie etwa Word-Dokumente in einem Feld einer Access-Tabelle zu speichern. Dies gelingt ganz einfach: Sie legen eine Tabelle mit einem OLE-Feld wie in Bild 1 an. Wechsel Sie dann in die Datenblattansicht und klicken Sie mit der rechten Maustaste auf das OLE-Feld (siehe Bild 2).

Wählen Sie dann aus dem Kontextmenü den Eintrag **Objekt einfügen ...** aus und es erscheint der Dialog zum Hinzufügen eines OLE-Objekts. Wählen Sie dort wie in Bild 3 die

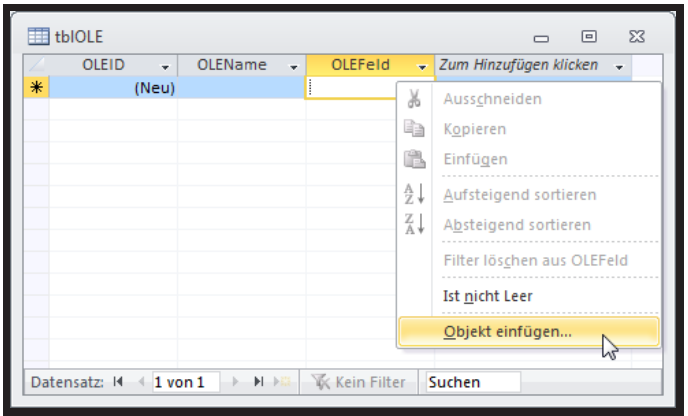


Bild 2: Füllen eines OLE-Feldes

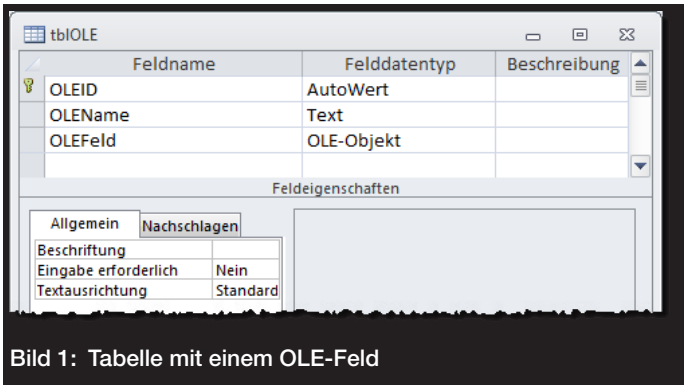


Bild 1: Tabelle mit einem OLE-Feld

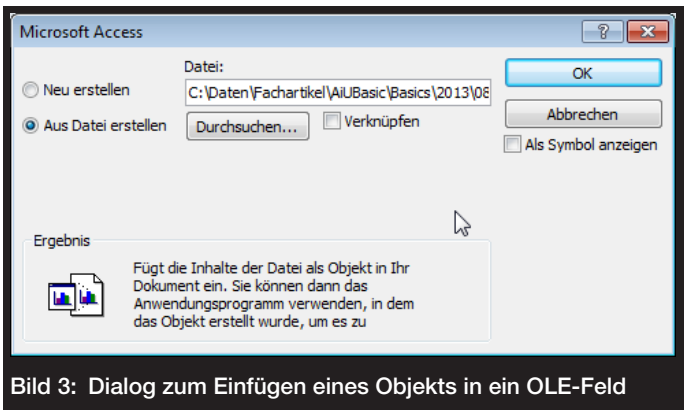


Bild 3: Dialog zum Einfügen eines Objekts in ein OLE-Feld

Option **Aus Datei erstellen** aus. Mit der Durchsuchen-Schaltfläche öffnen Sie einen **Datei öffnen**-Dialog, der bei der Auswahl der einzufügenden Datei hilft.

Nach der Auswahl können Sie mit der Option **Verknüpfen** noch festlegen, ob sich Änderungen an der Originaldatei auf die in der Tabelle gespeicherte Version auswirken sollen.

Nach dem Speichern zeigt das OLE-Feld ganz banal den Wert **Paket** an (gelegentlich auch **Package**). Ein Doppelklick auf dieses Feld bietet die Möglichkeit, die Datei mit der verknüpften Anwendung zu öffnen. Diesen Dialog können Sie für spätere Aufrufe deaktivieren (siehe Bild 4).

Wenn Sie ein Word-Dokument in das OLE-Feld einfügen, zeigt dieses nicht **Paket** oder **Package** an, sondern **Word Document**. Am Beispiel eines Word-Dokuments wollen wir kurz zeigen, wie sich die **Verknüpfen**-Option auswirkt. Dazu fügen Sie ein Word-Dokument bei aktivierter **Verknüpfen**-Option hinzu. Öffnen Sie das Word-Dokument testweise per Doppelklick auf das OLE-Feld.

Schließen Sie es und öffnen das Dokument dann über das Dateisystem. Ändern Sie den enthaltenen Text und speichern Sie das Dokument. Klicken Sie dann erneut doppelt auf das Dokument im OLE-Feld und überzeugen Sie sich davon, dass die Änderungen auch in das im OLE-Feld gespeicherte Dokument übernommen wurden. Davon abgesehen öffnet Access nun scheinbar das Ursprungsdokument, was dadurch sichtbar wird, dass die übliche temporäre Datei eines geöffneten Word-Dokuments im Windows Explorer angezeigt wird (siehe Bild 5).

Problematisch wird es, wenn Sie die Originaldatei im Dateisystem löschen: Dann meldet Access einen Fehler, wenn Sie versuchen, die Datei per Doppelklick auf die Verknüpfung im OLE-Feld zu öffnen.

OLE-Bild im Formular

Wie nun zeigen wir ein in einem

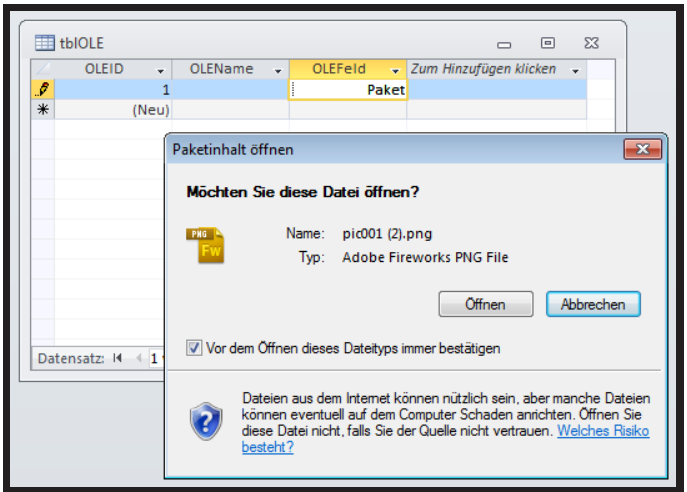


Bild 4: Öffnen einer in einem OLE-Feld gespeicherten Datei

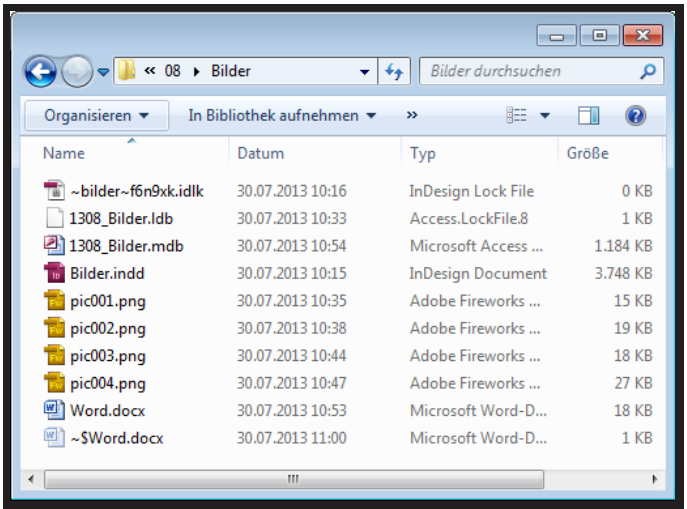


Bild 5: Temporäre Word-Datei

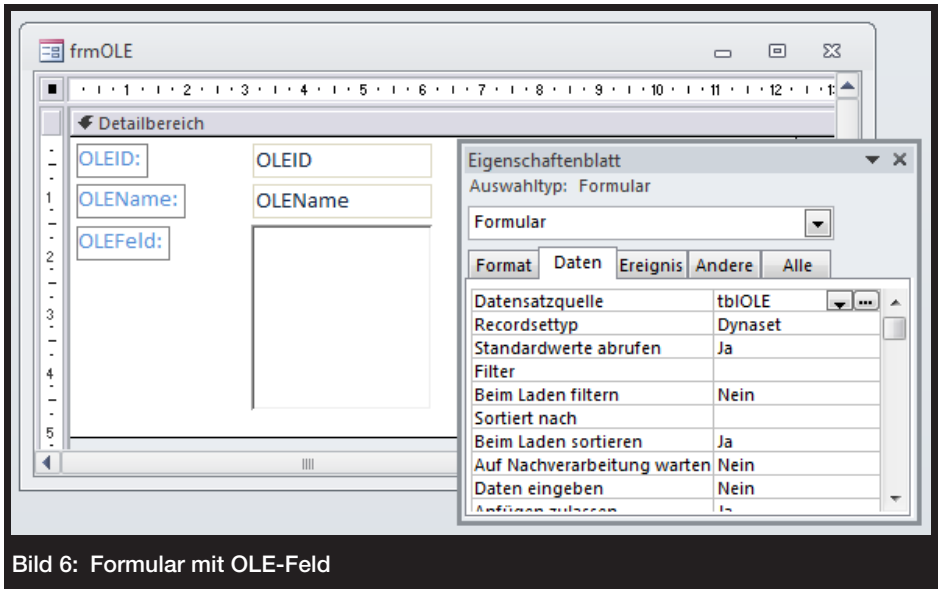


Bild 6: Formular mit OLE-Feld

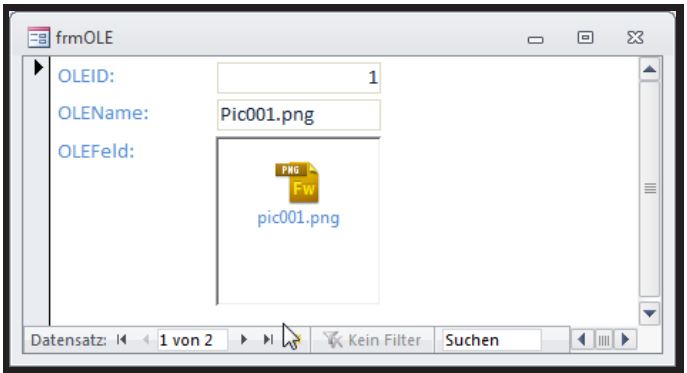


Bild 7: Formular mit OLE-Feld in der Formularansicht

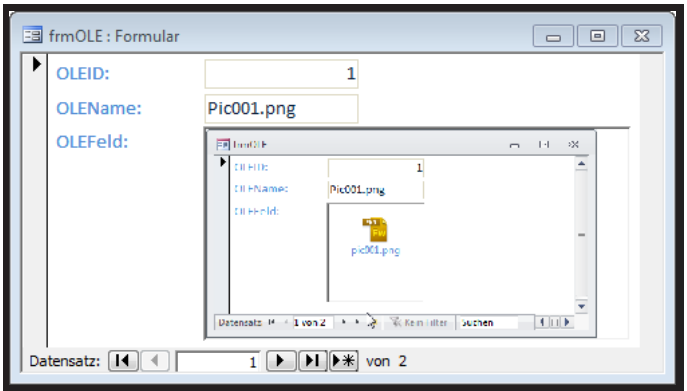


Bild 8: OLE-Feld mit Abbildung unter Access 2003

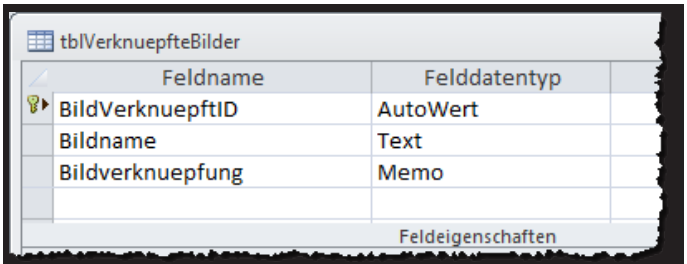


Bild 9: Speichern des Dateipfades einer Bilddatei in der Tabelle

OLE-Feld gespeichertes Bild in einem Formular an? Dazu erstellen wir zunächst ein neues Formular und speichern dieses unter dem Namen **frmOLE**.

Stellen Sie die Eigenschaft Datenherkunft auf die Tabelle **tblOLE** ein. Ziehen Sie dann alle Felder der Datenherkunft aus der Feldliste in den Formularentwurf (siehe Bild 6).

Ein Wechsel in die Formularansicht liefert leider nicht den erwarteten Effekt: Das OLE-Feld zeigt lediglich ein Icon mit dem Namen des gespeicherten Dokuments an (siehe Bild 7).

Warum gelingt dies nicht? Die aktuellste Access-Version, mit der uns dies gelungen ist, war Access 2003 (siehe Bild 8).

Und selbst dort wurden teilweise nur **.bmp**-Bilder oder solche Bilder, die mit einem OLE-Server verknüpft waren, angezeigt.

Das bedeutet, dass Sie für eine Bilddatei mit der entsprechenden Dateiendung standardmäßig eine Bearbeitungssoftware auswählen mussten, die als OLE-Server genutzt werden kann. Dies ist die Grundvoraussetzung, um Bilder aus OLE-Felder in einem gebundenen Objektfeld anzuzeigen.

Um es kurz zu machen: Zur Behandlung von Bilddateien, die nativ in OLE-Feldern gespeichert werden, gibt es ausreichend Quellen im Internet, und in neueren Access-Versionen arbeitet man ohnehin mit anderen Varianten. Die hier vorgestellte Technik sollten Sie wirklich nur einsetzen, wenn keine halbwegs aktuelle Access-Version zur Verfügung steht – also Access 2007 und neuer.

Bild per Verknüpfung anzeigen

Eine versionsübergreifend funktionierende Variante zum Anzeigen von Bildern in einem Access-Formular ist das Anzeigen verknüpfter Bilder. Dabei ist Verknüpfung nicht im Sinne des vorhergehenden Beispiels gemeint, sondern so: Sie speichern in einem Feld einer Tabelle den Pfad zu der anzuzeigenden Bilddatei und zeigen diese im Formular an, wenn der entsprechende Datensatz aufgerufen wird.

Die Tabelle zu diesem Beispiel sieht im Entwurf wie in Bild 9 aus. Das Feld **Bildverknuepfung**, das wegen der Begrenzung der Zeichenzahl auf 255 nicht als Textfeld, sondern als Memofeld ausgelegt wird, nimmt dabei etwa einen Ausdruck wie den folgenden auf:

```
C:\Daten\Fachartikel\AiUBasic\Basics\2013\08\Bilder\pic001.png
```

Damit diese Bild in einem Formular angezeigt wird, müssen Sie dieses zunächst an die Tabelle **tblVerknuepfteBilder** binden.

Danach ziehen Sie alle Felder der Datenherkunft in den Entwurf des Formulars. Zur Anzeige des Bildes verwenden wir ein Bildsteuerelement.

Das Anlage-Feld

Mit Access 2007 hat Microsoft mit dem Anlage-Feld einen neuen Felddatentyp eingeführt, der gleichzeitig ein neues Steuerelement mitbringt. Mit dem Anlage-Feld speichern Sie alle möglichen Dateien in einem Feld der Datenbank – und bei Bedarf sogar mehrere Dateien pro Datensatz. Dieser Artikel liefert alle Informationen rund um das Anlage-Feld und zeigt, welche Aufgaben Sie damit erledigen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1308_Anlagefeld.accdb**.

Voraussetzungen

Die wichtigste Voraussetzung für die Nutzung dieses Artikels ist das Vorhandensein von Access 2007 oder neuer. Unter Access 2003 und älter gibt es das Anlagefeld noch nicht.

Der Felddatentyp

Wenn Sie unter Access 2007 oder neuer Felder zu einer Tabelle in der Entwurfsansicht hinzufügen, finden Sie dort einen neuen Datentyp namens Anlage (siehe Bild 1).

Zu Beispielzwecken erstellen wir eine neue Tabelle namens **tblDateien**, die das Primärschlüsselfeld **DateiID**, das Textfeld **Dateiname** und das Anlage-Feld **Datei** enthält (siehe Bild 2). Wie Sie der Abbildung entnehmen können, liefert dieser Datentyp nur zwei Eigenschaften: **Beschriftung** und **Eingabe erforderlich**. Warum das so ist, erfahren Sie weiter unten.

Schauen wir uns zunächst an, wie sich das Einfügen eines Attachments anfühlt. Dazu wechseln Sie zunächst zur Datenblattansicht der Tabelle. Hier wird schnell deutlich, dass das Anbringen einer Beschriftung für dieses Fle durchaus sinnvoll ist: Sonst zeigt der Spaltenkopf nämlich lediglich ein Büroklammer-Symbol an (siehe Bild 3). Stellen Sie also die Eigenschaft **Beschriftung** auf einen Text wie **Dateianlage** ein.

Auch das Feld selbst unterscheidet sich von herkömmlichen Feldern in Access-Tabellen. Es zeigt ebenfalls ein Büroklammer-Symbol an, gefolgt von der Anzahl der enthaltenen Dateien.

Das bedeutet, dass ein Anlagefeld je Datensatz mehr als eine Tabelle aufnehmen kann.

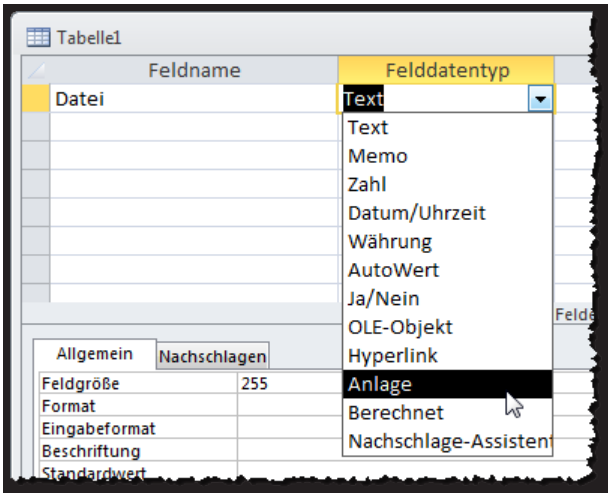


Bild 1: Anlegen eines neuen Anlage-Feldes

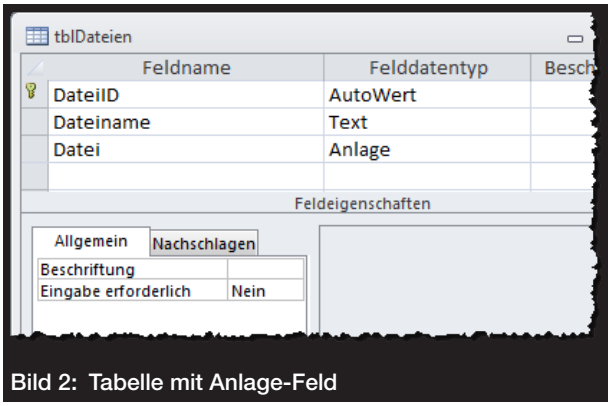


Bild 2: Tabelle mit Anlage-Feld

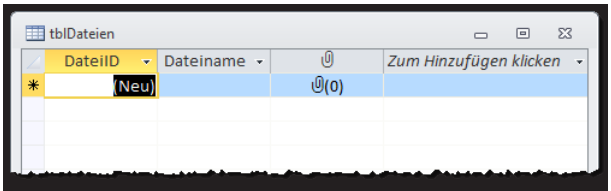


Bild 3: Das Anlage-Feld in der Datenblattansicht

Anlage hinzufügen

Anlagen können Sie über diese Ansicht auf zwei Arten hinzufügen: entweder durch einen Klick mit der rechten Maustaste und anschließende Auswahl des Eintrags **Anlagen verwalten ...** des nun erscheinenden Kontextmenüs (siehe Bild 4) oder durch einen einfachen Doppelklick in das Feld. Beides ruft die Anzeige des Dialogs **Anlagen** aus

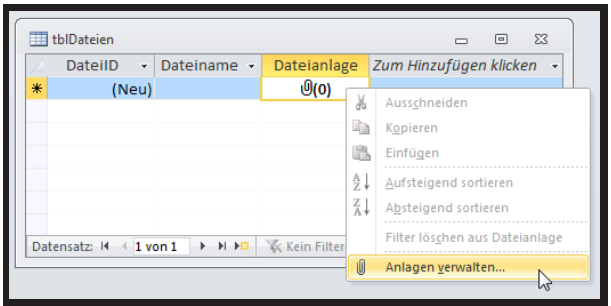


Bild 4: Kontextmenü-Eintrag zum Öffnen des Dialogs zum Verwalten der Anlagen

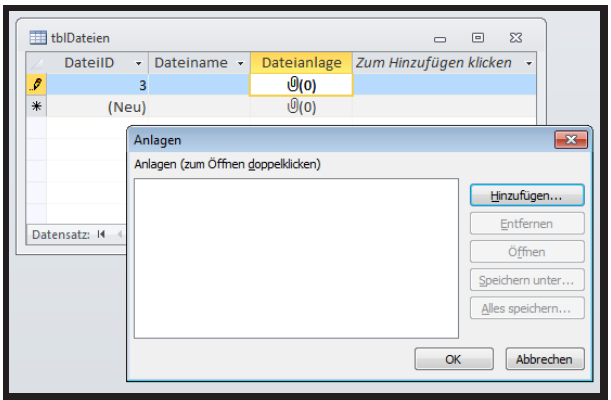


Bild 5: Dialog zum Verwalten von Anlagen

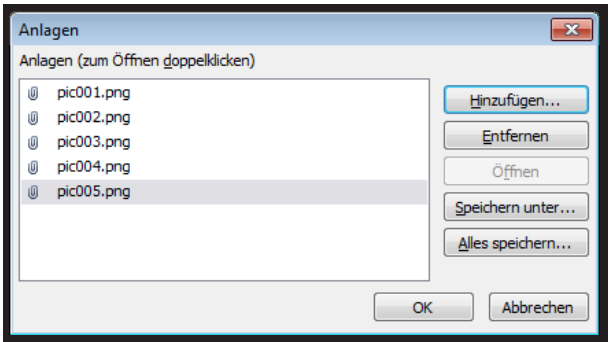


Bild 6: Anlagen-Dialog mit einigen Anlagen

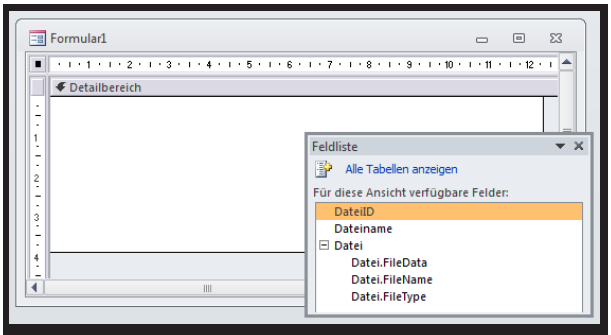


Bild 7: Hinzufügen eines Anlage-Felds zum Formularentwurf

Bild 5 hervor. Hier bietet sich nun zunächst nur die Schaltfläche **Hinzufügen...** an, mit der Sie einen klassischen **Datei auswählen**-Dialog öffnen. Mit diesem Dialog wählen Sie eine oder auch mehrere Dateien gleichzeitig aus und fügen diese durch Betätigen der **Öffnen**-Schaltfläche hinzu.

Anschließend werden die Dateinamen der hinzugefügten Dateien im Dialog **Anlagen** angezeigt (siehe Bild 6).

Damit eröffnen sich nun weitere Möglichkeiten: Sie können weitere Dateien hinzufügen, vorhandene Einträge entfernen, die Datei mit der dafür vorgesehenen Anwendung öffnen (diese Option bietet sich unter Umständen erst nach erneutem Anzeigen des **Anlagen**-Dialogs), die Datei im Dateisystem speichern oder alle Dateien im Dateisystem speichern – alles über die Schaltflächen rechts im Dialog.

Das Anlage-Steuerelement

Kommen wir nun zum Anlage-Steuerelement. Wenn Sie ein solches zu einem Formular hinzufügen möchten, gelingt dies am einfachsten, wenn sie zunächst das Formular über die Eigenschaft **Datenherkunft** an die entsprechende Tabelle binden – hier **tblDateien**. Wenn Sie dann zur Feldliste wechseln, zeigt dieses die ersten beiden Felder in der üblichen Art an. Das Anlage-Feld hingegen erscheint wie in Bild 7 mit drei „Unterfeldern".

Ziehen Sie alle drei Felder der Feldliste in den Entwurf (die Unterfelder werden dabei automatisch mit markiert), sieht das Ergebnis wie in Bild 8 aus.

Das Steuerelement, das an das Anlage-Feld gebunden ist, wird automatisch mit dem Datentyp **Anlage** angelegt. Wenn Sie das Steuerelement

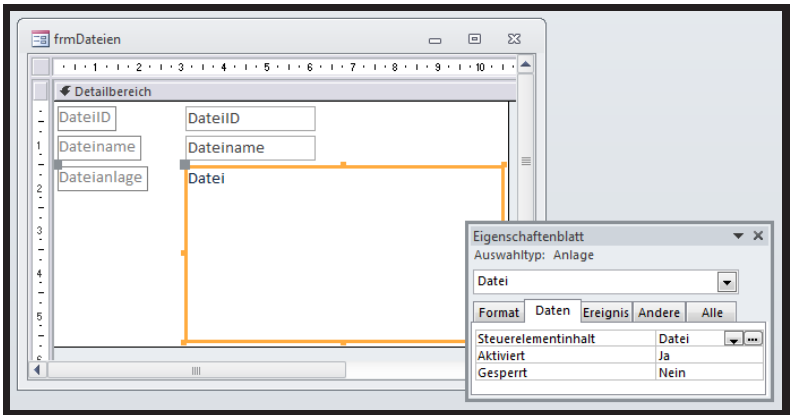


Bild 8: Das Steuerelement wird mit dem Typ Anlage ausgewiesen.

zuerst anlegen und dann an ein Feld binden möchten, markieren Sie in der Liste der Steuerelemente im Ribbon das mit dem Büroklammer-Symbol und platzieren das Steuerelement anschließend im Formular.

Ein Wechsel in die Formularansicht zeigt, dass das Anlage-Steuerelement Bilder direkt in der Vorschau anzeigt (siehe Bild 9). Wenn das Anlage-Feld, wie in diesem Fall, mehrere Dateien enthält, können Sie darin mit einer speziell dafür eingeblendeten Menüleiste blättern. Ein Doppelklick oder die Auswahl des Kontextmenü-Eintrags **Anlagen verwalten** öffnet den **Anlagen**-Dialog. In diesem Fall können Sie auch über das Kontextmenü durch die enthaltenen Dateien navigieren.

Eigenschaften des Steuerelements

Das Anlage-Steuerelement besitzt einige interessante Eigenschaften.

Mit der Eigenschaft **Standardbild** stellen Sie beispielsweise ein Bild ein, das angezeigt werden soll, wenn das an das Anlage-Steuerelement gebundene Feld keine Daten enthält. Dies gilt sowohl für vorhandene als auch für neue Datensätze. Klicken Sie einfach auf die Schaltfläche mit den drei Punkten (...) rechts neben der Eigenschaft und wählen Sie das anzuzeigende Bild aus. Auf diese Weise können Sie beispielsweise ein Bild einbauen, das die Anweisungen für den Umgang mit den Anlage-Steuerelement enthält (siehe Bild 10).

Die Eigenschaft **Hintergrundart** nimmt die Werte **Transparent** und **Normal** an. Der Wert **Transparent** sorgt dafür, dass Bilder mit transparenten Bereichen an den entsprechenden Stellen den Hintergrund des Formulars anzeigen.

Bei den Ereigniseigenschaften ist Microsoft ein kleiner Fehler passiert: Die Eigenschaft **On Attachment Current** wurde im Eigenschaftsfeld nämlich noch nicht übersetzt, daher finden Sie hier noch den englischen Namen (siehe Bild 11).

Dieses Ereignis beim Anzeigen einer Anlage ausgelöst, also sowohl beim ersten Anzeigen als auch beim Wechseln der aktuellen Anlage. Dies nutzen wir aus, indem wir mit einer kleinen Prozedur den Wert des Textfeldes **txtAktuelleDatei** auf den Namen der Anlage einstellen. Die Prozedur sieht wie in Listing 1 aus und fragt die Eigenschaft **Filename** des gebundenen Feldes ab.

VBA-Eigenschaften

Mit **FileName** haben Sie bereits die erste Eigenschaft kennengelernt, die nur per VBA ermittelt werden kann. Es gibt noch einige weitere:

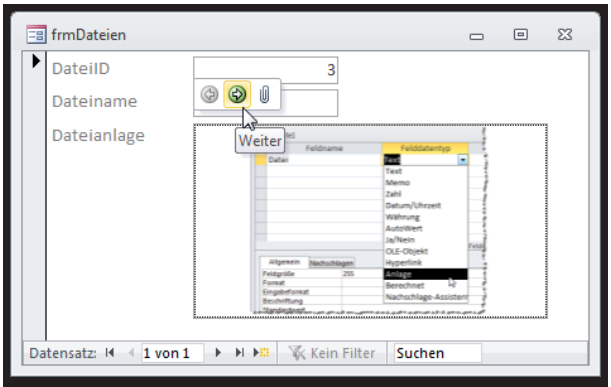


Bild 9: Anlage-Feld mit Bilddatei

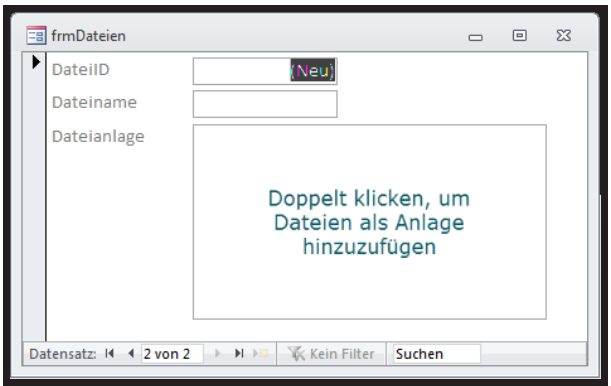


Bild 10: Anlage-Feld mit einem Standardbild

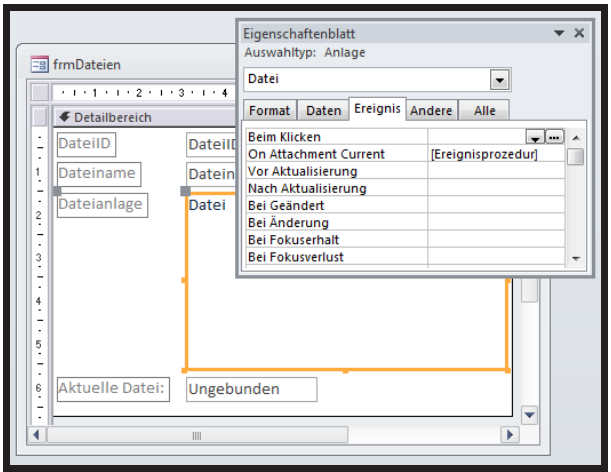


Bild 11: Das Ereignis wird beim Blättern in den Anlagen ausgelöst.

```
Private Sub Datei_AttachmentCurrent()
    Me!txtAktuelleDatei = Nz(Me!Datei.FileName)
End Sub
```

Listing 1: Einstellen eines Textfeldes auf den Namen der aktuellen Anlage

Anlage-Felder und VBA

Anlage-Felder bieten bereits ohne den Einsatz von VBA echten Nutzen. Wenn es jedoch darum geht, größere Operationen durchzuführen, um etwa eine Reihe von Bildern aus verschiedenen Verzeichnissen in ein oder mehrere Anlage-Felder zu kopieren, werden Sie dies kaum von Hand erledigen wollen. Stattdessen erstellen Sie eine VBA-Abfrage, die das Einfügen, Löschen oder Entfernen der Dateien in und aus dem Anlage-Feld erledigen soll.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1309_Anlage-felder.accdb**.

Hinweise

Anlage-Felder wurden mit Access 2007 eingeführt. Voraussetzung ist außerdem ein Verweis auf die Bibliothek **Microsoft Office x.0 Access Database Engine Object Library**, der normalerweise bei Datenbanken des Typs **.accdb** bereits voreingestellt sein sollte.

Sollte die Datenbank aus irgendwelchen Gründen den Verweis auf die Bibliothek **Microsoft DAO 3.6 Object Library** enthalten, entfernen Sie diesen Sie über den **Verweise**-Dialog (VBA-Editor, Menü **Extras|Verweise**) und fügen den Verweis auf die oben genannte Bibliothek hinzu (siehe Bild 1).

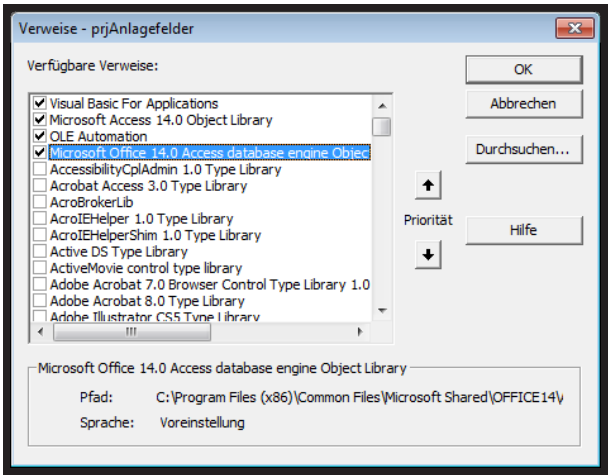


Bild 1: Verweis auf die neue DAO-Bibliothek

Struktur von Anlage-Feldern

Bevor wir uns mit VBA an das Lesen und Bearbeiten eines Anlage-Feldes heranwagen, schauen wir uns die zugrunde liegende Struktur an. Am besten gelingt dies, wenn Sie eine neue Abfrage erstellen und eine Tabelle als Datenherkunft hinzufügen, die ein Anlage-Feld enthält. Dies sieht etwa wie in Bild 2 aus.

Wechseln Sie zur Datenblattansicht dieser Abfrage, erkennen Sie, dass jeder Datensatz für jede enthaltene Anlage je einmal erscheint (siehe Bild

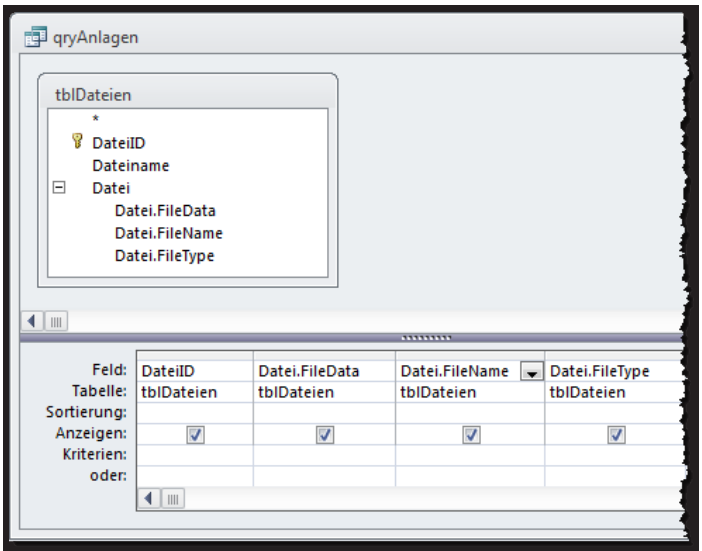


Bild 2: Das Anlage-Feld im Abfrageentwurf

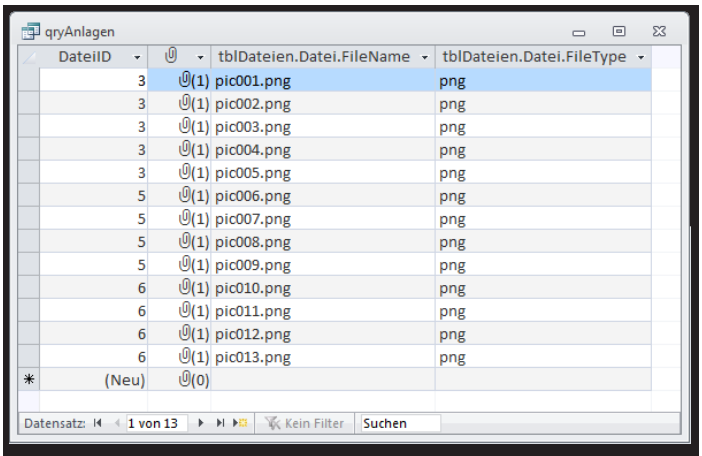


Bild 3: Abfrage, die jeden Datensatz entsprechend der Anzahl der enthaltenen Anlagen anzeigt.

```
Public Sub LesenderZugriff()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("qryAnlagen", dbOpenDynaset)
    Do While Not rst.EOF
        Debug.Print rst!DateiID, rst("tblDateien.Datei.FileName"), rst("tblDateien.Datei.FileType"), _
            Left(rst("tblDateien.Datei.FileData"), 100)
        rst.MoveNext
    Loop
End Sub
```

Listing 1: Einfacher lesender Zugriff über eine Abfrage mit allen Feldern der Tabelle mit dem Anlage-Feld

3). Der Datensatz mit dem Wert 3 im Feld **DateiID** enthält beispielsweise fünf Dateien im Anlagefeld. Dementsprechend erscheint der Datensatz fünf Mal, jeweils mit einer anderen Anlage.

Dies weist darauf hin, dass die in Anlage-Feldern gespeicherten Daten möglicherweise in einer weiteren, für Benutzeraugen nicht sichtbaren, Tabelle gespeichert werden. In der Tat findet sich in einer Datenbank, nachdem Sie eine Tabelle mit einem Anlage-Feld hinzugefügt haben, eine zusätzliche Tabelle mit einem kryptischen Namen wie **f_AE1ADE703D174BDAADDFFEBC068B79C8_Test** (diese lässt sich allerdings nicht öffnen).

Und wenn die Daten des Anlage-Feldes schon in einer verknüpften Tabelle landen, finden wir sicher auch unter DAO einen alternativen Weg für den Zugriff auf diese Daten.

Nur lesen

Wenn Sie die Daten nur lesen möchten, aber nicht bearbeiten, können Sie direkt mit einem Recordset auf Basis der oben beschriebenen Abfrage arbeiten. Für die VBA-Prozeduren legen Sie im VBA-Editor mit **Einfügen|Modul** ein neues Modul namens **mdlAnlagen** an.

Fügen Sie dann die Prozedur mit dem Code aus Listing 1 hinzu. Die Prozedur deklariert und erstellt zunächst die benötigten Objekte, also ein **Database**-Objekt namens **db** und ein **Recordset**-Objekt namens **rst** und durchläuft dann alle Datensätze der Abfrage **qryAnlagen** in einer **Do While**-Schleife.

Dabei gibt eine **Debug.Print**-Anweisung die Inhalte aller Felder im Direktbereich aus. Da der Inhalt des Anlagefeldes selbst, also die Datei, als Zeichenfolge ausgegeben wird, haben wir die Anzahl der Zeichen auf die ersten **100** begrenzt (**Left(...)**). Das Ergebnis finden Sie in Bild 4.

Die Namen der Felder können Sie dabei einfach der Datenblattansicht der Abfrage **qryAnlagen** entnehmen – oder Sie fragen diese etwa mit folgenden Ausdrücken im Direktfenster ab:

```
? CurrentDb.QueryDefs("qryAnlagen").
Fields(1).Name
tblDateien.Datei.FileData
```

Nun stellt sich allerdings die Frage, ob wir auch auf die Datei zugreifen und diese etwa speichern können. Aber mit welcher Methode? Zunächst einmal nutzen wir eines der aktualisierten Objekte der neuen DAO-Bibliothek, und zwar **Field2**.

Deklarieren Sie in der Prozedur von oben eine entsprechende Variable dieses Typs namens **fld**. Dann weisen Sie dieser Variablen innerhalb

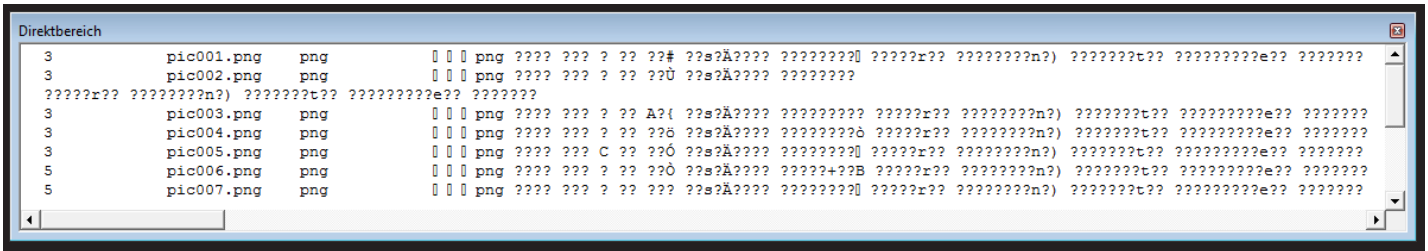


Bild 4: Inhalt eines Anlagefeldes

```
Public Sub LesenderZugriff()
    Dim fld As DAO.Field2
    ...
    Do While Not rst.EOF
        Set fld = rst.Fields("tblDateien.Datei.Filedata")
        fld.SaveToFile CurrentProject.Path & "\test_" & rst("tblDateien.Datei.Filename")
        rst.MoveNext
    Loop
End Sub
```

Listing 2: Diese Prozedur durchläuft alle Anlagen und speichert diese auf der Festplatte.

der Schleife den Inhalt des Feldes **tblDateien.Datei.Filedata** zu, also des für das Speichern der Datei zuständigen Feldes innerhalb der verborgenen, verknüpften Tabelle.

Das Objekt des Typs **Field2** liefert gegenüber der älteren Variante **Field** zum Beispiel die neue Methode **SaveToFile**, die wir in diesem Fall einsetzen um den Inhalt des Feldes in einer als Parameter angegebenen Datei zu speichern. Den angepassten Code der Prozedur finden Sie in gekürzter Form in Listing 2.

Anlagen in eigenem Recordset

Nun haben wir ja bereits ermittelt, dass die Anlagen in einer eigenen Tabelle gespeichert werden. Können wir auf die Elemente dieser Tabelle auch mit einem eigenen Recordset zugreifen? Ja, das ist möglich.

Dazu deklarieren Sie zunächst eine weitere **Recordset**-Variable, die aber dem Typ **Recordset2** entsprechen muss. Dieser weisen Sie dann wiederum den **Value**-Wert des Anlage-Feldes des ersten Recordsets zu. Wir wollen und den Aufbau des Recordsets einmal ansehen und lesen dazu zunächst alle Feldnamen und Felddatentypen dieses Recordsets aus, nach dem wir es referenziert haben.

Die passende Prozedur finden Sie in Listing 3. Die Prozedur referenziert die aktuelle Datenbank und erstellt eine Datensatzgruppe auf Basis der Tabelle **tblDateien**.

Mit der Variablen **rstAnlagen** referenziert sie dann die **Value**-Eigenschaft des Anlage-Feldes dieser Tabelle, also des Feldes **Datei**, und erzeugt so ein Recordset auf Basis der verknüpften Tabelle mit den Daten des Anlage-Feldes.

Da es sich hierbei tatsächlich um eine Tabelle handelt, können wir nun in einer **For Each**-Schleife alle Felder des Recordsets durchlaufen und dabei alle Feldnamen plus die Kennzahlen für die Felddatentypen im Direktbereich ausgeben.

Nachfolgend finden Sie das Ergebnis, ergänzt um die Datentypen im Klartext:

- **FileData**: **11** (Byte-Array)
- **FileFlags**: **4** (Long)
- **FileName**: **10** (Text)
- **FileTimeStamp**: **8** (Date/Time)
- **FileType**: **10** (Text)
- **FileURL**: **12** (Memo)

Anlagen auslesen

Nun wollen wir die im Anlage-Feld der Tabelle befindlichen Dateien direkt über das untergeordnete **Recordset2**-Objekt auslesen.

```
Public Sub FelderDerAnlagenTabelle()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim fld As DAO.Field2
    Dim rstAnlagen As DAO.Recordset2
    Set db = CurrentDb
    Set rst = db.OpenRecordset("tblDateien", dbOpenDynaset)
    If Not rst.EOF Then
        Set rstAnlagen = rst.Fields("Datei").Value
        For Each fld In rstAnlagen.Fields
            Debug.Print fld.Name, fld.Type
        Next fld
    End If
End Sub
```

Listing 3: Auslesen aller Felder eines Recordsets auf Basis eines Anlage-Feldes

Das TreeView-Steuerelement: Verknüpfte Daten, Teil I

Das **TreeView**-Steuerelement ist prädestiniert für die Anzeige von Daten aus hierarchisch verknüpften Tabellen. In diesem Artikel zeigen wir anhand der Daten aus einer Kunden-, Bestellungen- und Artikeltabelle in ein **TreeView**-Steuerelement einlesen und diese dabei hierarchisch anordnen. Dabei lernen Sie zwei verschiedene Arten kennen, die benötigen Daten zu durchlaufen und damit den Baum aufzubauen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1309_TreeView.mdb**.

TreeView mit Tabellendaten füllen

Als Access-Entwickler sind Sie natürlich interessiert, wie Sie das **TreeView**-Steuerelement mit den Daten aus einer oder mehreren Tabellen füllen. Im Wesentlichen gibt es folgende Szenarien:

- Die Daten stammen aus hierarchisch verknüpften Tabellen wie **tblKunden, tblBestellungen und tblBestellpositionen**.
- Die Daten stammen aus einer Tabelle mit reflexiver Verknüpfung.

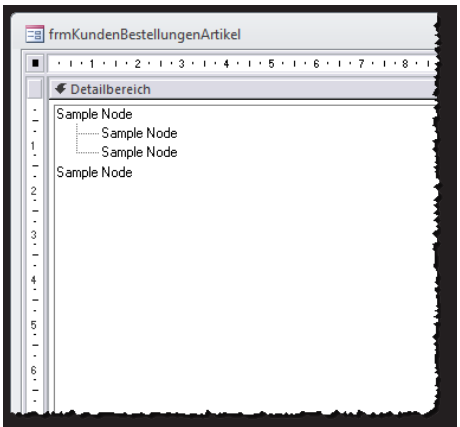


Bild 2: Formular mit dem frisch hinzugefügten TreeView-Steuerelement in der Entwurfsansicht

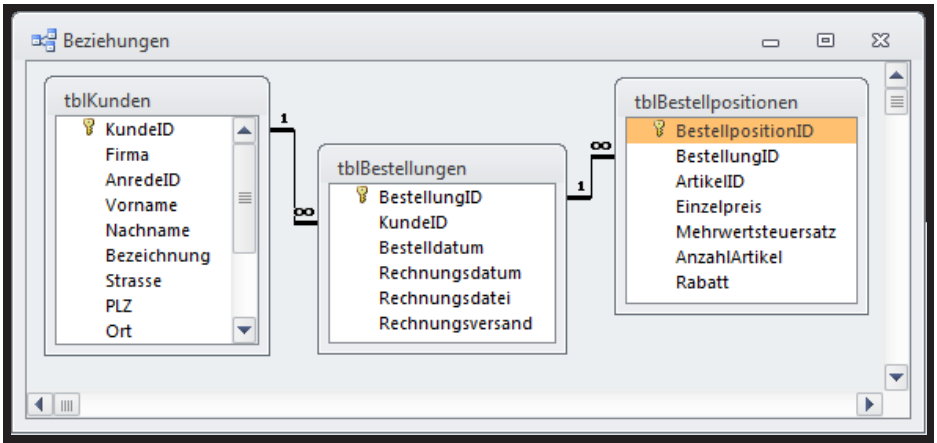


Bild 1: Datenherkunft für die Anzeige hierarchischer Daten im TreeView-Steuerelement

- Die beiden vorherigen Varianten werden kombiniert.

Wir schauen uns in der vorliegenden Artikelreihe zunächst die erste Variante an, um die weiteren kümmern wir uns in folgenden Beiträgen.

Kunden, Bestellungen und Artikel

Im ersten Beispiel soll das **TreeView**-Steuerelement die Daten der Tabellen **tblKunden**, **tblBestellungen**, **tblBestelldetails** und **tblArtikel** abbilden. Die Tabellen sind wie gefordert aufeinander aufbauend per 1:n-Beziehung miteinander verknüpft (siehe Bild 1). Das heißt, die Daten der Tabelle **tblKunden** sollen ganz links im **TreeView** angezeigt werden, die Daten der Tabelle **tblBestellungen** rechts unterhalb des jeweiligen Kunden und die Daten der Tabelle **tblBestellpositionen** wiederum rechts unterhalb der jeweiligen Bestellungen.

Dabei soll die Datenherkunft für diese Ebene im **TreeView**-Steuerelement nicht nur Daten der Tabelle **tblBestellpositionen** enthalten, sondern auch noch den Artikelnamen aus der mit dieser Tabelle verknüpften Tabelle **tblArtikel**.

Um dies zu erreichen, legen Sie zunächst ein neues Formular namens **frmKundenBestellungenArtikel** an und fügen diesem ein **TreeView**-Steuerelement hinzu, dass Sie mit dem Namen **ctlTreeView** versehen (siehe Bild 2).

Verweis auf das TreeView-Steuerelement

Das TreeView wollen wir wieder mit einer Variablen namens **m_TreeView** referenzieren, die wie folgt im Kopf des Klassenmoduls **Form_frmKundenBestellungenArtikel** deklariert wird (in einer Zeile):

```
Dim WithEvents m_Treeview As
MSComctlLib.TreeView
```

Damit diese Variable immer gefüllt ist und wir über den Ausdruck **objTreeView** immer darauf zugreifen können, also auch, wenn einmal ein Laufzeitfehler die Variable geleert hat, verwenden wir die **Property**-Prozedur **objTreeView** aus Listing 1.

Eigenschaften einstellen

Die Eigenschaften stellen wir gleich per Code ein – so brauchen Sie nur die Einstellungen vorheriger **TreeView**-Steuerelemente zu übernehmen. Die folgende Prozedur, die durch das Ereignis **Beim Laden** des Formulars ausgelöst wird, nutzt auch gleich die **Property**-Prozedur **objTreeView**, um das **TreeView**-Steuerelement zu referenzieren:

```
Private Sub Form_Load()
    With objTreeView
        .Appearance = ccFlat
        .BorderStyle = ccNone
        .LineStyle = tvwRootLines
        .Style = _
            tvwTreelinesPlusMinusText
    End With
    TreeViewFuellen
End Sub
```

Damit stellen wir die Eigenschaften zunächst wie in den Beispielen der bisherigen Artikel zum Thema **TreeView**-Steuerelement ein.

```
Private Property Get objTreeView() As MSComctlLib.TreeView
    If m_Treeview Is Nothing Then
        Set m_Treeview = Me!ctlTreeView.Object
    End If
    Set objTreeView = m_Treeview
End Property
```

Listing 1: Die Property-Prozedur objTreeView liefert immer einen aktuellen Verweis auf das TreeView-Steuerelement.

Kunden hinzufügen

Nun folgt jedoch der Aufruf der Prozedur **TreeViewFuellen**, die wie in Listing 2 aussieht.

Das Listing enthält drei ineinander verschachtelte **Do While**-Schleifen. Die erste durchläuft alle Datensätze eines Recordsets auf Basis der Tabelle **tblKunden** (wobei nur die beiden Felder **BestellungID** und **Firma** zurückgeliefert werden) und fügt für jeden Kunden ein neues Element zum **TreeView**-Steuerelement hinzu.

Der Aufruf der **Add**-Methode verwendet dabei nur zwei der sechs möglichen Parameter, nämlich **Key** und **Text**.

Eindeutiger Schlüssel

Key muss ja, wie in den vorherigen Artikeln beschrieben, innerhalb des **TreeView**-Steuerelements eindeutig sein. In der Regel ist es sinnvoll, zur eindeutigen Kennzeichnung den Primärschlüsselwert der abzubildenden Daten zu verwenden.

Aber wenn Sie, wie hier, Daten aus verschiedenen Tabellen im **TreeView** anzeigen möchten, kann es ja vorkommen, dass ein Primärschlüsselwert in mehreren Tabellen vorkommt. Aber das ist kein Problem: Die **Key**-Eigenschaft darf ja ohnehin nur Werte annehmen, die mit einem Buchstaben beginnen. Also denken wir uns einfach für jede Hierarchie-Ebene einen eindeutigen Buchstaben aus, zum Beispiel **k** für Kunden, **b** für Bestellungen und **p** für Bestellpositionen.

Die Elemente der ersten Ebene, welche die Kunden abbilden, sollen also einen **Key**-Wert erhalten, der sich aus dem Buchstaben **k** und dem jeweiligen Wert des Feldes **KundeID** aus der Tabelle **tblKunden** zusammensetzt.

Nach dem Anlegen des **Node**-Elements für den aktuellen Kundendatensatz erzeugt die Prozedur ein weiteres Recordset, das alle Einträge der Tabelle **tblBestellungen** enthält, deren Fremdschlüsselfeld **KundeID** mit dem Primärschlüsselfeld des aktuell in der äußeren Schleife durchlaufenen Datensatzes entspricht.

Dies erreichen wir durch die Formulierung eines entsprechenden Kriteriums für die Datenherkunft des Recordsets **rstBestellungen**. Dieses Recordset soll die beiden Felder **BestellungID** und **Bestelldatum** liefern – die **BestellungID** als Schlüssel und und das Bestelldatum als anzuzeigenden Wert.

TreeView und Unterformulare

Ein TreeView-Steuerelement kann zwar hierarchische Daten anzeigen, aber es ist nicht dazu geeignet, mehr als einige wenige Informationen zum jeweiligen Element zu liefern. Wenn Sie weitere Informationen zum jeweils markierten Element einblenden möchten, werden Sie dies entweder im gleichen Formular erledigen, in dem sich das TreeView-Steuerelement befindet oder in einem Unterformular-Steuerelement, das gegebenenfalls verschiedene Unterformulare anzeigen kann.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1309_TreeView.mdb**.

Beispielobjekte

In dieser Datenbank finden Sie ein Formular namens **frmKundenBestellungenArtikel** vor, das bereits im Artikel **Das TreeView-Steuerelement: Verknüpfte Daten, Teil I** beschrieben wurde und als Grundlage für das in diesem Artikel erstellte Formular **frmKundenBestellungenArtikel_Unterformular** dient.

Vom TreeView zum Unterformular

Wenn wir beispielsweise das TreeView-Steuerelement aus dem Artikel **Das TreeView-Steuerelement: Verknüpfte Daten, Teil I** betrachten, finden wir zwar bereits eine Menge Informationen vor (siehe Bild 1).

Es zeigt eine Übersicht aller Kunden an und liefert per Mausklick die Datumsangaben zu den bisherigen Bestellungen sowie die bestellten Artikel. Nun wäre es doch richtig praktisch, wenn das Formular, in dem sich das TreeView-Steuerelement befindet, weitere Informationen zum jeweils angeklickten Element anzeigen würde. Wenn der Benutzer auf einen Kunden klickt, soll das Formular beispielsweise die Details zu diesem Kunden liefern, wenn er eine Bestellung anklickt, soll die Bestellung erscheinen – mit Bestelldatum und gegebenenfalls mit Bestellpositionen. Wenn der Benutzer dann auf eine der Bestellpositionen im TreeView klickt, kann das Formular gleich den passenden Artikel samt Details anzeigen – die Bestellpositionen erscheinen ja bereits, wenn der Benutzer auf die Bestellung klickt.

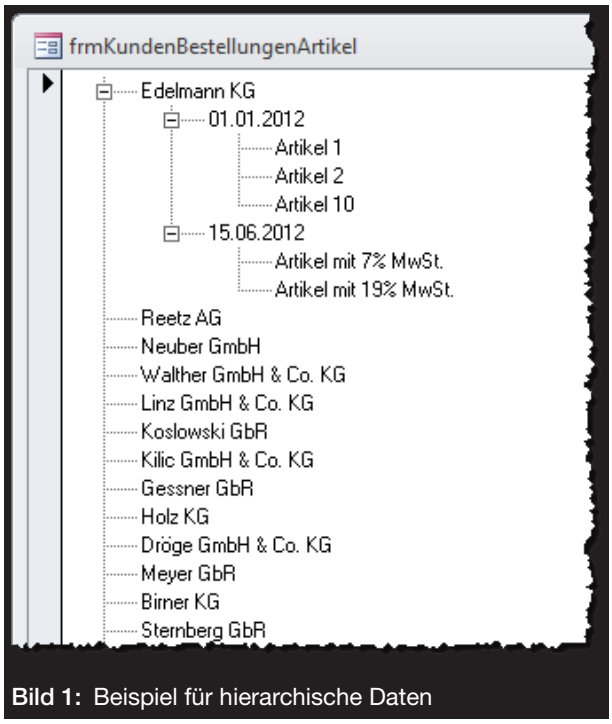


Bild 1: Beispiel für hierarchische Daten

Wie organisieren wir dies? Am einfachsten ist es, verschiedene Unterformular anzulegen, die bei Bedarf in einem Unterformular-Steuerelement eingeblendet werden und den passenden Datensatz anzeigen.

Wir benötigen also drei Unterformulare:

- eines für die Kunden,
- eines für die Bestellung samt Bestelldetails und
- eines für die Artikel.

Unterformulare

Wir werden diese Unterformular recht einfach halten, um den eigentlichen Fokus dieses Artikels nicht aus den Augen zu lassen. Daher in aller Kürze:

Das Unterformular **sfmKunden** soll die Daten der Tabelle **tblKunden** anzeigen.

Daher stellen Sie die Eigenschaft **Datenherkunft** auf diese Tabelle ein und fügen dem Formularentwurf die benötigten Felder der Tabelle hinzu (siehe Bild 2).

Das Unterformular **sfmArtikel** wird über die Eigenschaft Datenherkunft an die Tabelle **tblArtikel** gebunden. Ziehen Sie alle Felder der Tabelle Artikel in den Detailbereich des Formularentwurfs und ordnen Sie diese grob an (siehe Bild 3).

Das Unterformular **sfmBestellungenzeigt** im oberen Bereich die Daten der Bestellung selbst an, also Bestelldatum et cetera. Entsprechend stellen Sie als Datenherkunft die Tabelle **tblBestellungen** ein und fügen die gewünschten Felder zum Entwurf hinzu.

Es soll jedoch noch ein weiteres Unterformular enthalten, das die Bestellpositionen enthält. Dieses verwendet die Tabelle **tblBestellpositionen** als Datenherkunft. Stellen Sie dort außerdem die Eigenschaft **Standardansicht** auf **Datenblatt** ein (siehe Bild 4).

Wenn Sie das Unterformular **sfmBestellpositionen** in das übergeordnete Unterformular **sfmBestellungen** ziehen, sollten die beiden Eigenschaften **Verknüpfen von** und **Verknüpfen nach** des Unterformularsteuerelements automatisch jeweils auf das Feld **BestellungID** eingestellt werden.

Falls nicht, holen Sie dies manuell nach. Dazu markieren Sie das Unterformular-Steuerelement, nicht das Unterformular selbst – dies geht am einfachsten durch Auswählen des entsprechenden Eintrags in der Liste im Eigenschaftsfenster (siehe Bild 5).

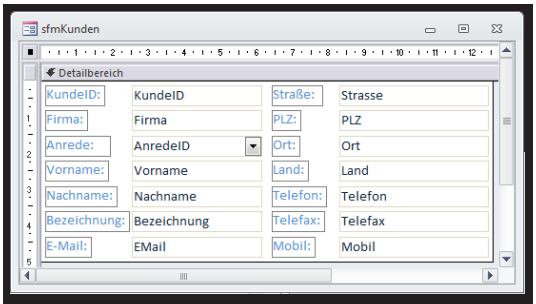


Bild 2: Das Unterformular **sfmKunden** in der Entwurfsansicht

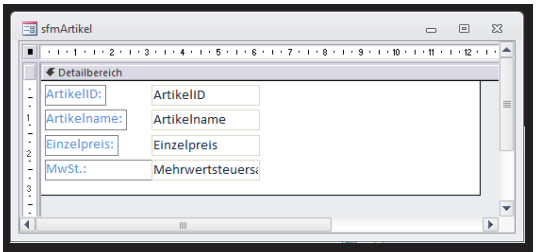


Bild 3: Unterformular zur Anzeige der Artikeldetails

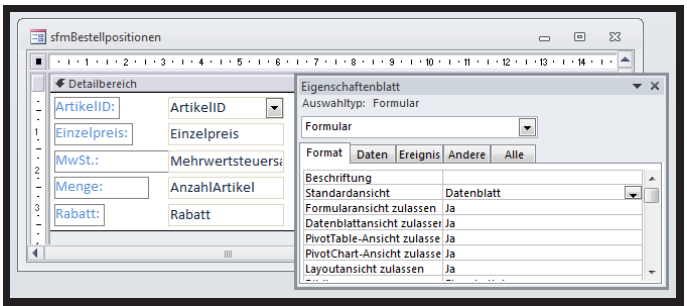


Bild 4: Das Unterformular **sfmBestelldetails** soll in der Datenblattansicht erscheinen.

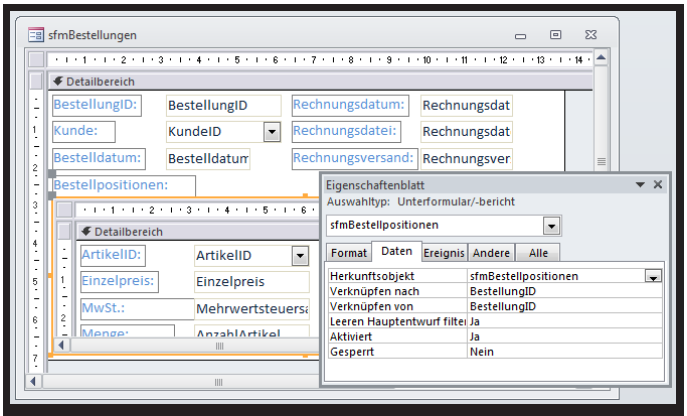


Bild 5: Verknüpfung zwischen Unterformular und übergeordnetem Hauptformular

Unterformular-Steuerelement

Nach diesen Vorbereitungsarbeiten kümmern wir uns um die Anzeige der verschiedenen Unterformulare in dem Hauptformular, das auch das TreeView-Steuerelement enthält. Dazu fügen Sie zunächst ein noch leeres Unterformular-Steuerelement zum Formular **frmKundenBestellungenArtikel_Unterformular** hinzu. Das Unterformular-Steuerelement soll den Namen **sfm** erhalten, das Bezeichnungsfeld benennen Sie in **lblUnterformular** um. Die Steuerelemente sollen schließlich etwa wie in Bild 6 angeordnet sein.

Aktion bei Mausklick

Nun haben wir ein TreeView-Steuerelement und ein leeres Unterformular-Steuerelement sowie eine Reihe von Unterformularen, die zu verschiedenen Zeitpunkten eingeblendet werden sollen. Damit dies geschieht, müssen wir für das TreeView-Steuerelement ein entsprechendes Ereignis anlegen.

Die Deklaration der Objektvariablen, welche das **TreeView**-Steuerelement

Bilder in Access, Teil II: Bilder in OLE-Feldern

Für alle, die Bilder flexibel in allen Access-Versionen einbinden und verwenden möchten, bietet sich das OLE-FEld an. Dabei füllen Sie die Daten nicht so hinein, wie es der erste Teil dieser Artikelreihe beschrieben hat, sondern als Byte-Array. Das heißt, dass Sie die komplette Datei, so wie sie auch auf der Festplatte liegt, in das OLE-Feld schreiben. Von dort aus können Sie die Datei beispielsweise wieder auf die Festplatte zurückschreiben oder diese als Bild an verschiedenen Stellen anzeigen – beispielsweise als Icons im TreeView- oder ListView-Steuerelement.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1309_Bilder.mdb**.

Bild binär im OLE-Feld speichern

Bevor wir uns diese Technik ansehen, gleich die schlechte Nachricht: Es gelingt ausschließlich mit VBA. Aber das ist kein Problem, auf diesem Wege erhalten Sie gleich einige interessante Grundlagen. Bevor wir beginnen, erstellen wir eine kleine Beispieltabelle namens **tblOLEBilder**, welche die drei Felder **OLEBildID**, **OLEBild** und **Bildname** enthält (siehe Bild 1).

Außerdem erstellen wir ein Formular, mit dem wir den Import und den Export der Bilder steuern. Dieses Formular soll **frmOLEBilder** heißen. Wir statten es zunächst mit einer Schaltfläche aus, mit der Sie einen Dateiauswahl-Dialog anzeigen können. Diese erhält die Beschriftung **Dateien hinzufügen** und den Namen **cmdDateienHinzufuegen**. Eine weitere Schaltfläche mit der Beschriftung **Datei exportieren** und dem Namen **cmdDateiexport** legen Sie gleich daneben an. Das Formular enthält außerdem ein Unterformular-Steuerelement, welches das Unterformular **sfmOLEBilder** aufnimmt. Dieses Formular verwendet die soeben erstellte Tabelle **tblOLEBilder** als Datenherkunft und zeigt diese in der Datenblatt-Ansicht an. Im Entwurf sieht das Formular wie in Bild 2 aus.

Einlesen einer Bilddatei

Die Schaltfläche **cmdDateiHinzufuegen** löst die Prozedur aus Listing 1 aus. Diese deklariert zunächst zwei Variablen: strDatei nimmt eine durch das Tabulator-Zeichen getrennte Liste mehrere Dateien auf, strDateien() ist ein String-Array, dass jeweils einen der durch die Tabulator-Zeichen getrennten Zeichenketten pro Datenfeld aufnimmt. Was steckt dahinter? Wir verwenden eine Funk-

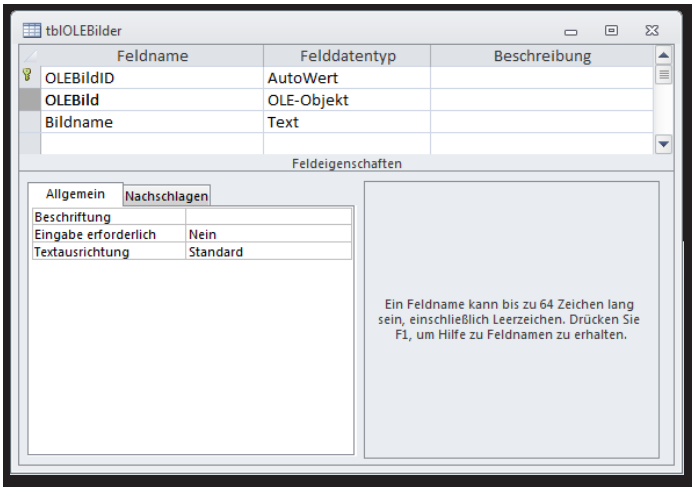


Bild 1: Tabelle mit einem OLE-Feld

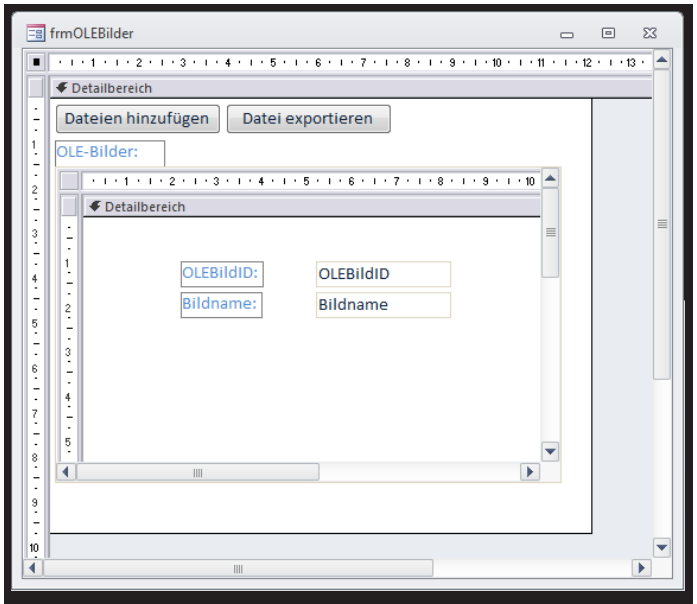


Bild 2: Formular zum Importieren und Exportieren von Bilddateien

tion namens **OpenFileNameMultipe** zum Anzeigen eines Dateiauswahl-Dialogs, der die Auswahl mehrerer Dateien gleichzeitig erlaubt. Diese Funktion ist so ähnlich aufgebaut wie die aus dem Artikel **Dialog zur Auswahl von Dateien anzeigen**. Sie finden diese Funktion im Modul **mdlTools** der Beispieldatenbank.

Der Aufruf der Funktion **OpenFileNameMultiple** erwartet als Parameter das Startverzeichnis (hier das mit **CurrentProject.Path** ermittelte aktuelle Anwendungsverzeichnis), einen Titel sowie die Angabe des Filters für die anzuzeigenden Dateien.

Das Ergebnis des Dialogs, der etwa wie in Bild 3 aussieht, ist die erwähnte Zeichenkette, welche Pfad und Name aller ausgewählten Dateien getrennt durch das Tabulator-Zeichen enthält. Wir verwenden nun die Split-Funktion von VBA, um die einzelnen Elemente dieser Zeichenkette auf die Felder eines Arrays aufzuteilen, und zwar genau an den Tabulator-Zeichen, die durch die Zeichenkonstante **vbTab** identifiziert werden.

Dieses Array durchlaufen wir nun in einer **For...Next**-Schleife, die den Index des ersten Elements mit der **LBound**-Funktion ermittelt und den des letzten Elements mit **UBound**. Für alle in den Feldern enthaltenen Dateinamen ruft die Prozedur jeweils einmal eine weitere Routine namens **DateiEinlesen** auf.

Bilddatei einlesen

Diese finden Sie in Listing 2. Die Prozedur erwartet den Dateinamen als Parameter. Sie deklariert ein **Database**- und ein **Recordset**-Objekt, um per DAO auf die Tabelle **tblOLE-Bilder** zugreifen und das neue Bild in einen neuen Datensatz einfügen zu können. Das dazu erstellte Recordset liefert allerdings keine Datensätze, da es als Kriterium den Ausdruck 1=2 verwendet. Der Grund ist, dass diese Prozedur lediglich einen Datensatz anfügen soll und nicht auf vorhandene Datensätze zugreift. Die Variable **lngFileID** nimmt die Nummer auf, mit der die später einzulesende Datei beim Öffnen gekennzeichnet wird. Damit eine noch nicht verwendete Nummer für den Zugriff verwendet wird, ermitteln wir diese mit der VBA-Funktion **FreeFile** und speichern sie in **lngFileID**.

Fehlt noch die Variable **Buffer()**, die als Array mit dem Datentyp **Byte** deklariert wird und den Inhalt der einzulesenden Datei aufnehmen soll. Die Prozedur öffnet dann mit der **Open**-Anweisung die mit **strDatei**

```
Private Sub cmdDateienHinzufuegen_Click()
    Dim strDatei As String
    Dim strDateien() As String
    Dim i As Integer
    strDatei = OpenFileNameMultiple(CurrentProject.Path, "Bild auswählen", _
        "Bilddateien (*.png;*.bmp,*.jpg,*.tif,*.gif)|Alle Dateien (*.*)")
    strDateien = Split(strDatei, vbTab)
    For i = LBound(strDateien) To UBound(strDateien)
        DateiEinlesen strDateien(i)
    Next i
End Sub
```

Listing 1: Vorbereitungen für das Einlesen einer oder mehrerer Dateien in ein OLE-Feld

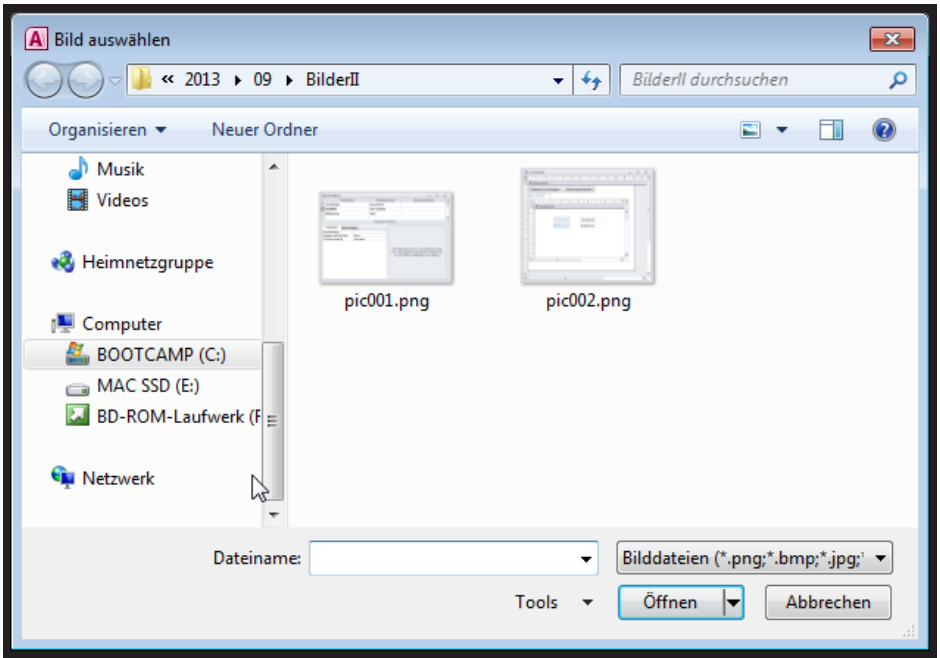


Bild 3: Dialog zum Auswählen der zu importierenden Dateien

übergebene Datei, und zwar für den binären Zugriff und referenziert diese mit der in **lngFileID** gespeicherten Dateinummer.

Die **LOF**-Funktion ermittelt dann die Länge der so geöffneten Datei und speichert die Länge in der Variablen **lngFileLen**. Damit das Byte-Array **Buffer** die Datei aufnehmen kann, wird sie mit **ReDim** auf die entsprechende Anzahl Felder redimensioniert.

Dann beginnt die Bearbeitung der Tabelle. Die **AddNew**-Methode fügt zunächst einen neuen Datensatz hinzu, dessen Feld **Bildname** zunächst mit dem Dateinamen ohne Verzeichnis gefüllt wird. Den Bildnamen ermittelt die **Mid**-Funktion, die von dem im **strDatei** gespeicherten Wert die ersten Zeichen abschneidet. Die Anzahl dieser Zeichen ergibt sich durch das Ergebnis der **InStrRev**-Funktion, welche die Position des ersten Backslash-Zeichen von hinten ermittelt – typischerweise gehören alle folgenden Zeichen zum Dateinamen. Damit das gefundene Backslash-Zeichen ebenfalls abgeschnitten wird, addiert der Ausdruck noch den Wert **1** hinzu.

```
Public Sub DateiEinlesen(strDatei As String)
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim lngFileID As Long
    Dim Buffer() As Byte
    Dim lngFileLen As Long
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM tblOLEBilder WHERE 1=2", dbOpenDynaset)
    rst.AddNew
    lngfileid = FreeFile
    Open strDatei For Binary Access Read Lock Read Write As lngFileID
    lngFileLen = LOF(lngFileID)
    ReDim Buffer(lngFileLen)
    rst.AddNew
    rst!OLEBild = Null
    rst!Bildname = Mid(strDatei, InStrRev(strDatei, "\") + 1)
    Get lngfileid, , Buffer
    rst!OLEBild.AppendChunk Buffer
    rst.Update
    rst.Close
    Set rst = Nothing
    Set db = Nothing
    Me!sfmOLEBilder.Form.Requery
End Sub
```

Listing 2: Einlesen einer Datei in ein OLE-Feld

Nun folgt das Einlesen der Datei, und zwar in zwei Schritten: Im ersten Schritt lädt die **Get**-Anweisung den Inhalt der Datei in das Byte-Array **Buffer**, die folgende Anweisung schließt die Daten, da diese nicht mehr benötigt wird. Der zweite Schritt ist das Übertragen des Inhalts des Byte-Arrays in das OLE-Feld. Dazu verwenden wir die **AppendChunk**-Methode des Feldes **OLEBild**.

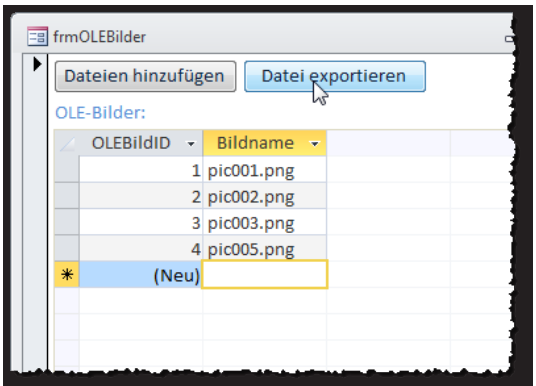


Bild 4: Unterformular mit einigen Datensätzen

Damit die Änderungen im neuen Datensatz gespeichert werden, führt die Prozedur noch die **Update**-Methode aus. Die übrigen Anweisungen schließen und leeren die verwendeten Objekte und aktualisieren schließlich die Daten des Unterformulars.

Das Formular sieht nun etwa wie in Bild 4 aus.

Vom OLE-Feld zur Datei

Wenn Sie schon Bilder im OLE-Feld einer Tabelle speichern, möchten Sie diese vielleicht auch wieder zurück in das Dateisystem exportieren. Dazu haben wir die zweite Schaltfläche zum Formular hinzugefügt, welche die Prozedur aus Bild 3 auslöst. Die Prozedur ermittelt zunächst den Primärschlüsselwert des aktuell im Unterformular markierten Datensatzes und speichert den gefundenen Wert in der Variablen **lngOLEBildID**. Befindet sich der Datensatzzeiger auf einem leeren, neuen Datensatz, erhält **lngOLEBildID** den Wert **0**. In diesem Fall wird die Prozedur mit einer entsprechenden Meldung abgebrochen. Anderenfalls ermittelt die Prozedur den Namen der Bilddatei aus dem Feld **Bildname** des Unterformulars und den Speicherort über eine weitere Hilfsfunktion zum Anzeigen eines Dateiauswahl-Dialogs,

Kontextmenüs

Mit Access 2007 hat Microsoft das Ribbon eingeführt und Access seiner Werkzeuge beraubt, Menüleisten und Kontextmenüs über die Benutzeroberfläche zu erstellen. Nun gut: Menüleisten gibt es nicht mehr, aber Kontextmenüs können in professionellen Anwendung doch sehr hilfreich und vor allem ergonomisch sein. Dieser Artikel zeigt den alternativen Weg zur Erstellung von Kontextmenüs – den über VBA.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1310_Kontextmenues.mdb**.

Kontextmenüs

In Access 2003 und jünger konnten Sie nicht nur Kontextmenüs, sondern auch andere Menüs zusammenklicken. Wenn Sie jedoch einmal dynamisch generierte Kontextmenüs benötigten, mussten Sie auch dort zu VBA greifen – beispielsweise, um verschiedene Werte einer Tabelle per Kontextmenü zur Auswahl anzubieten. Dementsprechend können Sie die in diesem Artikel vorgestellten Techniken auch in Datenbanken einsetzen, die mit Access älteren Jahrgangs als 2007 erstellt wurden.

Sollte Ihnen der Begriff Kontextmenü fremd sein: Das sind diejenigen Menüs, die aufgeklappt werden, wenn Sie mit der rechten Maustaste auf ein Objekt klicken (zum Beispiel auf eine Datei oder einen Ordner im Windows Explorer). Diese Kontextmenüs können Sie auch für Ihre eigenen Anwendungen anlegen und an den entsprechenden Stellen anbieten.

Einsatzbereiche

Access selbst bietet ja bereits eine ganze Reihe eingebauter Kontextmenüs für die verschiedenen Objekte an – für die Spaltenköpfe oder die Felder der Datenblattansicht, für Steuerelemente in Formularen, für Formulare selbst et cetera (siehe Bild 1). Warum sollten Sie also eigene Kontextmenüs hinzufügen? Oder sogar die vorhandenen Kontextmenüs ersetzen? Ganz einfach: Wenn Sie eine eigene Anwendung erstellen, dann enthält diese auch die von Ihnen definierte Benutzeroberfläche – und dieser können Sie neben den üblichen Ribbons, Formularen und Steuerelementen auch Kontextmenüs hinzufügen.

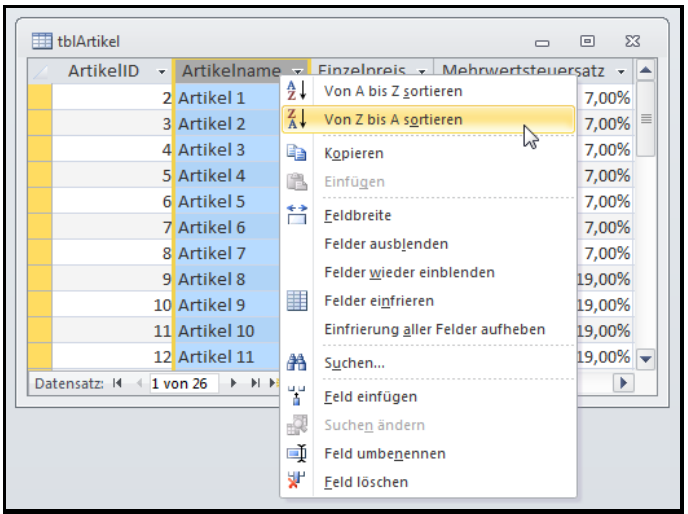


Bild 1: Beispiel für ein eingebautes Kontextmenü

Sehr sinnvoll sind Kontextmenüs beispielsweise in Zusammenhang mit TreeView-Steuerelementen, da Sie so leicht Funktionen zu den angezeigten Elementen hinzufügen können – etwa zum Bearbeiten des angeklickten Eintrags, zum Löschen des Beitrags oder auch zum Anlegen neuer Einträge.

Vielleicht möchten Sie auch nur eingebaute Steuerelemente anpassen, indem Sie vorhandene Einträge ausblenden oder deaktivieren oder eigene Einträge hinzufügen. So könnten Sie zum Beispiel das Kontextmenü eines Textfeldes um den Kopieren- oder den Ausschneiden-Befehl erleichtern, wenn Sie nicht möchten, dass der Benutzer auf einfache Weise die Inhalte des Textfeldes in die Zwischenablage einfügt. Für all diese Zwecke müssen wir zunächst einen Blick auf die VBA-Objekte zum Steuern des Aussehens von Kontextmenüs werfen.

Kontextmenüs per VBA

Dazu müssen Sie zunächst wissen, dass die Kontextmenüs keine Access-interne Komponente sind, sondern dass die notwendigen Objekte in der Office-Bibliothek enthalten sind. Und da diese einem VBA-Pro-

jekt standardmäßig nicht zugewiesen ist, fügen Sie den entsprechenden Verweis nun zunächst hinzu. Dazu öffnen Sie den VBA-Editor (zum Beispiel mit **Strg + G**) und betätigen den Menübefehl **Extras|Verweise**. Fügen Sie dort den Eintrag **Microsoft Office x.0 Object Library** hinzu (siehe Bild 2).

Anschließend können Sie im Code-Fenster auf die Objekte dieser Bibliothek zugreifen. Zu Testzwecken wollen wir zunächst eine Liste aller aktuell in der Datenbankdatei enthaltenen Kontextmenüs im Direktfenster ausgeben. Dabei lernen Sie dann gleich einige Objekte und Eigenschaften der Kontextmenüs unter VBA kennen. Der dazu notwendige Code steckt in der Prozedur **AlleMenues** und sieht wie folgt aus:

```
Public Sub AlleMenues()
    Dim cbr As Office.CommandBar
    For Each cbr In 7
                Application.CommandBars
        Debug.Print cbr.Name
    Next cbr
End Sub
```

Die Prozedur deklariert eine Variable des Typs **CommandBar**, was einer Menüleiste, einer Symbolleiste oder einem Kontextmenü entspricht. Mithilfe dieser Variablen durchläuft die Prozedur in einer **For Each**-Schleife alle Elemente der **CommandBars**-Auflistung des **Application**-Objekts, was der aktuellen Instanz von Access entspricht. Sie gibt dann über die Eigenschaft **Name** den Namen des aktuellen **CommandBar**-Objekts aus.

Wenn Sie diese Prozedur eingeben und starten, rauschen eine ganze Reihe Einträge durch den Direktbereich. Wieviele waren das? Das finden Sie durch Absetzen der folgenden Anweisung im Direktfenster heraus:

```
? Commandbars.Count
 209
```

Unter Access 2010 etwa gibt es also über 200 Menüs! Allerdings sind dies nicht alles Kontextmenüs. Informationen über die Art des Menüs liefert die Eigenschaft **Type**. Wollen wir die Namen der Menüs mit Typ ausgeben, erweitern Sie die **Debug.Print**-Zeile der obigen Prozedur wie folgt:

```
Debug.Print cbr.Name, cbr.Type
```

Eingebaut oder nicht?

Eine weitere interessante Eigenschaft des **CommandBar**-Objekts heißt **BuiltIn**. Diese Eigenschaft gibt den Wert **True** zurück, wenn es sich um ein benutzerdefiniertes Menü handelt, also eines, dass durch den Code der Anwendung, durch Import oder durch Erstellen mit den Tools von Access 2003 und älter erstellt wurde. Wenn Sie also beispielsweise einmal alle eingebauten Kontextmenüs entfernen möchten, liefert die Eigenschaft **Builtin** alle notwendigen Informationen.

Arten von Menüleisten

Um herauszufinden, welche der **CommandBar**-Objekte Kontext-

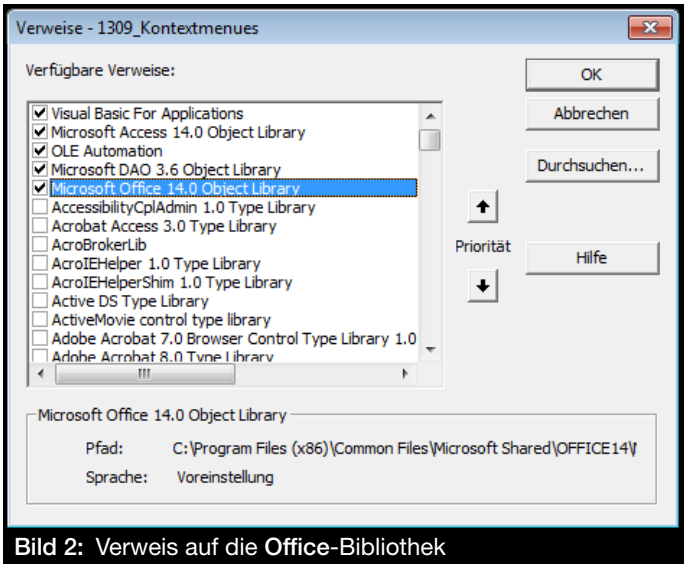


Bild 2: Verweis auf die **Office**-Bibliothek

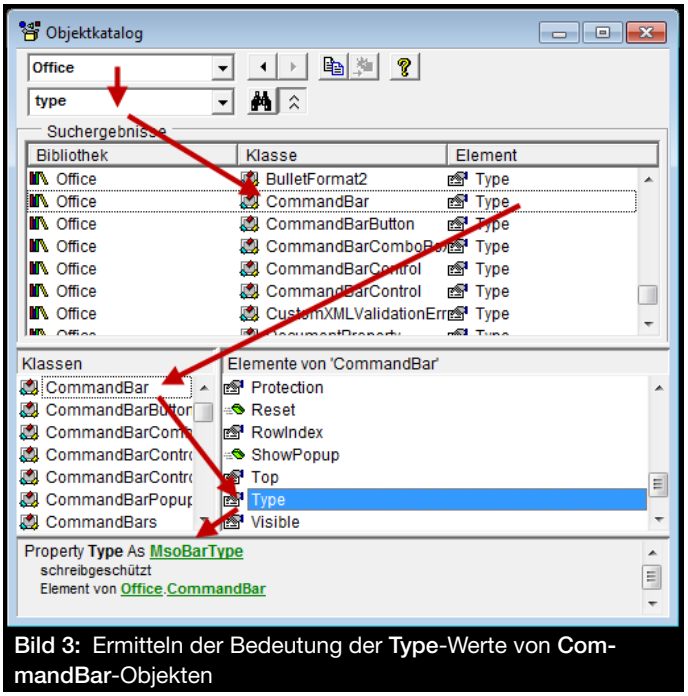


Bild 3: Ermitteln der Bedeutung der **Type**-Werte von **CommandBar**-Objekten

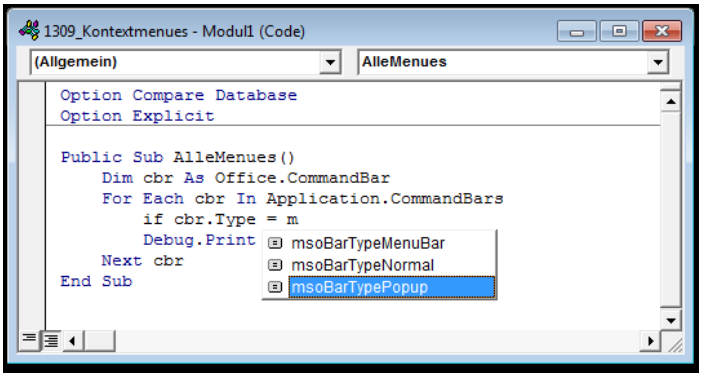


Bild 4: Auch IntelliSense liefert die Namen der Konstanten einer Enumeration.

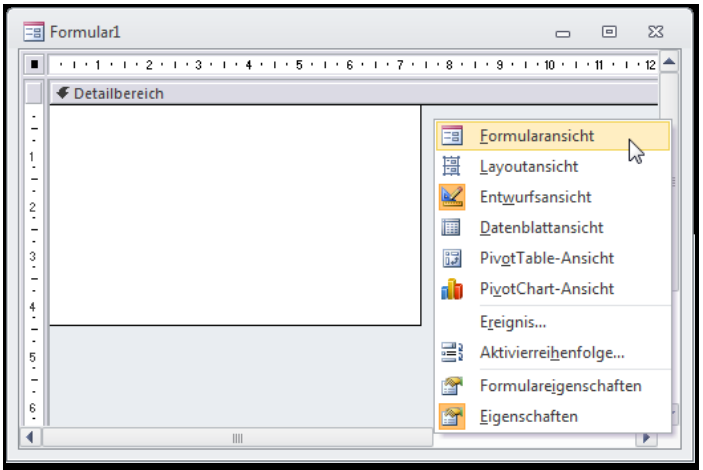


Bild 5: Kontextmenü der Entwurfsansicht eines Formulars

menüs repräsentieren, müssen wir zunächst ermitteln, was die Zahlenwerte **0**, **1** und **2** bedeuten. Dazu bemühen wir den Objekt-Explorer (**F2**), siehe Bild 3. Dort filtern wir zunächst nach der Bibliothek, in der sich die entsprechenden Konstanten befinden könnten, also **Office** (für **Microsoft Office x.0 Object Library**).

Dann suchen wir nach dem Namen der Eigenschaft, hier **Type**. Darüber gelangen wir an die **Type**-Eigenschaft des **CommandBar**-Objekts. Wählen Sie diese aus, erhalten Sie unten weitere Informationen – nämlich den Namen der Enumeration, welche die Konstanten für die Eigenschaft **Type** enthält (Sie hätten auch direkt den Weg über das **CommandBar**-Objekt gehen und dann nach der Eigenschaft **Type** suchen können). Klicken Sie dort auf die Enumeration **mso-BarType**, finden Sie schnell die möglichen Werte:

- **0**: **msoBarTypeNormal** (Symbolleiste)
- **1**: **msoBarTypeMenuBar** (Menüleiste)
- **2**: **msoBarTypePopup** (Kontextmenü)

Es gibt allerdings noch einen anderen Weg, um an die Konstanten zu gelangen. Dazu geben Sie einfach eine Anweisung in den VBA-Editor ein, mit der Sie der Eigenschaft **Type** einen Wert zuweisen wollen.

Dank IntelliSense liefert der VBA-Editor an dieser Stelle die gewünschten Informationen (siehe Bild 4).

```
Public Sub KontextmenuesMitSteuerelementen()
    Dim cbr As Office.CommandBar
    Dim cbc As Office.CommandBarControl
    Set cbr = CommandBars("Form Design Form")
    Debug.Print cbr.Name, cbr.Type
    For Each cbc In cbr.Controls
        Debug.Print "  " & cbc.Caption
    Next cbc
End Sub
```

Listing 1: Ausgabe der Steuerelemente eines Kontextmenüs

Fügen wir also eine **If...Then**-Bedingung hinzu, welche nur noch die Namen der Kontextmenüs ausgibt:

```
If cbr.Type = msoBarTypePopup Then
    Debug.Print cbr.Name, cbr.Type
End If
```

Steuerelemente ausgeben

Als Nächstes interessieren uns natürlich die Einträge der einzelnen Kontextmenüs. Um diese zu referenzieren und auszugeben, legen wir wiederum eine entsprechende Objektvariable fest, die diesmal den Namen **cbc** hat und den Datentyp **CommandBarControl** aufweist. Also geben wir einmal alle Steuerelemente eines bestimmten eingebauten Kontextmenüs aus, zum Beispiel **Form Design Form**. Dabei handelt es sich um das Kontextmenü, das erscheint, wenn Sie in den leeren Bereich im Formularentwurf klicken (siehe Bild 5).

Der Code, um die Namen der Steuerelemente auszugeben, sieht wie in Listing 1 aus. Die Prozedur stellt die Variable **cbr** auf das genannte Kontextmenü ein und durchläuft dann

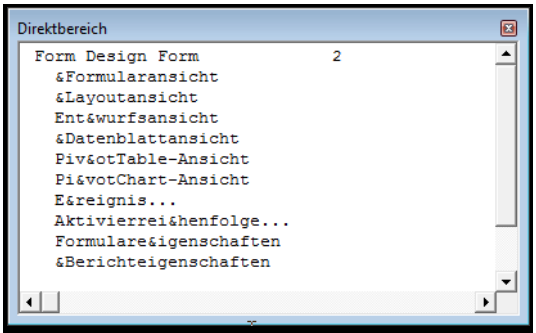


Bild 6: Ausgabe aller Controls eines CommandBars

über die Variable **cbc** alle Elemente der **Controls**-Auflistung des **CommandBar**-Objekts.

Das Ergebnis sieht dann wie in Bild 6 aus. Die Prozedur hat alle Elemente des **CommandBar**-Objekts in das Direktfenster geschrieben. Sie werden möglicherweise feststellen, dass ein Element nur im Direktbereich ausgegeben wird, aber nicht im Kontextmenü erscheint – nämlich der Eintrag **&Berichtseigenschaften**.

Eine Erklärung hierfür gibt es nicht – außer, dass Microsoft da wohl einen Eintrag vergessen hat. Gelegentlich kommt es auch vor, dass ein Eintrag in einer bestimmten Konstellation nicht im Kontextmenü angezeigt wird.

Woher aber weiß ich, dass das Kontextmenü **Form Design Form** tatsächlich im vermuteten Kontext angezeigt wird? Vielleicht wird dieses ja für einen ganz anderen Zweck benötigt, was auch den zusätzlichen Eintrag erklären würde? Um dies zu prüfen, gibt es einen einfachen Weg: Wir fügen dem Kontextmenü namens **Form Design Form** einfach einen Eintrag hinzu, beispielsweise mit dem Namen **Test**.

Das ist mit den Anweisungen aus Listing 2 schnell erledigt. Die erste **Set**-Anweisung referenziert das zu erweiternde Kontextmenü. Die zweite setzt einen Verweis auf das mit der **Add**-Methode hinzugefügte Element des Typs **msoControlButton**. Damit das hinzugefügte Element beim nächsten Öffnen von Access wieder verschwunden ist, erhält der letzte Parameter namens **Temporary** den Wert **True**. Dem neuen Eintrag verpassen wir mit der **Caption**-Eigenschaft noch die Bezeichnung **Test**.

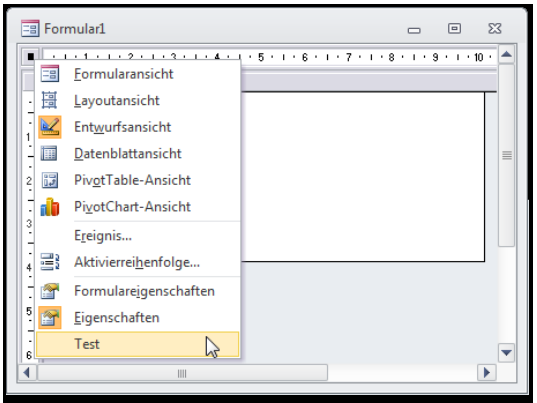


Bild 7: Kontextmenü mit einem benutzerdefinierten Eintrag

Wie **Bild 7** zeigt, erscheint der neue Eintrag im angegebenen Kontextmenü, das sich übrigens auch durch einen Rechtsklick auf den Schnittpunkt der beiden Lineale des Formularentwurfs anzeigen lässt.

Kontextmenü anlegen

Nun wollen wir uns der Erstellung komplett benutzerdefinierter Kontextmenüs zuwenden. Dazu müssen Sie zunächst ein neues **CommandBar**-Objekt erstellen und dieses mit einer Objektvariablen etwa namens **cbr** referenzieren (siehe Listing 3). Dazu

```
Public Sub KontextmenueErweitern()
    Dim cbr As Office.CommandBar
    Dim cbc As Office.CommandBarControl
    Set cbr = CommandBars("Form Design Form")
    Set cbc = cbr.Controls.Add(msoControlButton, , , , True)
    With cbc
        .Caption = "Test"
    End With
End Sub
```

Listing 2: Erweitern eines bestehenden Kontextmenüs

```
Public Sub KontextmenueAnlegen()
    Dim cbr As Office.CommandBar
    Dim cbc As Office.CommandBarControl
    On Error Resume Next
    CommandBars("cbrTest").Delete
    On Error GoTo 0
    Set cbr = CommandBars.Add("cbrTest", msoBarPopup, False, True)
    Set cbc = cbr.Controls.Add(msoControlButton, , , , True)
    With cbc
        .Caption = "Test"
        .OnAction = "=Test()"
    End With
    cbr.ShowPopup
End Sub
```

Listing 3: Anlegen eines neuen Kontextmenüs mit einem Beispieleintrag

Das TreeView-Steuerelement: Verknüpfte Daten, Teil II

Mit dem TreeView-Steuerelement zeigen Sie nicht nur die Daten aus mehreren hierarchisch verknüpften Tabellen an, sondern auch solche aus Tabellen mit reflexiver Verknüpfung. Das gängige Beispiel hierfür sind Vorgesetzte und ihre Mitarbeiter. Dieser Artikel zeigt die notwendigen Tabellen sowie die Funktion, mit der Sie ein TreeView-Steuerelement mit den gewünschten Daten füllen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1310_TreeView.mdb**.

Mitarbeiterhierarchie im TreeView-Steuerelement

Ziel dieses Artikels ist es, die Mitarbeiter einer Mitarbeiter-Tabelle hierarchisch in einem **TreeView**-Steuerelement anzuzeigen. Das Ergebnis sieht beispielsweise wie in Bild 1 aus.

Formular erstellen

Wir benötigen wieder ein Formular mit einem **TreeView**-Steuerelement namens **ctlTreeView**. Das Formular finden Sie in der Beispieldatenbank unter dem Namen **frmTreeViewMitarbeiter**. Für das Formular stellen Sie die Eigenschaften **Datensatzmarkierer**, **Navigationsschaltflächen** und **Bildlaufleisten** sowie **Trennlinien** auf **Nein**, die Eigenschaft **Automatisch zentrieren** auf **Ja** ein. Außerdem legen Sie für die Eigenschaft **Rahmenart** des **TreeView**-Steuerelements den Wert **Durchgezogen** fest (siehe Bild 2).

Die übrigen Eigenschaften des **TreeView**-Steuerelements definieren wir wie gewohnt über die entsprechenden VBA-Eigenschaften, die in der Ereignisprozedur **Form_Load** eingestellt werden (siehe Listing 1). Dabei deklarieren wir zunächst eine Objektvariable zum Referenzieren des **TreeView**-Steuerelements:

```
Dim WithEvents m_Treeview As 7
                              MSCom-
ctlLib.TreeView
```

Um diese Variable zu füllen und darauf zuzugreifen, legen wir eine **Property Get**-Prozedur an. Diese prüft, ob die Variable **m_TreeView** bereits mit einem Verweis gefüllt ist und füllt diesen gegebenenfalls. Der Inhalt von **m_Treeview** wird dann mit **objTreeView** zurückgegeben (siehe Listing 2).

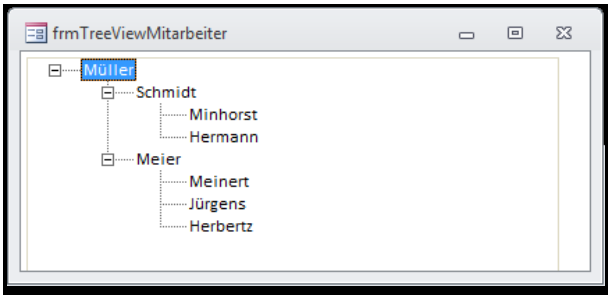


Bild 1: TreeView-Steuerelement zur Abbildung der Mitarbeiter-Hierarchie

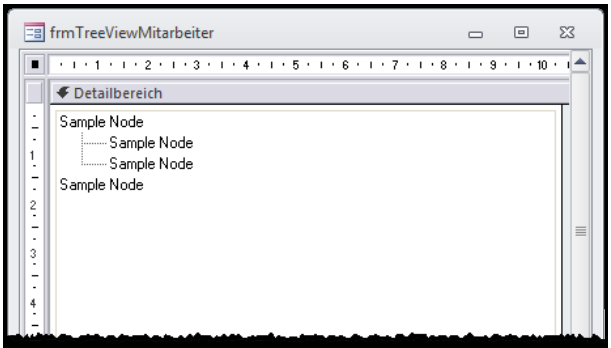


Bild 2: Basis für das zu füllende Formular mit **TreeView**-Steuerelement

```
Private Sub Form_Load()
    With objTreeView
        .Appearance = ccFlat
        .BorderStyle = ccNone
        .LineStyle = tvwRootLines
        .Style = tvwTreelinesPlusMinusText
        .Font.Name = "Calibri"
        .Font.Size = 9
    End With
    TreeViewFuellen
End Sub
```

Listing 1: Einstellen der Eigenschaften des **TreeView**-Steuerelements beim Laden des Formulars und Aufrufen der Funktion zum Füllen.

Reflexive Tabellen als Datenherkunft

Schauen wir uns nun zunächst die Datenherkunft für das **TreeView**-Steuerelement an. Diese besteht in unserem Fall aus ein oder zwei Tabellen – je nachdem, ob Sie das Feld zur Verknüpfung eines Mitarbeiter-Datensatzes mit einem anderen Mitarbeiter-Datensatz zur Mitarbeiter-Tabelle oder zu einer weiteren Tabelle hinzugefügt haben.

Im ersten Fall enthält die Tabelle **tblMitarbeiter** gleich das Feld **VorgesetzterID**, mit dem der Vorgesetzte

```
Private Property Get objTreeView() As MSComctlLib.TreeView
    If m_Treeview Is Nothing Then
        Set m_Treeview = Me!ctlTreeView.Object
    End If
    Set objTreeView = m_Treeview
End Property
```

Listing 2: Die **Property**-Prozedur **objTreeView** liefert immer einen aktuellen Verweis auf das **TreeView**-Steuerelement.

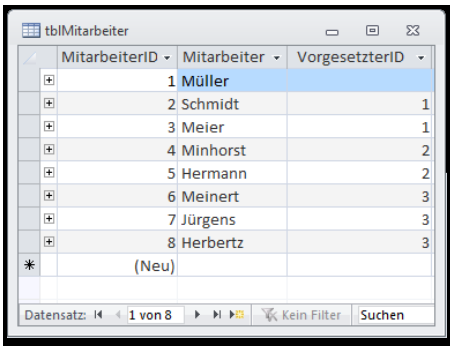


Bild 3: Die reflexiven Daten kommen aus einer einzigen Tabelle.

des aktuellen Mitarbeiters festgelegt wird (siehe Bild 3). Für die Darstellung der Daten aus dieser Tabelle sind keine weiteren Vorarbeiten nötig.

Es kann aber auch sein, dass Sie – was sinnvoller ist – die Informationen über die Beziehung zwischen den Mitarbeitern in einer weiteren Tabelle speichern. In der Beispieldatenbank heißt die Mitarbeitertabelle für diesen Fall **tblMitarbeiterOhne**. Die Tabelle **tblVorgesetzteUntergebene** enthält die beiden Felder **VorgesetzterID** und **UntergebenerID**, welche die Kombinationen aus Vorgesetztem und Untergebenem definieren. Dies sieht dann beispielsweise wie in Bild 4 aus.

Um die Daten so zusammenzuführen, dass diese wie in der einzelnen Tabelle **tblMitarbeiter** dargestellt weden, verwenden wir eine Abfrage wie in Bild 5. Diese enthält die Daten der Tabelle **tblMitarbeiterOhne** und fügt aus der Tabelle **tblVorgesetzteUntergebene** das Feld **VorgesetzterID** hinzu.

Wichtig ist, dass in der Abfrage allein die Beziehung zwischen dem Feld **MitarbeiterID** der Tabelle **tblMitarbeiterOhne** und das Feld **UntergebenerID** der Tabelle **tblVorgesetzteUntergebene** definiert ist.

Die Eigenschaften dieser Beziehung stellen Sie so ein, dass die Abfrage alle Datensätze der Tabelle **tblMitarbeiterOhne** liefert – unabhängig davon, ob es für diesen Mitarbeiter einen über die Tabelle **tblVorgesetzteUntergebene** festgelegten Vorgesetzten gibt. Ist kein solcher vorhanden, bleibt das Feld **VorgesetzterID** schlicht leer. Das ist hilfreich, wenn wir gleich die Mitarbeiter der obersten Ebene ermitteln wollen, also solche, für die kein Vorgesetzter festgelegt wurden.

TreeView füllen

Kommen wir zum Füllen des TreeView-Steuerelements mit den Mitarbeiter-Elementen. Dazu ruft die Prozedur **Form_Load** die Routine **TreeViewFuellen** auf (siehe Listing 3). Die Routine erstellt ein Recordset auf Basis der Abfrage **qryMitarbeiterVorgesetzte**, wobei allerdings nur diejenigen Datensätze berücksichtigt werden, deren Feld **VorgesetzterID**

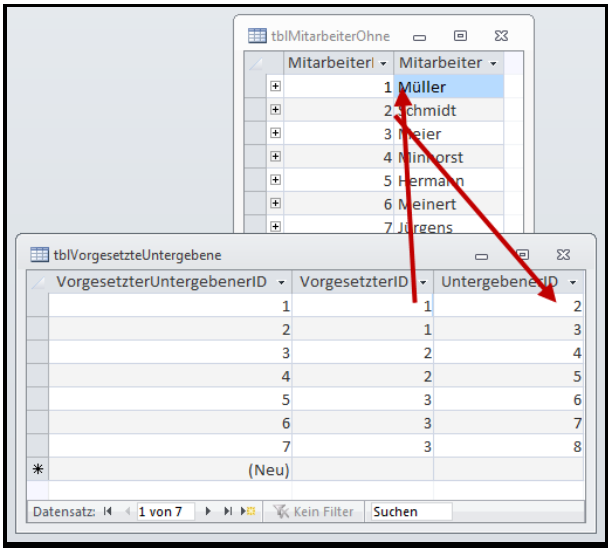


Bild 4: Die Beziehung kann aber auch über eine Verknüpfungstabelle hergestellt werden.

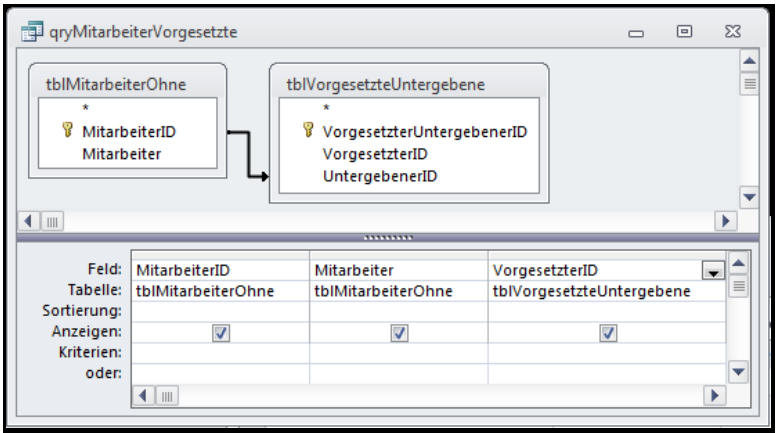


Bild 5: Im Falle der Verknüpfung führen wir die notwendigen Daten in einer Abfrage zusammen.

```
Public Sub TreeViewFuellen()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim objNode As MSComctlLib.Node
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM qryMitarbeiterVorgesetzte WHERE VorgesetzterID IS NULL", dbOpenDynaset)
    Do While Not rst.EOF
        Set objNode = objTreeView.Nodes.Add(, , "m" & rst!MitarbeiterID, rst!Mitarbeiter)
        objNode.Expanded = True
        TreeViewFuellen_Rekursiv db, rst!MitarbeiterID
        rst.MoveNext
    Loop
End Sub
```

Listing 3: Die Prozedur **TreeViewFuellen** füllt die erste Ebene des **TreeView**-Steuerelements.

den Wert **NULL** enthält. Dies entspricht den Mitarbeitern, die keinen Vorgesetzten haben und somit der ersten Hierarchieebene angehören.

Diese durchläuft die Routine in einer **Do While**-Schleife. Innerhalb der Schleife fügt sie mit der **Add**-Methode der **Nodes**-Auflistung des **TreeView**-Steuerelements jeweils ein neues Element zum **TreeView** hinzu und referenziert dieses mit der Objektvariablen **objNode**.

Die beiden Parameter **Key** und **Text** dieser Methode nehmen einen aus dem Buchstaben **m** und dem Primärschlüsselwert des Mitarbeiters und den Namen des Mitarbeiters entgegen. Damit die auch die untergeordneten Elemente angezeigt werden, stellt die Prozedur noch die Eigenschaft **Expanded** auf den Wert **True** ein. Bevor die **Do While**-Schleife sich um den nächsten Datensatz kümmert, ruft die Prozedur noch die rekursiv definierte Prozedur **TreeViewFuellen_Rekursiv** auf. Diese erwartet zwei Parameter:

- den in der Variablen **db** gespeicherten Verweis auf das aktuelle **Database**-Objekt und
- den Primärschlüsselwert für den soeben bearbeiteten Mitarbeiter.

Der Verweis auf das **Database**-Objekt wird übergeben, damit dieses nicht bei jedem rekursiven Aufruf neu ermittelt werden muss. Und der Mitarbeiter mit dem übergebenen Primärschlüsselwert entspricht dem Vorgesetzten für den beim aktuellen Aufruf der rekursiven Prozedur zu ermittelnden Mitarbeiter.

Die rekursive Prozedur **TreeViewFuellen_Rekursiv** ist prinzipiell genauso aufgebaut wie die Funktion **TreeViewFuellen** – mit dem Unterschied, dass diese beim Erstellen der **TreeView**-Einträge noch angeben muss, unter welchem vorhandenen Mitarbeiter der untergebene Mitarbeiter angelegt werden soll (siehe Listing 4). Dies sind die beiden ersten Parameter der **Add**-Methode:

Der Parameter **Relative** erwartet den **Key**-Wert des übergeordneten Elements, das in diesem Fall aus dem Buchstaben **m** und dem mit **lngVorgesetzterID** übergebenen Zahlenwert zusammengesetzt wird. Der zweite Parameter heißt **Relationship** und

```
Public Sub TreeViewFuellen_Rekursiv(db As DAO.Database, lngVorgesetzterID As Long)
    Dim rst As DAO.Recordset
    Dim objNode As MSComctlLib.Node
    Set rst = db.OpenRecordset("SELECT * FROM qryMitarbeiterVorgesetzte WHERE VorgesetzterID = " & lngVorgesetzterID, dbOpenDynaset)
    Do While Not rst.EOF
        Set objNode = objTreeView.Nodes.Add("m" & lngVorgesetzterID, tvwChild, "m" & rst!MitarbeiterID, rst!Mitarbeiter)
        objNode.Expanded = True
        TreeViewFuellen_Rekursiv db, rst!MitarbeiterID
        rst.MoveNext
    Loop
End Sub
```

Listing 4: Die Prozedur **TreeViewFuellen_Rekursive** füllt jeweils eine weitere Ebene des **TreeView**-Steuerelements.

Das Bild-Steuerelement

Wenn es um die Anzeige von Bildern in Access geht, nutzt man meist das eingebaute Bild-Steuerelement. Dieses ist vor allem seit Access 2007 interessant – ab dieser Version liefert es nämlich die Eigenschaft **Steuerelementinhalt**, der Sie einfach den Pfad eines anzuzeigenden Bildes übergeben können. Dieser Artikel liefert alle Informationen, die Sie für die Verwendung des Bild-Steuerelements benötigen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in den Datenbanken **1310_DasBildsteuerelement.mdb** und **1310_DasBildsteuerelement.accdb**.

Wichtig: Speicherformat anpassen

Wenn Sie mit Access 2007 oder jünger arbeiten, was für die meisten Themen in diesem Artikel Voraussetzung ist, sollten Sie bei der Arbeit mit Bilddateien unbedingt die Option **Bildeigenschaften-Speicherformat** in den Access-Optionen auf den Wert **Quellbildformat beibehalten** aktivieren (siehe Bild 1).

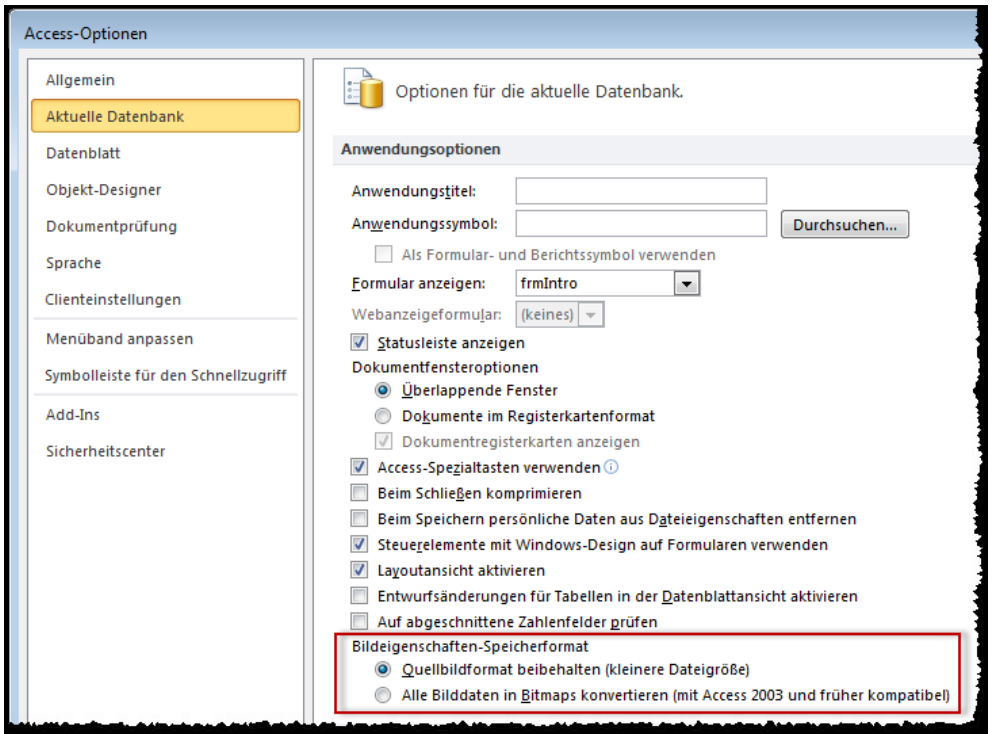


Bild 1: Einstellen des Speicherformats für Bilder

Bild-Steuerelement hinzufügen

Das Bild-Steuerelement fügen Sie wie die übrigen standardmäßig verfügbaren Steuerelemente zu einem Formular hinzu, indem Sie im Ribbon auf das Symbol mit dem TooltipText **Bild** klicken. Nach dem Einfügen in die Entwurfsansicht eines Formulars zeigt Access einen **Grafik einfügen**-Dialog an. Wenn Sie einfach nur ein statisches Bild zum Formular hinzufügen möchten, etwa um ein Icon im Kopf des Formulars zu platzieren, wählen Sie hier die gewünschte Datei aus.

Bild 2: Mit und ohne Anpassung des Speicherformats hinzugefügte Bilder

Anschließend erscheint das Bild an der gewünschten Stelle, beispielsweise wie in Bild 2. Dort haben wir auch dargestellt, wie sich das Einfügen von Bildern unter Beibehaltung des Quellbildformats und beim Konvertieren in Bitmaps verhält. Beide Male wurde das gleiche Bild mit der soeben beschriebenen Methode hinzugefügt. Das Bild, das im Originalformat eingefügt wurde, wird korrekt abgebildet. Das in das Bitmap-Format konvertierte Bild hingegen wurde verändert – und nicht zum Vorteil.

Eigenschaften des Bild-Steuerelements

Das Bild-Steuerelement in der Version ab Access 2007 hat die folgen-

den speziellen Eigenschaften (siehe Bild 3):

- **Bildtyp**: **Eingebettet** oder **Verknüpft** (ab Access 2010 zusätzlich **Freigegeben**)
- **Bild**: Dateiname oder ab Access 2010 Name des Bildes aus der Tabelle **MSysResources**
- **Bild nebeneinander**: Gibt an, ob das Bild mehrere Male nebeneinander dargestellt werden soll.
- **Größenanpassung**: Gibt an, wie das Bild angezeigt werden soll (**Abschneiden**, **Dehnen**, **Zoomen**). **Abschneiden** zeigt das Bild in Originalgröße, bei Bildern, die größer als das Bild-Steuerelement sind, aber nur einen Teil. **Dehnen** passt das Bild genau in das Bildsteuerelement ein, was zur Änderung der Proportionen führen kann. **Zoomen** passt das Bild so in das Bild-Steuerelement ein, dass es komplett sichtbar ist. Das Problem hierbei ist, dass man die Einstellungen nicht individuell festlegen kann, ohne etwa per VBA die Bildgröße zu ermitteln und die richtige Einstellung zu wählen.
- **Bildausrichtung**: Legt die Ausrichtung des Bildes im Steuerelement fest (**Links oben**, **Rechts oben**, **Mitte**, **Links unten**, **Rechts unten**)

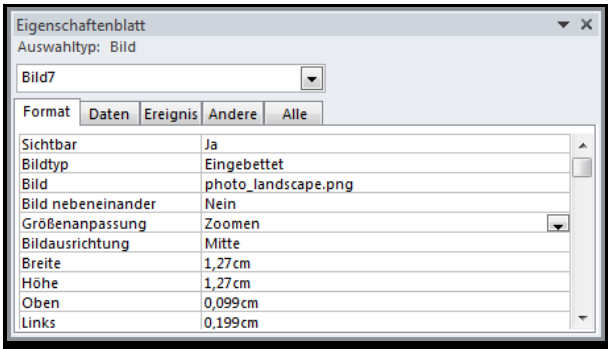


Bild 3: Eigenschaften des Bild-Steuerelements

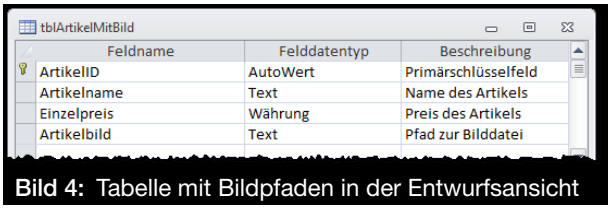


Bild 4: Tabelle mit Bildpfaden in der Entwurfsansicht

Ändern der Bild-Steuerelementgröße

Wenn Sie ein Bild statisch zu einem Bild-Steuerelement hinzugefügt haben und dann die Größe des Bild-Steuerelements anpassen, kann es sein, dass die enthaltenen Bilder unscharf werden. Sollten Sie also wie im obigen Beispiel etwa ein Icon für ein Formular verwenden wollen, sollten Sie seine Größe nach dem Hinzufügen nicht mehr ändern.

Bilder nach Pfad

Das Bild-Steuerelement bietet ab Access 2007 die Eigenschaft **Steuerelementinhalt** (unter VBA: **ControlSource**) an, mit der Sie das Steuerelement an ein Feld der Datenherkunft des Formulars binden können. In der Beispieldatenbank finden Sie eine Tabelle namens **tblArtikelMitBild**, deren Entwurf wie in Bild 4 aussieht.

Mit einige Datensätzen erhalten Sie eine Tabelle wie in Bild 5. Nun binden wir ein Formular an diese Tabelle und fügen dem Formular alle Felder der Tabelle hinzu. Außerdem legen Sie im Formular ein Bild-Steuerelement namens **ctlPicture** an.

Für die Eigenschaft **Steuerelementinhalt** des Bild-Steuerelements stellen Sie den Namen des Feldes **Bildpfad** ein. Das Ergebnis sieht wie in Bild 6 aus.

Ein Wechsel in die Formularansicht liefert nun ein mit einem Bild aus dem Dateisystem gefülltes Bild-Steuerelement (siehe Bild 7).

Fehlt noch eine Schaltfläche, mit der Sie die Pfade zu den Bilddateien auf einfache Weise hinzufügen können. Kein Problem: Fügen Sie neben dem

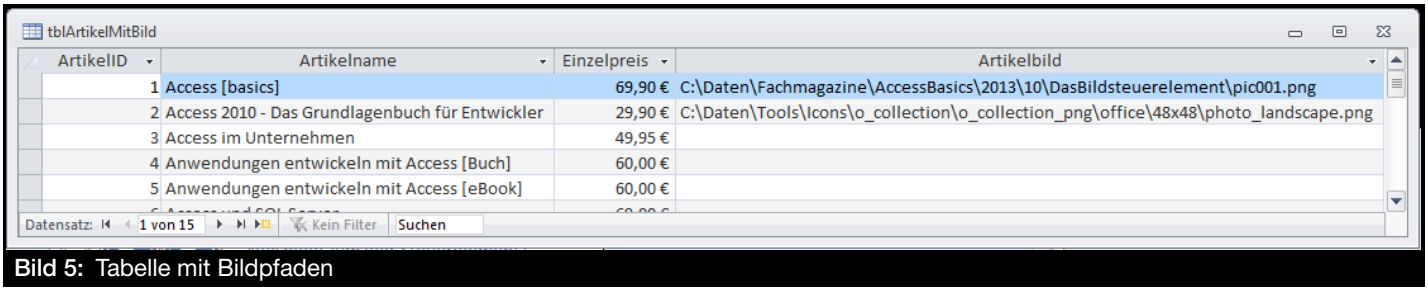


Bild 5: Tabelle mit Bildpfaden

Textfeld **Artikelbild** eine Schaltfläche namens **cmdBildAuswaehlen** hinzu, welche die Prozedur aus Listing 1 auslöst. Die dort verwendete Funktion **OpenFileName** finden Sie im Modul **mdlTools**. Sie zeigt einen Dialog zur Auswahl einer Bilddatei an.

Bild nach Steuerelementinhalt

Neben dem Zuweisen eines Feldnamen der Datenherkunft eines Formulars können Sie die Eigenschaft **Steuerelementinhalt** auch direkt mit dem Pfad zu einer Bilddatei füllen.

Bilder unter Access 2010/2013 hinzufügen

Unter Access 2010 und 2013 bietet Microsoft noch eine weitere Möglichkeit an, um Bilder zu einem Formular oder Bericht hinzuzufügen – und auch zu Schaltflächen. Im Gegensatz zu den vorherigen Beispielen benötigen Sie dazu eine Datenbank, die im Format von **Access 2007**, **2010** oder **2013** gespeichert ist.

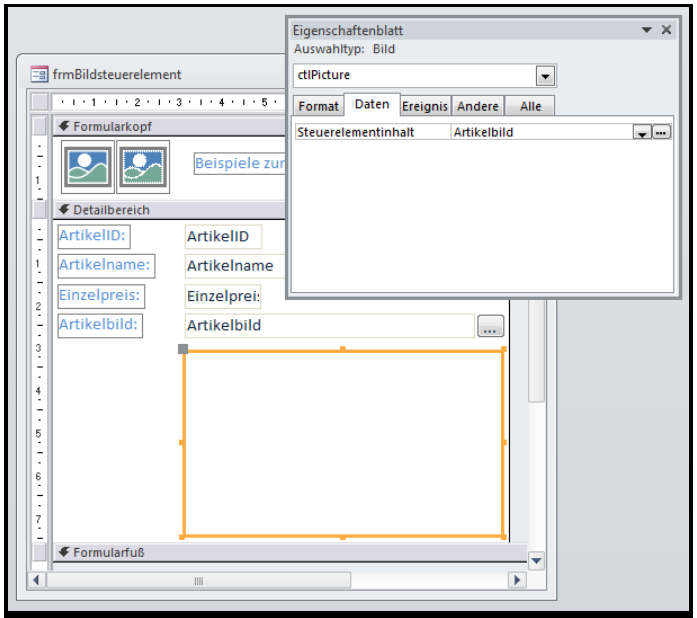


Bild 6: Formular mit gebundenem Bild-Steuerelement

Der Hintergrund ist, dass Sie die zu verwendende Datei in einer speziellen Tabelle namens **MSysResources** speichern. Bei dem Feld, dass die Bilddateien aufnimmt, handelt es sich um ein Anlage-Feld, das in Datenbankdateien mit der Endung **.mdb** et cetera nicht verwendet werden kann.

Bild 7: Formular mit Bild aus dem Dateisystem

Deshalb finden Sie im Download zu diesem Artikel eine weitere Beispieldatenbank mit der Endung **.accdb**.

Ist unter Access 2010 oder 2013 eine **.accdb**-Datenbank geladen, können Sie einem Formular oder Bericht in der Entwurfsansicht mit dem Ribbon-Eintrag **Bild einfügen** ein Bild hinzufügen (siehe Bild 8). Klicken Sie in einer Datenbank ohne Bilder auf die Schaltfläche, klappt ein Eintrag namens **Durchsuchen...** auf, mit dem Sie einen Dialog zum Hinzufügen der Bilddatei öffnen.

Wenn Sie bereits eines oder mehrere Bilder zur Datenbank hinzugefügt haben, erscheinende diese in einer Liste unter dem Eintrag **Durchsuchen...** und können direkt ausgewählt werden (siehe Bild 9).

Der Ribbon-Befehl **Bild einfügen** wird auch aktiviert, wenn Sie im Formular-Entwurf den Fokus auf eine Schaltfläche verschieben.

Wenn Sie dann ein Bild hinzufügen, wird dieses gleich als Bild der Schaltfläche hinzugefügt – das gilt natürlich auch für ein vorhandenes Bild-Steuerelement.

```
Private Sub cmdBildAuswaehlen_Click()
    Me!Artikelbild = OpenFileName(CurrentProject.Path, "Bild auswählen", "Bilddateien (*.jpg;*.bmp;*.png;*.gif;*.tif)")
End Sub
```
Listing 1: Aufruf eines Dialogs zum Auswählen des Bildpfades.

Bilder in Access, Teil III: Bilder aus OLE-Feldern anzeigen

Für alle, die Bilder flexibel in allen Access-Versionen einbinden und verwenden möchten, bietet sich das OLE-FEld an. Dabei füllen Sie die Daten nicht so hinein, wie es der erste Teil dieser Artikelreihe beschrieben hat, sondern als Byte-Array. Das heißt, dass Sie die komplette Datei, so wie sie auch auf der Festplatte liegt, in das OLE-Feld schreiben. Von dort aus können Sie die Datei beispielsweise wieder auf die Festplatte zurückschreiben oder diese als Bild an verschiedenen Stellen anzeigen – beispielsweise als Icons im TreeView- oder ListView-Steuerelement.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1311_Bilder.mdb**.

Bilder anzeigen

Grundsätzlich gibt es zwei verschiedene Arten, Bilder zu speichern: in einem OLE-Feld oder in einem Anlage-Feld. Dabei ist letztere die moderne Variante, die mit Access 2007 eingeführt wurde. Die Anzeige der gespeicherten Bilder erfolgt meist als Icon für Schaltflächen, Ribbons, **ListView**- und **TreeView**-Steuerelemente oder in Kontextmenüs. Bei den Schaltflächen gibt es ebenfalls seit Access 2007 Neuerungen, wobei sich mit Access 2010 weitere Änderungen ergeben haben. In dieser Artikelreihe beschreiben wir das Beziehen von Bildern sowohl aus OLE-Feldern als auch aus Anlage-Feldern.

Die Techniken für die Anzeige im Ribbon, in **TreeView**- und **ListView**-Steuerelementen sowie in Kontextmenüs sind über die Versionen gleich geblieben, bei den Schaltflächen oder einfachen Bildern von Formularen haben sich jedoch Änderungen ergeben. Wir beziehen uns in diesem Fall nur auf die aktuelle Version, die seit Access 2010 Verwendung findet.

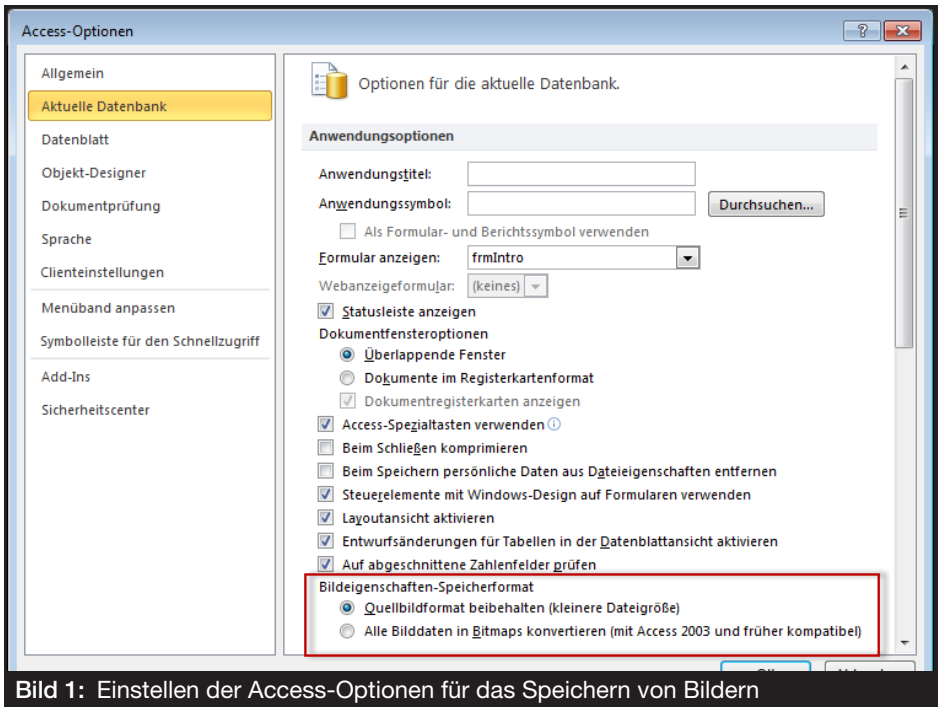


Bild 1: Einstellen der Access-Optionen für das Speichern von Bildern

Bilder aus dem OLE-Feld anzeigen

Zunächst schauen wir uns an, wie Sie Bilder aus einem OLE-Feld in verschiedenen Objekten anzeigen. Später betrachten wir dann die Anzeige von Bildern aus einem Anlage-Feld und speziell solche aus einer Tabelle namens **MSysResources** – einer Systemtabelle, die seit Access 2010 automatisch eingerichtet wird.

Bildformat einstellen

Nochmal zur Erinnerung: In neueren Access-Versionen (ab Access 2007) legen Sie fest, dass Dateien nicht in Bitmaps konvertiert werden sollen, sondern im Quellbildformat zu speichern sind. Dies erspart an manchen Stellen Ärger (siehe Bild 1).

Vorbereitung

Um die nachfolgenden Beispiele in Ihre eigenen Anwendungen zu integrieren, benötigen Sie drei Module, die Sie in der Beispieldatenbank zu diesem Artikel finden. Diese heißen:

- mdlOGL0713

Das ImageList-Steuerelement

Die beiden Steuerelemente **TreeView** und **ListView** sind praktische Helfer, wenn es um die Darstellung von Daten in hierarchischer oder Listenform geht. Beide können sogar Icons anzeigen. Diese beziehen sie ausschließlich aus dem **ImageList**-Steuerelement, das vorher mit diesen Icons gefüllt werden muss. Wie Sie dies erledigen, zeigt der vorliegende Artikel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in den Datenbanken **1311_ImageList.mdb** und **1311_ImageList.accdb** (für die Beispiele mit Anlage-Feld).

ImageList-Steuerelement hinzufügen

Das **ImageList**-Steuerelement finden Sie wie die übrigen Steuerelemente der Klasse **MSCOMCTL.ocx** in dem Dialog, den Sie mit einem Klick auf den Eintrag ActiveX-Steuerelemente öffnen (siehe Bild 1).

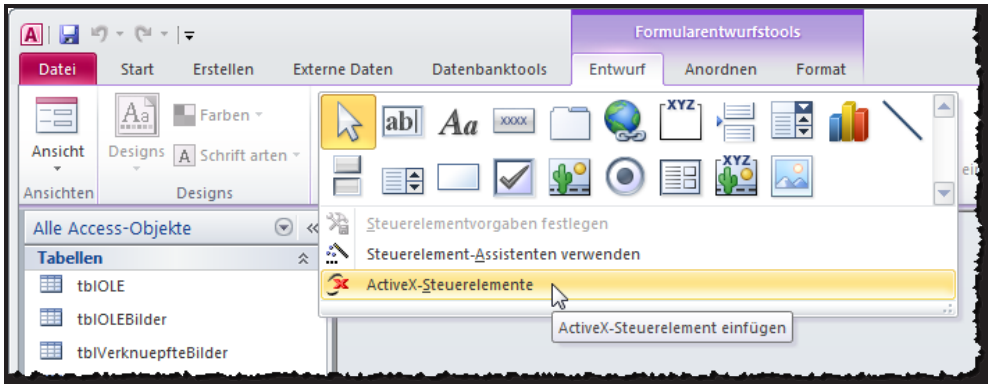


Bild 1: Öffnen des Dialogs zum Hinzufügen von ActiveX-Steuerelementen

Damit öffnen Sie einen Dialog, der alle verfügbaren ActiveX-Steuerelemente anzeigt. Hier wählen Sie den Eintrag **Microsoft ImageList Control, version 6.0** aus (siehe Bild 2).

Nach dem Hinzufügen entweder per Doppelklick oder durch Markieren und anschließendes Betätigen der OK-Schaltfläche erscheint das Steuerelement recht unscheinbar in der linken oberen Ecke des Zielformulars (siehe Bild 3).

Noch interessanter wird es, wenn Sie nun in die Formularansicht wechseln – das Steuerelement ist dann nämlich gar nicht mehr zu sehen. Das ist auch kein Wunder, denn es soll schlicht und einfach als Container

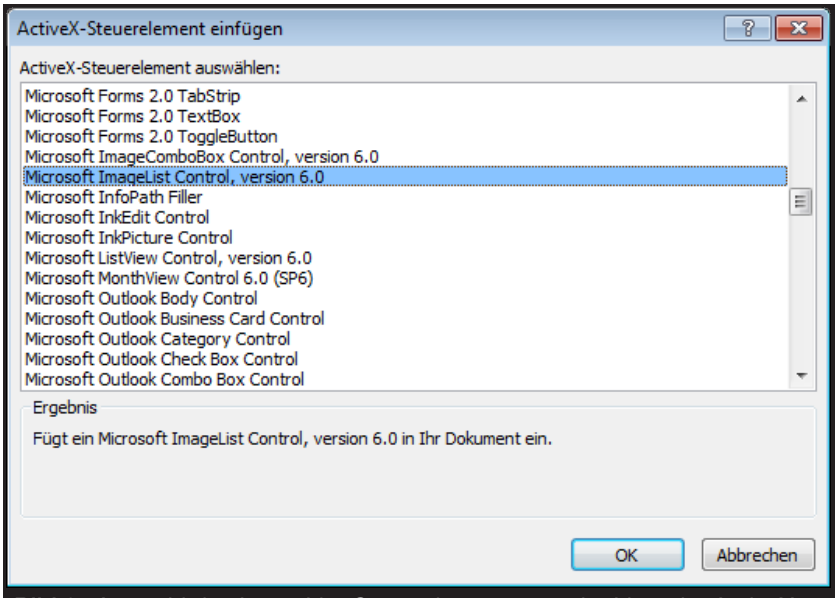


Bild 2: Auswahl des ImageList-Steuerelements aus der Liste der ActiveX-Steuerelemente

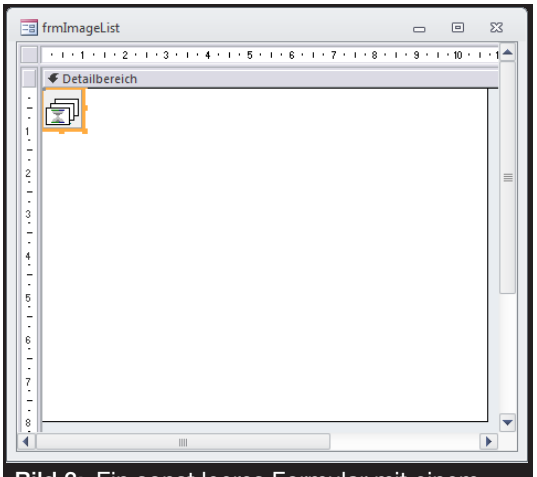


Bild 3: Ein sonst leeres Formular mit einem ImageList-Steuerelement

für die in **TreeView**- und **ListView**-Steuerelementen anzuzeigenden Icons dienen.

Bevor wir uns das Steuerelement genauer ansehen, geben wir ihm noch einen Namen – in diesem Fall **ctlImageList**.

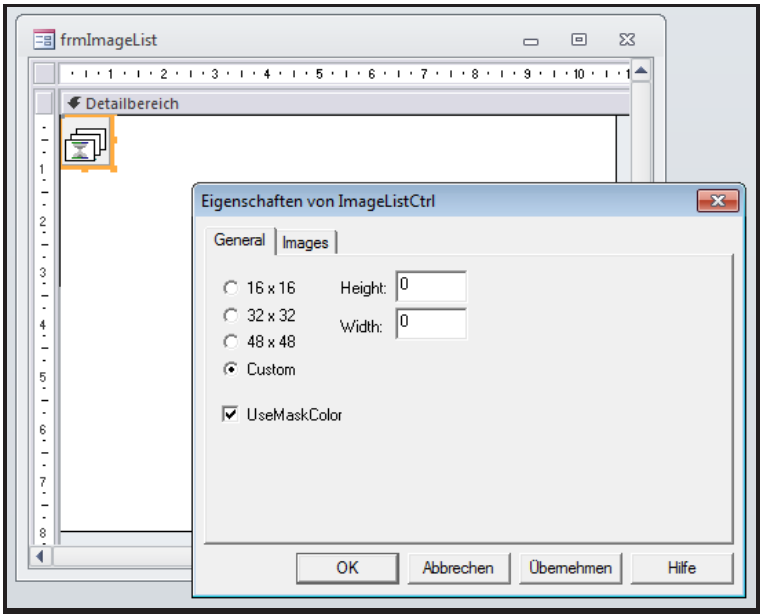


Bild 4: Eigenschaftsfenster des **ImageList**-Steuerelements

Eigenschaften des ImageList-Steuerelements

Wenn Sie in der Entwurfsansicht des Formulars mit der rechten Maustaste auf das **ImageList**-Steuerelement klicken, können Sie den Kontextmenü-Eintrag **ImageListCtl-Object|Properties** auswählen. Dies öffnet den Dialog aus Bild 4. Sie können aber auch einfach doppelt auf das Steuerelement klicken!

Der **Eigenschaften**-Dialog hat zwei Registerseiten. **General** enthält allgemeine Einstellungen wie etwa die Größe der gespeicherten Icons, **Images** ist für die Verwaltung der eigentlichen Bilder verantwortlich.

Auf der Seite **General** stellen sie zunächst die Größe ein, in der die Icons im **ListView**- oder **TreeView**-Steuerelement angezeigt werden sollen. Diese Einstellung können Sie nur vornehmen, wenn das **ImageList**-Steuerelement noch keine **Icons** enthält. Üblicherweise zeigt etwa ein **TreeView**-Steuerelement seine Icons in der Größe 16 x 16 Pixel an. Sie können diese aber nach Wunsch auch in der Größe 32 x 32 oder 48 x 48 Pixel anzeigen – oder eine ganz andere Größe verwenden. Dazu wählen Sie die Option **Custom** und geben die Höhe und die Breite für die beiden Eigenschaften **Height** und **Width** ein.

Wichtig ist an dieser Stelle der Hinweis, dass die Icons immer an die gewählte Größe angepasst werden.

ImageList-Steuerelement manuell füllen

Wenn Sie dem **ImageList**-Steuerelement Bilder hinzufügen möchten, wechseln Sie zur Registerseite **Images**. Dort finden Sie als wichtigstes Element eine Schaltfläche namens **Insert Picture** vor (siehe Bild 5).

Ein Klick auf diese Schaltfläche öffnet einen Dialog zum Auswählen eines oder mehrerer Icons. Das Problem bei der Sache ist: Abhängig von den Eigenschaften der Bilddatei, die Sie hinzufügen möchten, kommt es gelegentlich zur Fehlermeldung **Invalid Picture**.

Sollten Sie über Bilddateien verfügen, die sich problemlos hinzufügen lassen, können Sie zusätzlich noch die beiden Eigenschaften **Key** und **Tag** für ein Bild einstellen.

Wichtig ist die **Key**-Eigenschaft: Sie muss einen eindeutigen Wert enthaltenen und identifiziert etwa beim Hinzufügen eines im **ImageList**-Steuerelement gespeicherten Icons in ein **TreeView**-Steuerelement das gewünschte Bild.

Sie können allerdings auch den numerischen Index des hinzuzufügenden Bilds angeben.

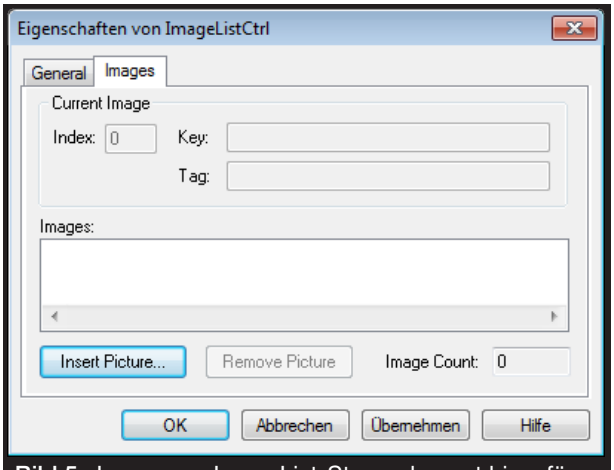


Bild 5: Icons zum ImageList-Steuerelement hinzufügen

ImageList-Steuerelement per VBA füllen

Wenn Sie über Icons verfügen, die in einem Format vorliegen, dass sich nicht manuell in das **ImageList**-Steuerelement importieren lässt, können Sie dies per VBA erledigen. Hier gibt es zahlreiche Varianten – Sie können die Bilder direkt aus einem Verzeichnis importieren, aus dem OLE-Feld einer Tabelle oder auch aus einem Anlage-Feld (siehe hierzu auch die Artikel **Bilder in Access, Teil II: Bilder in OLE-Feldern** und **Anlage-Felder und VBA**).

Bilder aus dem Dateisystem importieren

Im ersten Beispiel gehen wir davon aus, dass sich die Icons, die Sie in einem Steuerelement wie etwa einem **TreeView** anzeigen möchten, in einem Ordner im Dateisystem befinden, in diesem Fall im gleichen Verzeichnis wie die Datenbank-Datei.

Ziel ist es, die vier Icons wie in Bild 6 in einem **TreeView**-Steuerelement abzubilden. Das Beispiel finden Sie im Formular **frmImageListVBA_Dateisystem**.

Wir schauen uns zunächst den Vorgang für ein einzelnes Icon an. Dazu deklarieren Sie zunächst zwei Objektvariablen, um das **ImageList**-Steuerelement und das **TreeView**-Steuerelement zu referenzieren:

```
Dim objTreeView As MSComctlLib.TreeView
Dim objImageList As MSComctlLib.ImageList
```

Dann legen Sie eine Ereignisprozedur an, die durch das Ereignis **Beim Laden** des Formulars ausgelöst wird. Dieses deklariert zwei weitere Variablen, nämlich **objPicture**, um zwischenzeitlich einen Verweis auf das in aus der Datei in den Speicher geladene Icon zu referenzieren und **strPicture**, um den Namen der Bilddatei zu speichern:

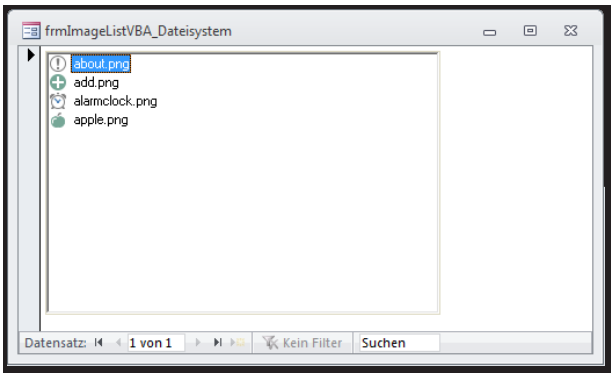


Bild 6: **TreeView**-Steuerelement mit einigen Beispielicons aus dem Dateisystem

```
Private Sub Form_Load()
    Dim objPicture As stdole.StdPicture
    Dim strPicture As String
```

Im ersten Schritt setzt die Prozedur dann den Verweis auf das **ImageList**-Steuerelement:

```
    Set objImageList = _
            Me!ctlImageList.Object
```

Dann speichert die Prozedur den Namen der hinzuzufügenden Bilddatei, hier **add.png**, in der Variablen **strPicture** (wir benötigen diesen Namen später mehrmals):

```
    strPicture = "add.png"
```

Nun liest die Prozedur die Bilddatei in die Variable **objPicture** ein. Dazu nutzt sie eine Funktion aus dem Modul **mdlOGL0713**, das einige für die Verwendung von Bildern wichtige Funktionen bereithält.

Dieses Modul müssen Sie, wenn Sie die Funktion in eigenen Datenbanken verwenden möchten, zum VBA-Projekt der jeweiligen Datenbank hinzufügen. Die Funktion erwartet den Pfad der einzulesenden Datei als Parameter:

```
    Set objPicture = _
        mdlOGL0713.LoadPictureGDIP(_
            CurrentProject.Path _
            & "\" & strPicture)
```

Damit haben Sie nun eine Variable **objPicture**, die mit einem Verweis auf ein im Speicher liegendes Objekt des Typs **StdPicture** gefüllt ist – unserem Bild. Dieses Bild können wir nun zur Liste der Bilder im **ImageList**-Steuerelement hinzufügen. Dazu verwenden wir die **Add**-Methode der **ListImages**-Auflistung des **ImageList**-Steuerelements. Dieser offeriert drei Parameter:

- **Index**: Gibt die Position an, an der das Bild eingefügt werden soll. Darf nicht größer sein als die Anzahl der bereits vorhandenen Bilder.
- **Key**: Gibt einen eindeutigen Schlüssel für das Bild an.
- **Picture**: Erwartet einen Verweis auf das **StdPicture**-Objekt.

In diesem Fall verwenden wir nur die letzten beiden Parameter und übergeben den namen der Bilddatei (**add.png**) sowie die Objektvariable

Kontextmenüs im TreeView, Teil I: Anlegen und Löschen

Das **TreeView**-Steuerelement eignet sich perfekt zur Anzeige hierarchischer Daten. Allerdings hat es einen Nachteil: Im Gegensatz zu gebundenen Formularen kann man darin nicht mal eben einen neuen Datensatz anlegen, einen Datensatz löschen oder bearbeiten. Dazu muss man Hand in Hand mit einem weiteren Element wie etwa einem Unterformular zur Anzeige des aktuellen Datensatzes arbeiten. Aber wie legt man mal eben einen neuen Datensatz von einem **TreeView**-Steuerelement aus an oder löscht ein Element? Dies gelingt am einfachsten per Kontextmenü. Dieser Artikel liefert das nötige Know-how und einige Beispiele.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1311_TreeView.mdb**.

Beispielformular

Als Beispielformular verwenden wir das im Artikel **TreeView und Unterformulare** aufgebaute Formular, das wir kopiert und **frmTreeViewKontextmenue** genannt haben.

Damit wir über das **TreeView**-Steueelement leicht neue Kunden hinzufügen können, haben wir es etwas umgestaltet: Statt wie zuvor alle Kunden in der ersten Ebene des **TreeView**-Steuerelements anzuzeigen, haben wir einen Root-Knoten namens **Kunden** angelegt und alle Kunden-Elemente unterhalb dieses Knotens hinzugefügt. Dazu waren die in Listing 1 dargestellten Änderungen nötig. Dort fügen wir zunächst mit der **Add**-Methode das Root-Element mit dem **Key**-Wert **r0** und dem Text **Kunden** hinzu. Dieses wird mit **Expanded = True** aufgeklappt. Die beim Durchlaufen der Tabelle **tblKunden** hinzugefügten Elemente legt die Prozedur dann unterhalb des Elements mit dem **Key**-Wert **r0** an. Das Ergebnis sieht schließlich wie in Bild 1 aus.

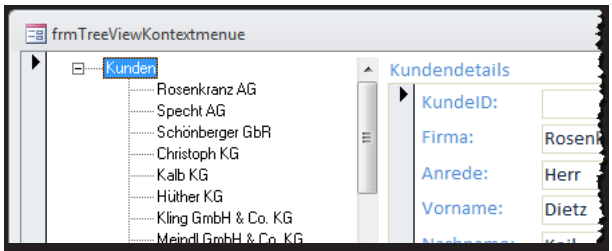


Bild 1: Hinzufügen eines Root-Knotens zum **TreeView**-Steuerelement

Element löschen

Als Erstes wollen wir einen Kontextmenü-Eintrag hinzufügen, mit dem der Benutzer eines der Kunden-Elemente löschen kann. Das Kontextmenü soll beim Anklicken des zu löschenden Elements angezeigt werden und einen Eintrag namens **Löschen** anbieten. Um das Herunterdrücken der rechtem Maustaste abzufangen, verwenden wir die Ereignisprozedur, die durch das Ereignis **MouseDown** ausgelöst wird. Für ActiveX-Steuerelemente legen Sie eine solche Ereignisprozedur nicht über die entsprechende Eigenschaft im Eigenschaftsfenster an, sondern direkt im VBA-Fenster.

Für das **TreeView**-Steuerelement liegt bereits eine Objektvariable namens **m_TreeView** vor, die mit dem

```
Private Sub TreeViewFuellen()
    ...
    Dim objNode As MSComctlLib.Node
    ...
    Set objNode = objTreeView.Nodes.Add(, , "r0", "Kunden")
    objNode.Expanded = True
    Do While Not rstKunden.EOF
        objTreeView.Nodes.Add "r0", tvwChild, "k" & rstKunden!KundeID, rstKunden!Firma
        ...
    Loop
End Sub
```

Listing 1: Änderungen am Beispielformular, damit dieses einen Kunden-Knoten in der ersten Ebene anzeigt

Schlüsselwort **WithEvents** deklariert wurde und somit auch Ereignisprozeduren ausführt.

Öffnen Sie also das Klassenmodul **Form_frmTreeviewKontextmenue** und wählen Sie dann erst im linken Kombinationsfeld den Eintrag **m_TreeView** und im rechten den Eintrag **MouseDown** aus (siehe Bild 2).

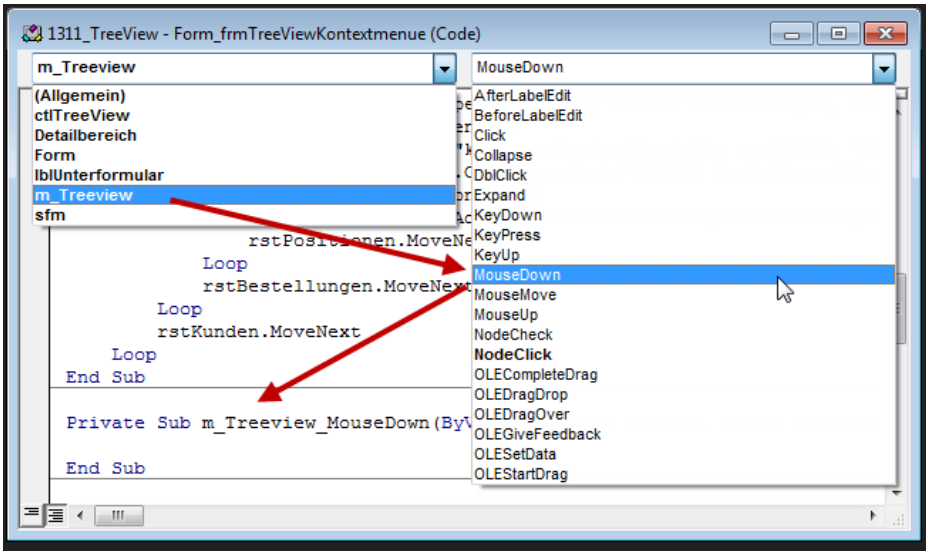


Bild 2: Anlegen der Ereignisprozedur MouseDown

Die auf diese Weise erstellte Ereignisprozedur **m_TreeView_MouseDown** liefert mit dem Parameter **Button** einen Wert, der über die betätigte Maustaste Auskunft gibt. Diesen können Sie über die beiden Konstanten **acLeftButton** oder **acRightButton** prüfen. In diesem Fall interessiert uns, ob der Benutzer die rechte Maustaste gedrückt hat. Wir können die Ereignisprozedur also mit einer Bedingung wie der folgenden anreichern:

```
Select Case Button
    Case acRightButton
        'Kontextmenü anzeigen
End Select
```

Fehlt nur noch der Code, der ermittelt, welches Element der Benutzer angeklickt hat und der das passende Kontextmenü zusammenstellt. Hat der Benutzer tatsächlich mit der rechten Maustaste auf das **Treeview**-Steuerelement geklickt, können verschiedene Fälle eintreten:

- Er hat mit einer anderen Maustaste als der rechten geklickt.
- Er hat mit der rechten Maustaste ein Element im **TreeView**-Steuerelement angeklickt.
- Er hat mit der rechten Maustaste auf einen leeren Bereich im **TreeView**-Steuerelement geklickt.

Wenn der Benutzer kein Element getroffen hat, soll nichts geschehen. Aber wie prüfen wir das? Indem wir die Funktion **HitTest** des **TreeView**-Steuerelements nutzen. Diese erwartet die Koordinaten für den Punkt, den der Benutzer angeklickt hat. Diese liefert die Ereignisprozedur **MouseDown** ebenfalls, und zwar mit den Parametern **x** und **y**.

Da wir Informationen von dem angeklickten **Node**-Element benötigen, in diesem Fall den Wert der **Key**-Eigenschaft, speichern wir den Verweis auf das **Node**-Element in einer Variablen zwischen.

Diese deklarieren wir vorher wie folgt:

```
Dim objNode As MSComctlLib.Node
```

Dem **Key**-Wert entnehmen wir zwei Informationen: den als erstes Zeichen angegebenen Buchstaben, der den Typ des Elements liefert (zum Beispiel **r** für Root, **k** für Kunde, **b** für Bestellung oder **p** für Bestellposition) und die folgende Zahlenkombination, die dem Primärschlüsselwert des Datensatzes in der passenden Tabelle entspricht.

Diese Informationen sollen ebenfalls gespeichert werden, und zwar in den folgenden Variablen:

```
Dim strTyp As String
Dim lngID As Long
```

Die folgenden Anweisungen ermitteln zunächst das angeklickte Element:

```
Set objNode = m_Treeview.HitTest(x, y)
```

Da **HitTest** den Wert **Nothing** übergibt, wenn kein Element angeklickt wurde, sondern ein leerer Bereich im **TreeView**-Steuerelement, prüfen wir vor den weiteren Schritten, ob **objNode** überhaupt mit einem Verweis auf ein angeklicktes Element gefüllt wurde und lesen dann die gewünschten Informationen aus:

```
Private Sub m_Treeview_MouseDown(ByVal Button As Integer, ByVal Shift As Integer, ByVal x As stdole.OLE_XPOS_PIXELS, _
        ByVal y As stdole.OLE_YPOS_PIXELS)
    Dim strTyp As String
    Dim lngID As Long
    Dim objNode As MSComctlLib.Node
    Select Case Button
        Case acRightButton
            Set objNode = m_Treeview.HitTest(x, y)
            If Not objNode Is Nothing Then
                strTyp = Left(objNode.Key, 1)
                lngID = Mid(objNode.Key, 2)
                Select Case strTyp
                    Case "k"
                        KontextmenueKundeAnzeigen(lngID)
                    Case "r"
                        KontextmenueRootAnzeigen
                    Case "b"
                        KontextmenueBestellungAnzeigen(lngID)
                    Case "p"
                        KontextmenuePositionAnzeigen(lngID)
                End Select
            End If
    End Select
End Sub
```

Listing 2: Aufrufen der Prozeduren zum Anzeigen der verschiedenen Kontextmenüs

```
If Not objNode Is Nothing Then
    strTyp = Left(objNode.Key, 1)
    lngID = Mid(objNode.Key, 2)
End If
```

Damit ausgestattet können wir eine **Select Case**-Bedingung anlegen, die den Wert der Variablen **strTyp** prüft. In unserem Beispiel enthält **strTyp** einen der Werte **r**, **k**, **b** oder **p**. Um die Übersichtlichkeit zu erhalten, rufen wir für jeden Wert eine eigene Prozedur zum Anzeigen des Kontextmenüs auf.

Diese heißen **KontextmenueRootAnzeigen**, **KontextmenueKundenAnzeigen** und so weiter. Alle Prozeduren außer das für das Root-Element nehmen den Primärschlüsselwert des angeklickten Elements als Parameter entgegen. Die vollständige Ereignisprozedur **m_TreeView_MouseDown** finden Sie in Listing 2.

```
Private Sub KontextmenueKundeAnzeigen(lngID As Long)
    Dim cbr As Office.CommandBar
    Dim cbc As Office.CommandBarButton
    On Error Resume Next
    CommandBars("TreeView_Kunde").Delete
    On Error GoTo 0
    Set cbr = CommandBars.Add("TreeView_Kunde", msoBarPopup, False, True)
    With cbr
        Set cbc = .Controls.Add(msoControlButton)
        With cbc
            .Caption = "Kunde löschen"
            .OnAction = "=KundeLoeschen(" & lngID & ")"
        End With
        .ShowPopup
    End With
End Sub
```

Listing 3: Anzeigen des Kontextmenüs für einen Kunden, hier zunächst mit der Mögilchkeit zum Löschen des Kunden

Wir wollen mit dem Kontextmenü-Eintrag zum Löschen des aktuell angeklickten Kunden beginnen. Die entsprechende Prozedur heißt **KontextmenueKundeAnzeigen** und sieht wie in Listing 3 aus. Die Prozedur deklariert eine **CommandBar**- und eine **CommandBarButton**-Variable.

Dann löscht sie ein eventuell bereits vorhandenes Kontextmenü namens **TreeView_Kunde**.

Da dies einen Fehler auslösen kann, deaktiviert die Prozedur die integrierte Fehlerbehandlung für die Durchführung dieser Anweisung.

Anschließend fügt sie mit der **Add**-Methode der **CommandBars**-Auflistung ein neues Kontextmenü namens **TreeView_Kunde** hinzu. Dabei gibt sie mit dem ersten Parameter den Namen und mit dem zweiten den Menütyp an. Der dritte Parameter legt fest, dass es sich nicht um eine Menüleiste handelt und der vierte sorgt dafür, dass das Kontextmenü nur temporär erstellt wird – damit wird es beim Schließen von Access gelöscht.

Für das mit **cbr** referenzierte Kontextmenü fügt die Prozedur mit der **Add**-Methode der **Controls**-Auflistung einen neuen Kontextmenü-Eintrag hinzu. Der Typ wird mit dem Wert **msoControlButton** festgelegt. Anschließend stellt die Prozedur noch die beiden Eigenschaften **Caption** (Beschriftung) und **OnAction** ein. Letztere erwartet den Aufruf der beim Auswählen des Kontextmenü-Eintrags auszuführenden VBA-Funktion. Dieser soll gleich den Primärschlüsselwert des angeklickten **TreeView**-Elements übergeben, daher wird der Aufruf aus **=KundeLoeschen**, einer öffnenden Klammer, dem Primärschlüsselwert und einer schließenden Klammer zusammengesetzt – also zum Beispiel als **=KundeLoeschen(123)**.

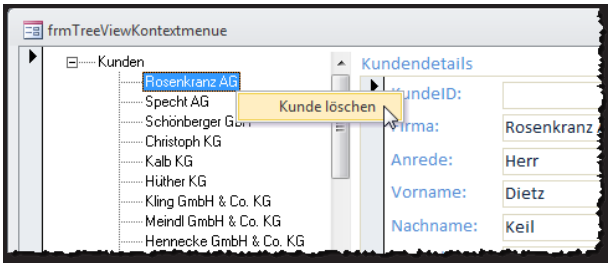


Bild 3: Anzeigen des Kontextmenüs für einen Kunden

Die **ShowPopup**-Methode des **CommandBar**-Objekts zeigt das soeben erstellte Kontextmenü schließlich an (siehe Bild 3).

Kunden löschen

Fehlt noch die durch das Betätigen des Kontextmenü-Eintrags ausgelöste Löschen des Kunden. Damit ist es allerdings nicht getan: Sie müssen ja nicht nur den Kundendatensatz aus der Tabelle **tblKunden** löschen, sondern auch noch weitere Aktionen einleiten:

- Entfernen des Eintrags aus dem **TreeView**-Steuerelement
- Setzen des Fokus auf einen anderen Eintrag im **TreeView**
- Aktualisieren des Unterformulars für den neu ausgewählten Eintrag

Dies alles erledigt die Prozedur aus Listing 4. Sie fragt erst mit einem Meldungsfenster ab, ob der Datensatz tatsächlich gelöscht werden soll. Falls ja, entfernt die Prozedur zunächst den Datensatz aus der Tabelle **tblKunden**, dessen Primärschlüsselfeld den Wert der Variablen **lngKundeID** enthält. Dann versucht die Prozedur nacheinander, den vorherigen oder folgenden Eintrag auf der gleichen Ebene des **TreeView**-Steuerelements mit der Variablen **objNode** zu referenzieren. Ist beides nicht möglich, referenziert sie das übergeordnete Element mit dem **Key**-Wert **r0** und leert das Unterformular-Steuerelement.

Wenn **objNode** jedoch gefüllt werden konnte, ruft die Prozedur die Ereignisprozedur **m_TreeView_NodeClick** auf und übergibt den Verweis auf

```
Public Function KundeLoeschen(lngID As Long)
    Dim db As DAO.Database
    Dim objNode As MSComctlLib.Node
    If MsgBox("Soll der Kunde mit der Kundennummer " & lngID _
            & " wirklich gelöscht werden?", vbYesNo + vbExclamation, _
            "Kunde löschen") = vbYes Then
        Set db = CurrentDb
        db.Execute "DELETE FROM tblKunden WHERE KundeID = " & lngID
        Set objNode = m_Treeview.Nodes("k" & lngID).Previous
        If objNode Is Nothing Then
            Set objNode = m_Treeview.Nodes("k" & lngID).Next
        End If
        If objNode Is Nothing Then
            Set objNode = m_Treeview.Nodes("r0")
            Me!sfm.SourceObject = ""
        Else
            m_Treeview_NodeClick objNode
            m_Treeview.SelectedItem = objNode
        End If
        m_Treeview.Nodes.Remove "k" & lngID
    End If
End Function
```
Listing 4: Löschen eines Kunden per Kontextmenü

das zu markierende **Node**-Element. Die Prozedur übernimmt alle notwendigen Schritte: Sie stellt das im Unterformular-Steuerelement **sfm** anzuzeigende Unterformular ein und filtert den gewünschten Datensatz.

Neuen Kunden anlegen

Nun fügen wir dem Root-Element des **TreeView**-Steuerelements einen Kontextmenü-Eintrag hinzu, der das Anlegen eines neuen Kunden ermöglicht. Auch dies hört sich leicht an – man braucht ja nur einen neuen Datensatz im Kunden-Unterformular einzutragen! In der Tat sollten aber eigentlich gleich zwei Schritte erfolgen: Der Kunde soll in einem leeren, neuen Datensatz im Unterformular angelegt werden können. Gleichzeitig muss aber auch ein neuer Eintrag im **TreeView**-Steuerelement her. Aber welchen Text soll dieser anzeigen, wenn der Benutzer noch gar keinen Kundennamen im Unterformular eingetragen hat? Wir gehen also am besten einen kleinen Kompromiss ein: Wir zeigen erst das Unterformular zum Eingeben eines neuen Kunden an und fügen den neuen Kunden erst nach dem Speichern zum **TreeView**-Steuerelement hinzu.

Dazu müssen wir den Zeitpunkt abfangen, zu dem der Benutzer den neuen Kunden speichert. Eine solche Ereignisprozedur würde man normalerweise im Klassenmodul des Unterformulars unterbringen, und zwar für das Ereignis **Nach Aktualisierung**. Dazu müsste das Unterformular aber auf das **TreeView**-Steuerelement im Hauptformular zugreifen. Eine solche Abhängigkeit ist nicht gut, denn das Unterformular sollte im besten Fall

```
Public Function NeuerKunde()
    Me!sfm.SourceObject = "sfmKunden"
    Me!sfm.Form.Recordset.AddNew
    Set objsfmKunde = Me!sfm.Form
    With objsfmKunde
        .AfterUpdate = "[Event Procedure]"
    End With
End Function
```

Listing 5: Hinzufügen eines neuen Kunden

```
Private Sub objsfmKunde_AfterUpdate()
    Dim objNode As MSComctlLib.Node
    On Error Resume Next
    Set objNode = m_Treeview.Nodes("k" & objsfmKunde!KundeID)
    On Error GoTo 0
    If objNode Is Nothing Then
        Set objNode = m_Treeview.Nodes.Add("r0", tvwChild, "k" _
            & objsfmKunde!KundeID, objsfmKunde!Firma)
    Else
        objNode.Text = objsfmKunde!Firma
    End If
    objNode.EnsureVisible
    Me!ctlTreeView.SetFocus
    Set m_Treeview.SelectedItem = objNode
End Sub
```

Listing 6: Hinzufügen des neuen Kunden zum TreeView-Steuerelement

überhaupt nichts vom Hauptformular wissen. Deshalb legen wir diese Ereignisprozedur im Hauptformular an. Dazu benötigen wir eine **Objektvariable**, die das Unterformular referenziert und mit dem Schlüsselwort **WithEvents** deklariert wurde. Die folgende Deklarationszeile kommt dementsprechend in das Klassenmodul **Form_frmTreeViewKontextmenue**:

```
Dim WithEvents objsfmKunde As Form
```

Im Moment des Aufrufs des Kontextmenü-Eintrags **Neuer Kunde** soll das Unterformular **sfmKunden** angezeigt werden und einen neuen, leeren Kundendatensatz liefern. Dies erledigt die Funktion **NeuerKunde** (siehe Listing 5). Den neuen Datensatz fügt die Funktion mit der **AddNew**-Methode des **Recordset**-Objekts des Formulars hinzu. Dann referenziert die Funktion das Unterformular mit der soeben deklarierten Variablen **objsfmKunden**. Für dieses legt die Funktion außerdem fest, dass im aktuellen Klassenmodul das Ereignis **AfterUpdate** gelauscht werden soll.

Für dieses legen Sie nun noch die passende Ereignisprozedur an, die wie in Listing 6 aussieht. Sie versucht zunächst, ein **Node**-Objekt zu referenzieren, dass einen **Key**-Wert bestehend aus dem Buchstaben **k** und dem Primärschlüsselwert aus dem Unterformular aufweist. Dies könnte der Fall sein, wenn der Benutzer den neuen Datensatz direkt nach dem Anlegen und Speichern erneut ändert und speichert. Wird kein solches **Node**-Objekt gefunden, legt die Prozedur dieses neu an. Als Parent wird dabei das Root-Element mit dem

Das Kontrollkästchen

Immer, wenn es um die Darstellung von Werten im Format **Ja/Nein**, **Wahr/Falsch**, **True/False** oder **-1/0** geht, kommt das Kontrollkästchen zum Zug. Es ist etwas unscheinbar, aber wenn Boolean-Werte angezeigt werden sollen, ist es in seiner Kompaktheit unschlagbar und sehr praktisch. Dieser Artikel zeigt, wie Sie mit dem Kontrollkästchen arbeiten, wie Sie es an eine Datenherkunft binden und welche weiteren Eigenschaften es offenbart.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1312_Kontrollkaestchen.mdb**.

Mit und ohne Bindung

Ein Kontrollkästchen können Sie wie die übrigen Steuerelemente mit und ohne Bindung an ein Feld der Datenherkunft des Formulars benutzen. Für den ersten Fall bietet es sich beispielsweise zum Festlegen von Optionen vor der Ausführung von Aktionen wie etwa dem Drucken eines Dokuments an.

Bei gebundenen Formularen zeigt es die Werte aus **Ja/Nein**-Feldern an und ermöglicht es, die enthaltenen Werte mit einem einfachen Mausklick zu ändern. Dabei stellt das Kontrollkästchen mit aktiviertem Häkchen den Wert **-1** (**True**, **Wahr**, **Ja**) dar. Ohne Häkchen repräsentiert es den Wert **0** (**False**, **Falsch**, **Nein**).

Ungebundene Kontrollkästchen

Im ersten Beispiel fügen Sie zunächst ein einfaches Kontrollkästchen zu einem Formular namens **frmKontrollkaestchen** hinzu.

Dazu klicken Sie im Ribbon unter **Entwurf|Steuerelemente** auf das entsprechende Symbol und platzieren es dann an der gewünschten Stelle im Formular. Im Gegensatz zu

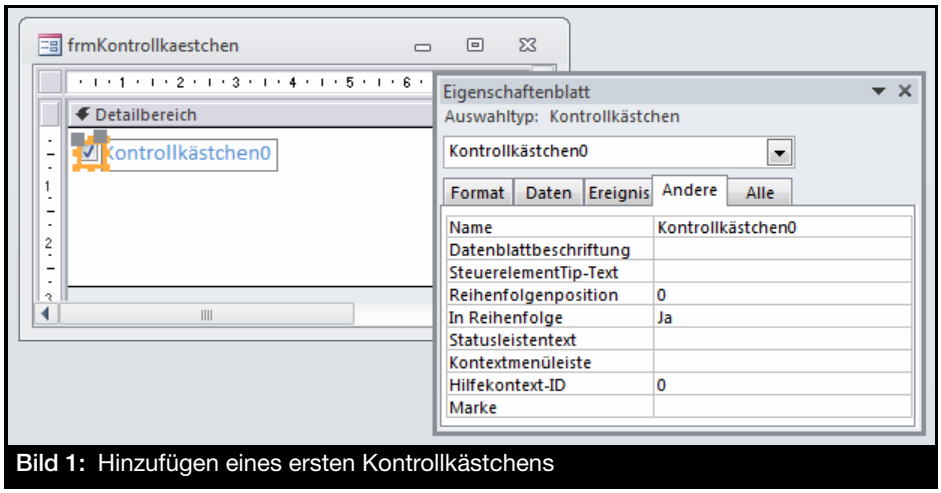


Bild 1: Hinzufügen eines ersten Kontrollkästchens

den meisten übrigen Steuerelementen platziert Access das Bezeichnungsfeld rechts neben dem eigentlichen Steuerelement und nicht links daneben.

Bild 2: Kontrollkästchen ohne Wert

Das ist manchmal praktisch, aber nicht immer – zum Beispiel dann nicht, wenn das Kontrollkästchen gebunden ist und zusammen mit den übrigen Feldern der Datenherkunft untereinander dargestellt werden soll. Wir vertauschen also zunächst die Position der beiden zusammengehörenden Steuerelemente.

Access stellt außerdem automatisch den Namen des Steuerelements sowie den Text des Bezeichnungsfeldes ein (siehe Bild 1). Diese beiden Werte passen wir als Erstes an: Das Kontrollkästchen soll den Namen **chkBeispiel1** und das Bezeichnungsfeld den Wert **Beispiel 1:** für die Eigenschaft **Beschriftung** erhalten.

Wenn Sie nun in den Entwurf des Formulars wechseln, finden Sie das Kontrollkästchen wie in Bild 2 vor.

Ups – das ist aber weder ein leeres noch ein angehaktes Kontrollkästchen! Wenn das Kontrollkästchen

den Wert **-1** (**True**) oder **0** (**False**) hätte, würde es wohl entweder einen Haken anzeigen oder leer sein. Welchen Wert mag es also nun enthalten? Dies können wir nur auf eine At herausfinden – per VBA.

Aktuellen Wert ermitteln

Wenn Sie den Wert eines Kontrollkästchens ermitteln möchten, müssen brauchen Sie zwei Informationen: den Namen des Formulars, in dem sich das Kontrollkästchen befindet sowie den Namen des Kontrollkästchens.

Dann öffnen Sie den Direktbereich im VBA-Editor mit **Strg + G** und geben dort den folgenden Ausdruck ein:

```
? Forms!frmKontrollkaestchen!
                        chkBeispiel1
```

Das Ergebnis lautet **Null**, wie auch der Screenshot aus Bild 3 zeigt.

Nun soll das Kontrollkästchen beim Anzeigen weder so aussehen, wie es aussieht, noch den Wert **Null** liefern, wenn wir seinen Wert abfragen. Dies können wir nur verhindern, wenn wir einen entsprechenden Standardwert für das Kontrollkästchen festlegen. Dies erledigen Sie auf der Registerseite **Daten** des Eigenschaftsfensters.

Bild 3: Das Kontrollkästchen enthält den Wert **Null**.

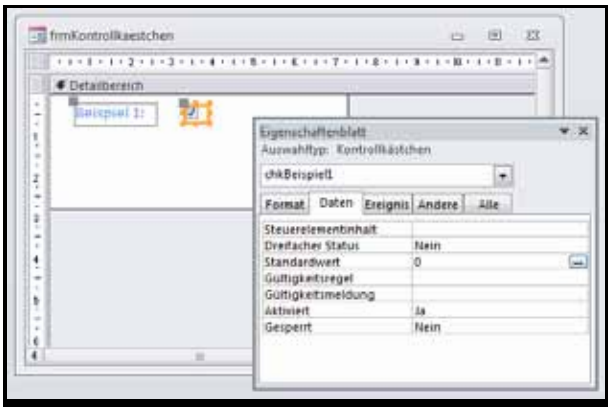


Bild 4: Einstellen des Standardwerts

Stellen Sie diese Eigenschaft auf den Wert **0** ein, wenn das Kontrollkästchen beim Anzeigen leer sein soll und auf **-1**, wenn es einen Haken enthalten soll (siehe Bild 4).

Der nächste Wechsel zur Formularansicht liefert dann auch das gewünschte leere Kontrollkästchen (siehe Bild 5).

Bild 5: Neuer Standardwert

Die Abfrage des Wertes mit **Forms!frmKontrollkaestchen!chkBeispiel1** im Direktbereich von VBA liefert nun den Wert **0**. Wenn Sie das Kontrollkästchen anhaken, liefert die Abfrage den Wert **-1** – es funktioniert also alles wie gewünscht.

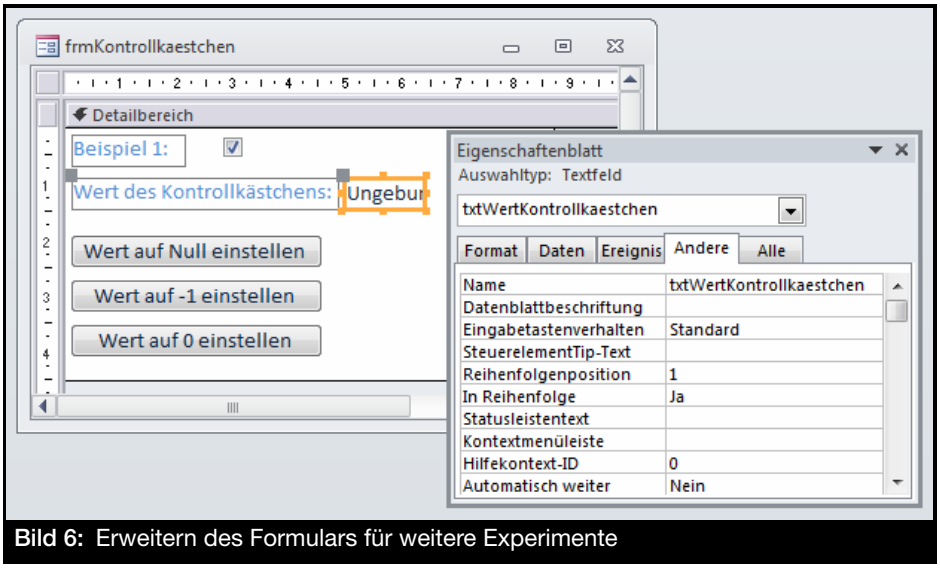


Bild 6: Erweitern des Formulars für weitere Experimente

True und False

An dieser Stelle nochmal der Hinweis, dass unter VBA die Zahl **-1** dem Wert **True** und **0** dem Wert **False** entspricht.

Dies entspricht auch den Werten eines **Ja/Nein**- beziehungsweise Boolean-Feldes in Access-Datenbanken.

Wenn Sie eine SQL Server-Datenbank verwenden, entspricht **0**

Anwendungsparts

Anwendungsparts sind eine der Neuerungen, die mit Access 2010 eingeführt wurden. Sie können damit Access-Objekte, also etwa Tabellen, Abfragen oder Formulare, in einer Vorlagen-Datenbank speichern und die Inhalte dann per Mausklick zur aktuell geöffneten Datenbank hinzufügen. Dieser Artikel zeigt, wie Sie diese Anwendungsparts nutzen und eigene Anwendungsparts erstellen.

Anwendungsparts

Wenn Sie viel mit Access arbeiten, stoßen Sie immer wieder auf ähnliche Aufgaben. Dementsprechend benötigen Sie auch immer wieder die gleichen Objekte – beispielsweise Standardtabellen wie **tblAnreden** oder **tblAdressen** oder auch Formulare, Standardmodule oder Klassenmodule mit immer wiederkehrenden Funktionen.

Die bisherige Vorgehensweise dürfte wie bei vielen entweder das Neuerstellen oder das Importieren der Objekte aus anderen Datenbankdateien sein.

Ab Access 2010 finden Sie ein interessantes Werkzeug, welches das letztere Vorgehen stark vereinfacht: Sie können eines oder mehrere vordefinierte Objekte per Ribbon-Schaltfläche zu Ihrer Anwendung hinzufügen.

Dazu klicken Sie einfach auf die Ribbon-Schaltfläche **Datei|Anwendungsparts** und öffnen so die Gallerie aus Bild 1.

Die Gallerie teilt die vorhandenen Anwendungsparts auf verschiedene Gruppen auf. Die ersten beiden Gruppen der Gallerie aus der Abbildung werden mit Access auf Ihrer Festplatte installiert.

Sie dienen im Wesentlichen als Anhaltspunkt für eigene Schöpfungen dieser Art. Die unteren beiden Gruppen

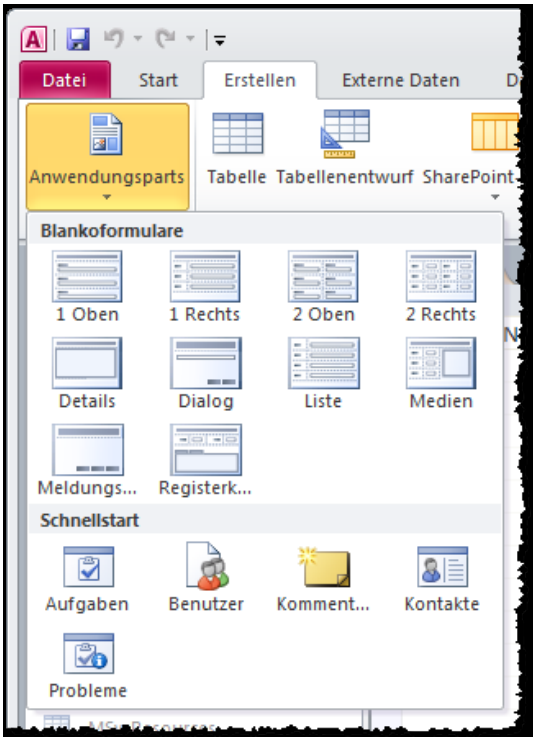


Bild 1: Auswahl vorgefertigter Anwendungsparts

enthalten bereits Beispiele für benutzerdefinierte, also selbst erstellte Anwendungsparts.

Wenn Sie beispielsweise auf einen der Einträge der Kategorie **Blankoformulare** klicken, erstellt Access ein entsprechendes Formular und speichert es in der aktuellen Datenbank.

Diese Formulare haben keine Bindung zu irgendeiner Tabelle, Sie müssten diese also zum größten Teil noch vollständig mit einer Datenherkunft verbinden. Da können Sie auch gleich ein neues Formular auf Basis

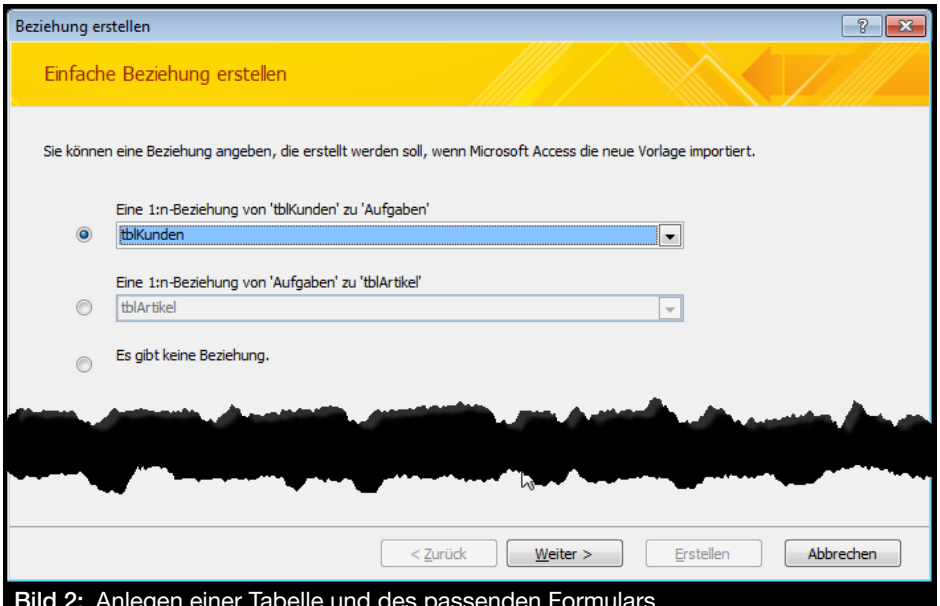


Bild 2: Anlegen einer Tabelle und des passenden Formulars

einer Tabelle der Datenbank erstellen und schnell die gewünschten Felder in das Formular ziehen.

Interessanter sind da schon die Elemente unterhalb von Schnellstart: Wenn Sie einen der dortigen Einträge auswählen, beispielsweise Benutzer, legt Access gleich mehrere Elemente an.

Klicken Sie etwa auf den Eintrag **Aufgaben**, liefert Access zunächst einen Dialog, der fragt, ob die mit diesem Anwendungspart gelieferte Tabelle **Aufgaben** mit einer der vorhandenen Tabellen verknüpft werden soll (siehe Bild 2).

Dazu gibt es zwei Möglichkeiten:

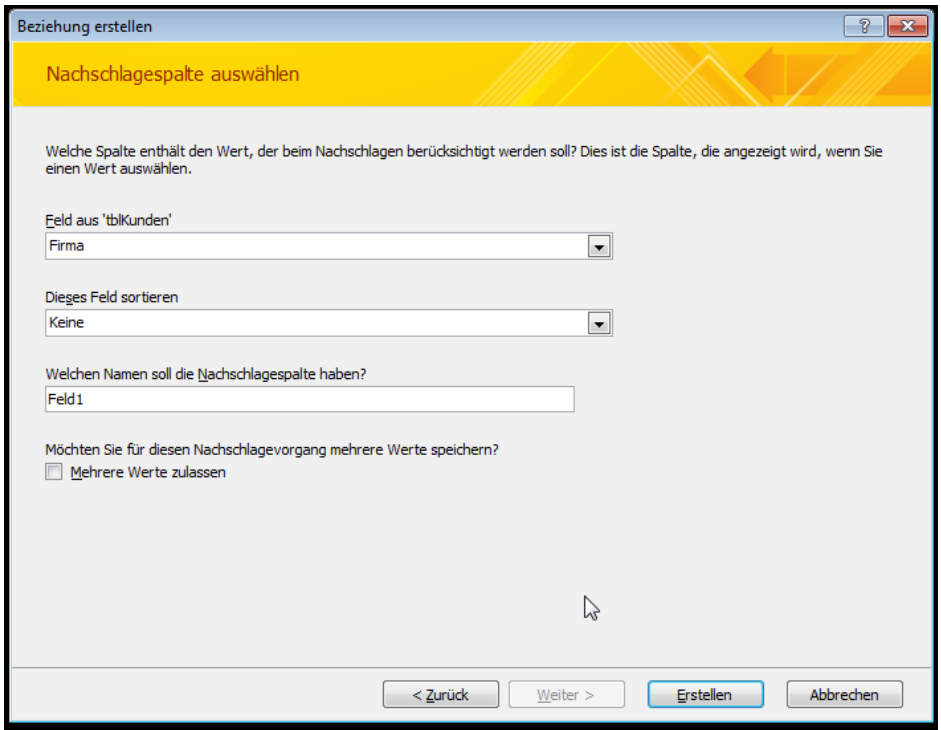


Bild 3: Erstellen einer Beziehung zwischen der neuen Tabelle der Tabelle **tblKunden**

- Sie legen fest, dass die Tabelle **Aufgaben** per 1:n-Beziehung mit einer der vorhandenen Tabellen verknüpft werden soll, also etwa mit der Tabelle **tblKunden**.
- Sie legen fest, dass eine der vorhandenen Tabellen, also etwa tblKunden, per 1:n-Beziehung mit der neu zu erstellenden Tabelle Aufgaben verknüpft werden soll.
- Alternativ geben Sie an, dass keine Beziehung angelegt werden soll.

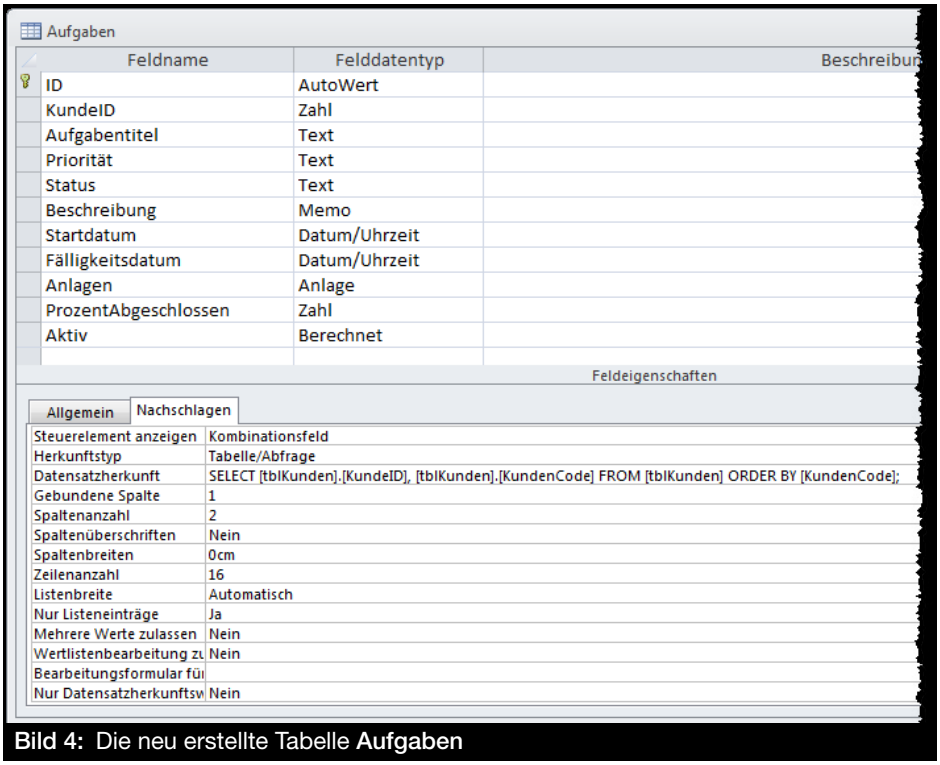


Bild 4: Die neu erstellte Tabelle **Aufgaben**

Wenn Sie etwa die erste Option markieren und dort die Tabelle **tblKunden** auswählen, bietet der folgende Dialog alle Felder der Tabelle **tblKunden** als das an der Beziehung beteiligte Primärschlüsselfeld an (siehe Bild 3). Dort geben Sie außerdem an, ob das resultierende Nachschlagefeld nach dem Feldinhalt aufsteigend oder absteigend sortiert werden soll. Schließlich geben Sie den Namen für das der Tabelle **Aufgaben** hinzuzufügende Fremdschlüsselfeld an (hier ebenfalls **KundeID**). Um es vorwegzunehmen: Dies liefert nicht das gewünschte Ergebnis. Der Assistent erkennt nämlich automatisch das Primärschlüsselfeld der zu verknüpfenden Tabelle (hier **KundeID**). Sie müssen also das Feld auswählen,

das in dem Nachschlagefeld, das für das Fremdschlüsselfeld angelegt wird, angezeigt und sortiert werden sollen. Wählen Sie also das Feld Kundencode aus und betätigen die Eingabe mit einem Klick auf die Schaltfläche **Erstellen**.

Dies legt zunächst eine Tabelle namens **Aufgaben** an, die in der Entwurfsansicht wie in Bild 4 aussieht. Das Nachschlagefeld enthält die beiden Felder **KundeID** und **Kundencode** als Felder und zeigt nur Letzteres beim Aufklappen an – soweit also alles in Ordnung!

Außerdem hat Access zwei Formulare angelegt. Das erste heißt **AufgabeDB** und sieht in der Entwurfsansicht wie in Bild 5 aus. Es scheint so, als enthielte das Formular nur ein einziges Steuerelement. Wenn man dann in die Formularansicht wechselt, zeigt es jedoch ein komplettes Datenblatt mit allen Feldern der Tabelle **Aufgaben** an. Ein Blick in die Liste der Steuerelemente und das Anklicken verschiedener Einträge zeigt dann, dass die verwendeten Bezeichnungsfelder und die gebundenen Steuerelemente alle entweder mit minimaler Breite und Höhe oder mit den Abmaßen des einzigen sichtbaren Steuerelements übereinander abgelegt wurden.

Wie auch immer – für ein Formular in der Datenblattansicht spielt dies keine Rolle, aber ein nachträgliches Editieren erfordert natürlich zunächst das Entwirren der vorhandenen Steuerelemente. Schließlich liefert der Anwendungspart noch das Formular aus Bild 6.

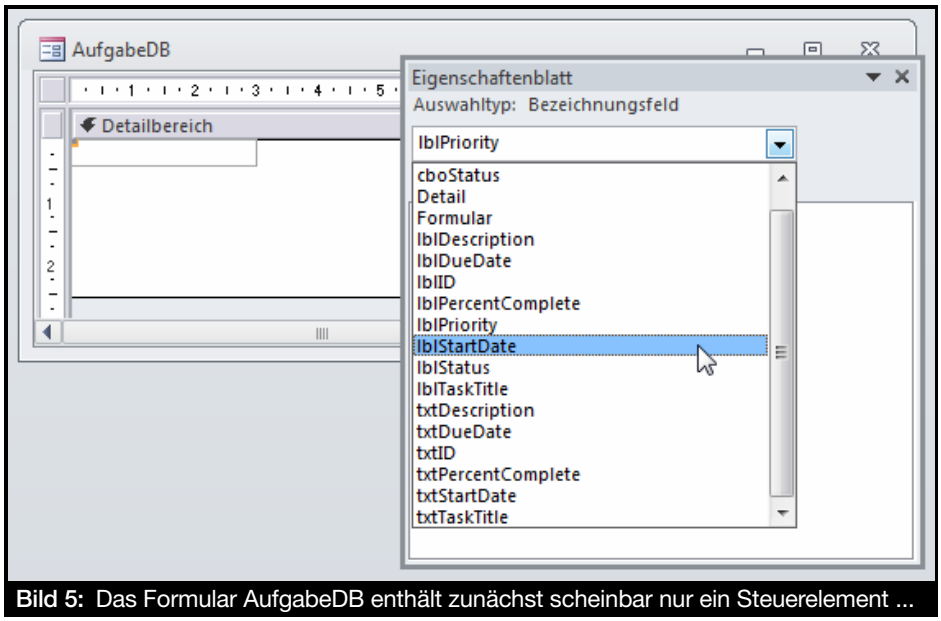


Bild 5: Das Formular AufgabeDB enthält zunächst scheinbar nur ein Steuerelement ...

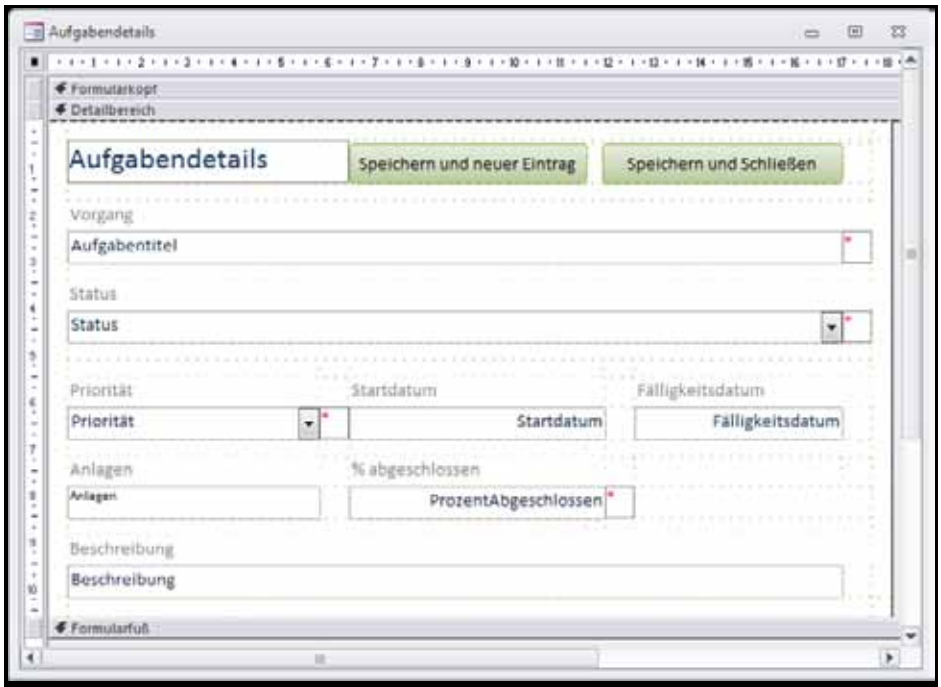


Bild 6: Entwurf des Formulars Aufgabendetails

Anwendungsparts selbst erstellen

Damit kommen wir zum interessanten Teil dieses Themas: Das Erstellen eigener Anwendungsparts und ihr Einsatz in Ihrem Programmieralltag. Bevor Sie ein neues Anwendungspart erstellen, entscheiden Sie, welche Elemente dieses enthalten soll. Beispiele für solche Zusammenstellungen sind die folgenden:

- Sie verwenden einfach nur eine einzige Tabelle wie etwa **tblAnreden** mit den beiden Werten **Herr** und **Frau** als Anwendungspart.

Autowerte

Access macht es dem Datenbankprogrammierer bei Anlegen von neuen Datensätzen sehr leicht: Es vergibt, wenn man dies so definiert hat, automatische Werte für bestimmte Felder eines Datensatzes. In der Regel ist dieses Feld identisch mit dem Primärschlüsselfeld, was auch durchaus sinnvoll ist – liefert die Autowert-Funktion von Access doch standardmäßig einen Wert, der um eins größer als der bisher größte Wert ist. Es gibt aber auch andere Einstellungen. Dieser Artikel wirft einen genauen Blick auf Autowerte in Access.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1312_Autowerte.mdb**.

Autowert anlegen

Wer schon ein paar Tabellen unter Access angelegt hat, verwendet diese Funktion wahrscheinlich bereits wie im Schlaf: Sie legen eine neue Tabelle an und fügen zunächst ein Feld hinzu, das als Primärschlüsselfeld dient und somit als Feld mit eindeutigen Werten.

Dies gewährleistet, dass jeder Datensatz zumindest anhand eines Feldes eindeutig identifiziert werden kann.

Damit dieses Feld auch zuverlässig mit einem bisher noch nicht vorhandenen Wert gefüllt wird, gibt es unter Access den Felddatentyp **Autowert**. Diesen legen Sie für das Primärschlüsselwert fest, in der Beispieldatenbank etwa für das Feld **AutowertID** der Tabelle **tblAutowerte** (siehe Bild 1).

Nachdem Sie dies erledigt haben, können Sie zur Datenblattansicht der Tabelle wechseln. Dort finden Sie nicht nur einen leeren Datensatz vor, sondern auch noch den Text **(Neu)** im Feld **AutowertID** (siehe Bild 2).

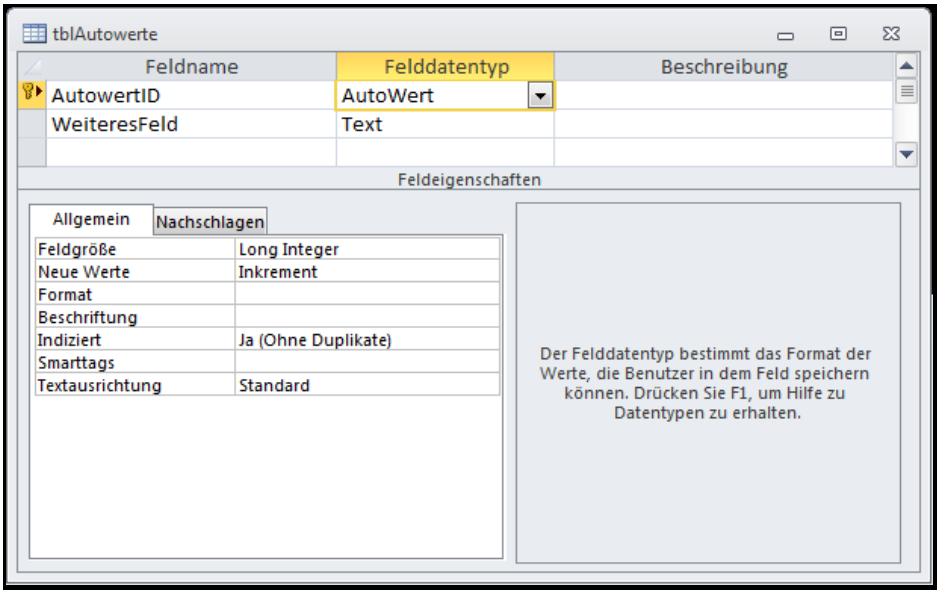


Bild 1: Anlegen einer Tabelle mit einem Autowert

Dieser ändert sich auch erst, nachdem Sie das erste Feld des frischen Datensatzes geändert haben. Das **Autowert**-Feld erhält dann, wenn die Tabelle bis dahin noch jungfräulich war, den Wert **1** (siehe Bild 3).

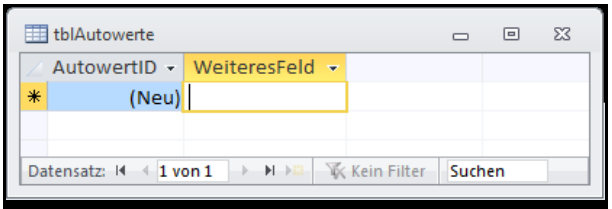


Bild 2: Das Autowert-Feld zeigt den Eintrag **(Neu)** an.

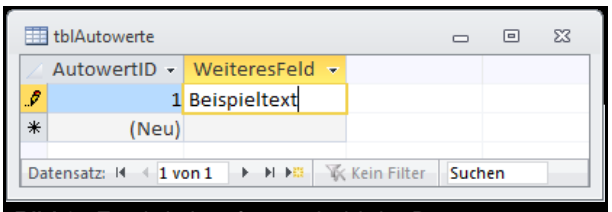


Bild 3: Er wird eingefügt, sobald der Datensatz geändert wurde.

Dieser Wert ist allerdings, wie der komplette Datensatz, noch nicht in der Tabelle gespeichert – dies geschieht erst, wenn Sie den Datensatz tatsächlich entweder durch einen Mausklick auf den Datensatzmarkierer des betroffenen Datensatzes oder durch Verlassen des geänderten Da-

tensatzes in der zugrunde liegenden Tabelle speichern (siehe Bild 4).

Lückenlos?

Dies geht nun so weiter – und wenn Sie keinen Datensatz löschen, erhalten Sie eine wunderschöne lückenlose Reihe von Primärschlüsselwerten (siehe Bild 5).

Dummerweise ist es damit vorbei, sobald Sie den ersten Datensatz löschen (siehe Bild 6). Aber was soll man an dieser Stelle sonst erwarten – wohl kaum, dass Access die Nummerierung der nachfolgenden Datensätze so ändert, dass sich die Lücke wieder schließt.

So mancher mag aber erwarten, dass zumindest der folgende Datensatz einen Autowert erhält, der die lückenlose Nummierung wiederherstellt. Das ist jedoch mitnichten der Fall: Der folgende Autowert orientiert sich brav an dem bisher größten vergebenen Wert und fügt diesem noch den Wert **1** hinzu (siehe Bild 7).

Anfügen nach Löschen?

Wenn Sie nun den zuletzt angefügten Datensatz löschen, könnten Sie zumindest theoretisch erwarten, dass Access den neuen Datensatz – wie auch die vorherigen Datensätze – wieder mit dem aktuell größten Primärschlüsselwert plus eins versieht.

Das Löschen des letzten und das erneute Anfügen eines Datensatzes liefert allerdings eine Enttäuschung: Der neue Datensatz erhält nicht den Primärschlüsselwert des zuvor gelöschten Datensatzes, sondern den um eins erhöhten Wert (siehe Bild 8).

Tabelle zurücksetzen

Dummerweise geschieht das Gleiche nicht nur, wenn Sie nur ein oder mehrere Datensätze am Ende einer Tabelle löschen, sondern auch dann, wenn Sie alle Datensätze löschen – beispielsweise, um eine bislang mit Testdaten gefüllte Datenbank zu leeren und diese dem Kunden zu übergeben.

Zum Glück gibt es eine Möglichkeit, um dies zu erledigen – nämlich durch Komprimieren der Datenbank. Unter Access 2010 erledigen Sie dies beispielswiese durch einen Mausklick auf den Reiter Datei des Ribbons und anschließendes Betätigen der Schaltfläche Daten komprimieren und reparieren im Backstage-Bereich von Access.

Nach dem Komprimieren der Datenbank fügt Access

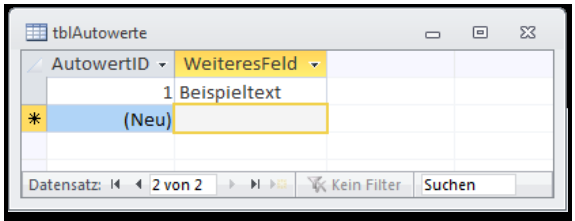


Bild 4: Weitere Datensätze erhalten den bisher größten Wert plus eins als Autowert.

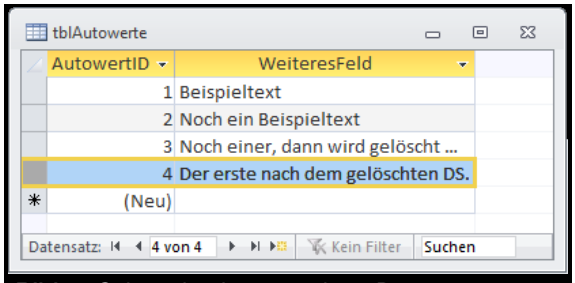


Bild 5: Schön durchnummerierte Datensätze

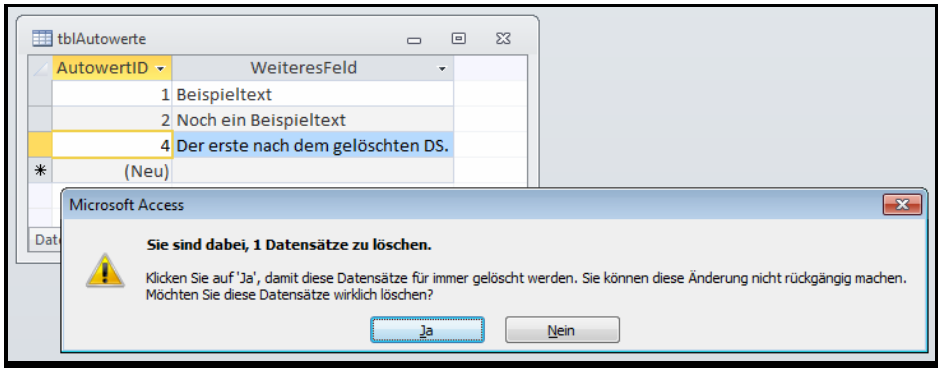


Bild 6: Aber wehe, ein Datensatz wird gelöscht!

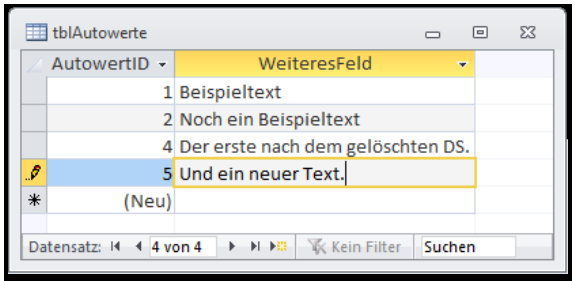


Bild 7: Die entstandene Lücke wird durch einen neuen Datensatz nicht geschlossen.

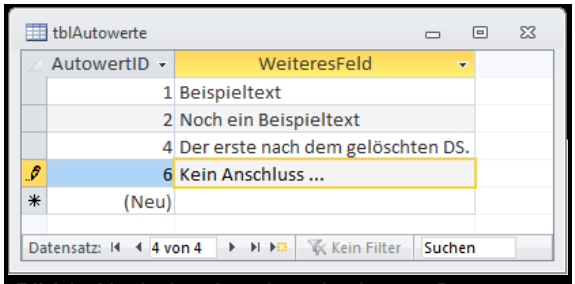


Bild 8: Nach dem Löschen des letzten Datensatzes wird dessen Autowert nicht neu vergeben

Datensätze kopieren mit Bordmitteln

Es gibt verschiedenste Anlässe, vollständige Datensätze zu kopieren. Ein Beispiel ist das Archivieren von Daten in eine andere Tabelle, ein anderes das Duplizieren von Datensätzen, um diese als Vorlage für weitere Datensätze in der gleichen Tabelle zu verwenden. Dieser Artikel zeigt, welche Möglichkeiten es zum Kopieren von Datensätzen gibt.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1312_DatensaetzeKopieren.mdb**.

Warum Datensätze kopieren?

Datensätze kopiert man aus unterschiedlichen Gründen. Der simpelste ist: Sie möchten schnell mal ein paar Testdaten erzeugen, um die Performance einer Datenbank zu testen. Oder Sie wollen einen neuen Artikel zu einer Artikeltabelle hinzufügen, der aber fast genau die gleichen Eigenschaften wie ein bereits vorhandener Artikel aufweist. Vielleicht möchten Sie auch ein oder mehrere Datensätze aus einer Tabelle in eine andere Tabelle kopieren, um die Daten zu sichern oder zu archivieren. In allen Fällen benötigen Sie eine zuverlässige Technik, die den Datensatz kopiert.

Schnelle Testdaten

Wenn Sie etwa in einer VBA-Prozedur Datensätze in einer Schleife durchlaufen, möchten Sie vielleicht wissen, wie lange dies für eine große Menge Daten dauert. Dann gibt es zwei Möglichkeiten: Entweder Sie erstellen sich eine kleine VBA-Prozedur, die ebenfalls per Schleife einige tausend Datensätze für den Test erzeugt. Oder Sie legen selbst Hand an und kopieren die vorhandenen Datensätze: Angenommen, die betroffene Tabelle enthielte nur einen einzigen Datensatz. Dann würden Sie zunächst diesen Datensatz kopieren und hätten bereits zwei Datensätze. Diese zwei kopieren Sie erneut und erhalten vier Datensätze. Diesen Vorgang wiederholen Sie so oft, bis die gewünschte Anzahl Datensätze vorliegt.

Bild 1: Markieren des zu kopierenden Datensatzes, Kopieren mit Strg + C, ...

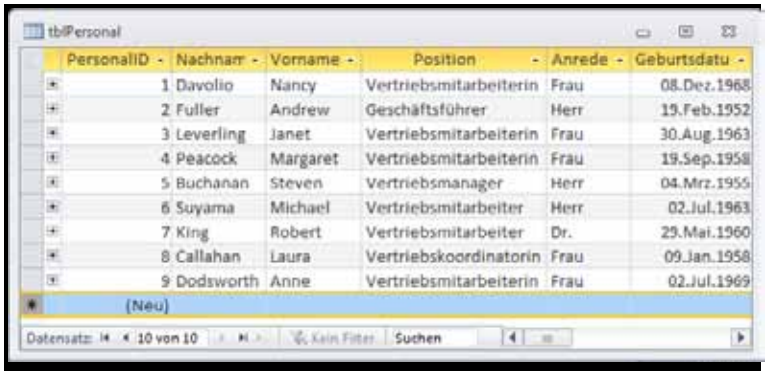


Bild 2: ... Markieren des Ziels, also eines leeren Datensatzes ...

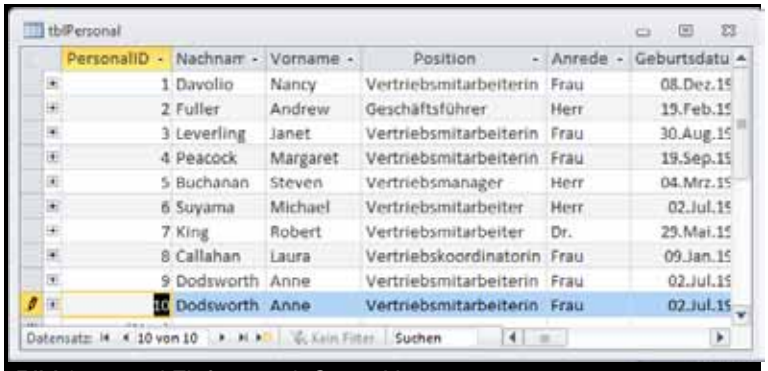


Bild 3: ... und Einfügen mit Strg + V.

Aber wie kopieren Sie einen Datensatz am einfachsten? Dazu öffnen Sie die Tabelle, in diesem Fall **tblPersonal**, markieren den zu kopierenden Datensatz durch einen Mausklick auf den Datensatzmarkierer und kopieren diesen mit **Strg + C** (siehe Bild 1).

Dann markieren Sie das Ziel, in diesem Fall den leeren, neuen Datensatz am Ende der Tabelle – wieder mit einem Maus-

klick auf den Datensatzmarkierer (siehe Bild 2).

Schließlich fügen Sie den Datensatz durch Betätigen der Tastenkombination **Strg + V** an der markierten Stelle ein (siehe Bild 3). Der Datensatz ist nun noch als „in Bearbeitung" markiert (Bleistift-Symbol im Datensatzmarkierer), da alle Felder neu gefüllt, der Datensatz aber noch nicht per Mausklick auf den Datensatzmarkierer oder durch den Wechsel zu einem anderen Datensatz verlassen wurde.

Bild 4: Markieren und kopieren mehrerer Datensätze, ...

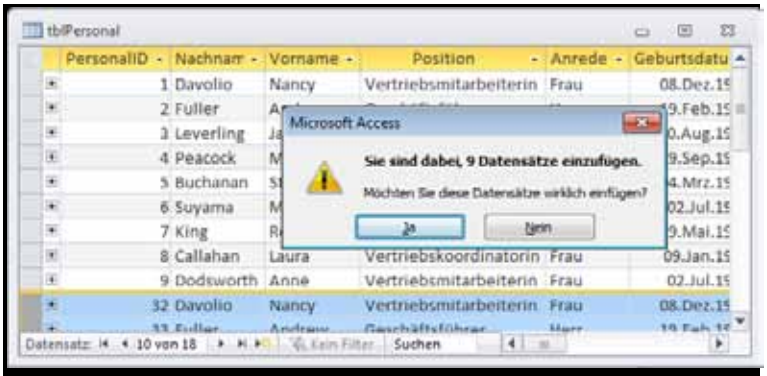


Bild 5: ... die allerdings nicht ohne vorherige Rückfrage kopiert werden.

Mehrere Datensätze kopieren

Auf die gleiche Art können Sie auch mehrere Datensätze gleichzeitig kopieren. Dazu markieren Sie einfach alle zu kopierenden Datensätze, indem Sie beispielsweise erst auf den Datensatzmarkierer für den ersten zu kopierenden Datensatz klicken und dann bei gedrückter Umschalt-Taste auf den letzten (siehe Bild 4).

Dann markieren Sie wiederum den neuen, leeren Datensatz der Tabelle und betätigen wieder die Tastenkombination **Strg + V**. Diesmal fragt Access nach, ob es die entsprechende Anzahl Datensätze kopieren soll (siehe Bild 5). Auch wenn die Datensätze zu diesem Zeitpunkt scheinbar schon in die Tabelle kopiert sind, so macht ein Klick auf die Schaltfläche **Nein** den Einfügevorgang wieder rückgängig – ein Klick auf **Ja** fügt die Datensätze natürlich an der gewünschten Stelle ein.

Datensatz über einen anderen Datensatz kopieren

Gegebenenfalls möchsten Sie auch einen Datensatz durch einen vorhandenen Datensatz ersetzen. In diesem Fall kopieren Sie den Datensatz wie zuvor beschrieben, markieren dann den Zieldatensatz (wiederum durch Anklicken des Datensatzmarkierers für diesen Datensatz) und fügen den Datensatz ein.

Access überschreibt dann den Zieldatensatz mit den Daten des Quelldatensatzes – mit Ausnahme des Primärschlüsselwertes, der beibehalten wird.

Dies gelingt auch für mehrere Datensätze. Dabei ist allerdings etwas zu beachten: Die Anzahl der Quelldatensätze sollte mit der Anzahl der Zieldatensätze übereinstimmen. Anderenfalls geschieht Folgendes: Wenn Sie etwa zwei Datensätze in einen markierten Bereich von drei Datensätze einfügen wollen, werden nur die ersten beiden überschrieben. Wenn Sie wiederum zwei Datensätze kopieren, aber nur einen Zieldatensatz markieren, kopiert Access nur den ersten Datensatz.

Datensatz per VBA kopieren

Natürlich gelingt dies auch per VBA. Wir wollen dem Benutzer ja nicht noch erläutern, wie er einen Datensatz in der Datenblattansicht kopieren kann – außerdem ist ja gar nicht sichergestellt, dass diese immer in der Datenblattansicht vorliegen. Vielleicht soll ja auch einmal ein Datensatz kopiert werden, der in einem Listenfeld erscheint.

Unter VBA gibt es die folgenden Möglichkeiten:

- Der aktuell im Datenblatt markierte Datensatz soll kopiert werden: Dann sind einige Befehle nötig, welche die zuvor manuell beschriebenen Schritte durchführen.

- Alternativ (und auch, wenn der zu kopierende Datensatz auf andere Weise ausgewählt wurde) erledigen Sie dies mit den entsprechenden DAO-Anweisungen (Öffnen des Recordsets mit dem zu kopierenden Datensatz, Erstellen eines neuen Datensatzes, Einfügen der Daten und Speichern des neuen Datensatzes in der Tabelle).

- Oder Sie verwenden eine einfache **INSERT INTO**-Anweisung.

- Gegebenenfalls wollen Sie ein oder mehrere Datensätze gleich in eine neue Tabelle kopieren. Dann verwenden Sie eine **SELECT INTO**-Anweisung.

Die folgenden Abschnitte beschreiben diese Vorgehensweisen – zunächst nach der Methode, die Sie mit dem Befehlsschaltflächen-Assistent erhalten.

Erstellen Sie dazu ein neues Formular, dass jeweils einen Datensatz der Tabelle **tblArtikel** in der Formularansicht soll (**frmDatensatzKopierenHF**). Dazu stellen Sie die Tabelle **tblArtikel** als Datenherkunft des Formulars ein und ziehen alle Felder der Tabelle aus der Feldliste in den Detailbereich des Formulars.

Nun wollen wir dem Formular eine Schaltfläche hinzufügen, mit dem wir den aktuellen Datensatz kopieren wollen. Dazu aktivieren Sie zunächst den Steuerelement-Assistenten (siehe Bild 6). Wenn Sie dann auf gewohnte Weise eine neue Schaltfläche im Formular anlegen, erscheint

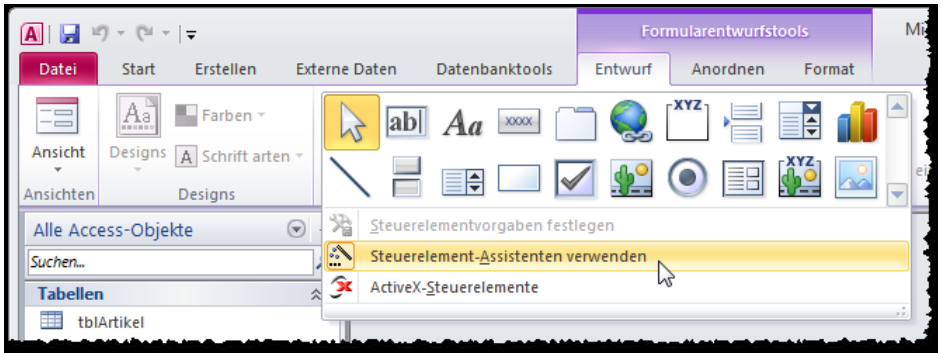


Bild 6: Aktivieren des Steuerelement-Assistenten

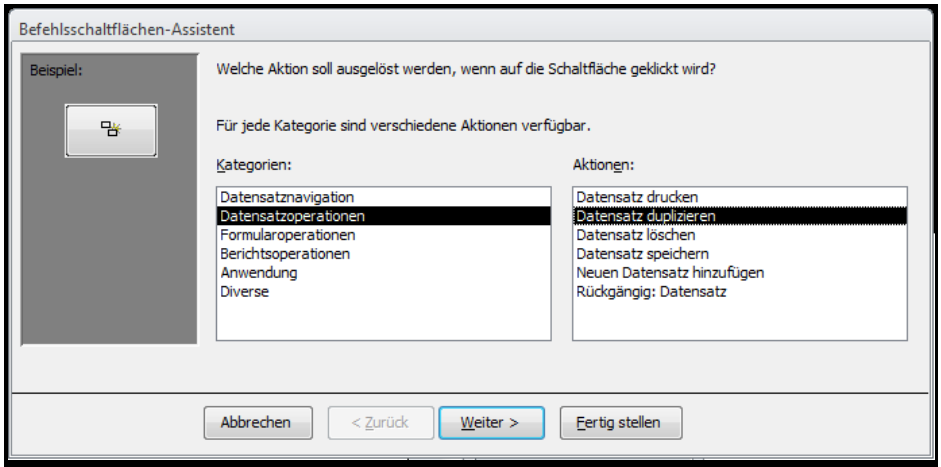


Bild 7: Der Befehlsschaltflächen-Assistent

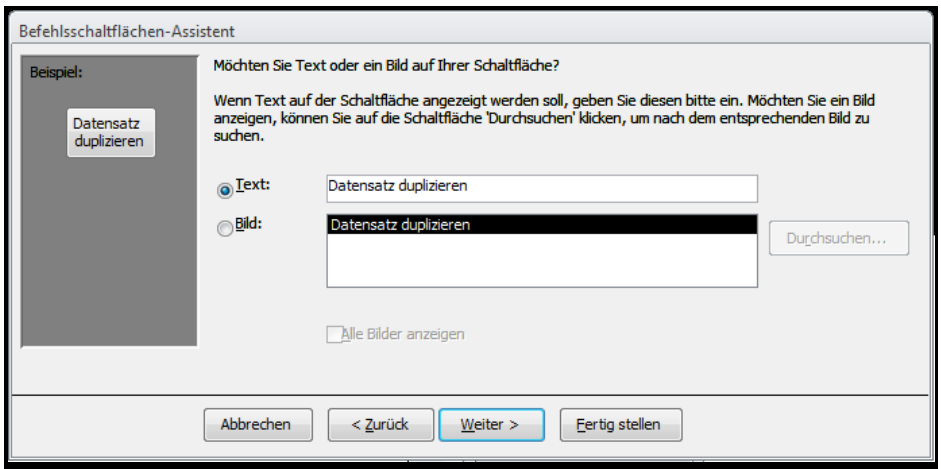


Bild 8: Schaltflächentext festlegen

Daraufhin erscheint der Befehlsschaltflächen-Assistent (siehe Bild 7). Hier wählen Sie die beiden Einträge **Datensatzoperationen** und **Datensatz duplizieren** aus.

Im nächsten Schritt fragt der Assistent Sie, ob und welche Beschriftung die Schaltfläche enthalten soll oder ob Sie ein Icon verwenden wollen (siehe Bild 8).

Neue Autowerte ermitteln und RecordsAffected

Mit der **Autowert**-Funktion liefert Access eine einfache Möglichkeit, eindeutige Werte für Primärschlüsselfelder zu vergeben. Für manche Zwecke benötigt man aber gleich nach dem Anlegen eines Datensatzes seinen Autowert. Wie diesen erhalten, erfahren Sie im vorliegenden Artikel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1401_Autowerte.mdb**.

ID des zuletzt hinzugefügten Autowertes ermitteln

Viele Anwendungsfälle verlangen danach, die ID eines gerade hinzugefügten Datensatzes zu ermitteln. Dies kann sowohl beim Hinzufügen per DAO (beziehungsweise ADO) also auch per SQL-INSERT INTO-Abfrage der Fall sein. Für beides gibt es Lösungen, die wir in diesem Artikel vorstellen. Als Beispieltabelle verwenden wir die Tabelle **tblAutowerte** (siehe Bild 1), die neben dem Autowert-Feld **AutowertID** nur noch ein weiteres Feld namens **WeiteresFeld** enthält.

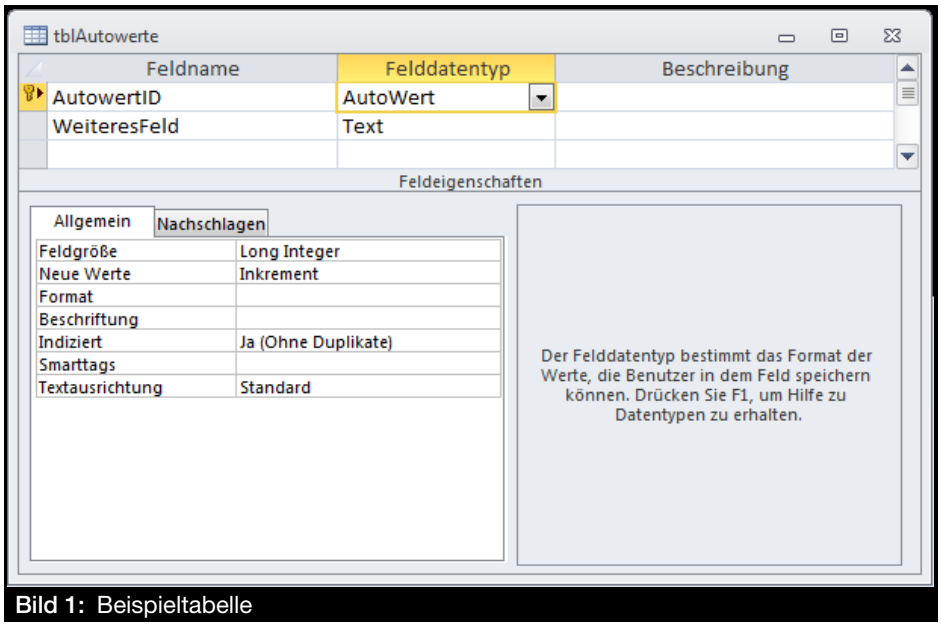


Bild 1: Beispieltabelle

Neuen Autowert unter DAO ermitteln

Unter DAO legen Sie neue Datensätze mit der **AddNew**-Methode an und fügt dann den einzelnen Feldern die gewünschten Werte hinzu. Wenn Sie anschließend den Autowert des neu angelegten Datensatzes benötigen, brauchen Sie diesen nur auszulesen – er ist beim Anlegen bereits vorhanden. Allerdings muss man dies vor dem Speichern des Datensatzes mit der **Update**-Methode erledigen.

Listing 1 zeigt, wie dies funktioniert. Hier legen wir zunächst mit der **AddNew**-Funktion einen neuen Datensatz an und weisen dem einzigen Nicht-Primärschlüsselfeld des Recordsets einen Wert zu.

Den Wert des Autowert-Feldes können Sie bereits nach dem Ausführen der **AddNew**-Methode abfragen – dies ist der früheste Zeitpunkt. Der späteste Zeitpunkt ist kurz vor dem Speichern des Datensatzes in der zugrunde liegenden Tabelle, also vor dem Aufrufen der **Update**-Methode.

Hier nutzen wir in der Beispielprozedur eine **MsgBox**-Funktion auf,

```
Public Sub AutowertMitAddNewUpdate()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("tblAutowerte", dbOpenDynaset)
    rst.AddNew
    rst!WeiteresFeld = "Test"
    MsgBox "Die neue ID lautet: " & rst!AutowertID
    rst.Update
    rst.Close
    Set rst = Nothing
    Set db = Nothing
End Sub
```

Listing 1: Autowert per DAO ermitteln

GUIDs

Normalerweise verwenden Sie zur eindeutigen Kennzeichnung eines Datensatzes in einer Tabelle einen numerischen Primärschlüssel, der inkrementell für jeden neuen Datensatz erhöht wird. Manchmal reicht dies allerdings nicht aus – zum Beispiel dann, wenn es mehrere Kopien der Datenbank gibt, in denen Datensätze hinzugefügt und anschließend zusammengeführt werden müssen. Die Anforderung ist durchaus realistisch – es gibt erstens nicht immer und überall Internetzugriff, und zweitens will längst nicht jedes Unternehmen seine Datenbank über das Internet verfügbar machen. Die Lösung für die problemlose Zusammenführung der Daten sind Primärschlüssel, die weltweit eindeutig sind – und dies erreichen Sie mit den sogenannten GUIDs.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1401_GUIDs.mdb**.

Die GUID

GUID ist die Abkürzung für **Globally Unique Identifier**. Während herkömmliche Primärschlüsselfelder normalerweise mit numerischen Werten des Datentyps **Long** gefüllt werden, was einem Wertebereich von -2.147.483.648 bis 2.147.483.647 entspricht (also insgesamt 2 hoch 32 Werten und einem Speicherplatz von 4 Byte entspricht, belegt ein GUID-Wert 16 Byte Speicher. Damit lassen sich allerdings auch eine ganze Reihe mehr Werte erzeugen.

Wie ist eine GUID nun aufgebaut? Sie besteht aus 32 Buchstaben beziehungsweise Zahlen, wobei die Zahlen von 0 bis 9 und die Buchstaben von A bis F verwendet werden (entsprechend dem hexadezimalen System). Die 32 Zeichen werden typischerweise in einer bestimmten Anordnung notiert, die etwa so aussieht und in geschweifte Klammern eingefasst wird:

```
{9E0C0079-213E-48F5-9432-BC17E2D2CB9F}
```

Zwei Zeichen ergeben jeweils ein Byte, sodass die Anordnung bezüglich der Anzahl der verwendeten Bytes wie folgt aussieht:

4-2-2-2-6

Weltweit eindeutig?

Was aber macht eine GUID nun weltweit eindeutig und somit zu einem bevorzugten Primärschlüsselwert, wenn es zum die Aufteilung und Synchronisierung von Datenbanken geht? Die Eindeutigkeit ist nur gewährleistet, wenn die GUID durch einen Rechner erzeugt wird, der eine Ethernet-Netzwerkkarte verwendet. Wenn Sie sich selbst GUIDs ausdenken, können Sie natürlich leicht zwei gleiche Bytefolgen erzeugen – Gleiches gilt, wenn Sie einen eigenen Algorithmus zur Erstellung von GUIDs verwenden.

Um die Eindeutigkeit zu gewährleisten, wird ein Teil der GUID in Abhängigkeit von der physikalischen Adresse der Ethernet-Netzwerkkarte erzeugt, ein anderer Teil wir inkrementell erhöht – so ist, die Eindeutigkeit der Adresse der Ethernetkarte vorausgesetzt, die doppelter Vergabe einer GUID ausgeschlossen.

GUIDs als Autowert

Es gibt zwei Möglichkeiten, eine GUID zu erzeugen. Die erste liefert Access von Haus aus mit – es handelt sich hierbei um die Möglichkeit, Autowerte des Typs **Replikations-ID** zu erzeugen. Dazu legen Sie ein Feld einer Tabelle, etwa das Primärschlüssel, mit dem Datentyp **Autowert** und der Feldgröße **Replikations-ID** an (siehe Bild 1).

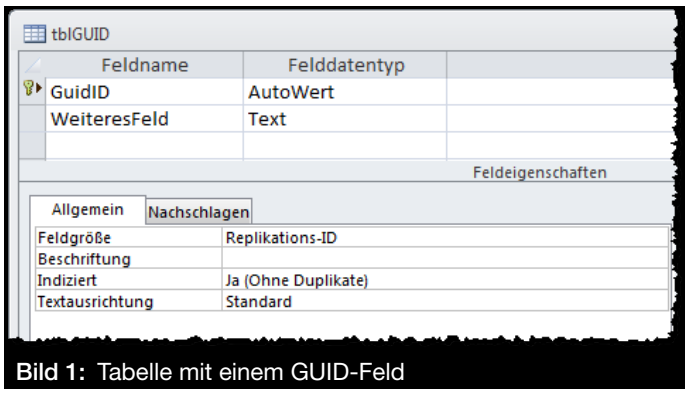


Bild 1: Tabelle mit einem GUID-Feld

Wenn Sie nun in die Datenblattansicht wechseln und einige neue Datensätze anlegen, zeigt das Feld **GuidID** frisch erzeugte GUID-Werte an (siehe Bild 2).

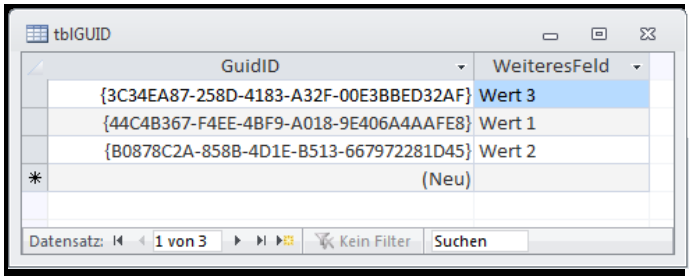


Bild 2: GUID-Werte einer Tabelle

Aber warum eigentlich Replikations-ID? Dies ist in einer von den aktuellen Versionen von Access nicht mehr unterstützten Technik begründet. Früher konnten Sie Access-Datenbanken replizieren und wieder synchronisieren, das heißt, dass Access Funktionen angeboten hat, mit denen Sie mehrere Versionen (Replikate) einer Datenbank erstellen konnten. Diese haben etwa Außendienstmitarbeiter zur Erfassung von Kunden, Aufträgen et cetere verwendet. Später wurden die ermittelten Daten per Synchronisation wieder zu einer Datenbank zusammengeführt.

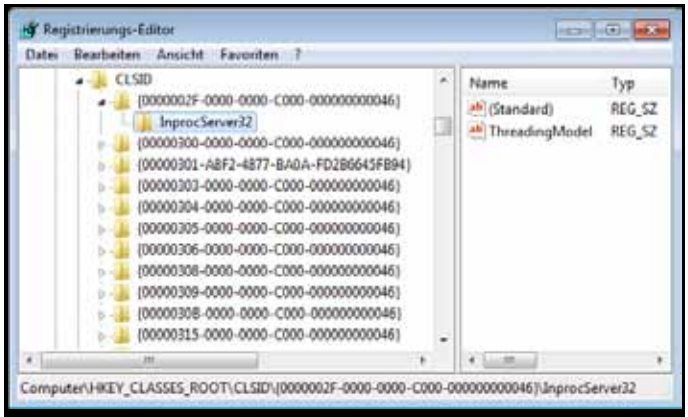


Bild 3: GUIDs in der Registry von Windows

GUID per API

Es gibt auch eine API-Funktion, mit der Sie GUIDs erstellen können. API-Funktionen sind Windows-Funktionen, die Sie aber auch von VBA aus verwenden können (mehr dazu in einem weiteren Artikel). Windows verwendet GUIDs beispielsweise in der Registry (die Sie mit dem Befehl **RegEdit** im Startmenü von Windows anzeigen und bearbeiten können), wie Bild 3 zeigt.

Um eine GUID per VBA zu erzeugen, benötigen Sie zunächst die Deklaration der zu verwendenden API-Funktion. Diese fügen Sie am besten in ein Standardmodul ein (in der Beispieldatenbank unter den Namen **mdlGUID** zu finden):

```
Private Declare Function CoCreateGuid
Lib "ole32.dll" (tGUIDStructure As
GUID) As Long
```

Außerdem benötigen Sie eine weitere API-Funktion als Hilfsfunktion, die Sie an gleicher Stelle deklarieren:

```
Private Declare Function StringFrom-
GUID2 Lib "ole32.dll" (rGUID As Any,
ByVal lpstrClsId As Long, ByVal cbMax
As Long) As Long
```

Die API-Funktion **CoCreateGuid** liefert die GUID in Form eines **Type**-Objekts, dass Sie zuvor noch deklarieren müssen. Die folgende Deklaration landet ebenfalls im Standardmodul **mdlGUID**:

```
Private Type GUID
    Data1 As Long
    Data2 As Integer
    Data3 As Integer
    Data4(0 To 7) As String * 1
End Type
```

Die Funktion **GUIDErzeugen** aus Listing 1 erzeugt schließlich eine GUID. Dazu deklariert die Prozedur zunächst eine Konstante namens **cLngLen** zum Speichern der Länge der Zeichenkette der GUID, eine Variable namens **strGUID**, welche die GUID aufnehmen soll, eine Variable namens **typeGUID**, die ein Objekt des zuvor deklarierten Typs **GUID** aufnimmt und ein Array des Datentyps **Byte** mit dem Namen **bytGuid()**. Die **Long**-Variable **lngReturn** schließlich nimmt das Ergebnis der Funktion **CoCreateGUID** auf.

Diese ruft die Funktion als erstes auf, und zwar mit dem Inhalt der Variablen **typeGUID** als Parameter. Wenn das Ergebnis **0** ist, ist etwas schiefgegangen und die Funktion führt den Teil innerhalb der **If...Then**-Bedingung nicht aus. Anderenfalls füllt die Funktion zunächst das Byte-Array **bytGUID** mit der dem Wert der Konstanten **cLngLen** entsprechenden Anzahl des Wertes 0. Dann erfolgt der Aufruf der zweiten deklarierten API-Funktion **StringFromGUID2**. Dieser übergeben wir als Parameter die mit der Variablen **typeGUID** gespeicherten GUID-Struktur, die mit **VarPtr** ermittelte Adresse der Variab-

Datensätze per VBA kopieren

Im Artikel **Datensätze kopieren** haben Sie bereits erfahren, wie Sie Datensätze ohne eine einzige selbstgeschriebene Zeile VBA-Code kopieren können. Der vorliegende Artikel zeigt, wie Sie dies mit VBA erledigen und damit einige Flexibilität gewinnen. Wir stürzen uns dabei zunächst auf die Datensätze, die in einem Unterformular angezeigt werden und schauen uns danach die übrigen Methoden von VBA zum Kopieren von Datensätzen an.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1401_DatensaetzeKopierenVBA.mdb**.

Datensätze im Unterformular kopieren
Im Artikel **Datensätze kopieren** haben wir zuletzt per Schaltflächen-Assistent eine Schaltfläche mit einer Prozedur erstellt, die fünf **DoCmd.RunCommand**-Befehle mit verschiedenen Parametern aufruft, um den im aktuellen Formular angezeigten Datensatz zu kopieren und als neuen Datensatz in die Tabelle einzufügen.

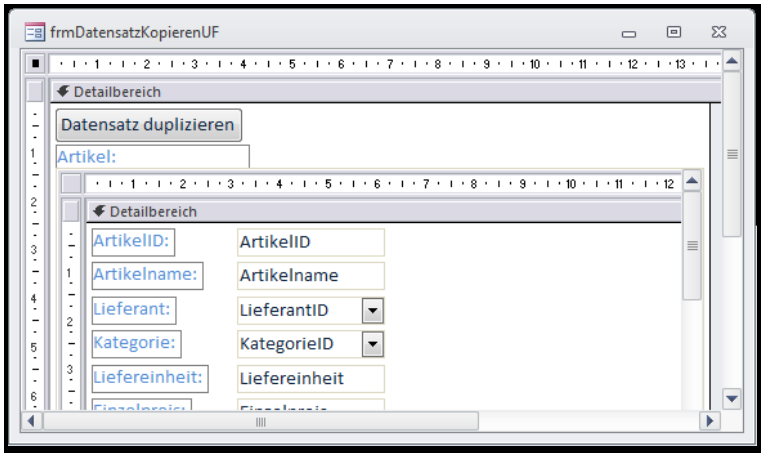


Bild 1: Haupt- und Unterformular mit der Schaltfläche zum Kopieren im Hauptformular und den Daten im Unterformular

```
Private Sub cmdDatensatzKopierenUF_Click()
    'Fehlerbehandlung entfernt
    DoCmd.RunCommand acCmdSelectRecord
    DoCmd.RunCommand acCmdCopy
    DoCmd.RunCommand acCmdRecordsGoToNew
    DoCmd.RunCommand acCmdSelectRecord
    DoCmd.RunCommand acCmdPaste
End Sub
```

Listing 1: Kopieren des Datensatzes im Unterformular

Eigentlich, so nimmt man naiverweise an, sollte dies doch genau in dieser Form auch funktionieren, wenn wir den Datensatz in einem Unterformular, das seine Daten in der Datenblattansicht anzeigt, kopieren wollen. Wenn wir also ein Hauptformular erstellen und in dieses ein Unterformular einfügen, dass die Tabelle **tblArtikel** als Datenherkunft verwendet, alle Felder dieser Tabelle anzeigt und den Wert **Datenblatt** als **Standardansicht** nutzt, brauchen wir nur noch eine Schaltfläche im Hauptformular, die den Kopiervorgang durchführt (siehe Bild 1).

Diese nennen wir **cmdDatensatzKopierenUF** und legen dafür, wieder mit dem Schaltflächen-Assistent, die Prozedur zum Duplizieren des aktuellen Datensatzes an. Die Prozedur erleichtern wir noch um die Fehlerbehandlung, um direkt zu erkennen, welche Zeile einen eventuellen Fehler auslöst (siehe Listing 1).

Anschließend starten Sie die Prozedur mit einem Mausklick auf die Schaltfläche **cmd-**

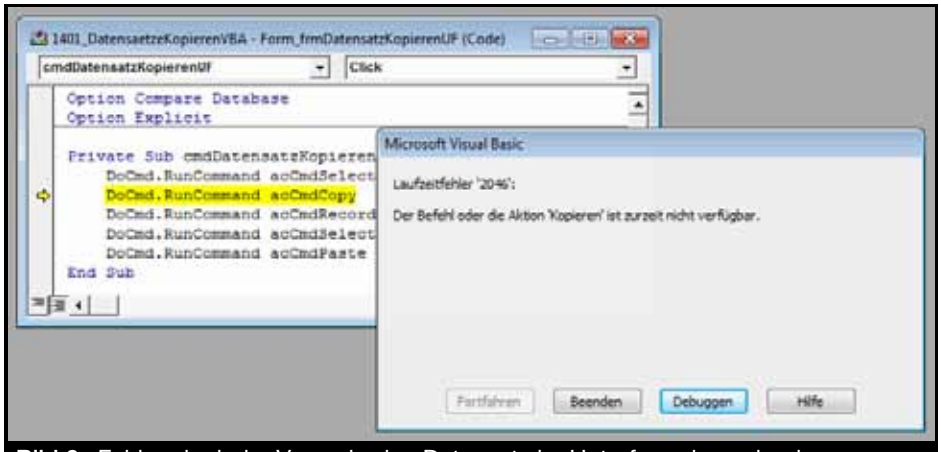


Bild 2: Fehler, der beim Versuch, den Datensatz im Unterformular zu kopieren, ausgelöst wird.

Datensatzkopieren-UF und erhalten die Fehlermeldung aus Bild 2. Anscheinend findet Access also nichts, was es aktuell kopieren könnte – was bei genauerer Betrachtung auch logisch ist: Immerhin sollte das zu kopierende Objekt auch den Fokus besitzen oder in einer anderen Form markiert sein. Aber selbst wenn Sie noch vor dem Mausklick auf die Schaltfläche **cmdDatensatzKopierenUF** den zu kopierenden Datensatz im Unterformular markiert haben, so verliert das Unterformular doch zum Zeitpunkt des Mausklicks auf die Schaltfläche den Fokus. Das ist jedoch kein großes Problem: Es gibt nämlich ebenfalls Befehle, um den Fokus auf die meisten Elemente der benutzerdefinierten Formulare zu verschieben – in diesem Fall auf das Unterformular-Steuerelement. Damit die Prozedur ihren Dienst tut, müssen Sie einfach nur die folgende Anweisung als erste Zeile der Prozedur einfügen:

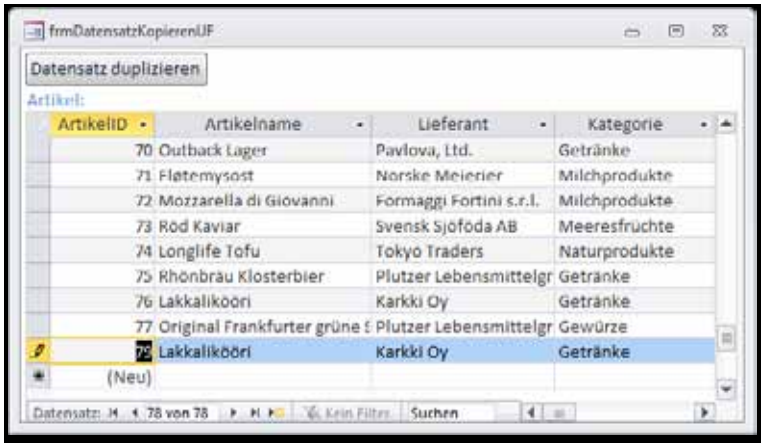


Bild 3: Kopieren und Einfügen eines Datensatzes im Unterformular per Schaltfläche

```
Me!sfmDatensatzKopieren.SetFocus
```

Nun erhält das Unterformular nach dem Anklicken der Schaltfläche den Fokus und die folgenden fünf **DoCmd.RunCommand**-Befehle können ihren Dienst verrichten. Der kopierte Datensatz landet dann am Ende des Datenblatts und wird gleich markiert (siehe Bild 3).

Datensatz per DAO kopieren

Im nächsten Beispiel wollen wir eine Variante vorstellen, die per direktem Zugriff auf die zu kopierenden Daten

```
Private Sub cmdDatensatzKopierenDAO_Click()
    Dim db As DAO.Database
    Dim rstOriginal As DAO.Recordset
    Dim rstDuplikat As DAO.Recordset
    Dim lngOriginalID As Long
    Dim lngDuplikatID As Long
    lngOriginalID = Me!sfmDatensatzKopieren.Form!ArtikelID
    Set db = CurrentDb
    Set rstOriginal = db.OpenRecordset("SELECT * FROM tblArtikel WHERE ArtikelID = " & lngOriginalID, dbOpenDynaset)
    Set rstDuplikat = db.OpenRecordset("SELECT * FROM tblArtikel WHERE 1 = 2", dbOpenDynaset)
    rstDuplikat.AddNew
    rstDuplikat!Artikelname = rstOriginal!Artikelname
    rstDuplikat!LieferantID = rstOriginal!LieferantID
    rstDuplikat!KategorieID = rstOriginal!KategorieID
    rstDuplikat!Liefereinheit = rstOriginal!Liefereinheit
    rstDuplikat!Einzelpreis = rstOriginal!Einzelpreis
    rstDuplikat!Lagerbestand = rstOriginal!Lagerbestand
    rstDuplikat!BestellteEinheiten = rstOriginal!BestellteEinheiten
    rstDuplikat!Mindestbestand = rstOriginal!Mindestbestand
    rstDuplikat!Auslaufartikel = rstOriginal!Auslaufartikel
    rstDuplikat!AusgelaufenAm = rstOriginal!AusgelaufenAm
    lngDuplikatID = rstDuplikat!ArtikelID
    rstDuplikat.Update
    Me!sfmDatensatzKopieren.Form.Recordset.Requery
    Me!sfmDatensatzKopieren.Form.Recordset.FindFirst "ArtikelID = " & lngDuplikatID
End Sub
```

Listing 2: Kopieren eines Datensatzes per DAO

Verknüpfte Daten kopieren

Das Kopieren einfacher Datensätze ist schnell erledigt. Markieren, kopieren, einfügen – schon liegt der neue Datensatz vor. Was aber geschieht, wenn an dem zu kopierenden Datensatz noch weitere Daten hängen wie etwa solche aus verknüpften Tabellen? Dann gilt es erst einmal, die Beziehung zu prüfen und dann zu entscheiden, ob die verknüpften Daten ebenfalls dupliziert werden müssen. Und schließlich benötigen Sie auch noch etwas VBA-Code, um die verknüpften Daten in einem Rutsch zu kopieren. All dies finden Sie im vorliegenden Artikel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1401_VerknuepfteDatenKopieren.mdb**.

Daten aus verknüpften Tabellen kopieren

Bevor wir uns auf die Programmierung der VBA-Routinen stürzen, wollen wir uns erst einmal ansehen, welche Konstellationen verknüpfter Daten es gibt und wann überhaupt auch die Inhalte der verknüpften Tabellen kopiert werden müssen.

Der einfachste Fall liegt vor, wenn etwa eine Tabelle namens **tblKunden** über das Fremdschlüsselfeld **AnredeID** mit einem Datensatz der Tabelle **tblAnreden** verknüpft ist (siehe Bild 1).

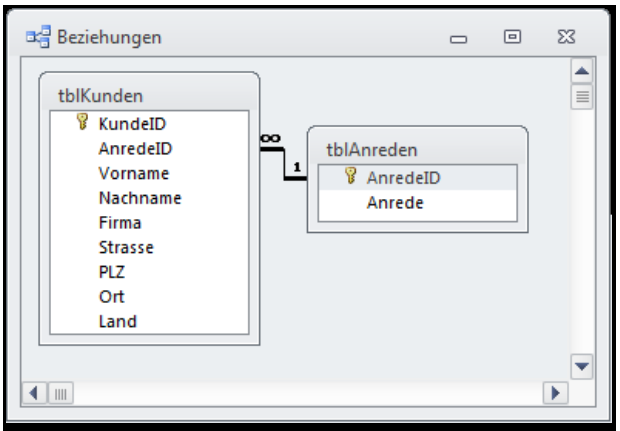


Bild 1: Beziehung zwischen Kunden und Anreden

Die Tabelle **tblAnreden** enthält jede benötigte Anrede einmal, was schon darauf hindeutet, dass beim Kopieren eines Datensatzes einer Kundentabelle kein neuer Datensatz in der verknüpften Tabelle **tblAnreden** angelegt werden muss. Und so ist es: Sie kopieren einfach den Kundendatensatz, wobei der neue Datensatz im Fremdschlüsselfeld **AnredeID** den gleichen Verweis auf einen Datensatz der Tabelle **tblAnreden** enthält wie der Originaldatensatz.

Dies gilt eigentlich für alle Beziehungen, in denen die über ein Fremdschlüsselfeld referenzierte Tabelle lediglich Lookup-Werte für ein Feld der Haupttabelle liefert – also beispielsweise auch für Tabellen wie **tblKategorien**, **tblGeschlecht** et cetera. Dies kann sich aber auch auf Tabellen beziehen, die mehr als nur einen Lookup-Wert liefern, also beispielsweise einer Tabelle wie **tblLieferanten**, die etwa mit einer Tabelle namens **tblArtikel** verknüpft ist.

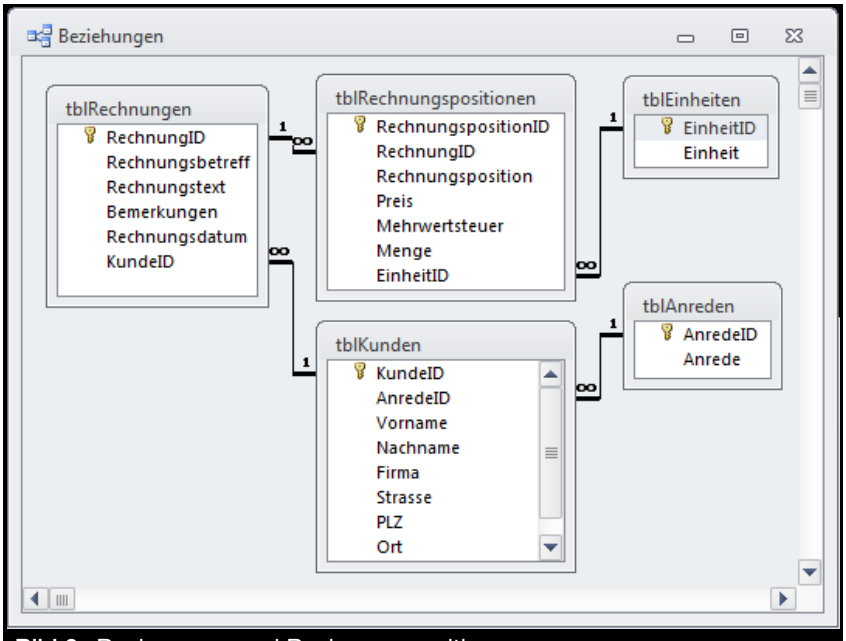


Bild 2: Rechnungen und Rechnungspositionen

Individuelle Rechnungen

Anders sieht es aus, wenn die verknüpfte Tabelle Daten enthält, die erst in

Zusammenhang mit dem Datensatz der Haupttabelle angelegt werden. Wenn Sie also etwa eine Rechnung zu einem Auftrag in der Tabelle **tblRechnungen** anlegen und dazu einige Rechnungspositionen in der damit verknüpften Tabelle **tblRechnungspositionen**, ändert sich die Situation (siehe Bild 2): Wenn Sie einen Rechnung kopieren, werden Sie wohl auch die Rechnungspositionen kopieren, um diese gegebenenfalls individuell anpassen zu können.

In diesem Fall gehen wir davon aus, dass es sich um Rechnungen für individuelle Leistungen handelt – also beispielsweise das Programmieren einer Anwendung für einen Kunden. Und trotz aller Individualität: Wenn es sich um ein größeres Projekt handelt, werden Sie möglicherweise immer wieder ähnliche Positionen in Rechnung stellen, die Sie nicht jedes Mal erneut formulieren möchten. Dies ist der erste Fall, den wir uns in diesem Artikel anschauen werden.

Bestellungen und Artikel

Der zweite Fall ist die klassische Bestellung mit Bestellpositionen und Artikeln. Hier werden die Bestellungen und die Artikel über eine Tabelle etwa namens tblBestelldetails miteinander per m:n-Beziehung verknüpft.

Wenn Sie hier eine Bestellung neu auf einer bereits vorhandenen Bestellung anlegen möchten, welche die gleichen Bestellpositionen wie das Original enthält, werden Sie sich freuen, wenn Sie dies per Mausklick erledigen können statt die Bestellung manuell Bestellposition für Bestellposition anzulegen. Dies ist das zweite in diesem Artikel behandelte Beispiel (siehe Bild 3).

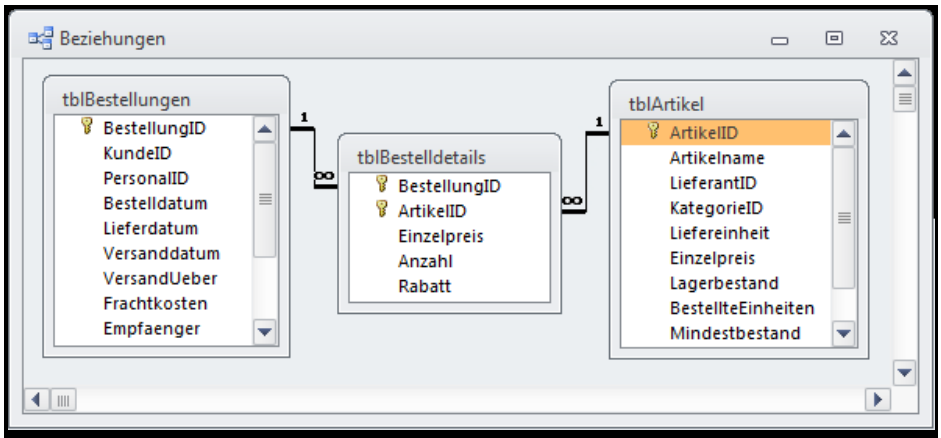


Bild 3: Bestellungen, Bestellpositionen und Artikel

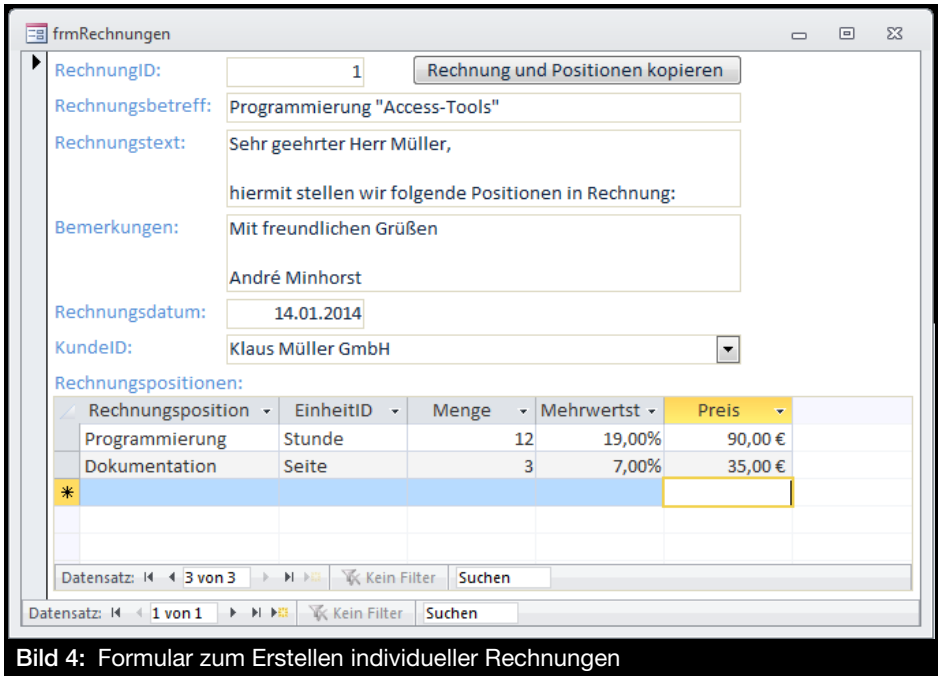


Bild 4: Formular zum Erstellen individueller Rechnungen

Daten aus 1:n-Beziehungen kopieren

Nun wollen wir uns das erste Beispiel der individuellen Rechnungen ansehen. Den relevanten Teil des Datenmodells haben Sie ja bereits weiter oben kennengelernt. Zusätzlich haben wir ein Formular erstellt, dass die Daten der Tabelle tblRechnungen anzeigt und die Daten der per 1:n-Beziehung verknüpften Tabelle tblRechnungspositionen zur aktuellen Rechnung in einem Unterformular anzeigt (siehe Bild 4). Haupt- und Unterformular sind über den Wert **RechnungID** in den Eigenschaften **Verknüpfen von** und **Verknüpfen nach** des Unterformular-Steuerelements miteinander verknüpft.

Wenn Sie nun für den gleichen Kunden eine neue Rechnung erstellen, können Sie dies natürlich manuell

erledigen. Wenn sich jedoch weder der Rechnungsbetreff noch die enthaltenen Rechnungspositionen nicht wesentlich geändert haben, können Sie auch den vorhandenen Rechnungsdatensatz kopieren – und ebenso die damit verknüpften Rechnungspositionen.

Dazu haben wir oben im Formular eine Schaltfläche namens **cmdRechnungKopieren** angelegt. Die durch das Anklicken dieser Schaltfläche ausgelöste Ereignisprozedur soll nun zwei Aufgaben erledigen:

- Kopieren des Datensatzes aus der Tabelle **tblRechnung** (also der im Hauptformular angezeigte Datensatz) und
- Kopieren der mit dem Datensatz im Hauptformular verknüpften Datensätze der Tabelle **tblRechnungspositionen** (also der Datensätze im Unterformular).

Es gibt (mindestens) zwei Möglichkeiten, diese Aufgabe zu erledigen:

- durch das Anlegen der neuen Datensätze mit den DAO-Methoden **AddNew**/**Update** oder
- mit entsprechenden SQL-Anweisungen.

Kopieren per DAO

Die erste Variante sieht wie in Listing 1 aus und kopiert die Da-

```
Private Sub cmdRechnungKopieren_Click()
    Dim db As DAO.Database
    Dim rstRechnungen As DAO.Recordset
    Dim rstPositionenAlt As DAO.Recordset
    Dim rstPositionenNeu As DAO.Recordset
    Dim lngAlteRechnungID As Long
    Dim lngNeueRechnungID As Long
    Set db = CurrentDb
    lngAlteRechnungID = Me!RechnungID
    Set rstRechnungen = db.OpenRecordset("SELECT * FROM tblRechnungen WHERE 1=2", dbOpenDynaset)
    With rstRechnungen
        .AddNew
        !Rechnungsbetreff = Me!Rechnungsbetreff
        !Rechnungstext = Me!Rechnungstext
        !Bemerkungen = Me!Bemerkungen
        !KundeID = Me!KundeID
        !Rechnungsdatum = Me!Rechnungsdatum
        lngNeueRechnungID = !RechnungID
        .Update
    End With
    Set rstPositionenAlt = db.OpenRecordset("SELECT * FROM tblRechnungspositionen WHERE RechnungID = " & lngAlteRechnungID, dbOpenDynaset)
    Set rstPositionenNeu = db.OpenRecordset("SELECT * FROM tblRechnungspositionen WHERE 1=2", dbOpenDynaset)
    With rstPositionenAlt
        Do While Not .EOF
            rstPositionenNeu.AddNew
            rstPositionenNeu!RechnungID = lngNeueRechnungID
            rstPositionenNeu!Rechnungsposition = !Rechnungsposition
            rstPositionenNeu!Menge = !Menge
            rstPositionenNeu!Preis = !Preis
            rstPositionenNeu!Mehrwertsteuer = !Mehrwertsteuer
            rstPositionenNeu!EinheitID = !EinheitID
            rstPositionenNeu.Update
            .MoveNext
        Loop
    End With
    Me.Requery
    Me.Recordset.FindFirst "RechnungID = " & lngNeueRechnungID
End Sub
```

Listing 1: Kopieren von Rechnungsdaten aus dem Haupt- und dem Unterformular

ten mit **AddNew/Update**. Dazu ist offensichtlich eine Menge Code erforderlich, aber wir wollen ja auch gleich die Daten aus zwei verknüpften Tabellen duplizieren.

Um diese Prozedur anzulegen, klicken Sie in der Entwurfsansicht des Formulars **frmRechnungen** auf die Schaltfläche **cmdRechnungKopieren** und wählen dann im Eigenschaftsfenster für die Eigenschaft **Beim Klicken** den Eintrag **[Ereignisprozedur]** aus.

Wenn Sie dann rechts auf die Schaltfläche mit den drei Punkten (...) klicken, zeigt der VBA-Editor ein neues Klassenformular für den Formularcode mit der benötigten Prozedur an. Diese füllen Sie dann mit dem nachfolgend beschriebenen Code.

Die Prozedur speichert zunächst den Primärschlüsselwert des aktuellen Datensatzes im Formular in der Variablen **lngAlteRechnungID** und erstellt ein Recordset auf Basis der Tabelle **tblRechnungen**, wobei dieses allerdings keine Daten enthalten soll (Bedingung **WHERE 1=2**). Dieser Datensatzgruppe fügt die Prozedur mit der **AddNew**-Methode einen neuen Datensatz hinzu. Dann stellt sie die einzelnen Felder des neuen Datensatzes auf die entsprechenden Werte der Steuerelemente im Formular ein.

Nun speichert sie den Primärschlüsselwert des neuen Datensatzes (also den Wert des Feldes **RechnungID**), der eigentlich schon nach der Ausführung der **AddNew**-Methode vorliegt, in der Variablen **lngNeueRechnungID** – wir benötigen diesen Wert gleich noch für mehrere Aktionen. Schließlich speichert die Prozedur den neuen Datensatz mit der **Update**-Methode.

Nun füllt die Prozedur zwei weitere **Recordset**-Variablen mit Verweisen auf neu geöffnete Recordsets. Das erste heißt **rstPositionenAlt** und enthält alle Datensätze der Tabelle **tblRechnungspositionen**, deren Feld **RechnungID** den in **lngAlteRechnungID** gespeicherten Wert enthält.

Dies entspricht allen Datensätzen der Tabelle **tblRechnungspositionen**, die mit dem zuvor kopierten Datensatz der Tabelle **tblRechnungen** verknüpft sind. Die zweite Datensatzgruppe basiert ebenfalls auf der Tabelle **tblRechnungspositionen**, soll aber leer sein und dient lediglich dem Anlegen der Kopien der Rechnungspositionen der Originalrechnung. Daher stellen wir auch hier als Kriterium den Ausdruck **1=2** ein.

Nun durchläuft die Prozedur alle Datensätze des ersten Recordsets mit den Original-Rechnungspositionen. Sie legt für jede Rechnungsposition zunächst einen neuen Datensatz an. Wichtig ist hier die Zeile, die das Feld **RechnungID** füllt: Dieses erhält nämlich mit dem in der Variablen **lngNeueRechnungID** gespeicherten Wert den Wert des Primärschlüsselfeldes des soeben angelegten Datensatzes in der Tabelle **tblRechnungen**. Die übrigen Feldinhalt werden einfach von den entsprechenden Feldern des aktuellen Datensatzes des Recordsets **rstPositionenAlt** übernommen.

Schließlich speichert die Routine den neuen Datensatz mit der **Update**-Methode des **Recordset**-Objekts und wechselt mit der **MoveNext**-Methode zum nächsten Datensatz – und dies wird solange wiederholt, solange die Abbruchbedingung in der **Do While**-Schleife (**rstPositionenAlt.EOF**) nicht erfüllt ist.

Schließlich zeigt die Prozedur den neu angelegten Rechnungsdatensatz im Hauptformular an, wobei erneut der in **lngRechnungID** gespeicherte Primärschlüsselwert des neuen Datensatzes zum Einsatz kommt.

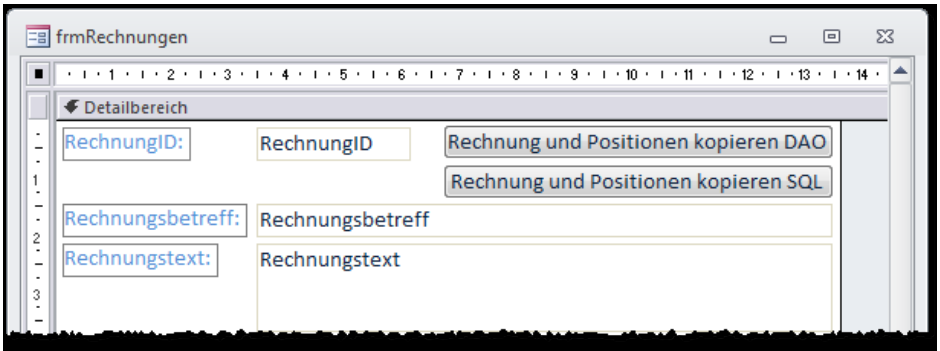


Bild 5: Neue Schaltfläche zum Kopieren per SQL-Anweisung

Verknüpfte Daten per SQL kopieren

Wir wollen uns noch eine zweite Variante zum Kopieren der Datensätze ansehen. Diese basiert auf der Ver-

Programmieren mit Makros, Teil I

Makros sind eine Möglichkeit, einfache Vorgänge unter Access zu automatisieren. Sie sind zwar erheblich unflexibler als etwa VBA-Code, haben aber dennoch ihre Berechtigung – zum Beispiel in Form zweier Spezialmakros namens **AutoExec** und **AutoKeys**, die beim Start beziehungsweise beim Ausführen von Tastenkombination aufgerufen werden können. Da Microsoft mit Access 2010 die Webdatenbanken eingeführt hat, die ausschließlich mit Makros programmiert werden können, wollen wir an dieser Stelle eine Einführung in die Programmierung mit Makros bieten.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1402_Makros.mdb**.

Makros und VBA

Makros und VBA eignen sich beide zur Programmierung von Ereignissen in Access. In Abgrenzung zu anderen Office-Anwendungen wie Word, Excel und Co. müssen wir hier jedoch eine kurze Begriffsklärung vornehmen: Bei den übrigen Office-Anwendungen spricht man in Zusammenhang mit einer VBA-Prozedur von einem Makro. Dort lassen sich fast alle Aktionen als Makro aufzeichnen, was aber bedeutet, dass die Office-Anwendung eine VBA-Prozedur erstellt, welche die in der Anwendung durchgeführten Tätigkeiten in Form von VBA-Anweisungen enthält.

Ein solches Word- oder Excel-Makro ist nicht mit einem Makro in Access zu vergleichen. Unter Access spricht man unter VBA zum Beispiel von Prozeduren oder Funktionen, gegebenenfalls auch von Routinen, aber niemals von Makros.

Ein Makro ist unter Access ein eigener Objekttyp, dessen Vertreter im Navigationsbereich eine eigene Kategorie besitzen. Die Anweisungen eines Makros werden nicht in Form von Codezeilen eingegeben, sondern als Makrobefehle eingegeben beziehungsweise ausgewählt.

Spezialmakros

Die einzigen beiden Fälle, in denen wir bislang auf die Programmierung von Makros eingegangen sind, finden Sie in den Artikeln **Aktion beim Start: Das AutoExec-Makro** und **AutoKeys: Globale Tastenkombinationen**. Das **AutoExec**-Makro wird direkt beim Start von Access ausgeführt.

Sie können damit also Aktionen auslösen, die in Zusammenhang mit dem Start ausgeführt werden sollen, also etwa das Neuverknüpfen mit einer Backend-Datenbank et cetera. Mit dem **AutoKeys**-Makro definieren Sie Tastenkombinationen, die global, also anwendungsweit, gültig sind.

Wenn Sie also etwa verhindern möchten, dass der Benutzer mit der Tastenkombination **Strg + G** das Direktfenster öffnet, können Sie diese Tastenkombination im **AutoKeys**-Makro entsprechend umbiegen. Im Gegensatz zu den Tastenkombinationen, die Sie in Formularen mit Ereignissen wie **Bei Taste ab** oder **Bei Taste auf** abfangen können, werden die mit dem **AutoKeys**-Makro definierten Tastenkombinationen zu jedem Zeitpunkt ausgelöst und nicht nur dann, wenn ein bestimmtes Formular den Fokus besitzt.

Makros ab Access 2007

Mit Access 2007 hat Microsoft begonnen, die Möglichkeiten der Makros aufzuwerten. Wenn Sie bereits früher mit Makros programmiert haben, sind die folgenden Hinweise für Sie interessant – aber auch für alle anderen:

- Makros können nicht nur gespeichert und als Wert von Ereigniseigenschaften angegeben werden. Sie können Makros auch in die Ereignisse von Formularen, Berichten und Steuerelementen einbetten. Das Makro wird dann nicht mehr als eigener Eintrag in der Kategorie Makros des Navigationsbereichs angezeigt, sondern direkt im Formular oder im Bericht gespeichert.

- Der VBA-Code und bestimmte Makrobefehle werden in Access-Datenbanken nicht ausgeführt, wenn diese nicht als vertrauenswürdig eingestuft sind. Sie können im Makro-Editor nun angeben, dass nur solche Makros angezeigt werden, die auch bei nicht vertrauenswürdigen Datenbanken ausgeführt werden können.

- Mit Access 2007 gibt es erstmals eine Fehlerbehandlung und die Möglichkeit, Makros zu debuggen. Makroaktionen wie **BeiFehler** oder **ClearMacroError** helfen dabei ebenso wie **MakroEinzelschritt**, mit der Sie innerhalb des Makros (ähnlich einer Haltemarke) in den Einzelschrittmodus wechseln können.

- Bis Access 2003 konnten Makros weder auf Variablen zugreifen noch solche mit Werten füllen. Mit Access 2007 wurden die sogenannten **Temporären Variablen** eingeführt. Diesen können Sie Werte zuweisen und diese auch wieder abfragen. Interessant ist, dass es auch in VBA ein entsprechendes Objekt gibt, sodass ein Austausch zwischen VBA-Routinen und Makros erfolgen kann.

Makros in Access 2010

Mit Access 2010 geht der Reigen der Neuerungen im Bereich der Makros weiter:

- Die wichtigste Neuerung ist der neue Makro-Editor. Er liefert eine wesentlich bessere Benutzeroberfläche zum Erstellen von Makros, hier vor allem bei der Umsetzung von Bedingungen und Verschachtelungen.

- Makroaktionen können Sie direkt aus der Liste im Bereich der zum Makro hinzugefügten Befehle auswählen (siehe Bild 1).

- Es gibt aber auch einen übersichtlichen Katalog, in dem die Makros nach Kategorien sortiert sind (siehe Bild 2).

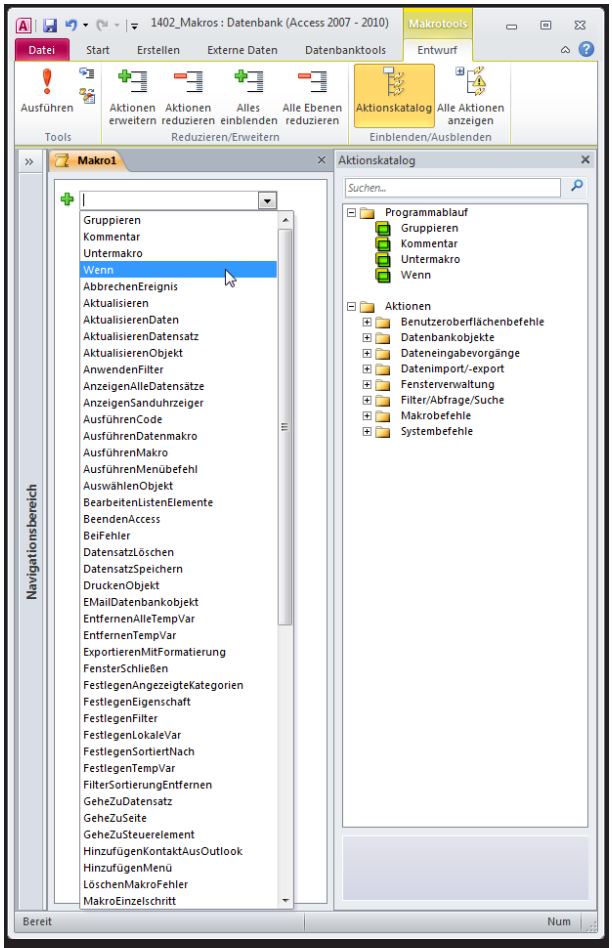


Bild 1: Auswählen einer Makroaktion direkt im Makro-Editor ...

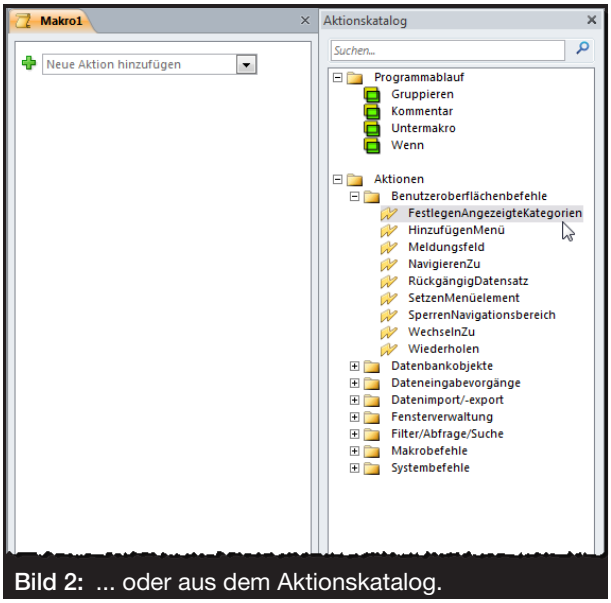


Bild 2: ... oder aus dem Aktionskatalog.

- Zusätzlich können Sie in Makros nun auch IntelliSense nutzen. Damit greifen Sie beispielsweise auf alle eingebauten Access-Funktionen zu, aber auch auf benutzerdefinierte VBA-Funktionen zu. Außerdem stehen die Objektnamen etwa von Tabellen, Abfrage, Formularen et cetera zur Auswahl (siehe Bild 3).

- Es gibt eine Wenn-Bedingung (mit Sonst- und Sonst Wenn-Zweigen), die auch verschachtelt angeordnet werden können.

- Sie können bereits erstellte Makros ganz einfach als Untermakros zu weiteren Makros hinzufügen. Diese werden im Aktionskatalog in der Kategorie **In diesen Datenbank|Makros** eingeblendet.

- Sie können Teile des Makrocodes kopieren und an anderer Stelle wieder einfügen. Das dabei verwendete Format ist XML. Sie können ein Makro also auch

kopieren und per E-Mail an einen anderen Entwickler versenden.

- Eine weitere wichtige Neuerung sind die Datenmakros, die ähnlich etwa wie die Trigger im SQL Server arbeiten. Diese Makros werden Tabellen zugeordnet und bei bestimmten Datenaktionen ausgelöst – wie etwa das Anlegen eines neuen oder das Ändern oder Löschen eines bestehenden Datensatzes.

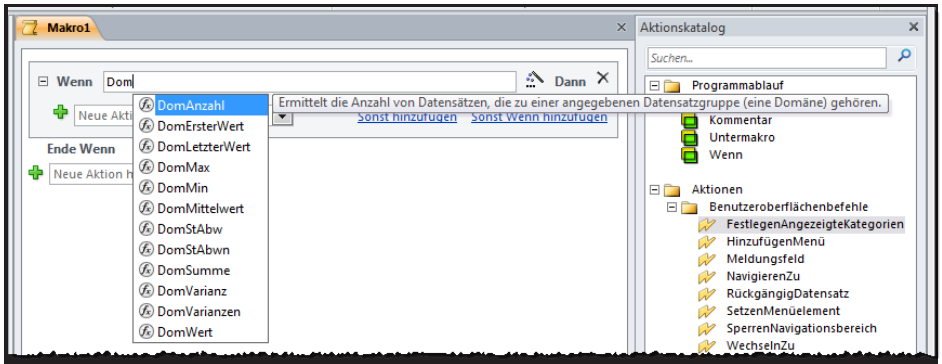


Bild 3: IntelliSense im Makro-Editor von Access 2010

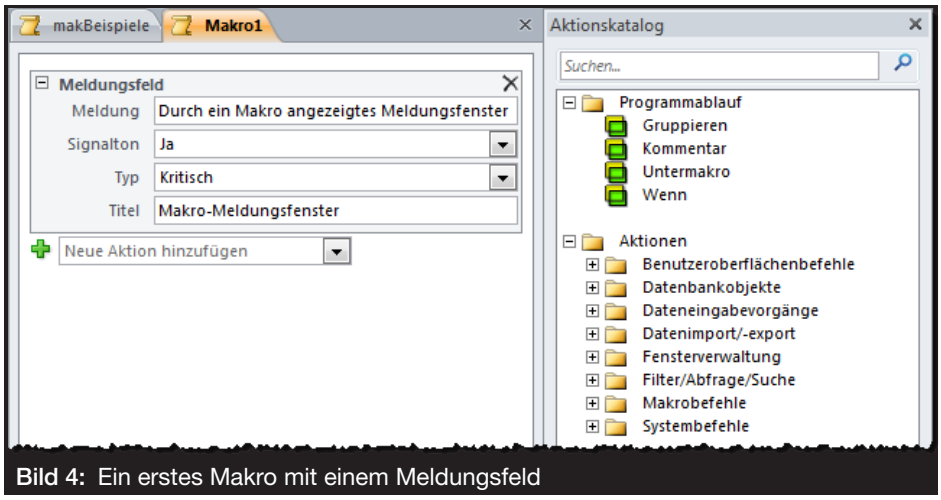


Bild 4: Ein erstes Makro mit einem Meldungsfeld

Makros in Access 2013

Unter Access 2013 gab es keine nennenswerten Neuerungen. Anscheinend sind sogar drei Makrobefehle weggefallen (mehr zu den verschiedenen Makro-Befehlen weiter unten):

- **SammelnDatenÜberEmail**
- **StartenNeuerWorkflow**
- **WorkflowAufgaben**

Verschiedene Makro-Arten

Unter Access gibt es die folgenden Makro-Arten:

- normale Makros, die über den Menübefehl **Erstellen|Makros und Code|Makro** angelegt und unter einem von Ihnen vergebenen Namen gespeichert und im Navigationsbereich angezeigt werden
- reine Ereignis-Makros, die als Makro für ein Ereignis eines Formulars, Berichts oder Steuerelements angelegt werden. Diese erscheinen nicht im Navigationsbereich.
- Datenmakros sind Makros, die durch Ereignisse in Tabellen ausgelöst werden, also beim Anfügen, Löschen oder Ändern von Datensätzen.

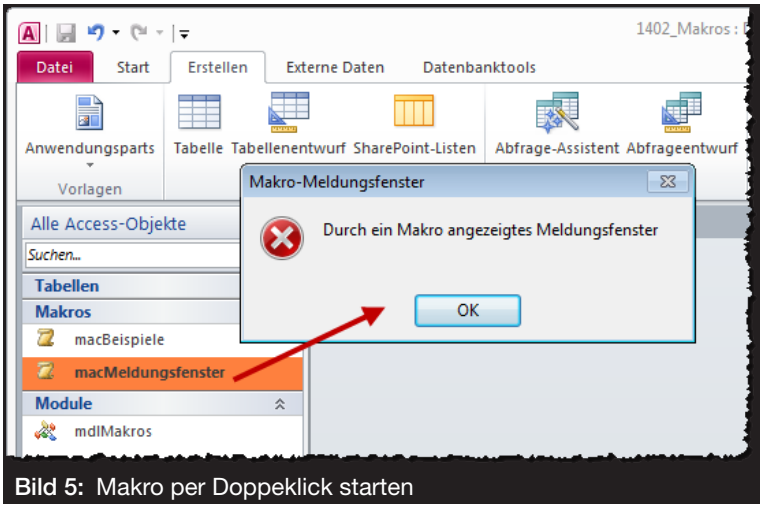


Bild 5: Makro per Doppeklick starten

- benannte Makros sind Makros, die Sie im Kontext einer bestimmten Tabelle anlegen und die entweder durch ein Datenmakro oder durch ein herkömmliches Makro ausgelöst werden können.

Makro erstellen

Die folgenden Beispiele wurden unter Access 2010 durchgeführt.

Wir möchten erstens die neuen Möglichkeiten untersuchen und zweitens die aktuelle Benutzeroberfläche zur Erstellung von Makros nutzen. Für einen ersten Überblick der Benutzeroberfläche erstellen Sie zunächst mit dem Ribbon-Eintrag **Erstellen|Makros und Code|Makro** ein neues Makro. Fügen Sie diesem nun per Auswahl über das Kombinationsfeld **Neue Aktion hinzufügen** einfach den Befehl **Meldungsfeld** hinzu.

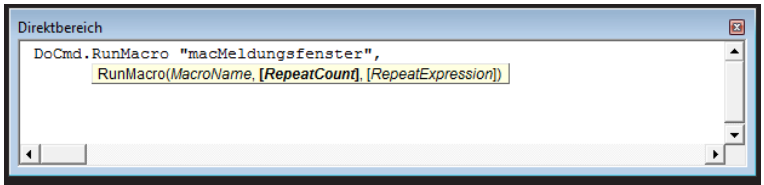


Bild 6: Aufruf eines Makros im Direktbereich des VBA-Editors

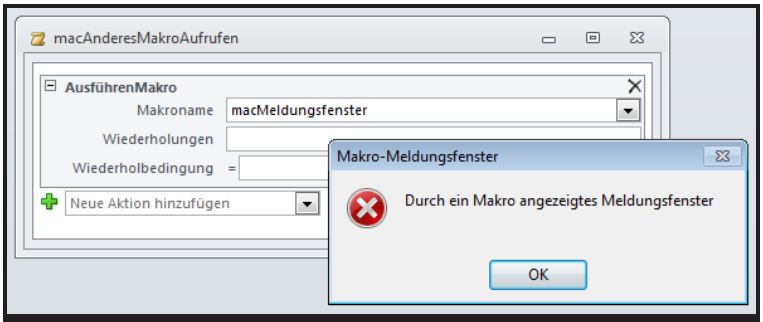


Bild 7: Aufrufen eines anderen Makros von einem Makro aus

Access zeigt nun direkt an Ort und Stelle die vier Eigenschaften für die **Meldungsfeld**-Makroaktion an (siehe Bild 4): **Meldung, Signalton**, **Typ** und **Titel**.

Die Eigenschaften sollten von der VBA-Funktion **MsgBox** bekannt sein, daher gehen wir an dieser Stelle nicht näher auf die Optionen ein.

Speichern Sie das Makro unter dem Namen **macMeldungsfeld**.

Makro aufrufen

Die einfachste Möglichkeit, ein Makro aufzurufen, ist ein Doppelklick auf den Namen des Makros im Navigationsbereich von Access (siehe Bild 5).

Natürlich können Sie ein Makro auch bereits beim Erstellen durch einen Mausklick auf den Ribbon-Befehl **Entwurf|Tools|Ausführen** aufrufen.

```
Private Sub Form_Load()
    Dim cbr As CommandBar
    Dim cbc As CommandBarControl
    On Error Resume Next
    CommandBars("cbrMakro").Delete
    On Error GoTo 0
    Set cbr = CommandBars.Add("cbrMakro", msoBarPopup, False, True)
    Set cbc = cbr.Controls.Add(msoControlButton)
    With cbc
        .Caption = "Makro ausführen"
        .OnAction = "macMeldungsfeld"
    End With
    cbr.ShowPopup
End Sub
```

Listing 1: Diese Prozedur zeigt gleich beim Laden eines Formulars ein Kontextmenü mit einem Befehl an, der ein Makro aufruft.

Makro per VBA aufrufen

Eine weitere Möglichkeit für den Aufruf eines Makros ist die Methode **RunMacro** des **DoCmd**-Objekts. Den folgenden Befehl können Sie beispielsweise im Direktbereich des VBA-Editors absetzen, aber auch in Funktionen oder Prozeduren integrieren:

```
DoCmd.RunMacro "<Makroname>"
```

Der Beispielaufruf aus Bild 6 zeigt, dass die **RunMacro**-Methode neben dem Makronamen noch zwei weitere Parameter anbietet:

- **RepeatCount**: Gibt an, wie oft das Makro ausgeführt werden soll.
- **RepeatExpression**: Enthält einen Ausdruck, der bei jeder der in **RepeatCount** angegebenen Ausführungen ausgewertet wird. Wenn der Ausdruck den Wert False enthält, wird die wiederholte Ausführung abgebrochen.

Sie können ein Makro auch von einem anderen Makro aus aufrufen. Wenn Sie also etwa unser Beispielmakro **macMeldungsfeld** von einem anderen Makro etwa namens **macAnderesMakroStarten** aus starten möchten, legen Sie dieses Makro an und fügen diesem die Makroaktion **AusführenMakro** mit dem Namen des auszuführenden Makros als Parameter hinzu (siehe Bild 7).

Makro per Kontextmenü aufrufen

Wenn Sie in Ihrer Anwendung Kontextmenüs verwenden, können Sie über die dort enthaltenen Einträge nicht nur VBA-Funktionen, sondern auch Makros aufrufen.

In den neueren Access-Versionen, auf die wir uns hier beziehen, legen Sie Kontextmenüs per VBA an (siehe Artikel **Kontextmenüs**). Dort geben Sie die aufzurufende Prozedur mit der Eigenschaft **OnAction** des jeweiligen Steuerelements an. Dies sieht dann etwa wie in Listing 1 aus – siehe das Formular **frmMakroPerKontextmenue** in der Beispieldatenbank. Die Prozedur benötigt einen Verweis auf die Bibliothek **Microsoft Office x.0 Object Library** und zeigt gleich beim Öffnen des Formulars das entsprechende Kontextmenü mit dem Eintrag zum Auslösen des Makros **macMeldungsfeld** an.

Makros per Ereignis aufrufen

Makros können in Desktop-Datenbanken als Alternative für VBA-Ereignisprozeduren verwendet werden. In den sogenannten Webdatenbanken können Sie ausschließlich Makros zur Implementierung von Ereignissen verwenden.

Wenn Sie ein Makro durch ein Ereignis eines Formulars, Berichts oder Steuerelements auslösen möchten, haben Sie zwei Möglichkeiten:

- Sie legen das Makro als eigenes Makro an (etwa mit dem Ribbon-Eintrag **Erstellen|Makros und Code|Makro**) und speichern es unter dem gewünschten Namen. Dann wählen Sie den Namen des Makros als Wert der entsprechenden Ereigniseigenschaft aus.
- Sie klicken auf die Schaltfläche rechts neben der Ereigniseigenschaft und wählen aus dem nun erscheinenden Dialog den Eintrag Makro-Generator aus (siehe Bild 8). Es erscheint nun der Makro-Editor, mit dem Sie das Makro programmieren können.

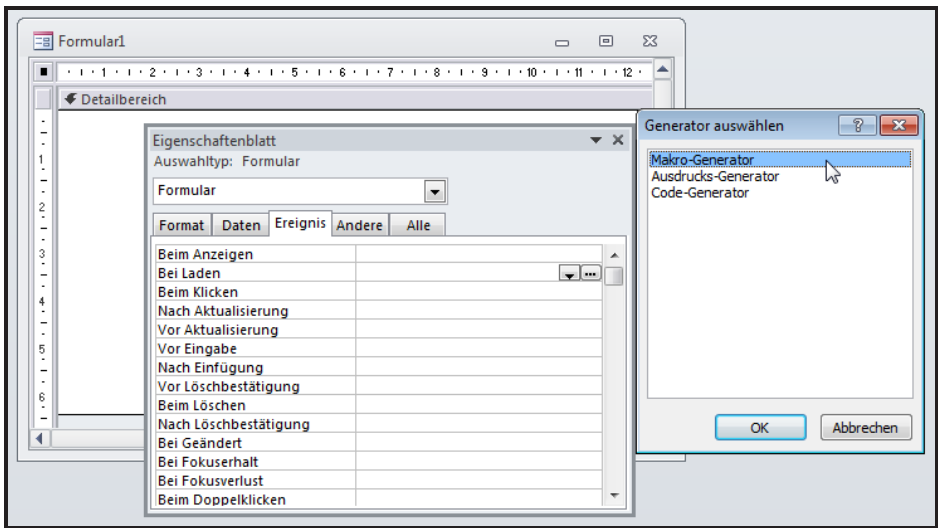


Bild 8: Makro für ein Ereignis festlegen

In letzterem Fall wird das Makro nicht unter einem von Ihnen vergebenen Namen gespeichert, sondern direkt mit dem Formular oder Bericht. Diese sogenannten eingebetteten Makros gab es in dieser Form zuvor nicht.

Mit den neueren Access-Versionen können Sie außerdem auch Makros für Steuerelemente festlegen, die quasi in die Steuerelemente eingebettet werden – das bedeutet, dass Sie ein Steuerelement von einem Formular zum nächsten kopieren können und das Makro gleich mit kopiert wird.

Datenmakros aufrufen

Datenmakros können Sie nicht „aufrufen", sondern lediglich auslösen – und zwar durch das entsprechende Ereignis einer Tabelle (Anfügen, Bearbeiten oder Löschen von Daten). Die einzelnen Ereignisse schauen wir uns später an.

Benannte Makros aufrufen

Benannte Makros, also solche Makros, die Sie im Kontext einer Tabelle erstellt haben, rufen Sie mit der Makroaktion **AusführenDatenmakro** auf.

Mit dem Makro-Editor arbeiten

Der Makro-Editor von Access 2010 und jünger bietet gegenüber dem Vorgänger eine Reihe neuer Features an (dieser Hinweis für alle, die bereits mit älteren Access-Versionen gearbeitet haben).

Wenn Sie einmal – wie weiter oben beschrieben – ein neues Makro angelegt haben, können Sie gleich mit dem Hinzufügen von Makroaktionen beginnen. Zunächst ist es hilfreich, die dabei zur Verfügung stehenden

Ribbon-Elemente kennenzulernen. Diese sehen etwa unter Access 2010 wie in Bild 9 aus.

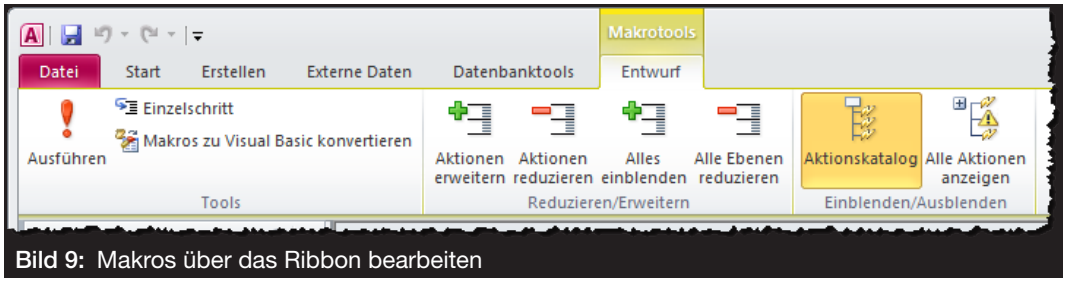


Bild 9: Makros über das Ribbon bearbeiten

Wenn Sie das erste Mal mit dem neuen Makro-Editor arbeiten, stellen Sie sich möglicherweise die eine oder andere Frage – zum Beispiel, wie man eine Makroaktion ganz vorne im Makro einfügt.

Solche Fragen beantwortet das Makro-Ribbon nicht. Es stellt lediglich den einen oder anderen Befehl zur Verfügung, der so nicht im Makro-Editor selbst zu finden ist. So bietet das Makro-Ribbon etwa die folgenden Befehle:

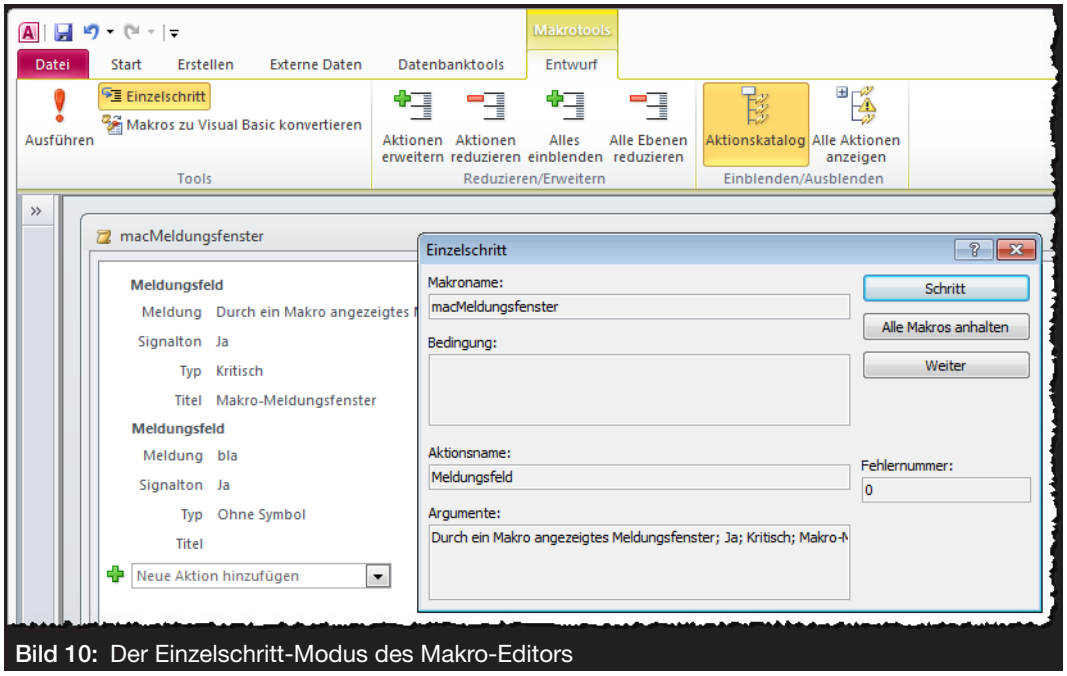


Bild 10: Der Einzelschritt-Modus des Makro-Editors

- **Ausführen**: Die schnellste Möglichkeit, um das aktuelle Makro zu starten. Achtung: Sie müssen ein Makro speichern, bevor Sie es ausführen können.

- **Einzelschritt**: Mit dieser Schaltfläche aktivieren oder deaktivieren Sie den Einzelschrittmodus beim Ausführen von Makros. Ist die Option aktiviert, zeigt Access vor der Ausführung eines jeden Befehls den Dialog **Einzelschritt** an, der Informationen über den nächsten auszuführenden Befehl ausgibt (siehe Bild 10).

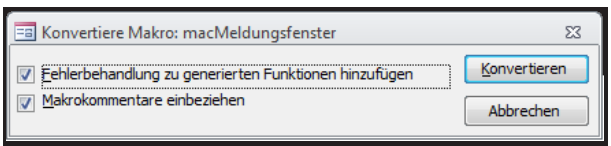


Bild 11: Optionen für die Konvertierung von Makros zu VBA-Prozeduren

- **Zu Visual Basic konvertieren**: Dieser Befehl startet die Konvertierung von Makros zu Visual Basic-Prozeduren (mehr dazu weiter unten).

- **Aktionen erweitern**: Erweitert alle markierten Aktionen (sind keine markiert, werden alle Aktionen erweitert).

- **Aktionen reduzieren**: Zeigt für alle markierten Aktionen nur noch die erste Zeile und das Plus-Zeichen zum Erweitern an. Sind keine Aktionen markiert, werden alle Aktionen reduziert dargestellt.

- **Alles einblenden**: Erweitert alle Aktionen.

- **Alle Ebenen reduzieren**: Stellt alle Ebenen nur noch in Form der ersten Zeile dar.

- **Aktionskatalog**: Blendet den Aktionskatalog ein und aus.

- **Alle Aktionen anzeigen**: Zeigt, wenn gedrückt, alle Aktionen im Aktionskatalog an. Ist **Alle**

Aktionen anzeigen nicht aktiviert, zeigt der Aktionskatalog nur die Aktionen an, die im aktuellen Kontext zulässig sind.

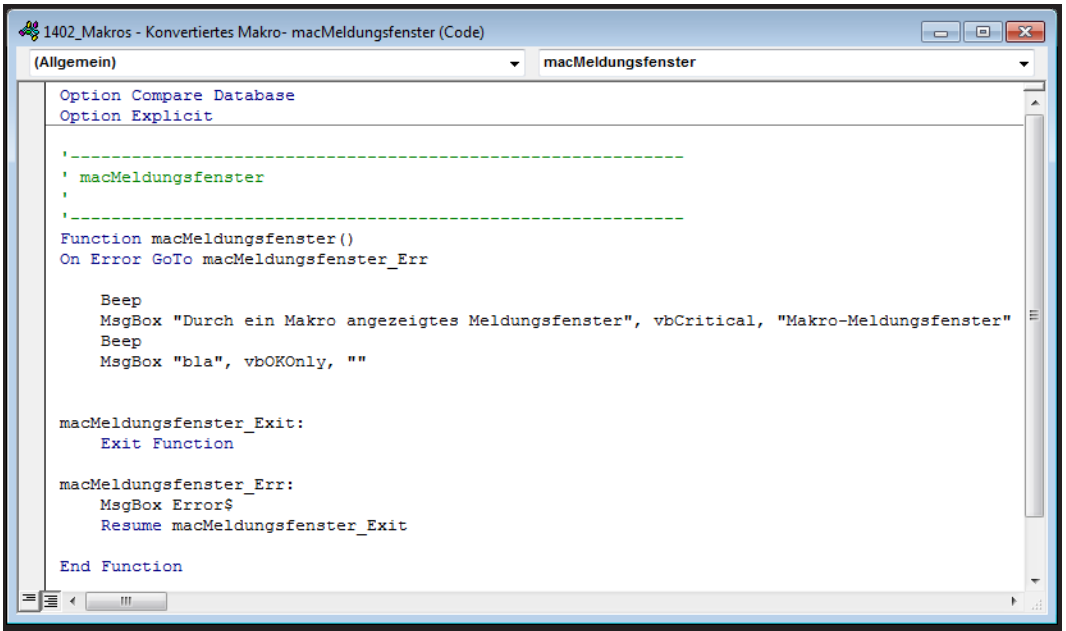


Bild 12: Ein nach VBA konvertiertes Makro

Makros konvertieren

Wenn Sie beispielsweise eine Datenbank bearbeiten sollen, indem Sie die vorhandenen Makros nach VBA konvertieren, können Sie dies zumindest teilweise automatisch erledigen lassen. Den dazu notwendigen Befehl **Zu Visual Basic konvertieren** haben Sie bereits weiter oben kennengelernt.

Wenn Sie diesen anklicken, liefert Access zunächst den Dialog aus Bild 11. Hier legen Sie fest, ob Sie eine Fehlerbehandlung zu der konvertierten VBA-Prozedur hinzufügen möchten oder ob die im Makro enthaltenen Kommentare im Code untergebracht werden sollen. Access legt für jedes konvertierte Makro ein neues VBA-Modul an, dessen Name aus Konvertiertes Makro- und dem Namen des Makros besteht, also beispielsweise **Konvertiertes Makro-macMeldungsfeld**.

Der Code sieht mit Fehlerbehandlung und Kommentaren beispielsweise wie in Bild 12 aus.

Makros eingeben

Die Eingabe von Makroaktionen ist einfach. Die erste Aktion können Sie direkt durch Auswahl im Kombinationsfeld eingeben, die Parameter fügen Sie über die dann erscheinenden Eingabefelder hinzu. Interessanter wird es, wenn Sie bereits einige Makroaktionen hintereinander eingegeben haben und dann weitere hinzufügen wollen, die nicht an der letzten Stelle eingefügt werden sollen. Wenn Sie etwa einen Makrobefehl an der ersten Position hinzufügen möchten oder zwischen zwei bestehenden Makroaktionen einfügen möchten, kommen Sie mit dem Kombinationsfeld nur auf Umwegen weiter. Sie können den neuen Befehl dann in der untersten Zeile eingeben und diesen dann per Drag and Drop an die gewünschte Position ziehen.

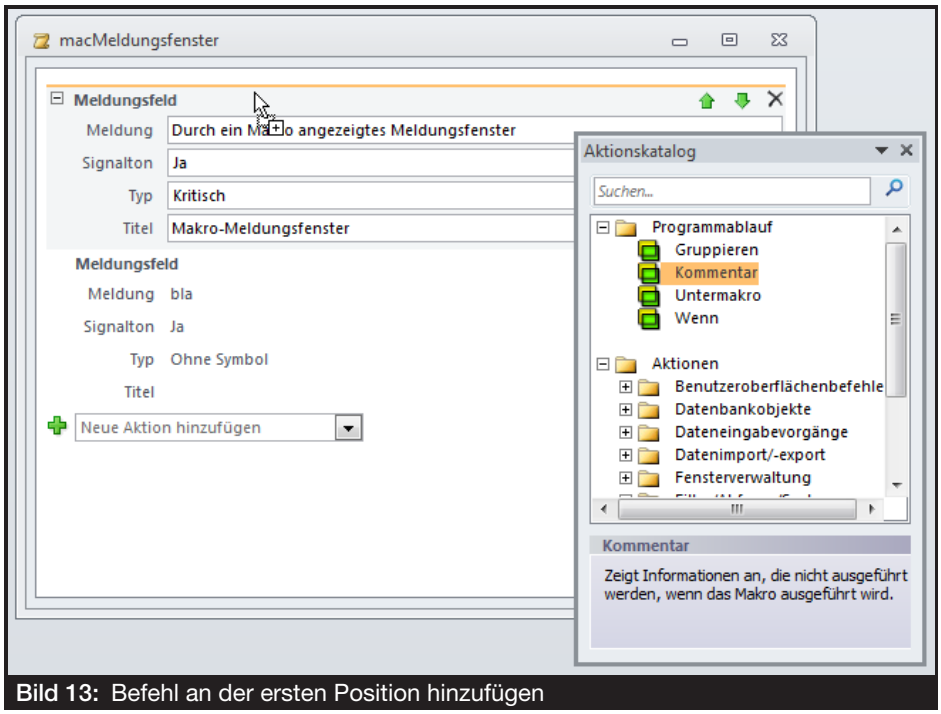


Bild 13: Befehl an der ersten Position hinzufügen

Wenn Sie eine Makroaktion direkt an der ersten Position oder zwischen zwei vorhandenen Makroaktionen einfügen möchten, benötigen Sie den Aktionskatalog. Von dort können Sie den einzufügenden Befehl per Drag and Drop an die gewünschte Position ziehen. Im Beispiel aus Bild 13 wird ein Kommentar aus dem Aktionskatalog gleich an die erste Position des Makros gezogen.

Wenn Sie mit einem der verschiedenen Blöcke wie etwa dem Wenn-Block arbeiten, gibt es verschiedene Techniken:

- Wenn Sie einen komplett leeren Block anlegen möchten, ziehen Sie diesen einfach aus dem Aktionskatalog an die gewünschte Stelle.

- Wenn das Makro bereits ein oder mehrere Makroaktionen enthält, die Sie mit einem Block wie einer Wenn-Bedingung einfassen möchten, markieren Sie die betroffenen Befehle und wählen dann aus dem Kontextmenü den zu verwendenden Block aus (zum Beispiel **If-Block erstellen**) – siehe Bild 14.

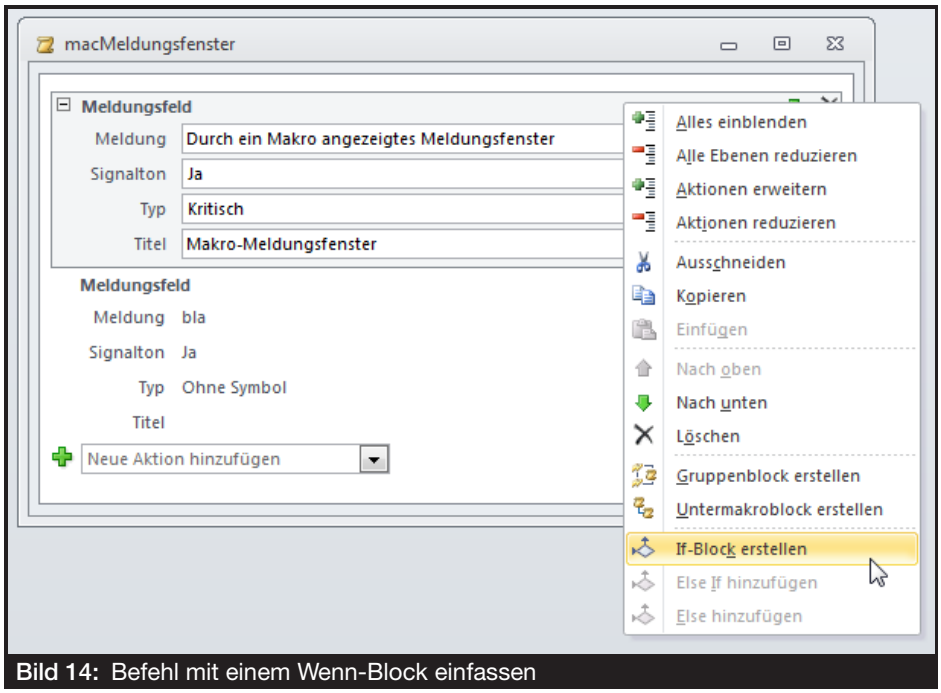


Bild 14: Befehl mit einem Wenn-Block einfassen

- Gegebenenfalls möchten Sie einen Block entfernen, aber nicht die enthaltenen Befehle. In diesem Fall markieren Sie alle enthaltenen Befehle, ziehen diese per Drag and Drop aus dem Block heraus und positionieren ihn entweder vor oder hinter dem Block (siehe Bild 15). Dann können Sie den Block löschen.

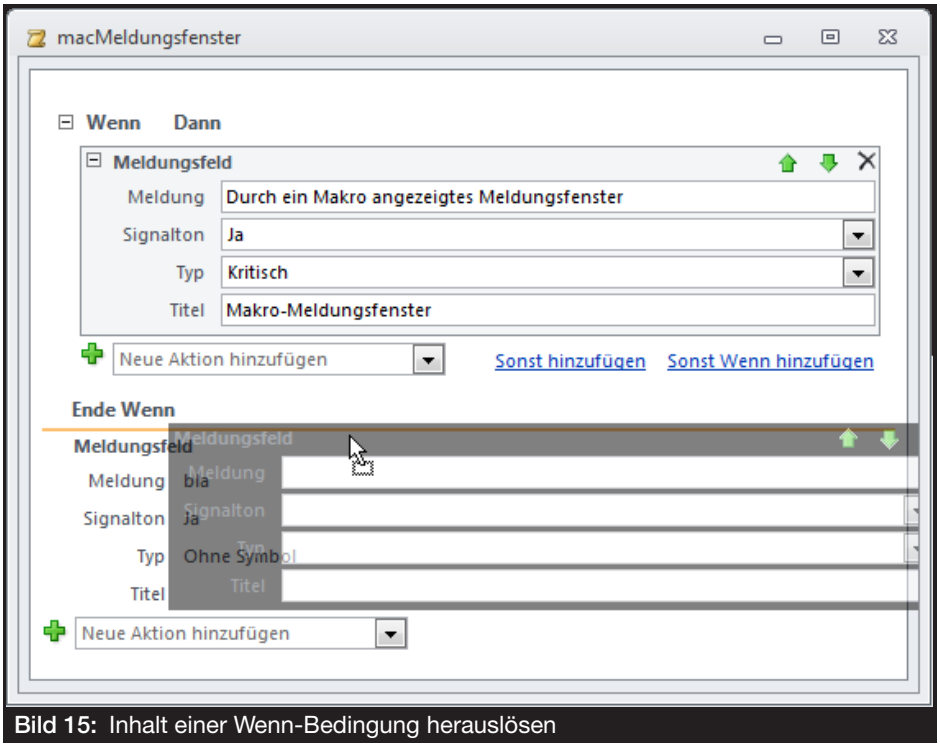


Bild 15: Inhalt einer Wenn-Bedingung herauslösen

Drag and Drop

Dem Ziehen und Fallenlassen von Elementen sind übrigens Grenzen gesetzt.

Während Sie einzelne Makroaktionen (oder auch zusammenängende einzelne Makroaktionen) oder auch Gruppierungen (siehe unten) problemlos überall hinziehen können, gelingt dies etwa mit Untermakros nicht: Diese lassen sich nicht an die

erste oder letzte Position oder an die Position zwischen zwei anderen Makroaktionen verschieben.

Reihenfolge ändern

Wenn Sie einmal die Reihenfolge zweier benachbarter Makroaktionen vertauschen möchten, können Sie dies ebenfalls per Drag and Drop erledigen. Sie haben allerdings auch die Möglichkeit, dies per Tastatur zu erledigen. Dazu markieren Sie einfach den zu verschiebenden Block und betätigen dann die Tastenkombination **Strg + Nach oben** oder **Strg + Nach unten** – je nachdem, in welche Richtung Sie das markierte Element bewegen möchten.

Zwischenablage

Hilfreich bei der Arbeit mit Makros ist auch die Zwischenablage. Sie können einzelne Makroaktionen, komplette Blöcke oder auch zusammenhängende Elemente markieren, kopieren (zum Beispiel mit der Tastenkombination **Strg + C**), ausschneiden (**Strg + X**) und an beliebiger Stelle wieder einfügen (**Strg + V**).

Was aber landet denn dort überhaupt in der Zwischenablage? Dies erfahren Sie, wenn Sie den Inhalt etwa in einen Texteditor einfügen (siehe Bild 16). Makros sind nämlich ab Access 2010 nichts anderes als mit XML definierte Anweisungen. Dies liegt auf der Hand: Sowohl die Struktur als auch die Eigenschaften und Parameter der Makro-Aktionen lassen sich per XML einfach abbilden.

```
Dokument1 *
<?xml version="1.0" encoding="UTF-16" standalone="no"?>
<UserInterfaceMacros xmlns="http://schemas.microsoft.com/office/accessservices/2009/11/application">
        <UserInterfaceMacro MinimumClientDesignVersion="14.0.0000.0000">
                <Statements>
                        <Action Name="MessageBox">
                                <Argument Name="Message">Durch ein Makro angezeigtes Meldungsfenster</Argument>
                                <Argument Name="Type">Critical</Argument>
                                <Argument Name="Title">Makro-Meldungsfenster</Argument>
                        </Action>
                </Statements>
        </UserInterfaceMacro>
</UserInterfaceMacros>
```

Bild 16: Inhalt der Zwischenablage nach dem Kopieren einer Makroaktion

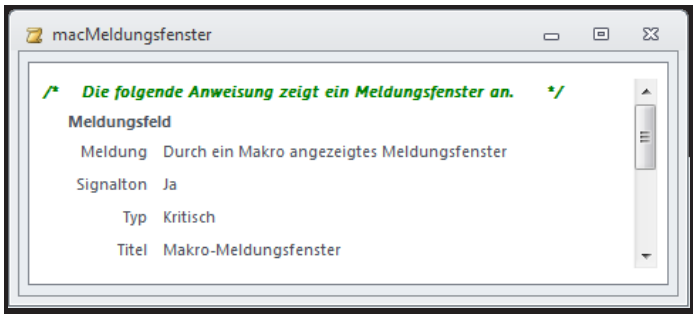


Bild 17: Kommentarzeile in einem Makro

Über die Zwischenablage können Sie auch Makro-Aktionen oder komplette Makros von einer Anwendung zu nächsten kopieren.

Wenn Sie ein oder mehrere Makroaktionen per Maus kopieren möchten, gehen Sie folgendermaßen vor:

- Markieren Sie die zu kopierenden Elemente.
- Ziehen Sie den markierten Bereich bei gedrückter **Strg**-Taste an die gewünschte Stelle. Fertig!

Kommentare in Makros

Seit Access 2010 und somit mit der neuen Benutzeroberfläche für die Programmierung von Makros können Sie Makros auch Kommentarzeilen hinzufügen. Dazu wählen Sie aus dem Kombinationsfeld zum Hinzufügen eines neuen Blocks den Eintrag Kommentar aus und geben den gewünschten Kommentar als Wert des einzigen Parameters dieser „Makroaktion" ein.

Das Ergebnis sieht wie in Bild 17 aus. Interessanterweise wird hier nicht das unter VBA als Kommentarzeichen verwendete Hochkomma verwendet, sondern das etwa von C# verwendete.

Makros gruppieren

Der Aktionskatalog des Makro-Editors von Access 2010 und neuer zeigt im oberen Bereich eine Kategorie namens Programmablauf an. Diese enthält als Erstes den Befehl **Gruppieren**. Damit können Sie ein oder mehrere Makroaktionen zu einer Gruppe zusammenfassen.

Um eine solche Gruppierung zu erstellen, haben Sie zwei Möglichkeiten:

- Sie ziehen das Gruppieren-Element aus dem Aktionskatalog in das Makro und fügen der Gruppierung dann die gewünschten Befehle hinzu (entweder durch Neuanlegen, durch Copy and

Paste oder auch per Drag and Drop).

- Sie markieren die zu gruppierenden Elemente und wählen dann aus dem Kontextmenü den Eintrag **Gruppenblock erstellen** aus.

Beides führt dazu, dass die betroffenen Makroaktionen in ein neues Element eingeschlossen werden, für das Sie einen Gruppennamen vergeben können (siehe Bild 18).

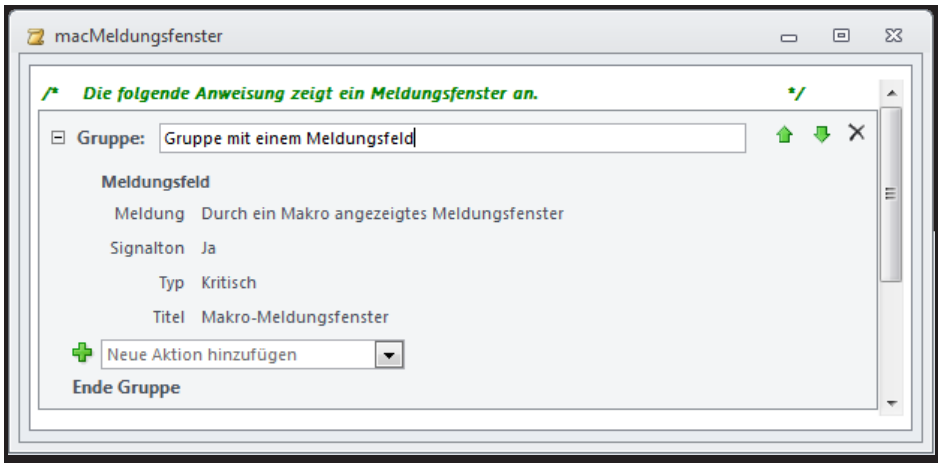


Bild 18: Gruppierung im Makro

Die Gruppierung hat folgenden Zweck:

- Sie geben zusammenhängenden Makroaktionen einen Namen.
- Sie können die gruppierten Makroaktionen nun en bloc ein- und ausblenden (siehe Bild 19).

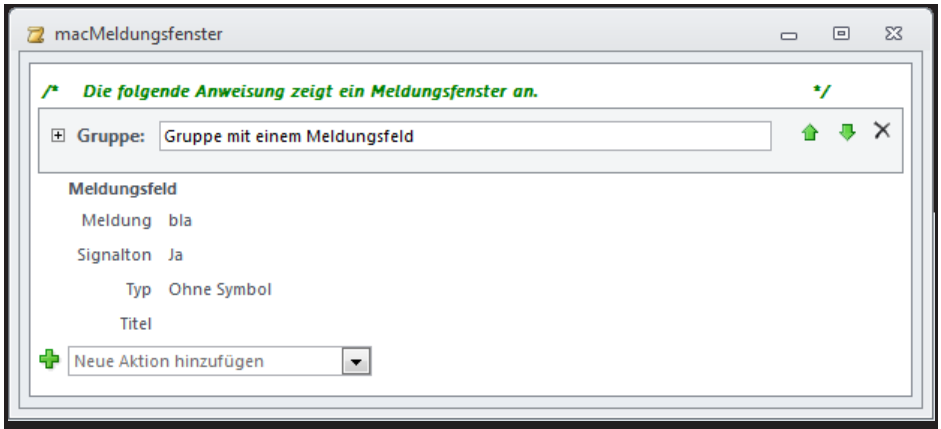


Bild 19: Minimierte Gruppierung

Sie können übrigens auch verschachtelte Gruppierungen definieren.

Untermakros

Wenn Sie mehrere inhaltlich zusammenhängende Makros erstellen, können Sie diese auch als Untermakros in einem einzigen Makro erstellen.

Das ist zum Beispiel sinnvoll, wenn Sie verschiedene Makros, die alle Datenoperationen bezogen auf eine bestimmte Tabelle enthalten, zusammenfassen wollen.

Bild 20 zeigt ein Makro mit zwei Untermakros, die jeweils die Makroaktion Meldungsfeld mit verschiedenen Meldungen enthalten. Nun gehen wir zunächst davon aus, dass Sie

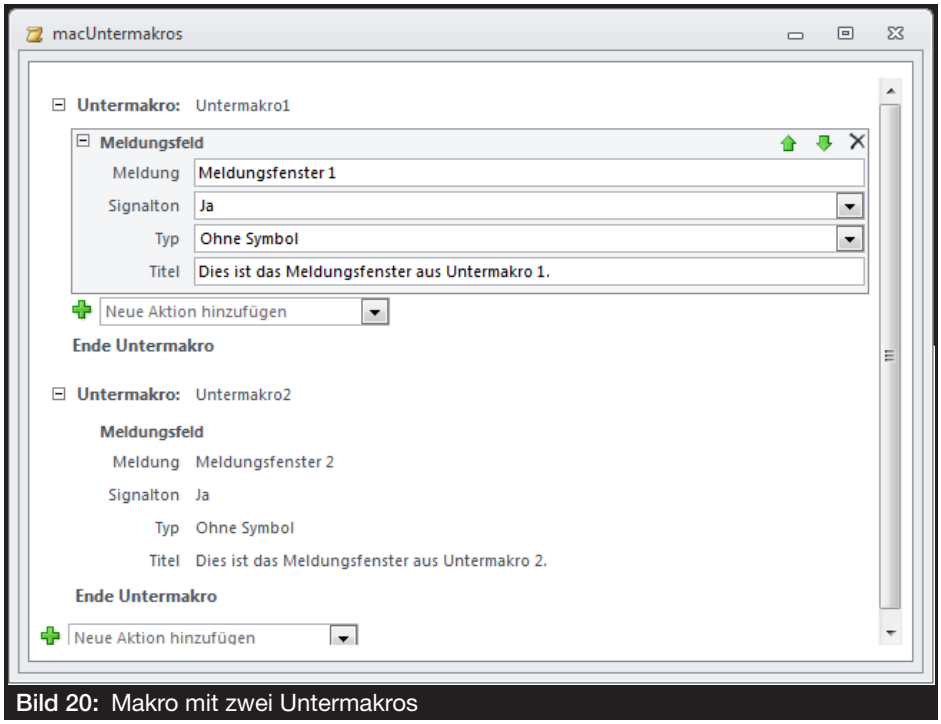


Bild 20: Makro mit zwei Untermakros

Programmieren mit Makros, Teil II

Makros sind eine Möglichkeit, einfache Vorgänge unter Access zu automatisieren. Sie sind zwar erheblich unflexibler als etwa VBA-Code, haben aber dennoch ihre Berechtigung – zum Beispiel in Form zweier Spezialmakros namens **AutoExec** und **AutoKeys**, die beim Start beziehungsweise beim Ausführen von Tastenkombination aufgerufen werden können. Da Microsoft mit Access 2010 die Webdatenbanken eingeführt hat, die ausschließlich mit Makros programmiert werden können, wollen wir an dieser Stelle eine Einführung in die Programmierung mit Makros bieten.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1403_Makros.mdb**.

Steuerelemente und Co.

Mit **FestlegenEigenschaft** stellen Sie den Wert einer Eigenschaft eines Steuerelements oder auch eines Formulars oder Berichts ein. Die Makroaktion erwartet die folgenden Parameter:

- **Steuerelementname**: Geben Sie hier den Namen des Steuerelements an, also beispielsweise **txtText**. Wenn Sie diesen Parameter leer lassen, bezieht sich die Makroaktion auf das Objekt, das aktuell den Fokus besitzt. Sie müssen dieses also gegebenenfalls zuvor noch aktivieren (mit **ÖffnenFormular**, **ÖffnenBericht** oder **AuswählenObjekt**).

- **Eigenschaft**: Stellen Sie hier die betroffene Eigenschaft ein. Sie können lediglich die folgenden Eigenschaften verwenden: Aktiviert, Sichtbar, Gesperrt, Links, Breite, Oben, Höhe, Vordergrundfarbe, Hintergrundfarbe, Beschriftung und Wert.

- **Wert**: Geben Sie hier den Wert für die Eigenschaft ein. Für Eigenschaften, welche die Werte **Wahr/Falsch** beziehungsweise **True/False** erwarten, geben Sie die Werte **-1** oder **0** an.

Ein Beispiel folgt weiter unten.

Die Makroaktion **BearbeitenListenElemente** öffnet den Dialog, mit dem Sie die Listeneinträge eines Kombinationsfeldes oder Listenfeldes bearbeiten können. Dabei gibt es folgende Voraussetzungen: Das Steuerelement muss zuvor den Fokus erhalten, es muss seine Daten in einer **Wertliste** anzeigen (Eigenschaft **Herkunftsart**) und die Eigenschaft **Wertlistenbearbeitung zulassen** muss den Wert **Ja** enthalten.

Der mit der Makroaktion geöffnete Dialog sieht wie in Bild 1 aus (siehe auch Formular **frmBearbeitenListenElemente**).

Datensatzoperationen

Wie Sie den Datensatzzeiger bewegen und wie Sie einen neuen Datensatz anzeigen, haben Sie in den vorherigen Abschnitten gelernt. Schauen wir uns nun an, welche Operationen mit Datensätzen Sie noch mit der Hilfe von Makros durchführen können.

Die Makroaktion **DatensatzLöschen** etwa erlaubt das Löschen des aktuellen Datenatzes. Um einen Datensatz zu löschen, aktivieren Sie im gleichen Makro zuvor das Objekt, in diesem sich der zu löschende Datensatz befindet. Wenn Sie eine Schaltfläche etwa in einem Detailformular anlegen, dass selbst den zu löschenden Datensatz anzeigt, braucht diese lediglich die Makroaktion **DatensatzLöschen** aufzurufen. Ein Beispiel dazu finden Sie im Formular **frmArtikelDetail**.

Wenn Sie hingegen einen Datensatz in einem Unterformular oder in einem anderen Formular speichern möchten, müssen Sie das entsprechende Element zuvor mit dem Fokus

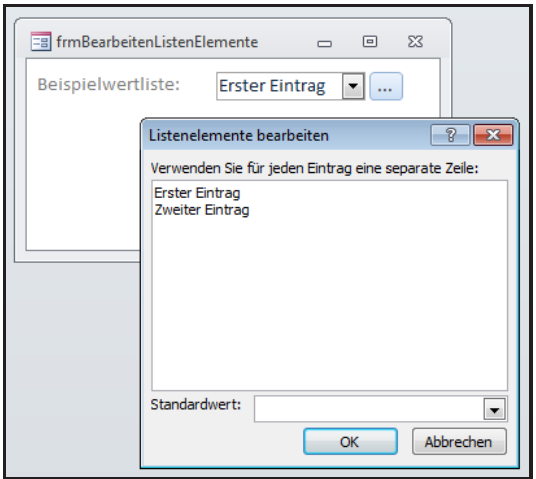


Bild 1: Öffnen des Dialogs zum Anzeigen einer Wertliste zur Bearbeitung

versehen (siehe Schaltfläche **Datensatz löschen** im Formular **frmArtikel**).

Die Makroaktion **DatensatzSpeichern** speichert den aktuellen Datensatz in dem Objekt, das den Fokus besitzt. Hier sehen wir uns ein etwas umfangreicheres Beispiel an (siehe Bild 2).

Das Makro aus der Abbildung (siehe Formular **frmArtikelDetail**) erledigt drei Schritte:

- Es erstellt einen neuen Datensatz.
- Es füllt das Steuerelement Artikelname mit einem Wert, damit der Datensatz in Bearbeitung und ungespeichert ist.
- Dann ruft es die Makroaktion **SpeichernDatensatz** auf (die eigentlich der Makroaktion **AusführenMenübefehl** mit dem Wert **DatensatzSpeichern** entspricht) und speichert somit den Datensatz.

Gegebenenfalls möchten Sie auch einmal die Änderungen an einem bereits in Bearbeitung befindlichen und noch nicht gespeicherten Datensatzes rückgängig machen. Dies erledigt man normalerweise mit der **Escape**-Taste, aber nicht jeder Benutzer ist sich dieser Funktion bewusst.

Also fügen wir dem Formular **frmArtikelDetail** noch eine Schaltfläche hinzu, welche die Änderungen seit dem letzten Speichervorgang rückgängig macht.

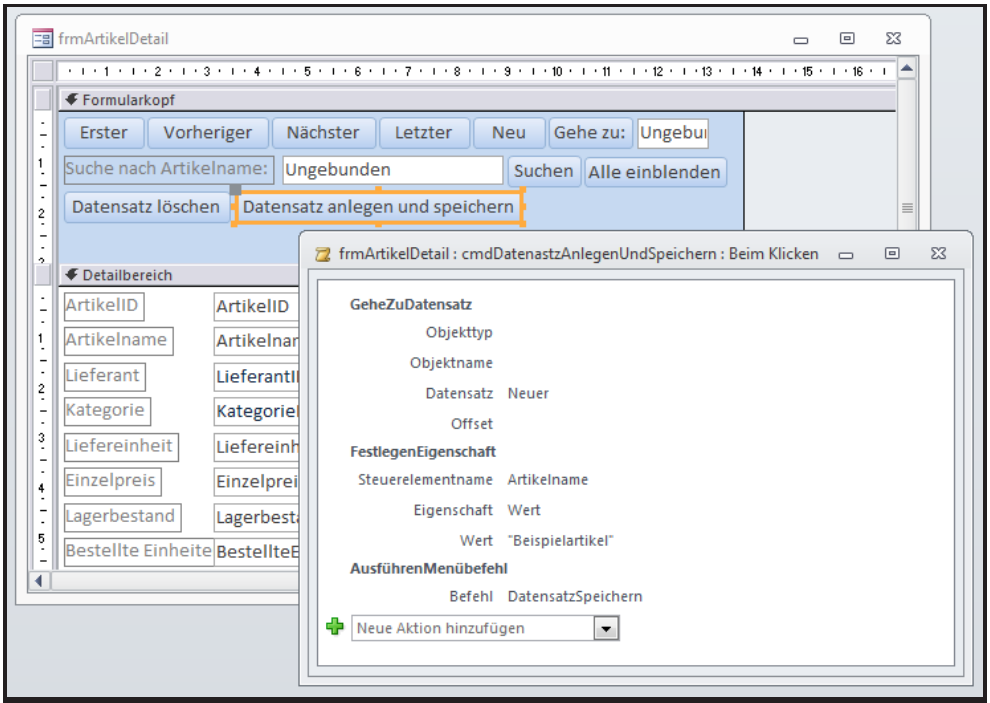


Bild 2: Anlegen und speichern eines neuen Datensatzes

Dazu verwenden wir schlicht und einfach die Makroaktion **RückgängigDatensatz**, die wiederum der Makroaktion **AusführenMenübefehl** mit dem Parameterwert **RückgängigDatensatz** entspricht. Ein Beispiel finden Sie hinter der Schaltfläche **Änderungen rückgängig machen** im Formular **frmArtikelDetail**.

Ereignisse abbrechen

VBA bietet in den Ereignisprozeduren von Ereignissen wie etwa **Beim Öffnen** oder **Vor Aktualisierung** einen Parameter namens **Cancel** an. Wenn Sie diesen auf **True** einstellen, wird die Aktion, die diesem Ereignis folgt, abgebrochen.

So wird beispielsweise das Formular nicht geöffnet, wenn bestimmte Bedingungen eintreten, unter denen **Cancel** auf **True** eingestellt wird oder ein Datensatz wird nicht aktualisiert.

Diese Möglichkeit gibt es auch bei Makros, die durch Formular- oder Berichtsereignisse ausgelöst werden. Nur sieht dies hier etwas anders aus, da durch Ereignisse ausgelöste Makros keine Parameter besitzen. Stattdessen rufen Sie in diesem Fall die Makroaktion **AbbrechenEreignis** auf.

Daten aktualisieren

Zum Aktualisieren der Daten in geöffneten Tabellen, Abfragen oder Formularen bietet Access verschiedene Methoden an.

Die Makroaktion **Aktualisieren** entspricht eigentlich der Makroaktion **AusführenMenübefehl** mit dem Wert Aktualisieren für den Parameter Wert und aktualisiert die Daten des aktuellen Objekts. Dabei wird die Datenherkunft nicht erneut abgefragt, sondern nur die aktuell angezeigten Datensätze. Das heißt, dass gelöschte oder

eventuell hinzugefügte Datensätze nicht berücksichtig werden.

Die Makroaktion **AktualisierenDatensatz** arbeitet genauso wie die Makroaktion **Aktualisieren**.

Die Makroaktion **AktualisierenDaten** erwartet als optionalen Parameter den Namen eines Steuerelements. Wenn Sie diesen festlegen, wird das genannte Steuerelement aktualisiert (typischerweise eines mit einer Datensatzherkunft wie ein Kombinationsfeld oder ein Listenfeld). Geben Sie kein Steuerelement an, aktualisiert die Makroaktion das aktuelle Objekt, also beispielsweise eine Tabelle, eine Abfrage oder ein Formular. Im Gegensatz zur Makroaktion Aktualisieren werden hier auch gelöscht und neu hinzugefügte Datensätze berücksichtigt.

Die Makroaktion **AktualisierenObjekt** aktualisiert die Anzeige im einem mit den beiden Parametern **Objekttyp** und **Objektname** angegebenen Objekt, in der Regel einem Formular. Damit erzwingen Sie die Anzeige etwa in berechneten Feldern, die sonst nicht sofort angezeigt würden, weil beispielsweise noch andere Aufgaben erledigt werden müssen.

Formular filtern

Wenn Sie in einem Formular nach bestimmten Daten suchen wollen, lässt sich dies auch per Makro realisieren. Dazu verwenden Sie die Makroaktion **FestlegenFilter**.

Wenn Sie beispielsweise alle Datensätze des Unterformulars anzeigen möchten, deren Feld **Artikelname** dem im Suchfeld **txtSuche** eingegebenen Begriff entspricht, verwenden Sie die folgenden Parameter für diese Makroaktion:

- **Filtername**: Name einer Abfrage, deren **WHERE**-Bedingung als Kriterium verwendet werden soll
- **Bedingung**: **[Artikelname] = [txtSuche]**
- **Steuerelementname**: **sfmArtikel**

Sie können dieser Makroaktion entweder den Parameter **Filtername** oder **Bedingung** übergeben. **Filtername** erwartet beispielsweise den Namen einer Abfrage, die eine **WHERE**-Bedingung zum Einschränken des Suchergebnisses enthält, also beispielsweise die folgende:

```
SELECT * FROM tblArtikel
WHERE Artikelname LIKE 'Ch*'
```

Für **Steuerelementname** geben keinen Wert ein, wenn sich die Schaltfläche, die **FestlegenFilter** auslöst, im gleichen Formular befindet wie die zu filternden Daten. Wenn sich diese Schaltfläche hingegen etwa im Hauptformular befindet, die zu durchsuchenden Daten hingegen in einem Unterformular, geben Sie den Unterformularnamen an.

Alternativ geben Sie unter **Bedingung** gleich den Filterausdruck an, also wie in diesem Beispiel **[Artikelname] = [txtSuche]**.

Letzteres ist etwas gewöhnungsbedürftig. Immerhin geben wir in der Bedingung mit **Artikelname** einen Feldnamen an, der in der Datenherkunft des Unterformulars existiert, ohne der Makroaktion mitzuteilen, dass es sich um ein Feld im Unterformular handelt. Wir tragen zwar den Namen des Unterformulars, also **sfmArtikel**, für die Eigenschaft **Steuerelementname** ein, was vermutlich dafür sorgt, dass beim Filtern das Feld des Unterformulars herangezogen wird.

Allerdings geben wir im Filter weiterhin das Textfeld **txtSuche** als Teil des Kriteriums an – dieses befindet sich allerdings im Hauptformular. Es ist an dieser Stelle nicht ganz klar, woher die Makroaktion weiß, dass **txtSuche** im Hauptformular und **Artikelname** im Unterformular zu suchen ist. Es kann aber wohl nicht schaden, darauf zu achten, dass die Bestandteile des Filterausdrucks in Haupt- und Unterformular eindeutig sind.

Wie auch immer: Die Makroaktion aus Bild 3 filtert die Daten des Unterformulars zuverlässig nach dem in **txtSuche** angegebenen Ausdruck.

Die **AnwendenFilter**-Makroaktion verwendet die gleichen Parameter wie **FestlegenFilter** und funktioniert auch weitgehend identisch. Allerdings scheint für den Parameter **Bedingung** die Angabe absoluter Werte für Steuerelement et cetera nötig.

Sie können also nicht einfach **txtSuche** als Vergleichswert angeben, sondern müssen auch noch das Formular referenzieren:

```
[Artikelname] Wie [Formulare]![frmArtikelDetail]![txtSuche]
```

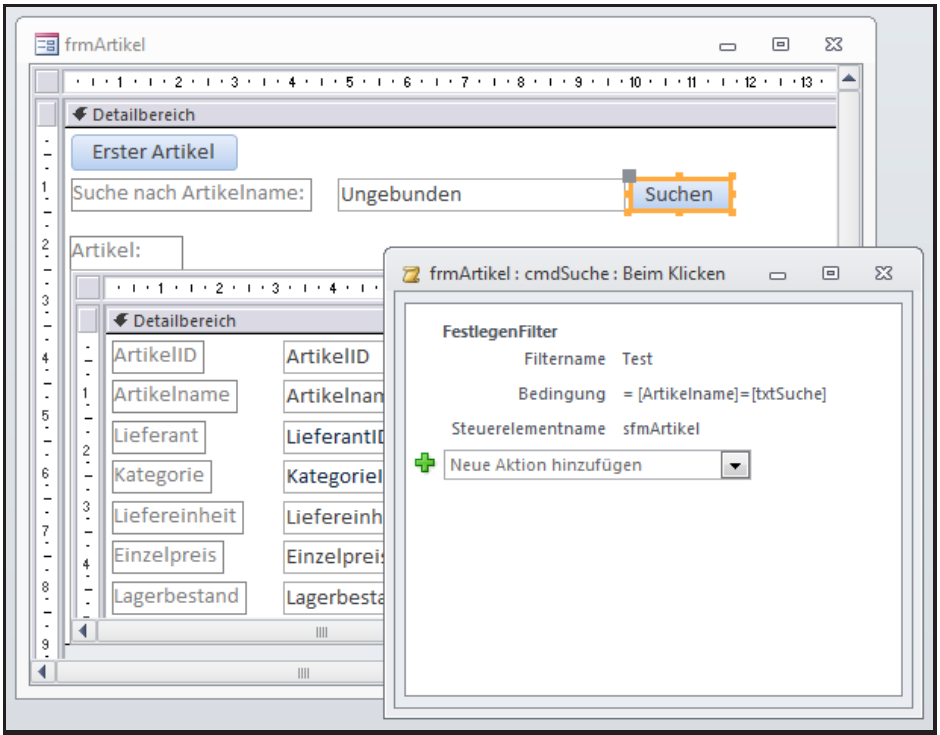


Bild 3: Makro zum Durchsuchen des Unterformulars

Filter aufheben

Wenn Sie nach dem Filtern eines Formulars oder Unterformulars wieder alle enthaltenen Daten anzeigen möchten, verwenden Sie die Makroaktion **FilterSortierungEntfernen**.

Diese können Sie auch verwenden, um auf beliebige Art festgelegte Sortierungen zurückzusetzen.

Eine weitere Makroaktion, mit der Sie wieder alle Datensätze im Formular oder Unterformular anzeigen, heißt **AnzeigenAlleDatensätze**. Diese Aktion erwartet keine Parameter, hebt aber auch nur den Filter auf, nicht die Sortierung.

Sortierung festlegen

Die Makroaktion **FestlegenSortiertNach** legt eine Sortierung für ein Formular oder einen Bericht fest. Sie erwartet zwei Parameter:

- **Sortiert nach**: Erwartet den Feldnamen, nach dem sortiert werden soll. Außerdem geben Sie hinter dem Feldnamen entweder ASC (für aufsteigende Sortierung) oder DESC (absteigend) an. Wenn Sie keines von beiden angeben, sortiert Access aufsteigend.

- **Steuerelementname**: Falls ein Unterformular oder Unterbericht sortiert werden soll, geben Sie hier den Namen des Unterformulars/Untersteuerelements an.

Leider können Sie nicht einfach etwa das zu sortierende Feld per Kombinationsfeld auswählen lassen. **Sortiert Nach** erwartet in jedem Fall eine Zeichenkette.

Datensatz suchen

Während Sie mit den Filter-Aktionen gleich alle nicht mit dem angegebenen Kriterium übereinstimmenden Datensätze ausblenden, ermöglichen einige weitere Makro-Aktionen das gezielte Springen zu bestimmten Datensätzen. Eine dieser Aktion lautet **SuchenDatensatz**. Diese erwartet die folgenden Parameter:

- **Suchen nach**: Suchausdruck

- **Vergleichen**: Auswahl, ob der gesamte Feldinhalt verglichen werden soll (das Feld muss komplett mit dem Suchbegriff übereinstimmen), nur ein Teil des Feldinhalts (der Suchbegriff muss irgendwo im Feld vorkommen) oder nur der Beginn des Feldinhalts.

- **Groß-/Kleinschreibung**: Gibt an, ob bei der Suche auf die Groß-/Kleinschreibung geachtet werden soll.

- **Suchen**: Gibt an, ob alle Datensätze vom ersten Datensatz an, vom aktuellen Datensatz nach oben oder vom aktuellen Datensatz nach unten gesucht werden soll.

- **Wie formatiert**: Gibt an, ob durch Formatierungen hinzugefügte Zeichen bei der Suche berücksichtigt werden sollen. Eine Zahl wie **1000** kann ja beispielsweise mit Tausendertrennzeichen angezeigt werden (**1.000**). Wenn Sie Wir Formatiert auf Ja einstellen, wird in diesem Fall das Tausendertrennzeichen berücksichtigt.

- **Nur aktuelles Feld**: Gibt an, ob nur das aktuelle Feld durchsucht werden soll oder alle Felder.

Fehlerbehandlung in Makros

Wenn Sie mit Makros programmieren, was zumindest beim Einsatz von Webdatenbanken (ab Access 2010) oder beim Einsatz von Datenmakros nötig ist, möchten Sie vielleicht auch eine Fehlerbehandlung zu Ihren Makros hinzufügen. Dieser Artikel zeigt, wie Fehlerbehandlung in Makros funktioniert und welche Makroaktionen dafür benötigt werden.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1403_Makros.mdb**.

Makroaktionen zur Fehlerbehandlung

Für die Fehlerbehandlung stellt Access ab Version 2010 die beiden folgenden Makroaktionen zur Verfügung:

- **BeiFehler**: Wird vor dem Aufruf der mit einer Fehlerbehandlung zu versehenen Makroaktionen aufgerufen und nimmt die beiden Parameter **Gehe zu** und **Makroname** entgegen.

- **LöschenMakroFehler**: Löscht die Informationen des zuletzt aufgetretenen Fehlers

Makroaktion BeiFehler

Die Makroaktion **BeiFehler** erlaubt verschiedene Aktionen, wenn ein Fehler in einer der folgenden Makroaktionen ausgelöst wird. Diese können Sie mit dem Parameter **Gehe zu** einstellen (siehe Bild 1):

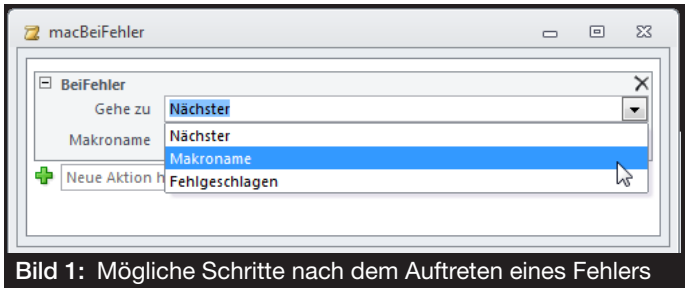


Bild 1: Mögliche Schritte nach dem Auftreten eines Fehlers

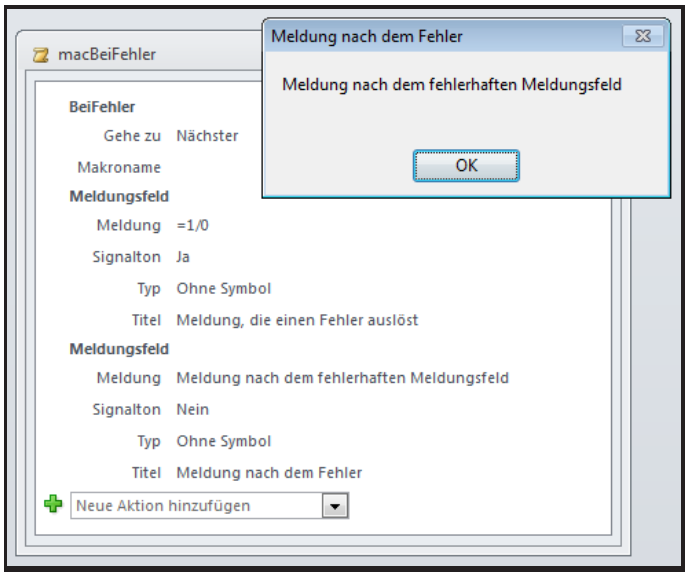


Bild 2: Gehe zu: Nächster fährt nach dem Fehler mit der folgenden Makroaktion fort.

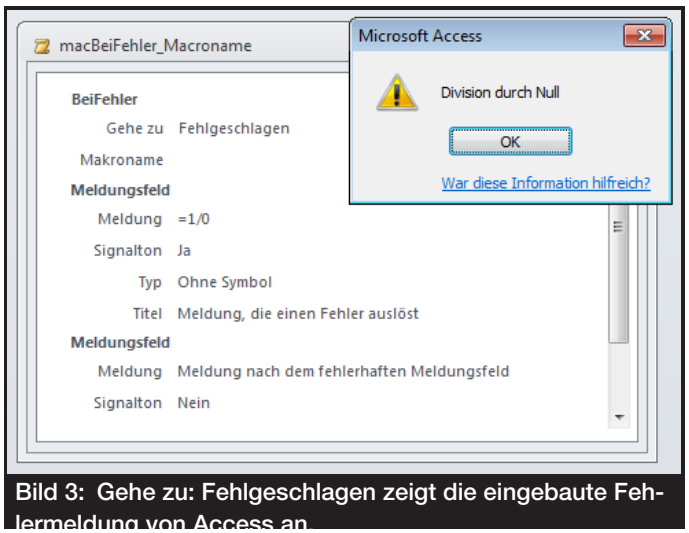


Bild 3: Gehe zu: Fehlgeschlagen zeigt die eingebaute Fehlermeldung von Access an.

- **Nächster**: Sorgt dafür, dass die fehlerhafte Anweisung ignoriert und die folgende Anweisung ausgeführt wird. In Bild 2 löst die Division durch 0 einen Fehler aus und zeigt gleich die folgende Meldung an. Dies entspricht der Anweisung **On Error Resume Next** unter VBA.

- **Macroname**: Sie können eine Fehlerbehandlung in einem Makro oder einem Untermakro programmieren. Wenn Sie diese Option auswählen, geben Sie für den zweiten Parameter **Makroname** den Namen des aufzurufenden Makros oder Untermakros an.

- **Fehlgeschlagen**: Löst den Fehler aus und zeigt die Standardmeldung von Access an, so als ob die Fehlerbehandlung gar nicht aktiviert wäre. In der Tat dient diese Variante dazu, eine zuvor aktivierte benutzerdefinierte Fehlerbehandlung wieder zu deaktivieren (entspricht der Anweisung **On Error Goto 0** unter VBA)

In letzterem Fall würde Access die gleiche Fehlermeldung anzeigen, als wenn das Makro keine Fehlerbehandlung enthielte. Diese sieht wie die Fehlermeldung aus, welche eine entsprechende VBA-Anweisung auslösen würde – nur ohne Angabe der Fehlernummer und ohne die Möglichkeit, den Fehler zu debuggen (siehe Bild 3).

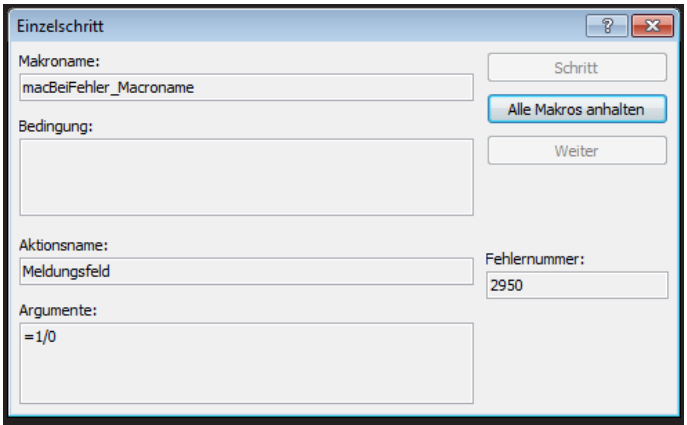


Bild 4: Fehlermeldung eines Makros

Anschließend erscheint dann noch ein Dialog mit weiteren Informationen (siehe Bild 4). Dieser enthält den Makronamen, die Bedingung, den Aktionsnamne, der den Fehler ausgelöst hat, die Argumente und die Fehlernummer. Außerdem finden Sie dort die Möglichkeit, verschiedene Aktionen auszuführen.

Das MacroError-Objekt

Zusätzlich gibt es ein Objekt namens **MacroError**, das ähnliche Informationen liefert wie das von VBA bekannte **Err**-Objekt:

- **Number**: Fehlernummer (entspricht **Err.Number**)
- **Description**: Fehlermeldung (entspricht **Err.Description**)
- **MacroName**: Name des Makros, das den Fehler auslöste
- **ActionName**: Name der Makroaktion, die den Fehler auslöste
- **Arguments**: Parameter der Makroaktion
- **Condition**: Bedingung beim Aufruf der Makroaktion

Wie Sie dieses Objekt nutzen können, versuchen wir anhand einer Fehlerbehandlung herauszufinden. Dazu legen wir ein neues Makro namens macFehlerbehandlung an und fügen ein Untermakro namens Fehlerbehandlung hinzu. Dieses soll eine Meldungsfeld-Makroaktion enthalten, welche die Fehlerinformationen in einem Meldungsfenster ausgibt.

Um den auszugebenden Ausdruck zusammenzustellen, machen wir ausgiebigen Gebrauch der mit Access 2010 eingeführten IntelliSense-Funktionen für Ausdrücke. Dies sieht dann etwa wie in Bild 5 aus, wo wir die einzelnen Elemente des **MacroError**-Objekts zu einer Meldung zusammenstellen. Insgesamt sieht diese dann wie folgt aus:

```
="Fehler '"
& [MacroError].[Beschreibung]
& "' in der Aktion '"
& [MacroError].[Actionname]
& "' des Makros '"
& [MacroError].[MacroName]
& "'. Bedingung: "
& [MacroError].[Condition]
& ", Fehlernummer "
& [MacroError].[Number]
```

Makro-Fehlerbehandlung aufrufen

Nun haben wir uns gedacht, dass es ja sinnvoll wäre, die Fehlerbehandlung in einem eigenen Makro unterzubringen – nämlich im Untermakro **Fehlerbehandlung** im Makro **macFehlerbehandlung** –, damit wir diese von verschiedenen Makros aus aufrufen können. Also probieren wir das in einem weiteren Makro namens **macBeiFehler_ExterneFehlerbe-**

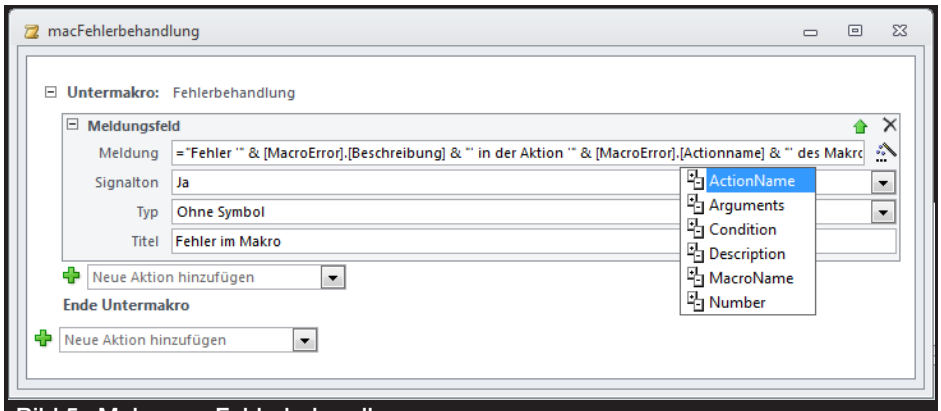


Bild 5: Makro zur Fehlerbehandlung

Datenmakros I: Grundlagen

Eine der interessantesten Neuerungen mit Access 2010 sind die Datenmakros. Damit können Sie sowohl endlich auch die üblichen Bearbeitungen und Zugriffe auf Tabellen durchführen (mit benannten Makros) als auch eine Art Ereignisse für Tabellen definieren, die beim Anlegen, Ändern oder Löschen von Daten in Tabellen ausgelöst werden (Datenmakros). Dieser Artikel zeigt, wie Sie Datenmakros und benaanlegen und was es dabei zu beachten gibt.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1403_Datenmakros.mdb**.

Einführung

Datenmakros kommen gleich in zwei Ausführungen: Die erste entspricht etwa den Triggern anderer Datenbanksysteme. Trigger werden durch Änderungen an den Datensätzen einer Datenbank ausgelöst. Die zweite kann einfache Datenoperationen abbilden, wie Sie sie auch per VBA mit den Elementen der DAO-Bibliothek programmieren können – also etwa das Hinzufügen, Ändern oder Löschen eines Datensatzes, aber auch das Auslesen von Feldinhalten eines zuvor ermittelten Datensatzes.

Ereignisse in Tabellen

Die folgenden Ereignisse in Zusammenhang mit der Änderung von Datensätzen in Tabellen können Datenmakros auslösen (sofern diese definiert sind):

- Nach dem Anlegen eines Datensatzes
- Vor dem Ändern eines oder mehrerer Felder eines Datensatzes
- Nach dem Ändern eines Datensatzes
- Vor dem Löschen eines Datensatzes
- Nach dem Löschen eines Datensatzes

Das Schöne an Datenmakros ist, dass diese immer ausgelöst werden, wenn die entsprechende Änderung erfolgt – egal, wodurch die Änderung geschieht.

Es gibt beispielsweise die folgenden Möglichkeiten, eine entsprechende Änderung zu bewirken und dadurch ein Datenmakro auszulösen:

- Änderungen direkt in der Datenblattansicht der Tabelle
- Änderungen über die Datenblattansicht einer Abfrage
- Änderungen über die Eingabe in ein Formular
- Änderungen per DAO oder ADO
- Änderungen durch die direkte Ausführung einer Aktionsabfrage über den Navigationsbereich (**INSERT INTO**, **UPDATE**, **DELETE**, **SELECT INTO**)
- Änderung durch eine Aktionsabfrage, die per **DoCmd.RunSQL** oder per **<Database>.Execute** ausgelöst wurde

Der Artikel **Datenmakros III: Ereignisse** liefert einige Beispiele für den Einsatz von Ereignissen in Tabellen.

Benannte Makros

Die benannten Makros legen Sie immer in Zusammenhang mit einer Tabelle an. Diese müssen sich aber nicht auf die aktuelle Tabelle beziehen, sondern Sie können damit auch die Daten anderer Tabellen bearbeiten oder einlesen. Benannte Makros können Sie entweder von Datenmakros aus auslösen oder auch mit der Makroaktion **AusführenDatenmakro** von einem herkömmlichen Makro aus.

Auch die Ausführung per VBA ist möglich. Dazu rufen Sie einfach die Methode **RunDataMacro** des **DoCmd**-Objekts auf – beispielsweise wie folgt:

```
DoCmd.RunDataMacro "macDatenmakro"
```

Später erfahren Sie noch ein weiteres Detail hierzu – nämlich zur Übergabe von Parametern.

Datenmakros anlegen

Datenmakros werden nicht wie herkömmliche Datenmakros über

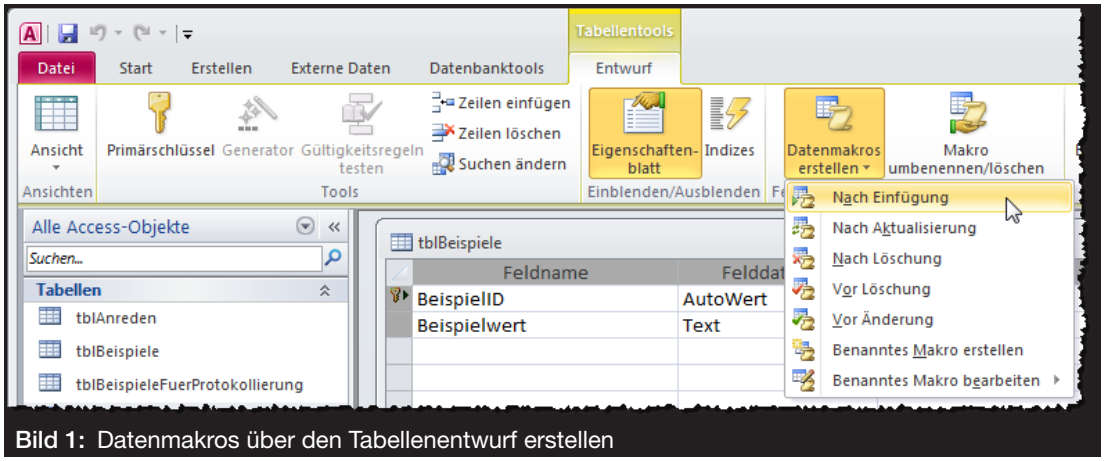


Bild 1: Datenmakros über den Tabellenentwurf erstellen

den Ribbon-Eintrag **Erstellen|Makros und Code|Makros** erstellt, sondern im Kontext der jeweiligen Tabelle. Dazu wählen Sie in der Entwurfsansicht den Ribbon-Eintrag **Entwurf|Feld-, Datensatz- und Tabellenereignisse|Datenmakros erstellen** und öffnen so eine Liste der verfügbaren Datenmakros sowie die Möglichkeit, ein benanntes Makro zu erstellen (siehe Bild 1).

Später werden Sie an dieser Stelle die Möglichkeit erhalten, bereits erstellte Datenmakros und benannte Makros zu bearbeiten.

Mit den neueren Access-Versionen (ab Version 2007) können Sie Tabellen auch in der Datenblattansicht bearbeiten. Das gilt auch für das Anlegen und Bearbeiten von Datenmakros und benannten Makros. Wenn Sie die Ansicht der Tabelle zur Datenblattansicht wechseln, finden Sie im Ribbon unter dem Tab **Tabelle** alle notwendigen Befehle direkt in der Übersicht vor (siehe Bild 2).

Hier werden die Makros sogar noch in zwei Ribbon-Gruppen unterteilt, nämlich in Vorabereignisse (**Vor Änderung** und **Vor Löschung**) und Nachfolgeereignisse (**Nach Einfügung**, **Nach Aktualisierung** und **Nach Löschung**).

Einfache Protokollierung

Wählen Sie hier den Eintrag **Nach Einfügung**, zeigt Access ein neues, leeres Makro an, das nach dem Einfügen eines neuen Datensatzes ausgeführt wird. In diesem Fall wollen wir noch keine Änderung am Feld **AngelegtAm** vornehmen, sondern ein sehr schönes Feature der Datenmakros nutzen: Es gibt nämlich eine Tabelle namens **USysApplicationLog**, mit der Sie Informationen aus Makros heraus speichern können, und zwar auf einfachste Weise.

Dazu klicken Sie in der Datenblattansicht auf den Ribbon-Eintrag **Tabelle|Nachfolgeereignisse|Nach Einfügung**. Es erscheint ein neues, leeres Makro, dem Sie den Befehl **ProtokollierenEreignis** hinzufügen. Dieses erwartet als Parameter lediglich den Text des zu protokollierenden Ereignisses, in diesem Fall den folgenden (siehe Bild 3):

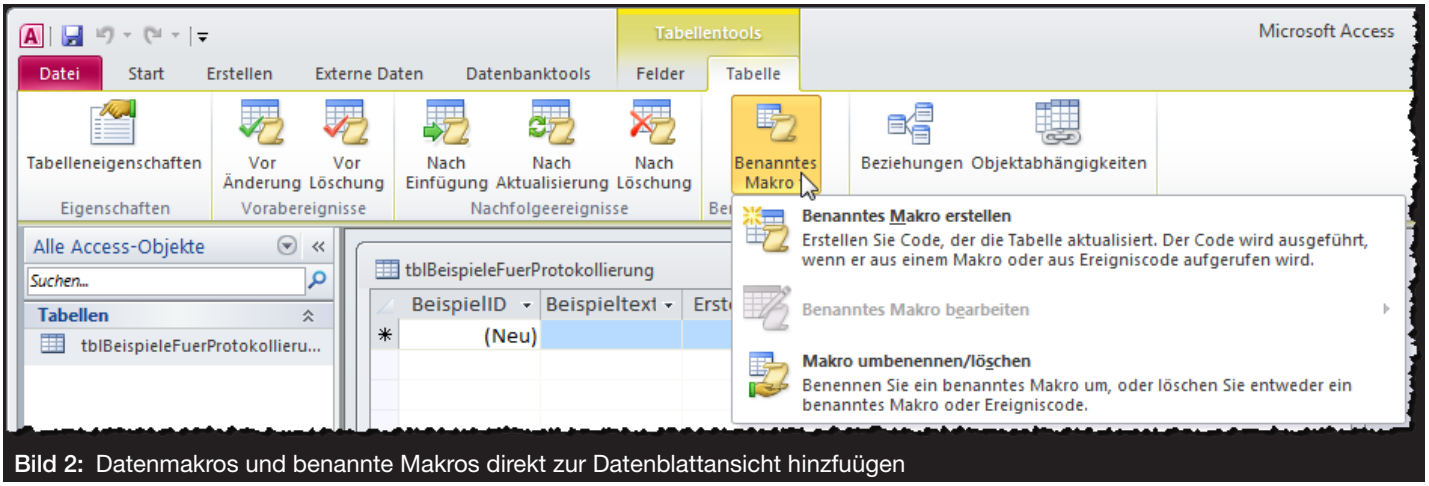


Bild 2: Datenmakros und benannte Makros direkt zur Datenblattansicht hinzufügen

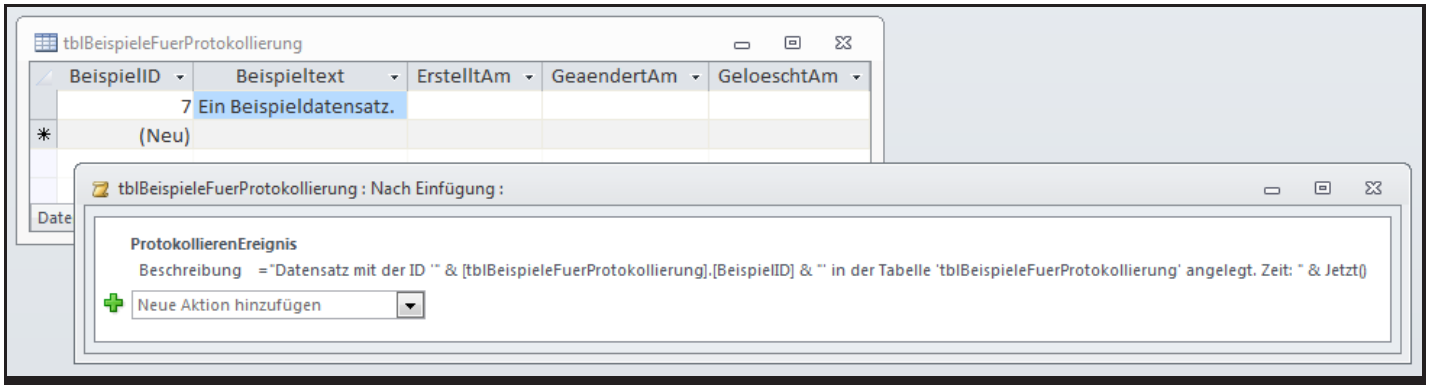


Bild 3: Datenmakros und benannte Makros direkt zur Datenblattansicht hinzfuügen

```
="Datensatz mit der ID '" & [tblBeispieleFuerProtokollierung].[BeispielID] & "' in der Tabelle 'tblBeispieleFuerProtokollierung' angelegt. Zeit: " & Jetzt()
```

Darin stecken gleich mehrere Informationen, nämlich der Primärschlüsselwert des neu hinzugefügten Datensatzes und die aktuelle Zeit mit Datum (**Jetzt()**).

Wenn Sie das Makro speichern und schließen, können Sie es gleich ausprobieren. Dazu müssen Sie lediglich einen neuen Datensatz zur Tabelle hinzufügen. Aber was geschieht nun? Zunächst einmal nichts Sichtbares.

Erst wenn Sie zufällig zum Backstage-Bereich von Access 2010 und neuer wechseln, finden Sie einen passenden Hinweis: Über den üblichen Schaltflächen etwa zum Komprimieren und Reparieren der aktuellen Datenbank finden Sie dort nämlich jetzt eine Schaltfläche mit der Beschriftung **Anwendungsprotokolltabelle anzeigen** (siehe Bild 4).

Klicken Sie diese Schaltfläche an, erscheint die Tabelle **USysApplicationLog** (siehe Bild 5). Diese enthält einige wichtige Informationen:

- **SourceObject**: Objekt, das den Eintrag verursacht hat – in diesem Fall das **AfterInsert**-Makro der Tabelle **tblBeispieleFuerProtokollierung**
- **Data Macro Instance ID**: Eindeutiger Identifizierer der Makro-Instanz
- **Error Number**: Fehlernummer
- **Category**: Kategorie des Fehlers, hier **User** (also benutzerdefiniert)
- **Object Type**: Typ des auslösenden Objekts, hier **Macro**
- **Description**: Beschreibung (in diesem Fall benutzerdefiniert entsprechend dem Wert des Parameters Beschreibung der Makroaktion **ProtokollierenEreignis**)
- **Context**: Kontext des Fehlers, in diesem Fall leer

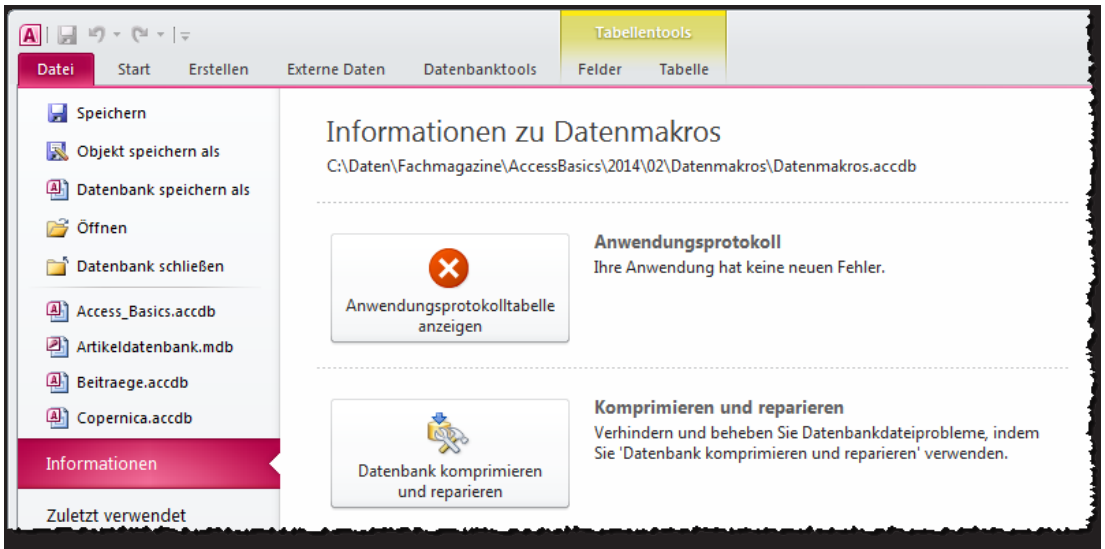


Bild 4: Der Backstage-Bereich liefert einen Hinweis auf die Anwendungsprotokolltabelle

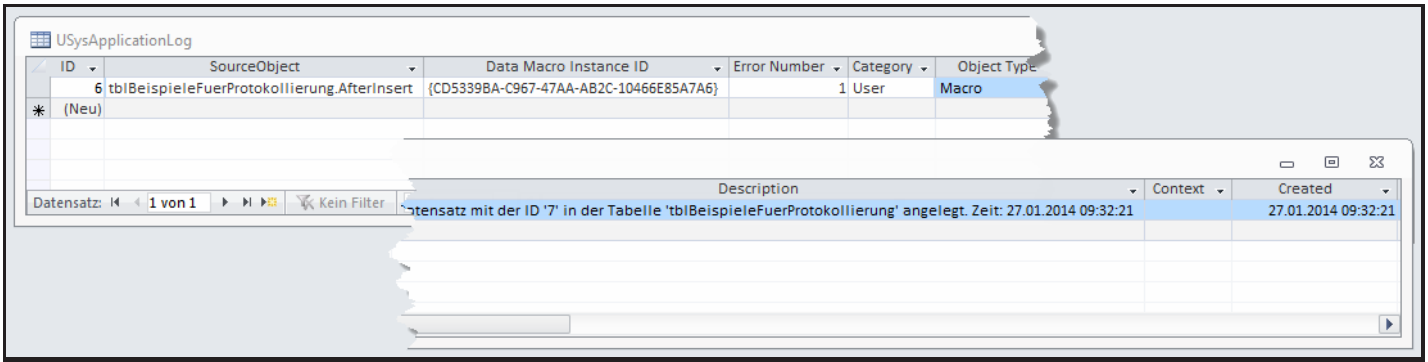


Bild 5: Die Tabelle **USysApplicationLog**

- **Created**: Zeitpunkt der Erstellung des Datensatzes. Die Angabe des Datums in der Beschreibung hätten wir uns somit sparen können.

Die Tabelle **USysApplicationLog** dient auch der Aufzeichnung von Fehlern, die in Datenmakros ausgelöst werden (Fehler in herkömmlichen Makros führen ja zu entsprechenden Meldungen, wenn diese nicht anderweitig behandelt werden, Datenmakros lösen keine offensichtlichen Fehlermeldungen aus).

Datenmakros verwalten

Auch wenn wir bislang noch keine Datenmakros und benannte Makros angelegt haben, wollen wir uns bereits ansehen, wie Sie die vorhandenen, in Zusammenhang mit einer Tabelle erstellten Makros verwalten können.

Dazu klicken Sie im Ribbon auf die Schaltfläche **Tabelle|Benannte Makros|Benannte Makros** und wählen dort den Eintrag **Makro umbenennen/löschen** aus (siehe Bild 6).

Dies öffnet wiederum den Dialog **Datenmakro-Manager** aus Bild 7.

Im Screenshot zeigt dieser zwar erst ein einziges Beispielmakro an, aber wenn Sie einmal mehrere Datenmakros und/oder benannte Makros zu einer Datenbank hinzugefügt haben, ist der **Datenmakro-Manager** ein hilfreiches Tool. Sie können damit jedes Datenmakro oder benanntes Makro der Tabellen Ihrer Datenbank anzeigen, bearbeiten oder löschen.

Aktionen in Datenmakros und benannten Makros

Nachfolgend schauen wir uns die in Datenmakros zur Verfügung stehenden Datenmakros im Detail an. Zunächst einmal finden Sie dort einige Elemente, die genau wie in einfachen Makros arbeiten:

- **Gruppieren**: Fasst eine oder mehrere Makroaktionen zu einer Gruppe zusammen. Dies ermöglicht das gemeinsame Ein- und Ausblenden mehrerer Befehle aus Gründern der Übersichtlichkeit.

- **Kommentar**: Gestattet das Hinzufügen eines Kommentars zu einem Makro.

- **Wenn**: Dient dem Anlegen von **If...Then**-Strukturen in Datenmakros und benannten Makros.

- **Bei Fehler**: Mit der **Bei Fehler**-Makroaktion können Sie angeben, was beim Auslösen eines Fehlers in einer der folgenden Makroaktionen geschehen soll. Dies funktioniert genau wie in herkömmlichen Makros, allerdings können Sie

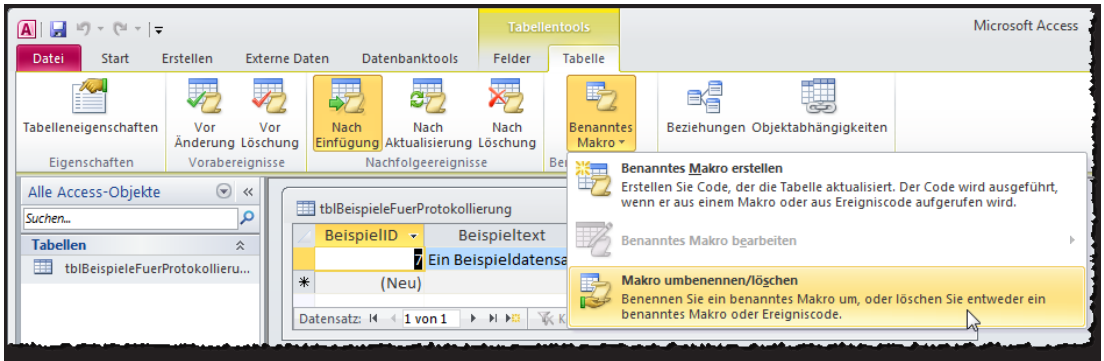


Bild 6: Aufruf der Benutzeroberfläche zur Verwaltung von Makros

Vertrauenswürdige Datenbanken

Mit Access 2007 hat Microsoft einige Features eingeführt, die auf die sichere Ausführung von Access-Anwendungen abzielen. Sichere Ausführung bedeutet dabei, dass eine Access-Datenbank, die Sie etwa aus dem Internet heruntergeladen oder per E-Mail erhalten haben, nicht einfach gestartet werden kann. Der Benutzer muss, um eine solche Datenbankanwendung mit Access zu öffnen, erst noch selbst tätig werden.

Sicher ist sicher

Wenn Sie nicht aufpassen und unbedacht eine Access-Datenbank ausführen, können Sie großen Schaden anrichten. Anwendungen können nämlich auf verschiedenen Ebenen Anweisungen ausführen, mit denen Dateien gelöscht, Daten verändert oder das System infiltriert werden kann – zum Beispiel die Folgenden:

- VBA-Code
- Makros
- ActiveX-Steuerelemente
- SQL-Code
- Aktionsabfragen
- Ausdrücke in Abfragen, Formularen, Berichten oder Steuerelementen

Um zu verhindern, dass der Aufruf oder der Einsatz eines der genannten Elemente keinen Schaden auf Ihrem System anrichten kann, hat Microsoft einige Sicherheitsfeatures in Access installiert.

So erscheint beispielsweise, wenn Sie eine frisch heruntergeladene oder erstellte Datenbank mit Access öffnen, der gelbe Balken aus Bild 1 direkt unter dem Access-Ribbon und weist darauf hin, dass einige aktive Inhalte der geöffneten Datenbank deaktiviert wurden. Klicken Sie hier auf den Text in der Mitte, erscheinen etwa unter Access 2010 weitere Informationen (siehe Bild 2).

Diese besagen, dass zum Beispiel VBA-Code deaktiviert wurde (dass die Datenbank soeben erst erstellt wurde und gar keinen VBA-Code enthält, ignoriert der Mechanismus). Wenn Sie hier auf die Schaltfläche Inhalt aktivieren klicken, erhalten Sie wie in Bild 3 die folgenden Möglichkeiten:

- **Alle Inhalte aktivieren**: Damit fügen Sie die Anwendung der Liste der vertrauenswürdigen Dateien hinzu – mehr dazu weiter unten.
- **Erweiterte Optionen**: Öffnet den Dialog aus Bild 4. Damit können

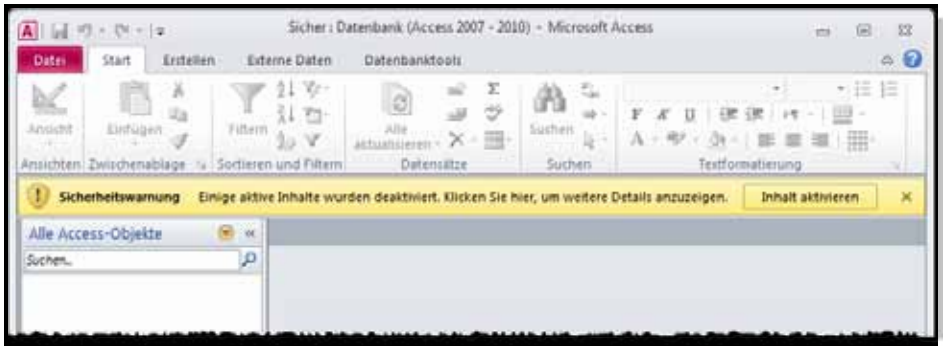


Bild 1: Hinweis auf eine nicht als vertrauenswürdig eingestufte Datenbankdatei

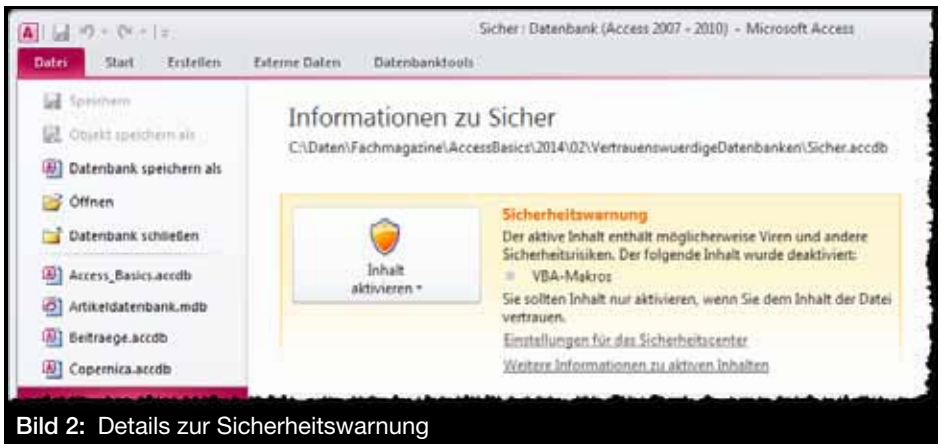


Bild 2: Details zur Sicherheitswarnung

Sie die Anwendung für eine Sitzung „entsichern". Beim nächsten Öffnen erscheint erneut der gelbe Balken mit der Sicherheitswarnung.

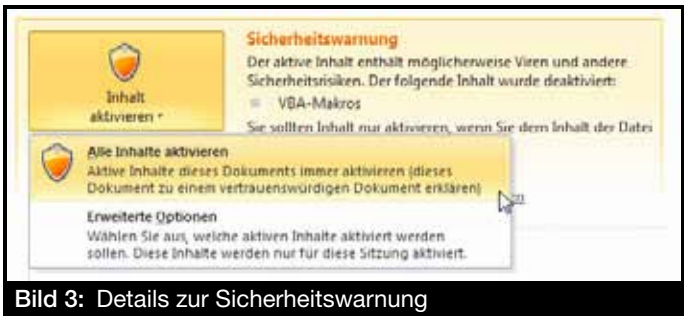


Bild 3: Details zur Sicherheitswarnung

Wenn Sie den Dialog **Microsoft Office-Sicherheitsoptionen** mit der Option **Vor unbekanntem Inhalt schützen (empfohlen)** schließen, können Sie in dieser Sitzung keinen VBA-Code in dieser Anwendung ausführen.

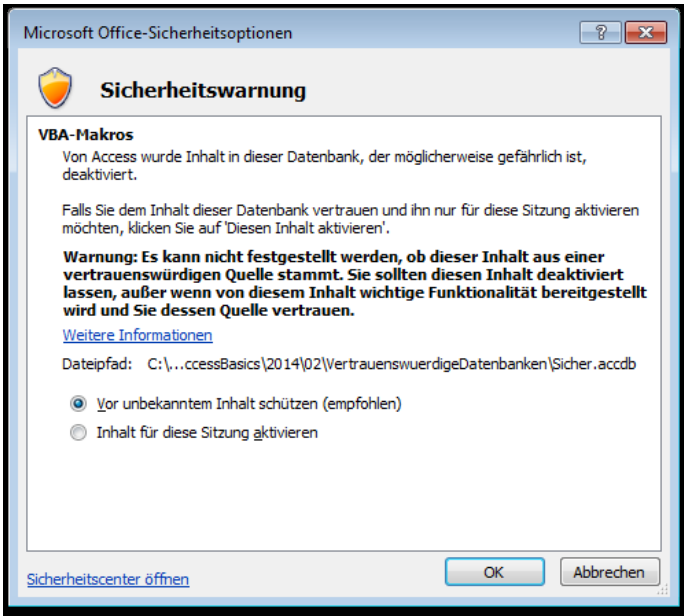


Bild 4: Aktivieren der Datenbank für eine Sitzung

Dies lässt sich leicht prüfen. Erstellen Sie in einer jungfräulichen (und somit nicht als vertrauenswürdig eingestuften) Datenbank ein VBA-Standardmodul und fügen Sie dieser eine einfache Prozedur hinzu. Wenn Sie diese nun ausführen, erhalten Sie die Meldung aus Bild 5. Das bedeutet, dass Sie die Datenbank auf irgendeine Weise zumindest übergangsweise für die Dauer einer Sitzung als vertrauenswürdig einstufen müssen.

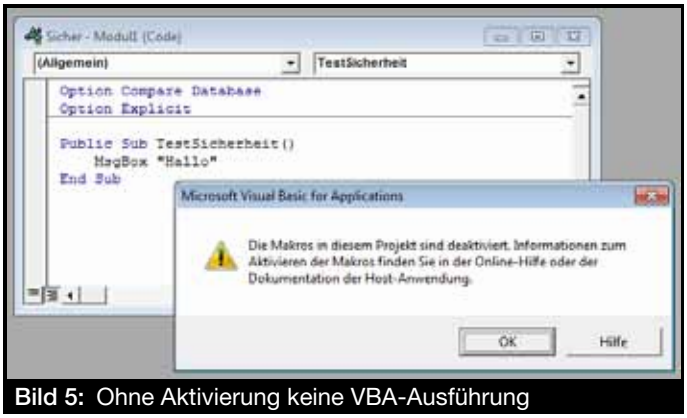


Bild 5: Ohne Aktivierung keine VBA-Ausführung

Alle Inhalte immer aktivieren

Die erste Option unter der Schaltfläche **Inhalt aktivieren** stuft die Anwendung dauerhaft als vertrauenswürdig ein. Für uns ist interessant, was dort im Hintergrund geschieht beziehungsweise an welchem Ort diese Einstufung gespeichert wird. Vielleicht möchten wir diese Einstellung ja später einmal rückgängig machen?

Access-Optionen

Dazu schauen wir intuitiv erstmal in den Access-Optionen nach. Unter Access 2010 etwa finden Sie diese im Backstage-Bereich von Access unter dem Eintrag **Sicherheitscenter**. Dort klicken Sie auf die Schaltfläche **Einstellungen für das Sicherheitscenter...** und öffnen damit den Dialog **Sicherheitscenter**.

Wenn wir uns die Einträge auf der linken Seite ansehen, scheint **Vertrauenswürdige Dokumente** ein guter Startpunkt zu sein. Ein Klick auf diesen Eintrag und das Öffnen der entsprechenden Registerseite ernüchtert allerdinge: Keine Liste der als vertrauenswürdig eingestuften Dokumente, stattdessen nur zwei Optionen und eine Schaltfläche (siehe Bild 6).

Mit **Vertrauenswürdige Dokumente in einem Netzwerk zulassen** erlauben Sie, dass auch Access-Datenbanken von Netzlaufwerken zur Liste der vertrauenswürdigen Dokumente hinzugefügt werden können.

Vertrauenswürdige Dokumente deaktivieren sorgt dafür, dass Sie Access-Datenbanken dauerhaft nur für die jeweils aktuelle Sitzung aktivieren können. Ein Hinzufügen zur Liste der vertrauenswürdigen Dokumente ist nicht mehr möglich. Es gibt jedoch mindestens eine Ausnahme – wenn Sie die Datenbank, wie weiter unten

erläutert, zu einem vertrauenswürdigen Verzeichnis hinzufügen, wird diese dennoch immer geöffnet und ist auch voll funktionstüchtig.

Die Schaltfläche **Bereinigen** sorgt dafür, dass alle Dokumente aus der Liste der vertrauenswürdigen Dokumente entfernt werden. Das hilft uns allerdings wenig, wenn wir gar nicht wissen, welche Elemente überhaupt in dieser Liste enthalten sind – und wie viele der zuvor mühevoll aktivierten Datenbanken dann erneut aktiviert werden müssten.

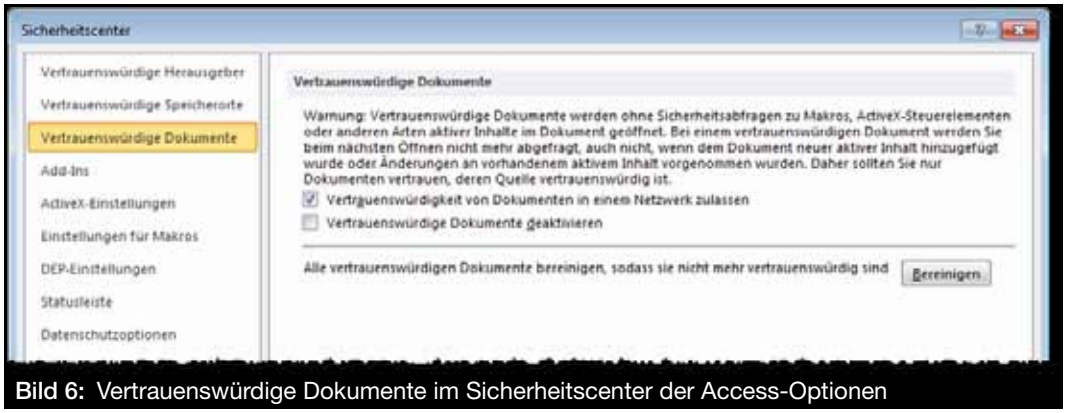


Bild 6: Vertrauenswürdige Dokumente im Sicherheitscenter der Access-Optionen

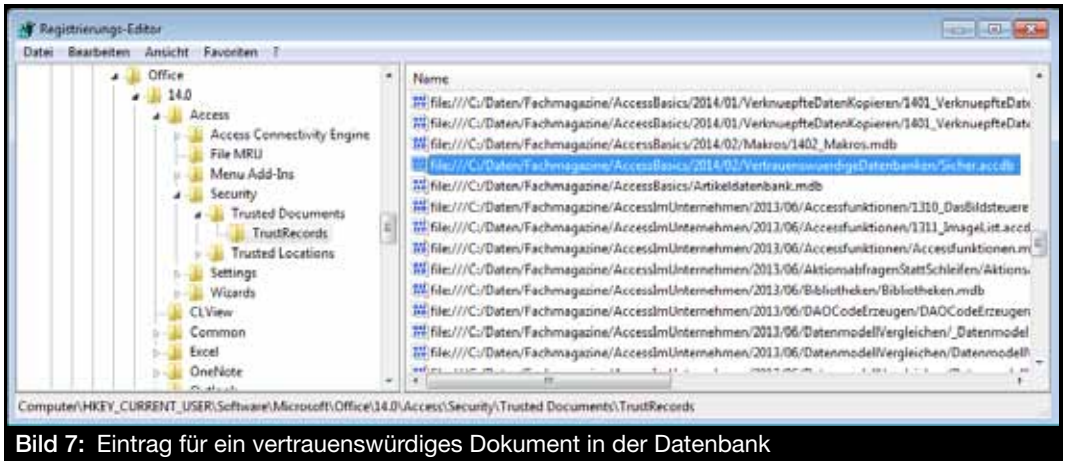


Bild 7: Eintrag für ein vertrauenswürdiges Dokument in der Datenbank

Aus Mangel an Alternativen werfen wir also einen Blick in die Windows Registry – dem Ort, an dem solche Informationen gern dauerhaft gespeichert werden. Und wir werden schnell fündig, und zwar unter **HKEY_CURRENT_USER\Software\Microsoft\Office\14.0\Access\Security\Trusted Documents\TrustRecords**. Dort finden wir die Liste aus Bild 7 mit allen als vertrauenswürdig eingestuften Dateien vor. Sollten Sie also einmal eine einzelne Datei aus dieser Liste entfernen wollen, ohne gleich mit Brachialgewalt die komplette Liste zu leeren, ist der Gang zur Registry die einzige Alternative.

Wenn Sie die Access-Instanz schließen und wieder öffnen, weist Access dann übrigens wieder auf den Status als deaktivierte Datenbank hin.

Vertrauenswürdige Verzeichnisse

Neben den vertrauenswürdigen Dokumenten gibt es auch vertrauenswürdige Verzeichnisse. Sie müssen also nicht jedes einzelne Dokument entriegeln, sondern können auch gleich komplette Verzeichnisse samt Unterverzeichnissen als vertrauenswürdig einstufen. Dies erledigen Sie ebenfalls im Dialog **Sicherheitscenter**, wo Sie nun den Bereich **Vertrauenswürdige Speicherorte** anzeigen (siehe Bild 8). Hier finden Sie die folgenden Optionen:

- **Vertrauenswürdige Speicherorte im Netzwerk zulassen (nicht empfohlen)**: Erlaubt den ungeschützten Zugriff auf Access-Datenbanken, die auf Netzlaufwerken gespeichert sind
- **Alle vertrauenswürdigen Speicherorte deaktivieren**: gleiche Funktion wie bei den vertrauenswürdigen Dokumenten

Außerdem liefert dieser Bereich noch weitere Elemente – zum Beispiel die bei den vertrauenswürdigen Dokumenten schmerzlich vermisste Liste der Elemente. Außerdem finden Sie noch einige Schaltflächen:

- **Neuen Speicherort hinzufügen...**: Öffnet einen Dialog zum Hinzufügen eines neuen vertrauenswürdigen Verzeichnisses, das als vertrauenswürdig eingestuft

Vertrauenswürdigkeit per Makro prüfen

Mit Access 2007 hat Microsoft Techniken zur Prüfung der Vertrauenswürdigkeit einer Datenbank eingeführt, wobei die Datenbank durch den Benutzer als solche klassifiziert werden kann/muss (es gibt auch noch die Möglichkeit der digitalen Signatur, die aber praktisch kaum eingesetzt wird). Leider gibt es unter Access Einstellungen, die das Ausführen von VBA verhindern, dem Benutzer aber andererseits keine Information darüber liefern, dass der Funktionsumfang der Datenbank aufgrund von Restriktionen eingeschränkt ist. Dieser Artikel zeigt, wie Sie dies umgehen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1402_VertrauenswuerdigkeitPerMakro.mdb**.

Datenbank funktioniert nicht

Ist Ihnen das auch schon einmal passiert? Sie schicken einem Kunden eine Datenbank, die auf Ihrem Rechner einwandfrei funktionierte, aber beim Kunden geht plötzlich gar nichts mehr – außer vielleicht das Öffnen des Startformulars? Dies deutet darauf hin, dass der Kunde Sicherheitseinstellungen verwendet, die das Ausführen von VBA-Code komplett unterbinden. Bei den Standardeinstellungen zeigt Access für Datenbanken, die nicht als vertrauenswürdiges Dokument eingestuft wurden und sich auch nicht in einem vertrauenswürdigen Verzeichnis befinden, die gelbe Statusleiste aus Bild 1 an.

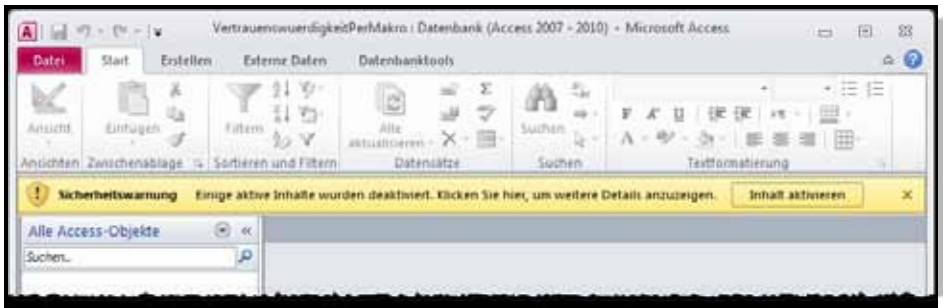


Bild 1: Diese Sicherheitswarnung wird gern übersehen ... oder erscheint erst gar nicht.

Es kann allerdings auch sein, dass ein allzu sicherheitsbedürftiger Admin oder auch Benutzer die Sicherheitseinstellungen in den Access-Optionen so anpasst, dass selbst diese Meldung nicht angezeigt und der enthaltene VBA-Code schlicht nicht ausgeführt wird (Details hierzu finden Sie im Artikel **Vertrauenswürdige Datenbanken**).

Spätestens nach der Lektüre diese Artikels wissen Sie als Entwickler natürlich, was zu tun ist, wenn sich der Benutzer Ihrer Anwendung bei Ihnen meldet und Ihnen mitteilt, dass diese nicht funktioniert. Sie liefern dem Benutzer dann die nötigen Informationen, damit dieser die Datenbankdatei entweder zur Liste der vertrauenswürdigen Dokumente hinzufügt oder das Verzeichnis der Anwendung als vertrauenswürdig kennzeichnet.

Benutzer beim Start informieren

Sie können allerdings auch gleich in der Datenbank eine entsprechende Meldung einbauen und dem Benutzer damit die benötigten Anweisungen übergeben. Es gibt nämlich die Möglichkeit, gewisse Abläufe trotz strengster Sicherheitseinstellungen durchzuführen. Dabei handelt es sich um einige spezielle Makro-Befehle.

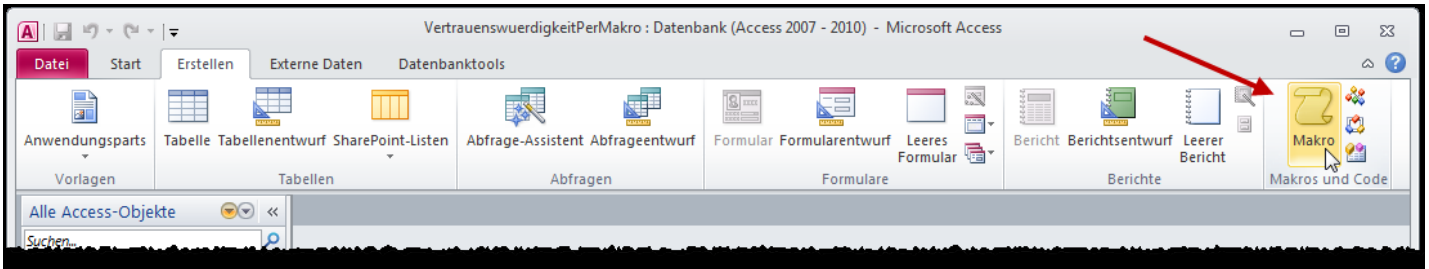


Bild 2: Anlegen eines neuen Makros

Datenmakros II: Datenoperationen

Datenmakros und benannte Makros liefern vor allem die interessante Möglichkeit, grundlegende Datenoperationen durchzuführen – also beispielsweise das Anlegen neuer Datensätze, das Bearbeiten oder Löschen vorhandener Datensätze, das Durchlaufen von Datensatzgruppen oder das Auffinden bestimmter Datensätze. Prinzipiell bilden Sie damit die Basisoperationen der DAO-Bibliothek auf Makro-Ebene ab. Dieser Artikel stellt die Datenoperationen im Detail vor.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1402_Makros.mdb**.

Datenoperationen
Die neuen Datenmakros sollen es vor allem ermöglichen, endlich auch per Makro einfache Datenoperationen auszuführen – also etwa einen neuen Datensatz anzulegen, vorhandene Datensätze zu ändern und zu löschen oder auch einfach auf die Felder eines Datensatzes zuzugreifen.

Diese Operationen bilden Sie typischerweise unter VBA ab – und zwar mit dem **Recordset**-Objekt und Befehlen wie **AddNew**, **Edit**, **Update** oder **FindFirst**.

Zu Testzwecken haben wir eine kleine Tabelle namens **tblKunden** angelegt, die allerdings nur das Primärschlüsselfeld **KundeID**, das Fremdschlüsselfeld **AnredeID** sowie die beiden Textfelder **Vorname** und **Nachname** enthält (siehe Bild 1).

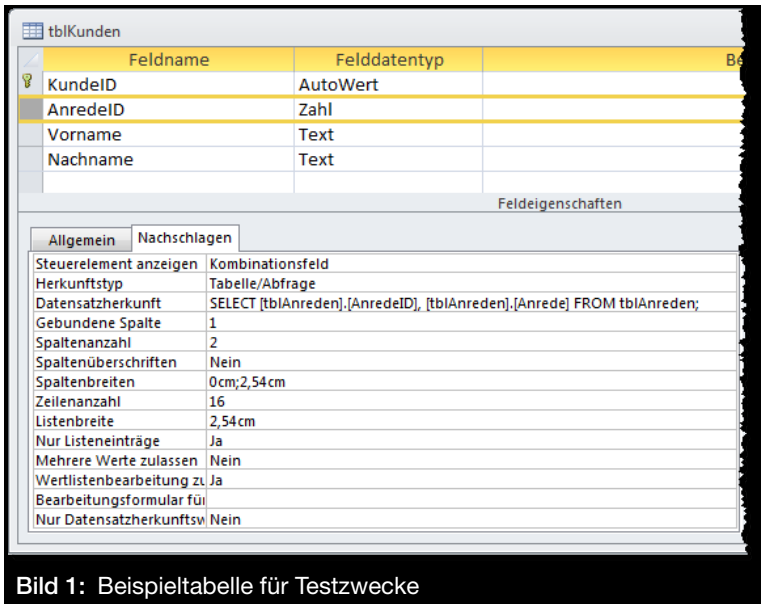


Bild 1: Beispieltabelle für Testzwecke

Wenn Sie nun beispielsweise einen neuen Datensatz in der Tabelle anlegen möchten, benötigen Sie ein benanntes Makro für diese Tabelle. Dieses legen Sie bei in der Datenblattansicht geöffneter Tabelle über den Ribbonbefehl **Tabelle|Benannte Makros|Benanntes Makro|Benanntes Makro erstellen** an.

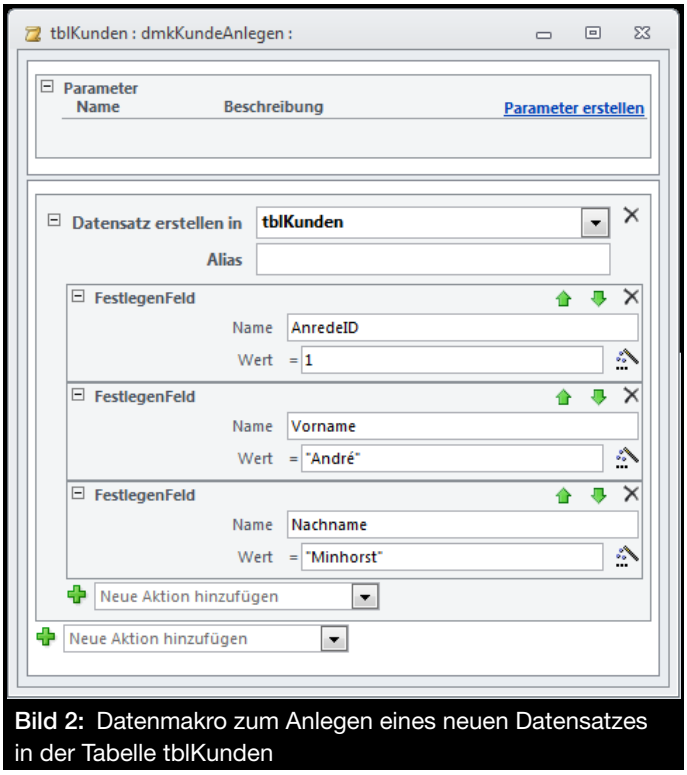


Bild 2: Datenmakro zum Anlegen eines neuen Datensatzes in der Tabelle tblKunden

In diesem Makro fügen Sie nun zunächst den Befehl **Datensatz erstellen in** ein und wählen die Tabelle **tblKunden** als Ziel der Aktion aus.

Dieser Aktion können Sie weitere Befehle unterordnen, und zwar **Gruppieren**, **Kommentar**, **Wenn**, **AbbrechenDatensatzänderung**, **FestlegenFeld** und **FestlegenLokaleVar**.

In diesem Fall wollen wir die drei Felder **AnredeID**, **Vorname** und **Nachname**

mit den entsprechenden Werten füllen. Dies erledigen wir, indem wir drei Mal die Makroaktion **FestlegenFeld** hinzufügen, unter **Name** das Zielfeld auswählen und unter **Wert** den einzutragenden Wert angeben (siehe Bild 2).

Dabei ist zu beachten, dass Sie Literale in Anführungszeichen einfassen. Anderenfalls sucht Access nach einem entsprechenden Ausdruck und löst einen Fehler aus.

Dieses Makro soll nun also einen Datensatz zur Tabelle **tblKunden** hinzufügen. Aber wie rufen wir das Makro auf? Dazu benötigen wir ein weiteres Makro, dem wir die Makroaktion **AusführenDatenmakro** zuweisen. Diese erwartet als Parameter den Namen des aufzurufenden Datenmakros mit führendem Tabellennamen, also hier **tblKunden.dmkKundeAnlegen** (siehe Bild 3). Ein erster Testaufruf des Makros **makKundeAnlegen** bringt das gewünschte Ergebnis – der Datensatz wird angelegt.

Datenmakro mit Parametern

Als Nächstes wollen wir das doch recht statische Makro etwas dynamischer gestalten – und zwar, indem wir die in die Tabelle **tblKunden** einzufügenden Werte per Parameter übergeben.

Dazu erstellen wir ein weiteres benanntes Makro für die Tabelle **tblKunden**. Diesmal klicken wir jedoch zunächst drei Mal auf den Text **Parameter erstellen** oben rechts

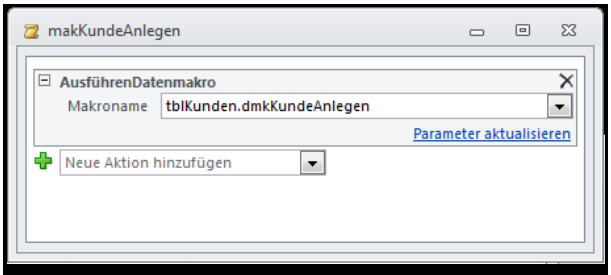


Bild 3: Dieses Makro ruft das Datenmakro zum Anlegen eines Kundendatensatzes auf.

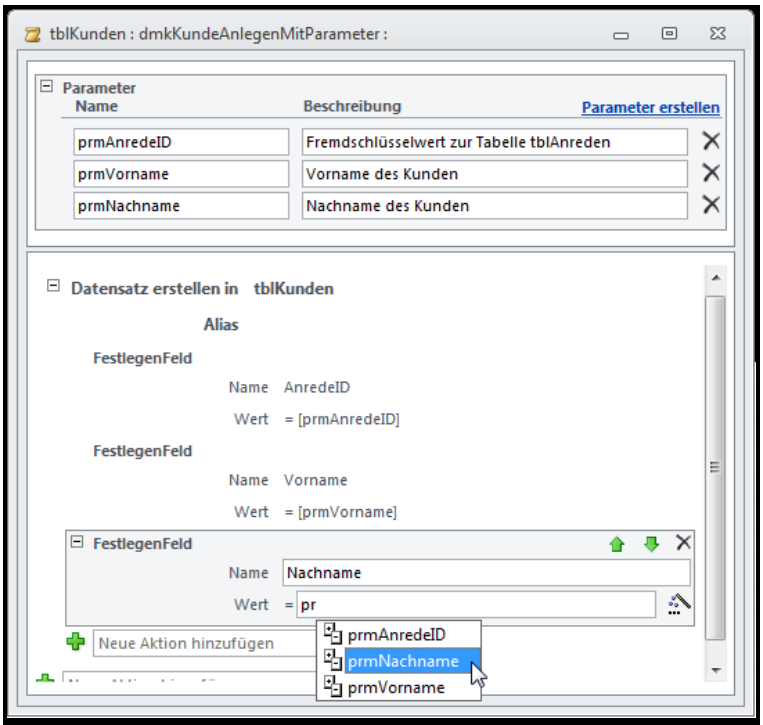


Bild 4: Benanntes Makro mit Parametern

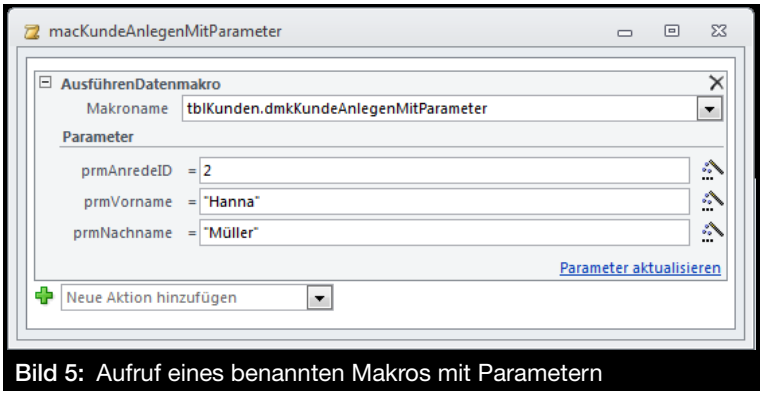


Bild 5: Aufruf eines benannten Makros mit Parametern

im Makro-Entwurf und erzeugen so drei Zeilen zur Eingabe von Parametern. Diese nennen wir **prmAnredeID**, **prmVorname** und **prmNachname**.

Nun fügen Sie wieder die Makroaktion **Datensatz erstellen in** hinzu, legen die Tabelle **tblKunden** als Ziel fest und tragen unterhalb dieser Makroaktion drei Mal den Befehl **FestlegenFeld** ein. Die Feldnamen lauten wieder wie im vorherigen Beispiel. Allerdings geben wir diesmal nicht direkt die einzufügenden Werte an, sondern die Namen der weiter oben definierten Parameter – also etwa den Parameter **prmAnredeID** für das Feld **AnredeID**. Die Parameter lassen sich bequem per IntelliSense aus der Liste der verfügbaren Elemente auswählen (siehe Bild 4).

Nun folgt der zweite Teil: die Definition des Makros, mit dem wir das

Datenmakro aufrufen und diesem die Werte für die Parameter übergeben.

Dazu erstellen Sie ein neues Makro namens **macKundeAnlegenMitParameter** und fügen diesem wieder die Makroaktion **AusführenDatenmakro** hinzu (siehe Bild 5). Diesmal wählen Sie das Makro **tblKunden.dmkKundeAnlegenMitParameter** für den Parameter **Makroname** aus. Nach der Auswahl des auszuführenden Makros erscheinen gleich die drei für das benannte Makro definierten Parameter. Hier geben wir wieder die gewünschten Werte ein (Anführungszeichen bei Zeichenketten nicht vergessen) und führen das Makro dann aus. Es funktioniert – auch dies legt den Datensatz mit den angegebenen Daten an.

Parameter per VBA übergeben

Nun möchten wir aber vielleicht das benannte Makro per VBA aufrufen. Wie übergeben wir die notwendigen Parameter mit der Methode **RunDataMacro**? Der Versuch, dieses einfach ohne Angabe von Parametern aufzurufen, führt direkt zu einem Laufzeitfehler (siehe Bild 6).

Die Antwort lautet: Wir übergeben die Parameter gar nicht mit dieser Methode. Stattdessen legen wir die Parameter vorher fest, und zwar mit einer eigenen **DoCmd**-Methode. Das sieht dann für dieses Beispiel wie in Bild 7 aus. Wichtig ist an dieser Stelle, dass Sie Zeichenketten nicht nur in Anführungszeichen, sondern zusätzlich in Hochkommata einfassen (oder zusätzliche doppelte Anführungszeichen: **"""Maria"""**).

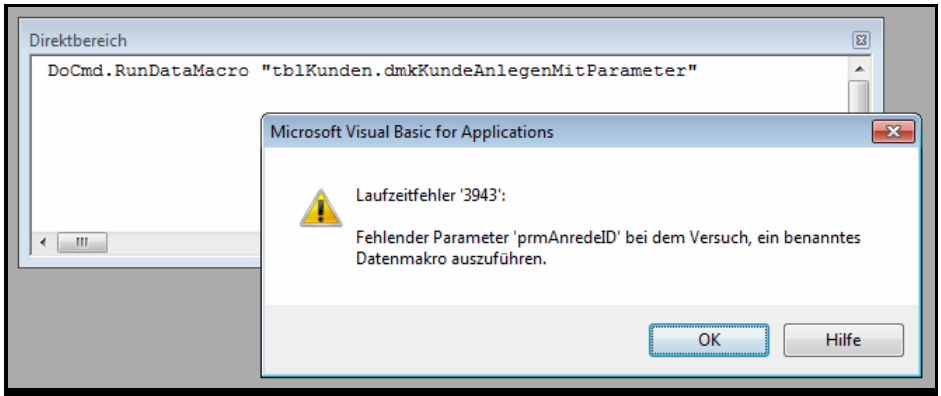


Bild 6: Aufruf eines Datenmakros ohne Angabe der Parameter

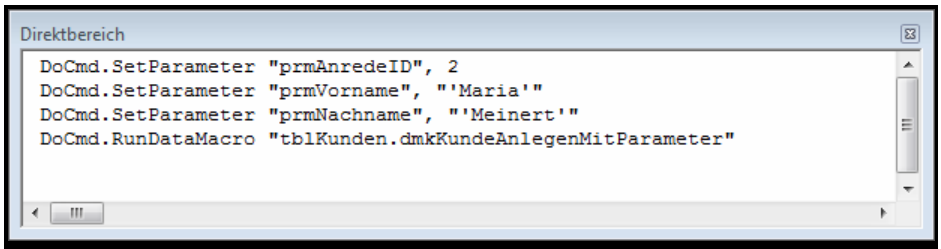


Bild 7: Vorherige Angabe der Parameter

Nach dem Aufruf der drei **DoCmd.SetParameter**-Anweisungen sowie der **RunDataMacro**-Methode wird der neue Datensatz in der Tabelle **tblKunden** angelegt. Anzumerken ist noch, dass die mit **SetParameter** gesetzten Parameter mit dem Aufruf der **RunDataMacro**-Methode geleert werden, sodass ein erneuter Aufruf des benannten Makros einen Fehler auslöst.

DLookup im Makro

Nun erhöhen wir den Schwierigkeitsgrad: Ein neues benanntes Makro soll nur die Anrede, aber nicht den Primärschlüsselwert des entsprechenden Datensatzes in der Tabelle **tblAnreden** als Parameter erhalten. Das heißt, dass das benannte Makro selbst die **AnredeID** für die angegebene Anrede ermitteln muss.

Das ist eigentlich recht einfach – wenn man weiß, auf welche Kleinigkeiten man achten muss. In diesem Fall erstellen Sie wiederum ein neues benanntes Makro für die Tabelle **tblKunden**. Diesmal geben Sie statt des Parameters **prmAnredeID** den Parameter **prmAnrede** an – es soll also nicht mehr direkt die **AnredeID** übergeben werden, sondern die Bezeichnung der Anrede.

Um die **AnredeID** aus der Tabelle **tblAnreden** zu ermitteln, gehen wir wie folgt vor: Zunächst legen Sie im benannten Makro **dmcKundeAnlegenMitAnrede** die Makroaktion **Datensatz nachschlagen in** an. Dieser weisen Sie als Quelle die Tabelle **tblAnreden** zu. Die erste Hürde, die den meisten etwas Kopfzerbrechen bereiten dürfte, ist die Formulierung der Bedingung, die genau den Datensatz mit der angegebenen Anrede liefert. Intuitiv würde man hier einen Ausdruck wie den folgenden unterbringen:

```
="Anrede = '" & prmAnrede & "'"
```

Dieser Versuch misslingt jedoch. An dieser Stelle ist dann guter Rat teuer. Wie soll ich ohne meine gewohnten VBA-Werkzeuge mal eben debuggen und beispielsweise herausfinden, welches Ergebnis mir der hier verwendete Ausdruck liefert? Aber das ist kein Problem: Es gibt ja die Makroaktion **ProtokollierenEreignis**, mit dem wir einen beliebigen Ausdruck in die Tabelle **USysApplicationLog** eintragen können. Also verwenden wir diesen wie in Bild 8 und lassen uns das Ergebnis des ermittelten Ausdrucks in dieser Tabelle ausgeben.

Dabei ist noch anzumerken, dass wir mit einem **Alias** für das mit Datensatz nachschlagen in ermittelte Recordset arbeiten, das wir mit **rstAnreden** bezeichnen. Dadurch können wir über diesen Bezeichner auf die im Recordset enthaltenen Feldwerte zugreifen.

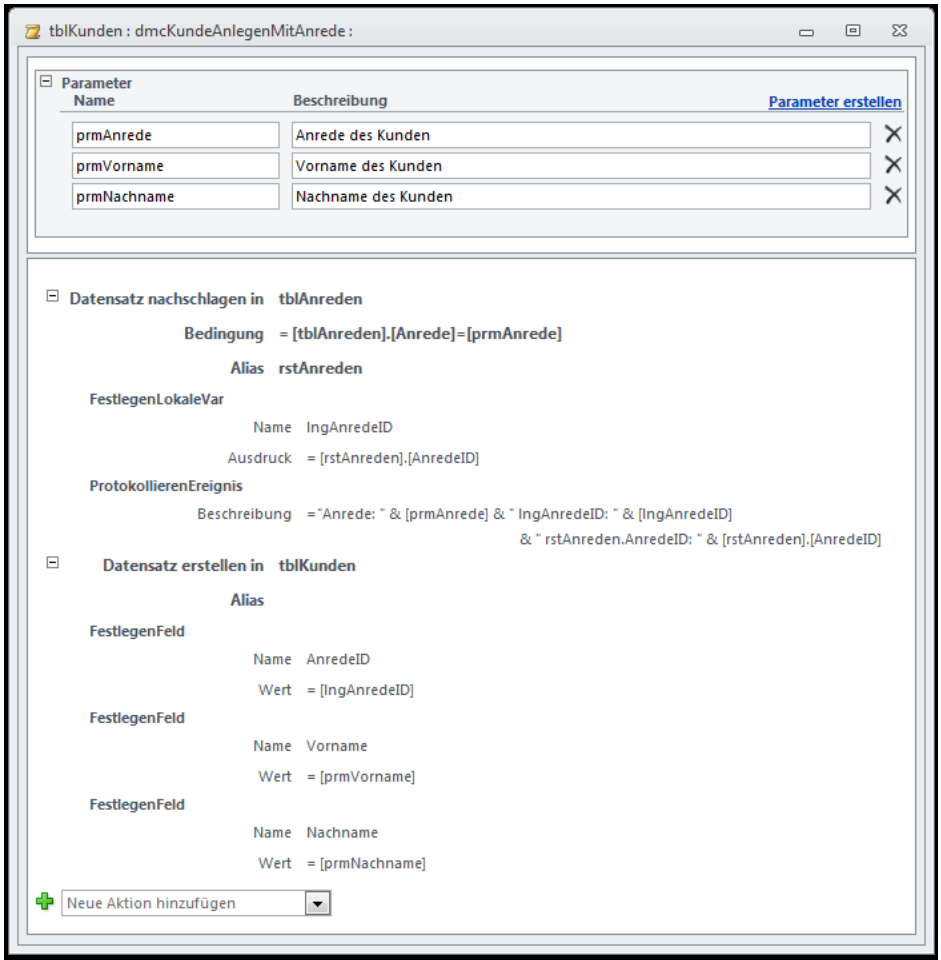


Bild 8: Benanntes Makro inklusive Ermittlung von Lookup-Werten

Nun: Der genannte Ausdruck liefert nicht das gewünschte Ergebnis, stattdessen einfach den ersten Datensatz. Wenn Sie also **Frau** als **Anrede** übergeben, erhalten Sie nicht den Wert **2** aus dem Feld **AnredeID**, sondern den Wert **1** (für **Herr**). Dies ist ein gutes Beispiel dafür, dass niemals nur mit Standardwerten (in diesem Fall den ersten Datensatz der Tabelle **tblAnreden**) testen sollte, da man sonst später nachkorrigieren muss. Der korrekte Ausdruck, um mit Datensatz nachschlagen in tatsächlich die **AnredeID** für den mit Anrede übergebenen Wert zu erhalten, lautet wie folgt:

```
[tblAnreden].[Anrede]=[prmAnrede]
```

Die Vergleichswerte werden also nicht in Abhängigkeit vom Datumstyp etwa in Anführungszeichen eingefasst.

rstAnreden enthält also nun den Datensatz der Tabelle **tblAnreden**, dessen Feld **Anrede** den mit dem Parameter **prmAnrede** übergebenen Wert enthält. Aus diesem Datensatz möchten wir nun den Wert des Feldes **AnredeID** auslesen und beim Anlegen des neuen Datensatzes in der Tabelle **tblKunden** nutzen. Dazu verwenden wir eine lokale Variable, die wir mit der Makroaktion **FestlegenLokaleVar** festlegen. Diese Aktion erwartet zwei Parameter: **Name** erhält den zu verwendenden Variablennamen, in diesem Fall **lngAnredeID**. Ausdruck erwartet den Wert, den die Makroaktion der Variablen zuweisen soll. Dieser Ausdruck lautet wie folgt:

```
[rstAnreden].[AnredeID]
```

Sie können diesen Ausdruck wie die meisten anderen mit IntelliSense zusammenklicken.

Nun müssen wir nur noch den Teil des benannten Makros, der den neuen Datensatz zur Tabelle **tblKunden**

hinzufügt, programmieren. Dieser sieht fast genauso aus wie bei den vorherigen Beispielen – mit dem kleinen Unterschied, dass das Feld **AnredeID** diesmal aus der lokalen Variablen **lngAnredeID** gefüllt wird und nicht mehr mit dem Wert des Parameters **prmAnredeID**.

Das Makro **macKundeAnlegenMitAnrede** zum Ausführen des Datenmakros **dmcKundeAnlegenMitAnrede** sieht schließlich wie in Bild 9 aus.

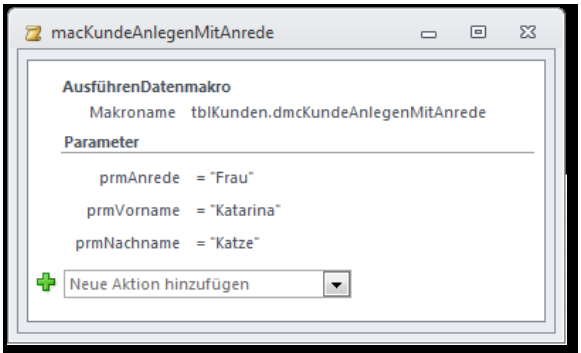


Bild 9: Aufruf des Makros mit der Anrede als Zeichenkette

Neue Lookup-Werte anlegen

Wir gehen noch einen Schritt weiter und nehmen an, dass gelegentlich neue Anreden zur Tabelle **tblAnreden** hinzukommen (das geschieht in der Praxis zwar so gut wie nie, aber wir benötigen noch eine Beispielkonstellation zum Anlegen neuer Lookup-Werte ...).

Dazu erstellen wir zunächst ein neues benanntes Makro für die Tabelle **tblKunden** und speichern es unter dem Namen **dmcKundeAnlegenMitNeuerAnrede**. Öffnen Sie dann das zuvor erstellte benannte Makro **dmcKundeAnlegenMitAnrede**, kopieren Sie den kompletten Inhalt mit den Tastenkombinationen **Strg + A** (um alle Elemente zu markieren) und **Strg + C** (um den markierten Bereich in die Zwischenablage zu kopieren).

Dann legen Sie ein neues Makro an, welches das neue benannte Makro aufruft. Dieses sieht wie in Bild 10 aus und ruft das Makro **dmcKundeAnlegenMitNeuerAnrede** auf. Als Anrede übergibt das Makro dabei mit dem Parameter **prmAnrede** einen noch nicht in der Tabelle **tblAnreden** verwendeten Wert (hier **Herr Dr.**).

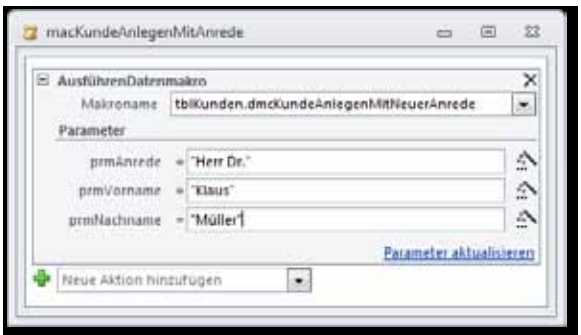


Bild 10: Makro zum Anlegen von Kunden mit noch nicht vorhandener Anrede

Damit wechseln wir zum benannten Makro **dmcKundeAnlegenMitNeuerAnrede** (siehe Bild 11). Hier wurde bisher die **AnredeID** für eine vorhandene Anrede ermittelt. Nun müssen wir zuvor prüfen, ob die mit **prmAnrede** übergebene Anrede überhaupt vorhanden ist und diese gegebenenfalls noch anlegen. Erst dann können wir den Wert des Feldes **AnredeID** in die Variable **lngAnredeID** schreiben und damit den neuen Datensatz in der Tabelle **tblKunden** erstellen.

Wir verwenden dazu wieder die Makroaktion **Datensatz Nachschlagen in**. Nun würden Sie normalerweise versuchen, zu ermitteln, ob diese Makroaktion ein Ergebnis geliefert hat und dann **lngAnredeID** mit den entsprechenden Datensatz füllen – also wie zuvor. Liefert **Datensatz nachschlagen in** kein Ergebnis, soll der Datensatz in der Tabelle **tblAnreden** neu angelegt werden. Dummerweise gibt es keine wirkliche Möglichkeit, zu prüfen, ob **Datensatz nachschlagen in** ein Ergebnis geliefert hat. Die einzige Möglichkeit ist die folgende: Sie legen zuvor mit **FestlegenLokaleVar** einen Wert für **lngAnredeID** fest, der in **tblAnreden** nicht vorkommt. Dies ist bei Verwendung eines inkrementellen Autowertes typischerweise beim Wert **0** der Fall.

Danach rufen wir **Datensatz nachschlagen in** auf. Dann machen wir von der Tatsache Gebrauch, dass die dieser Makroaktion untergeordneten Anweisungen nur ausgeführt werden, wenn ein Ergebnis gefunden wurde. In diesem Fall tragen wir den Wert des Feldes **AnredeID** des gefundenen Datensatzes in die Variable **lngAnredeID** ein. Anschließend können wir dann bequem prüfen, ob **lngAnredeID** den Wert **0** aufweist (was dafür spricht, dass keine passende Anrede gefunden wurde). Ist dies der Fall, müssen wir die Anrede neu anlegen. Dies erledigen wir mit der bereits bekannten Makroaktion Datensatz erstellen in, wobei wir dem Feld **Anrede** den Wert aus **prmAnrede** zuweisen.

Neuen Primärschlüssel ermitteln

Nun müssen wir nur noch den Primärschlüsselwert der Tabelle **tblAnreden** für den neuen Datensatz ermitteln. Dazu gibt es zwei Möglichkeiten:

- Wir kennen ja den Wert des Feldes **Anrede**. Darüber könnten wir mit einer erneuten **Datensatz nachschlagen in**-Makroaktion den entsprechenden Wert des Feldes **AnredeID** ermitteln.

- Es gelingt jedoch auch noch eine Spur eleganter: Bei Verwendung von Makros können Sie mit der Funktion **LetztesErstellenDatensatzID** auf den zuletzt hinzugefügten Primärschlüsselwert zugreifen (entspricht der SQL-Abfrage **SELECT @@IDENTITY**). Diese nutzen wir und tragen das Ergebnis in **lngAnredeID** ein.

Anschließend brauchen wir nur noch den neuen Datensatz in die Tabelle **tblKunden** eintragen, wobei wir gleich die **AnredeID** des neuen Datensatzes der Tabelle **tblAnreden** verwenden.

Datensätze in Schleife durchlaufen

Mit der Makroaktion **Für jeden Datensatz in** können Sie eine **Do While**-Schleife implementieren. Dabei geben Sie als ersten Parameter wieder die Tabelle an, deren Datensätze Sie durchlaufen möchten und tragen für **Bedingung** ein, welche Datensätze dabei berücksichtigt werden sollen – in diesem Fall all diejenigen, für die das Feld **EMail** nicht leer ist. Schließ-

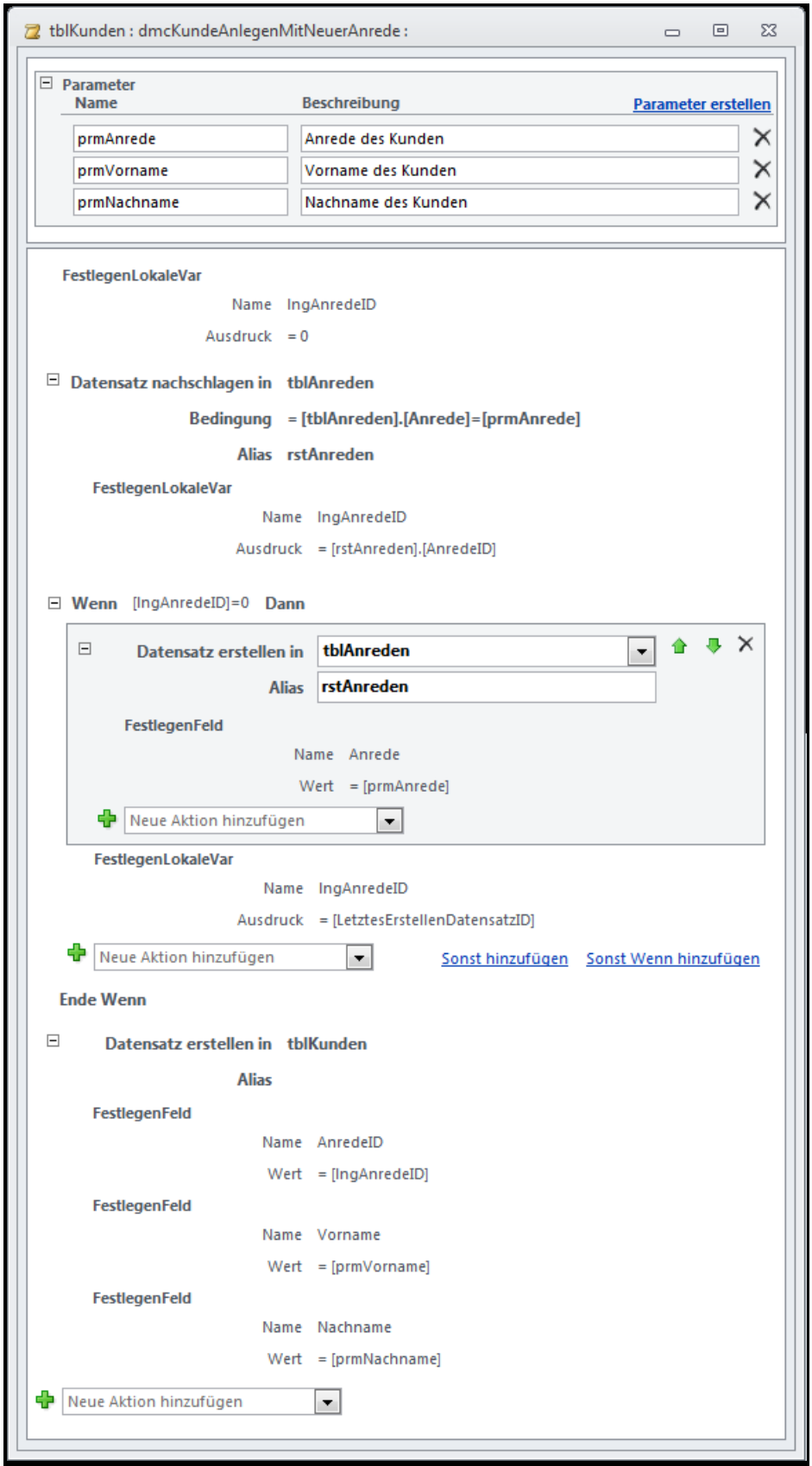


Bild 11: Makro zum Anlegen von Kunden mit noch nicht vorhandener Anrede

TempVars unter VBA

Im Artikel **Programmieren mit Makros** haben Sie bereits die so genannten temporären Variablen, auch **TempVars** genannt, kennen gelernt. Im vorliegenden Artikel vertiefen wir dieses Thema und schauen uns vor allem an, wie Sie die **TempVars** unter VBA nutzen und wie Sie damit die Verbindung zwischen VBA und Makros herstellen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1405_Tempvars.accdb**.

Voraussetzungen

TempVars wurden erst mit Access 2007 eingeführt. Dementsprechend können Sie als Nutzer von Access 2003 und älter nicht auf dieses Feature zugreifen.

Temporäre Variablen

Temporäre Variablen sind eigentlich ein Sonderfall des **Collection**-Objekts.Die Unterschiede sind, dass Sie dieses Objekt nicht erst wie ein **Collection**-Objekt deklarieren und instanzieren müssen, sondern dass es immer und überall bereitsteht.

Warum nun sollte ich einen Wert in einer temporären Variablen speichern und nicht in einer herkömmlichen Variablen?

Dafür gibt es gleich mehrere Gründe:

- Temporäre Variablen werden im Gegensatz zu herkömmlichen Variablen im Falle eines Laufzeitfehlers nicht gelöscht – ein Beispiel folgt später. Sie werden erst beim Schließen der Access-Datei gelöscht (sofern Sie dies nicht vorher per Makro oder Code erledigt haben).
- Temporäre Variablen können Sie sowohl von VBA als auch von Makros aus nutzen.

Temporäre Variablen nehmen Variablen des Typs **Variant** auf, also auch Strings, Zahlen et cetera, aber keine Objekte.

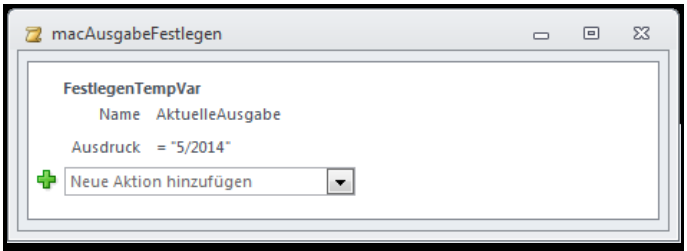


Bild 1: Festlegen einer temporären Variablen per Makro

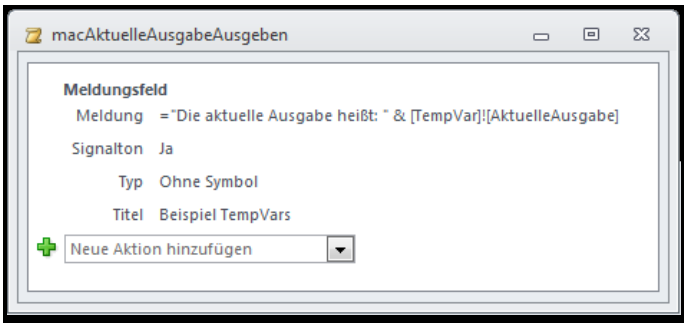


Bild 2: Ausgabe des Inhalts der temporären Variablen

Temporäre Variablen im Makros

Schauen wir kurz auf die Verwendung von temporären Variablen in Makros. Diese legen Sie wie in mit dem Makrobefehl **FestlegenTempVar** fest (siehe Bild 1). In diesem Fall heißt die Variable **AktuelleAusgabe** und wird mit dem Wert **5/2014** gefüllt.

Das Makro **macAktuelleAusgabeAusgeben** sieht wie in Bild 2 aus und gibt den Inhalt der temporären Variablen in einem Meldungsfenster aus.

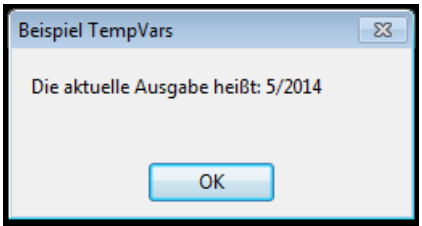


Bild 3: Per Makro angezeigtes Meldungsfenster mit dem Inhalt einer temporären Variablen

Das Ergebnis ist das Meldungsfenster aus Bild 3. Der Wert des Ausdrucks **[TempVar]![AktuelleAusgabe]** erscheint als Teil der auszugebenden Zeichenkette. Wichtig: Der Zugriff auf die Elemente der **TempVar**-Auflistung (die hier unsinnigerweise im Singular anzugeben ist) erfolgt über die Ausrufezeichen-Syntax.

Temporäre Variable per VBA festlegen

Wenn Sie eine temporäre Variable per VBA festlegen möchten, erledigne Sie dies interessanterweise nicht mit einer der Methoden des **DoCmd**-

Objekts, welches ja fast alle in Makros verwendeten Befehle anbietet.

Stattdessen nutzen Sie direkt die **TempVars**-Auflistung, in diesem Fall die **Add**-Methode dieser Auflistung:

```
TempVars.Add ↵
 "AktuelleAusgabe", "5/2014"
```

Eine auf diese Weise festgelegte Variable können Sie auch mit dem soeben definierten Makro **macAktuelleAusgabeAusgeben** anzeigen. Damit ist schon einmal sichergestellt, dass Makros und VBA auf die gleichen temporären Variablen zugreifen.

Per VBA auf eine temporäre Variable zugreifen

Wenn Sie von VBA aus den Wert einer temporären Variablen zugreifen möchten, verwenden Sie die **Item**-Eigenschaft der **TempVars**-Auflistung und übergeben dieser den Namen der benötigten temporären Variablen:

```
MsgBox "Die aktuelle Ausgabe heißt " ↵
    & TempVars.Item("AktuelleAusgabe")
```

Das Ergebnis sieht nun genauso aus wie beim Aufruf des Makros **macAktuelleAusgabeAusgeben**.

Es gibt noch einen weiteren Weg, auf die Inhalte des **TempVars**-Objekts zuzugreifen – und zwar, indem Sie einfach direkt den Namen der Variablen angeben:

```
MsgBox "Die aktuelle Ausgabe heißt " ↵
        & TempVars("AktuelleAusgabe")
```

Die nächste Variante ist der Zugriff über den numerischen Index:

```
Public Sub AlleTempVarsAusgeben()
    Dim objTempVar As TempVar
    For Each objTempVar In TempVars
        MsgBox "Name: " & objTempVar.Name & vbCrLf & "Wert: " & objTempVar.Value
    Next objTempVar
End Sub
```

Listing 1: Ausgeben aller temporären Variablen und ihrer Werte

```
Public Sub AlleTempVarsAusgeben_II()
    Dim i As Integer
    For i = 0 To TempVars.Count - 1
        MsgBox "Name: " & TempVars(i).Name & vbCrLf & "Wert: " & TempVars(i).Value
    Next i
End Sub
```

Listing 2: Durchlaufen aller temporären Variablen per For...Next-Schleife

```
MsgBox "Inhalt der ersten TempVar: " ↵
                    & TempVars(0)
```

Damit greifen Sie aber natürlich nicht mehr gezielt auf eine benannte temporäre Variable zu.

Es gibt einen wichtigen Unterschied beim Zugriff über den Namen der Variablen und über den Index: Wenn Sie über den Namen auf die temporäre Variable zugreifen, wird schlicht eine leere Zeichenfolge zurückgegeben. Wenn Sie es über den Index versuchen und für den Index keine Variable hinterlegt ist, löst dies einen Fehler aus (Fehler **2467**, **In dem von Ihnen eingegebenen Ausdruck wird auf ein Objekt verwiesen, das geschlossen oder nicht vorhanden ist**).

Wert einer temporären Variablen ändern

Den Wert einer temporären Variablen ändern Sie auf zwei Arten – entweder durch direkte Zuweisung oder indem Sie die Variable einfach mit der Add-Methode neu erstellen. Die erste Version sieht so aus:

```
TempVars("AktuelleAusgabe") = "6/2014"
```

Oder so:

```
TempVars(0) = "6/2014"
```

Beim Überschreiben verwenden Sie diesen Befehl:

```
TempVars.Add("AktuelleAusgabe") ↵
                                "6/2014"
```

Wichtig: Nicht nur die letztere Variante legt eine neue temporäre Variable an, wenn diese noch nicht vorhanden ist – auch die einfache Zuweisung erzeugt gegebenenfalls eine neue temporäre Variable.

Alle TempVars durchlaufen

Die einfachste Möglichkeit, die Werte aller temporären Variablen auszugeben, ist das Durchlaufen aller **TempVars** in einer **For Each**-Schleife. Dabei verwenden wir eine Variable des Typs **TempVar**, um die aktuelle temporäre Variable zu referenzieren. Die Prozedur aus Listing 1 zeigt, wie dies funktioniert.

Das **TempVar**-Objekt nimmt jeweils eine temporäre Variable auf und liefert über die beiden Eigenschaften

Programmieren mit Collections

In einem früheren Artikel haben Sie bereits den Umgang mit Arrays kennengelernt. Damit können Sie ein- oder mehrdimensionale Felder von Daten verwalten. Nun schauen wir uns eine weitere Möglichkeit an, um Daten in einer Art Feld unterzubringen: das **Collection**-Objekt. Erfahren Sie, wie Sie ein solches Objekt mit Daten füllen, diese abrufen und welche Unterschiede es zum herkömmlichen Array gibt.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1405_Collections.mdb**.

Collections erstellen

Im Gegensatz zu einem Array, dass einen der Basisdatentypen wie beispielsweise String oder Integer als Datentyp verwenden kann und dann auf diesen Typ begrenzt ist, ist eine **Collection** ein eigenes Objekt. Dementsprechend deklarieren Sie dieses zunächst etwa wie folgt:

```
Dim colKunden As Collection
```

Der Zugriff auf die Eigenschaften und Methoden des Objekts gelingt somit noch längst nicht – die Ausgabe der Anzahl der enthaltenen Elemente mit der **Count**-Eigenschaft, die ja aktuell **0** betragen sollte, führt gleich zu einem Fehler (siehe Bild 1). Der Grund ist einfach: Sie müssen das **Collection**-Objekt erst noch instanzieren.

Dies können Sie in der gleichen Zeile erledigen, welche auch das Objekt deklariert:

```
Dim colKunden As New Collection
```

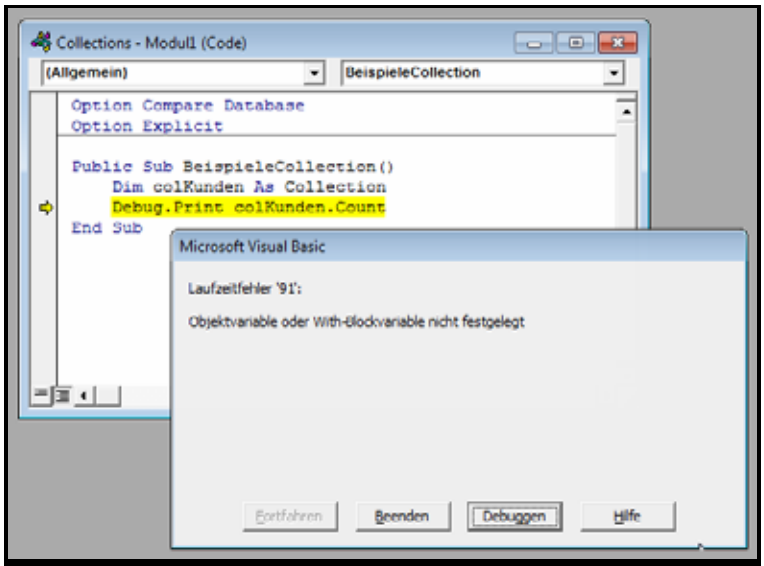


Bild 1: Fehler beim Zugriff auf ein nicht instanziertes Collection-Objekt

Wir haben es allerdings gern etwas kontrollierter und zu dem Zeitpunkt, zu dem wir das Objekt benötigen. Daher instanzieren wir es in einer eigenen Zeile mit der **New**-Anweisung und der anschließenden Zuweisung zu der dafür deklarierten Objektvariablen **colKunden**:

```
Dim colKunden As Collection
Set colKunden = New Collection
Debug.Print colKunden.Count
```

Keine Dimensionierung

Im Gegensatz zu einem Array, dass Sie ja entweder gleich beim Deklarieren dimensionieren oder dynamisch während des Aufbaus des Arrays mit der **ReDim**-Anweisung, brauchen Sie ein Array nicht zu dimensionieren. „Dimensionieren" bedeutet in diesen Zusammenhang, dass Sie nicht zuvor festlegen müssen, wieviele Elemente das Objekt einmal enthalten soll.

Das **Collection**-Objekt bietet die folgenden Methoden und Eigenschaften an:

- **Add**: Fügt ein Element zur **Collection** hinzu. Die Methode erwartet zumindest den Parameter **Item** und somit den hinzuzufügenden Wert und bietet drei weitere optionale Paramter an: **Key**, **Before** und **After**.
- **Count**: Gibt die Anzahl der aktuell im **Collection**-Objekt enthaltenen Elemente aus.
- **Item**: Erlaubt den gezielten Zugriff auf eines der Elemente über den **Index** oder über den **Key**.
- **Remove**: Entfernt ein Element der **Collection**. Das zu entfernende Objekt wird über den Parameter **Item** angegeben – mehr dazu weiter unten.

Einfaches Element zur Collection hinzufügen

Die einfachste Art, ein Element zu einer **Collection** hinzuzufügen, ist die Angabe des Elements als ersten Parameter der **Add**-Methode:

```
colKunden.Add "Kunde 1"
```

Auf diese Weise fügen Sie ein oder mehrere Elemente zur **Collection** hinzu.

Zugriff über den Index

Um die Elemente abzufragen, die Sie auf diese Weise hinzugefügt haben, greifen Sie über die **Item**-Eigenschaft auf das Element mit dem entsprechenden Index zu.

Der Index ist **1**-basiert, sodass Sie den mit der obigen Anweisung hinzugefügten Eintrag mit folgender Anweisung auslesen können:

```
Debug.Print colKunden.Item(1)
```

Element über den Index löschen

Wenn Sie das Element löschen wollen, können Sie dies ebenfalls nur mit der Kenntnis des Indexes des Elements erledigen:

```
colKunden.Remove 1
```

Bislang haben wir nur mit **Collection**-Elementen gearbeitet, die wir ohne Angabe eines Keys angelegt haben.

Damit können wir ausschließlich die Funktion eindimensionaler Arrays nachbilden. Nun schauen wir uns an, wie Sie die **Key**-Eigenschaft nutzen können.

```
Public Sub AlleTempVarsAusgeben()
    Dim objTempVar As TempVar
    For Each objTempVar In TempVars
        MsgBox "Name: " & objTempVar.Name & vbCrLf & "Wert: " & objTempVar.Value
    Next objTempVar
End Sub
```

Listing 1: Ausgeben aller temporären Variablen und ihrer Werte

Anlegen mit Key

Beim Anlegen eines Elements können Sie auch die **Key**-Eigenschaft nutzen. Diese entspricht etwa dem Namen bei einem Name-Wert-Paar.

Dies sieht etwa wie folgt aus:

```
colKunden.Add "André", "Vorname"
colKunden.Add "Minhorst", "Nachname"
```

Wir fügen der **Collection** so zwei Elemente hinzu, die unterschiedliche Keys und Werte haben.

Anlegen per Reihenfolge

Die beiden letzten Parameter der **Add**-Methode, **Before** und **After**, können Sie verwenden, um Elemente vor oder hinter bestehenden Elementen einzufügen.

Im folgenden Beispiel fügen wir beispielsweise erst einen Wert **1** und dann einen Wert **2** zur **Collection** hinzu. Dann sortieren wir hinter dem Element mit dem Index 1 ein weiteres Element ein:

```
Dim col As Collection
Dim var As Variant
Set col = New Collection
col.Add "1"
col.Add "2"
col.Add "Zwischen 1 und 2", , , 1
For Each var In col
    Debug.Print var
Next var
```

Zugriff auf die Werte

Den Wert des Elements mit dem **Key** namens **Vorname** können Sie nun auf folgende Arten ausgeben:

```
Debug.Print colKunden(1)
Debug.Print colKunden.Item(1)
Debug.Print colKunden("Vorname")
Debug.Print colKunden.Item("Vorname")
```

Interessanterweise können Sie aber auch die Ausrufezeichen-Notation verwenden, um über den Key auf ein Element zuzugreifen:

```
Debug.Print colKunden!Vorname
```

Auf den Key zugreifen Einträge ändern

Der Schwachpunkt des **Collection**-Objekts ist, dass Sie die Werte vorhandener Elemente nicht ändern können – also über eine einfache Zuweisung. Dies hier klappt also nicht (Laufzeitfehler **424**, **Objekt erforderlich**):

```
col.Add "Wert", "Name"
col.Item("Name") = "Neuer Wert"
```

Wenn Sie allerdings ohnehin mit dem Key arbeiten, können Sie das betroffene Element ja schnell löschen und dann ersetzen:

```
col.Add "Wert", "Name"
col.Remove "Name"
col.Add "Neuer Wert", "Name"
```

Mehrere Formularinstanzen anzeigen

Die Datensätze einer Tabelle zeigen Sie meist in einer Übersicht wie einem Datenblatt oder einem Listenfeld an. Für die Bearbeitung öffnen Sie den gewünschten Datensatz in einem Detailformular, das die Felder der Tabelle übersichtlich anzeigt. Was aber, wenn Sie einmal mehr als einen Datensatz in der Detailansicht betrachten oder bearbeiten möchten? In diesem Fall hilft die Lösung aus dem vorliegenden Artikel weiter. Wir zeigen Ihnen, wie Sie mehrere Instanzen des gleichen Formulars mit verschiedenen Datensätzen öffnen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1405_Mehrere-Formularinstanzen.mdb**.

Hintergrund

Es kann verschiedene Gründe geben, einmal mehr als einen Datensatz im Detail anzuzeigen. Vielleicht möchten Sie beispielsweise einfach einmal die Daten zweier oder mehrerer Datensätze vergleichen oder diese sogar aneinander anpassen, indem Sie die Feldinhalte von einem Formular zum anderen kopieren.

Vermeintlich einfache Lösung

Der einfachste Weg, mehrere Datensätze gleichzeitig anzuzeigen, ist die Erstellung einiger entsprechender Formulare. Diese erhalten dann unterschiedliche Namen wie etwa **frmArtikel_1**, **frmArtikel_2**, **frmArtikel_3** et cetera. Dies hat aber gleich mehrere Nachteil: Erstens müssen Sie beim Öffnen der Formulare aufpassen, welches schon geöffnet ist, welches der Benutzer vielleicht wieder geschlossen hat und vor allen, welcher Datensatz in welchem Formular angezeigt wird. Der zweite Nachteil ist, dass mehrere gleiche Objekte immer auch einen höheren Wartungsaufwand mit sich bringen: Sollten Sie also etwa vier gleiche Formulare verwenden, müssen Sie eine Änderung entweder immer gleich an allen vier Formularen durchführen oder die mit einem Formular erledigen und die übrigen Formulare löschen und durch die neue Version ersetzen. Der dritte Nachteil ist, dass Sie immer nur eine begrenzte Anzahl von Formularen verwenden können. Gut, dieser Nachteil fällt nicht allzusehr ins Gewicht, denn ab einer bestimmten Anzahl verliert man ja auch den Überblick. Das Fazit lautet jedenfalls: Wir suchen eine bessere Lösung.

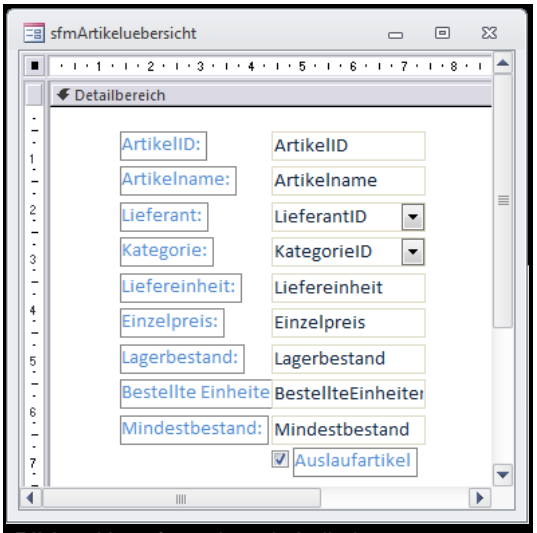


Bild 1: Unterformular mit Artikeln

Bessere Lösung

Unser Ansatz sieht so aus, dass wir nur ein Detailformular erstellen und dieses zur Anzeige verschiedener Datensätze mehrfach öffnen. Darin steckt, wie Sie sich vorstellen können, etwas mehr Hirnschmalz und außerdem gehören dazu einige Spezialkenntnisse, die Ihnen der vorliegende Artikel vermitteln wird.

Wir erstellen zunächst ein Übersichtsformular **frmArtikelUebersicht**, dass die Datensätze in einem Unterformular in der Datenblattansicht anzeigt. Der Benutzer soll nun einen der Datensätze markieren und diesen durch einen Klick auf eine Schaltfläche in einem Detailformular öffnen können.

Das Unterformular, dass die Datensätze der Tabelle **tblArtikel** in der Datenblattansicht anzeigen soll, heißt **sfmArtikeluebersicht** und sieht wie in Bild 1 aus. Stellen Sie die Tabelle **tblArtikel** als **Datenherkunft** des Formulars ein und ziehen Sie alle Felder aus der Feldliste in den Formularentwurf. Legen Sie außerdem für die Eigenschaft **Standardansicht** den Wert **Datenblatt** fest.

Nun erstellen Sie das Hauptformular **frmArtikeluebersicht** und ziehen das Unterformular, nachdem Sie es

gespeichert und geschlossen haben, aus dem Navigationsbereich in die Entwurfsansicht des Hauptformulars. Fügen Sie dem Hauptformular außerdem eine Schaltfläche namens **cmdArtikelAnzeigen** mit der Beschriftung **Artikel anzeigen** hinzu. Das Hauptformular sieht dann wie in Bild 2 aus.

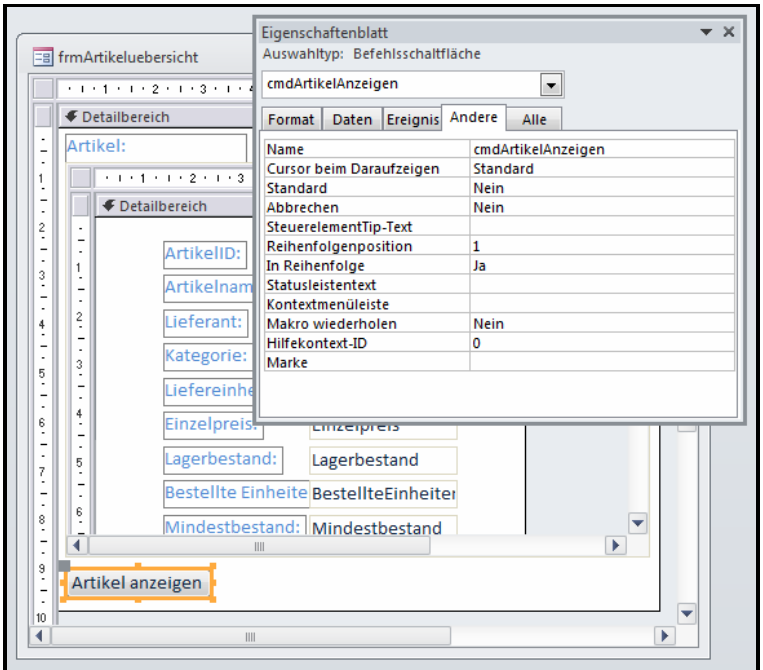


Bild 2: Haupt- und Unterformular zur Auswahl der Detailansichten

Detailformular erstellen

Nun benötigen wir noch das Detailformular namens **frmArtikeldetails**, von dem wir später mehrere Instanzen anzeigen möchten. Dieses füllen Sie ebenfalls mit der Tabelle **tblArtikel** als Datenherkunft. Ziehen Sie auch hier alle Felder der Datenherkunft in den Detailbereich der Entwurfsansicht. Allerdings dürfen Sie hier ein etwas ordentlicher bei der Anordnung sein, denn die Felder sollen ja nicht in der Datenblattansicht erscheinen, sondern so, wie diese auch im Entwurf aussehen – also wie in Bild 3.

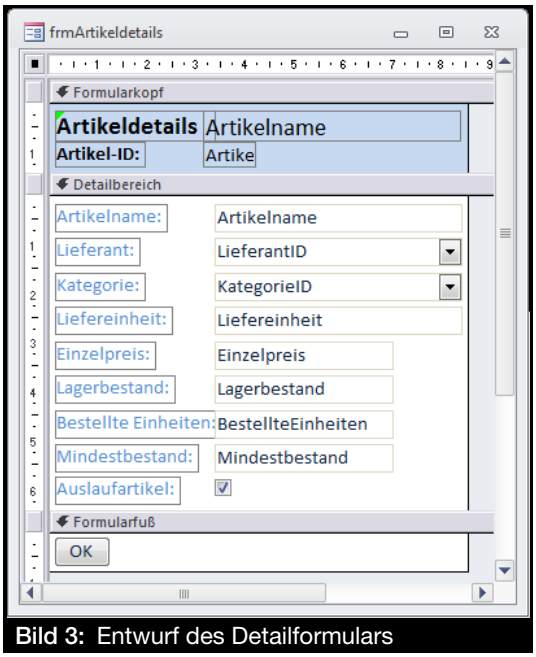


Bild 3: Entwurf des Detailformulars

Dort haben wir den Artikelnamen nochmals an oberster Stelle im Formularkopf untergebracht, damit der Benutzer direkt sieht, um welchen Artikel es sich handelt. Stellen Sie die Eigenschaften **Navigationsschaltflächen**, **Bildlaufleisten** und **Datensatzmarkierer** jeweils auf den Wert **Nein** ein, damit das Formular wie in Bild 4 aussieht.

Herkömmlicher Aufruf

Die Schaltfläche **cmdArtikelAnzeigen** füllen wir nun zunächst mit der üblicherweise verwendeten Methode zum Anzeigen eines Formulars mit einem speziellen Datensatz. Dazu wählen Sie im Eigenschaftsfenster des Formulars **frmArtikeluebersicht** in der Entwurfsansicht den Wert **[Ereignisprozedur]** aus und klicken dann auf die Schaltfläche rechts neben der Eigenschaft.

Dies öffnet den VBA-Editor mit einem neuen Klassenmodul für das Formular **frmArtikeluebersicht**. Dort finden Sie direkt die gewünschte Prozedur **cmdArtikelAnzeigen_Click** vor, die Sie nun wie in Listing 1 mit Code füllen.

Die Prozedur deklariert eine Variable namens **lngArtikelID**, welche den Primärschlüsselwert des aktuell im Unterformular markierten Artikels aufnehmen soll. Die folgende Anweisung

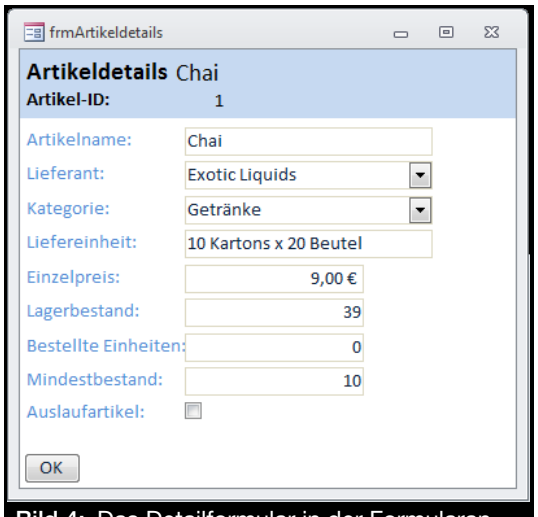


Bild 4: Das Detailformular in der Formularansicht

ermittelt diesen Wert mit dem Ausdruck **Nz(Me!sfmArtikeluebersicht.Form!ArtikelID)**. Der Ausdruck in der Klammer arbeitet sich über das aktuelle Formular (**Me**) über das Unterformular-Steuerelement (**sfmArtikeluebersicht**) zu dem enthaltenen Formular (**Form**) bis zu dem Feld **ArtikelID** des aktuellen Datensatzes durch.

Dabei dürfen Sie nicht das Unterformular-Steuerelement und das Unterformular selbst verwechseln, auch wenn beide den gleichen Namen tragen – das Unterformular selbst referenzieren Sie erst mit der Eigenschaft **Form** des Unterformular-Steuerelements.

Wenn das Unterformular leer ist oder der Benutzer den Datensatzzeiger gerade auf dem leeren, neuen Datensatz platziert hat, liefert dies den Wert **Null** zurück. In diesem Fall sorgt die Funktion **Nz**, welche den obigen Ausdruck umschließt, dafür, dass in diesem Fall die Zahl **0** in der Variablen **lngArtikelID** landet.

Sollte **lngArtikelID** danach einen Wert ungleich **0** enthalten, ruft die Prozedur im ersten Teil einer entsprechend formulierten **If...Then**-Bedingung die Methode **OpenForm** des **DoCmd**-Objekts auf, die zwei Parameter verwendet: Mit **"frmArtikeldetails"** den Namen des zu öffnenden Formulars und mit **WhereCondition:="ArtikelID = " & lngArtikelID** die Bedingung für die Datenherkunft des zu öffnenden Formulars. Letztere lautet nach dem Zusammensetzen etwa so:

```
ArtikelID = 12
```

```
Private Sub cmdArtikelAnzeigen_Click()
    Dim lngArtikelID As Long
    lngArtikelID = Nz(Me!sfmArtikeluebersicht.Form!ArtikelID)
    If Not lngArtikelID = 0 Then
        DoCmd.OpenForm "frmArtikeldetails", WhereCondition:="ArtikelID = " & lngArtikelID
    Else
        MsgBox "Wählen Sie zunächst einen Artikel aus der Liste aus."
    End If
End Sub
```

Listing 1: Öffnen des Formulars **frmArtikeldetails**

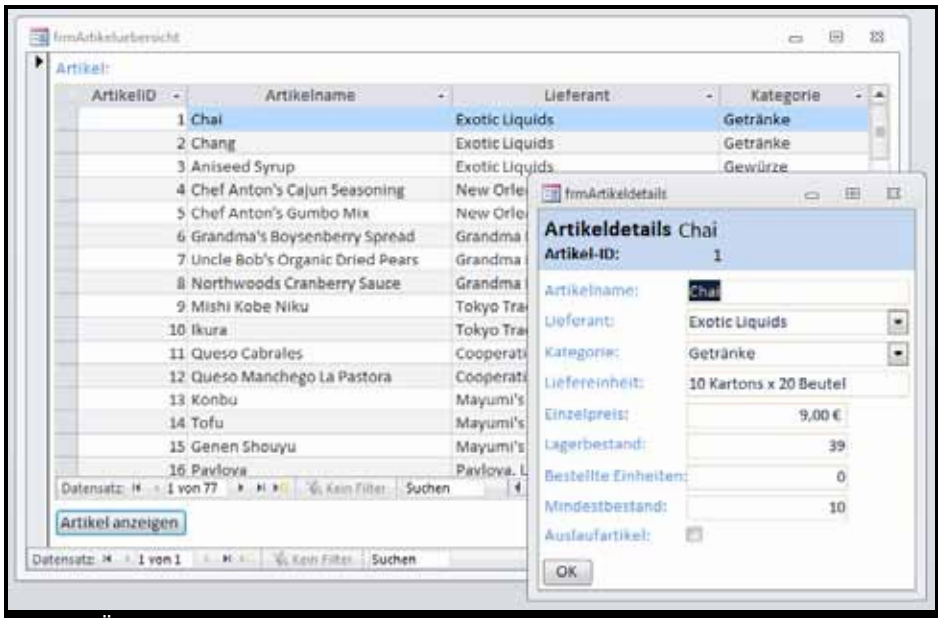


Bild 5: Öffnen eines Detailformulars für den aktuell markierten Artikel

Sollte sich der Datensatzzeiger im Unterformular gerade nicht auf einem Datensatz befinden, erhält **lngArtikelID** den Wert **0**. Dies führt zum **Else**-Teil der **If...Then**-Bedingung, in dem die Prozedur lediglich eine Meldung ausgibt, dass aktuell kein Datensatz markiert ist.

Das Ergebnis sehen Sie in Bild 5: Die Prozedur öffnet das Formular **frmArtikeldetails** und zeigt den aktuell im Unterformular markierten Datensatz an.

Probieren wir nun aus, was geschieht, wenn wir ein zweites Mal auf die Schaltfläche **cmdArtikelAnzeigen** klicken. Das Formular **frmArtikeldetails** flackert kurz, dann zeigt es den aktuell im Unterformular ausgewählten Datensatz an. Wir können also immer wieder einen anderen Datensatz anzeigen, ohne das Detailformular zu schließen, aber nicht mehrere Artikel gleichzeitig.

Mehrere Formulare, erster Versuch

Um mehrerer Detailformulare zu öffnen, gehen wir einen anderen Weg. Dazu bedarf es zunächst einiger Erläuterungen. Genau wie beispielsweise das **Recordset**-Objekt ist auch ein Formular nichts anderes als ein Objekt, basierend auf einer Klasse.

Sie können mehrere **Recordset**-Objekte erzeugen, und dies gelingt auch mit Formularen – allerdings nicht mit der oben verwendeten Methode **DoCmd.OpenForm**.

Während wir damit eine Instanz des Formulars öffnen, die ohne weiteres Zutun existiert, müssen wir beim expliziten Erzeugen eines Formular-Objekts erst eine Objektvariable anlegen, die einen Verweis auf das Formular speichert – anderenfalls würde das Formular gleich wieder im Nirvana verschwinden.

Diese Variable deklarieren wir sinnvollerweise direkt im Kopf des Klassenmoduls **Form_frm-Artikeluebersicht**, damit diese nicht gleich beim Beenden der Prozedur, die das Formular erzeugt und die Variable füllt, gelöscht wird – und damit auch das Formular.

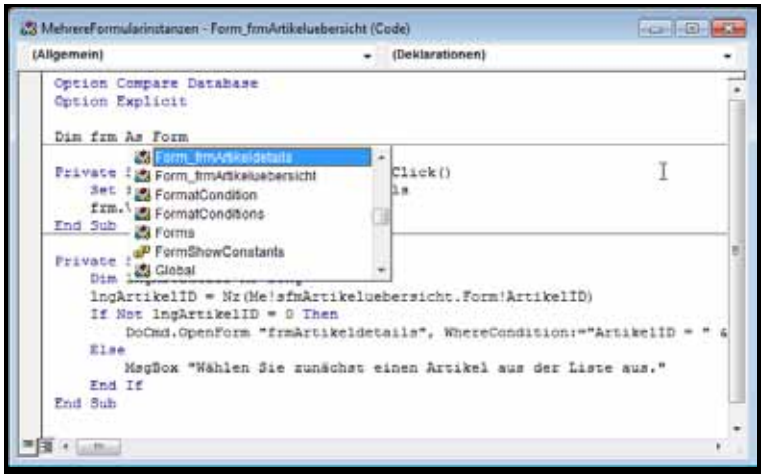


Bild 6: Datentypen für Formularklassen

```
Private Sub cmdMehrereArtikelAnzeigen_Click()
    Set frm = New Form_frmArtikeldetails
    frm.Visible = True
End Sub
```

Listing 2: Erzeugen eines Objekts auf Basis der Klasse **Form_frm-Artikeldetails**

```
Private Sub cmdArtikelAnzeigenII_Click()
    Dim lngArtikelID As Long
    lngArtikelID = Nz(Me!sfmArtikeluebersicht.Form!ArtikelID)
    If Not lngArtikelID = 0 Then
        Set frm = New Form_frmArtikeldetails
        With frm
            .Visible = True
            .Filter = "ArtikelID = " & lngArtikelID
            .FilterOn = True
        End With
    Else
        MsgBox "Wählen Sie zunächst einen Artikel aus der Liste aus."
    End If
End Sub
```

Listing 3: Neues Formular mit dem richtigen Artikel anzeigen

Nun wissen wir aber noch gar nicht, welchen Datentyp ein solches Formular hat. Die allgemeine Klasse **Form**, mit der Sie Formulare auch referenzieren können, hilft uns an dieser Stelle nicht weiter.

Der Grund ist: Wir möchten ja ein neues Objekt auf Basis dieser Klasse erstellen. VBA wüsste aber beim allgemeinen Typ **Form** nicht, welches Formular überhaupt gemeint ist.

Die Lösung ist die Verwendung des Namens des Klassenmoduls als Datentyp, also beispielsweise **Form_frmArtikeldetails**. Dieses Element finden Sie im VBA-Editor ganz einfach mithilfe von IntelliSense (siehe Bild 6).

Die Deklarationszeile im Kopf des Klassenmoduls des aufrufenden Formulars **frmArtikeluebersicht** sieht also so aus:

```
Dim frm As Form_frmArtikeldetails
```

Nun fügen wir dem Übersichtsformular eine weitere Schaltfläche namens **cmdArtikelAnzeigenII** hinzu und füllen diese wie in Listing 2. Die Prozedur erzeugt eine neue Instanz der Klasse **Form_frmArtikeldetails** und speichert diese in der Variablen **frm**. Außerdem stellt sie die Eigenschaft **Visible** auf den Wert **True** ein, da das Formular sonst gar nicht sichtbar wäre.

Allein der erste Test enttäuscht: Wir erzeugen durch mehrfaches Anklicken der Schaltfläche **Artikel anzeigen II** nach wie vor immer nur ein einziges Formular und zweitens zeigt dieses immer den gleichen Datensatz an – nämlich den ersten Datensatz der Datenherkunft.

Also erweitern wir die Prozedur etwas – und zwar wie Listing 3. Dort fügen wir zunächst die Ermittlung des anzuzeigenden Datensatzes ein. Nach dem Einblenden des Formulars stellen wir dessen Eigenschaft **Filter** auf den gleichen Audruck ein,

Ereignisse in Formularen: Zeitgeber

Immer wieder kann es notwendig werden, in Formularen in regelmäßigen Abständen bestimmte Aktionen durchzuführen – oder einfach nur eine bestimmte Zeit nach dem Öffnen verstreichen zu lassen, bevor eine weitere Aktion erfolgt. Dieser Artikel zeigt die Grundlagen der Programmieren von zeitgesteuerten Ereignissen in Formularen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1405_EreignisseZeitgeber.mdb**.

Zeitgeber einschalten

Um den Zeitgeber eines Formulars (in der Beispieldatenbank **frmZeitgeber**) einzuschalten, müssen Sie lediglich die Eigenschaft **Zeitgeberintervall** des Formulars auf einen Zahlenwert einstellen, welcher der Anzahl der Millisekunden bis zum nächsten Auslösen des entsprechenden Ereignisses entspricht. Wenn Sie also etwa nach dem Öffnen eines Formulars 60 Sekunden verstreichen lassen wollen, legen Sie für die Eigenschaft **Zeitgeberintervall** den Wert **60.000** fest.

Damit schaut Access nun nach 60 Sekunden nach, ob Sie auch eine Prozedur für das Ereignis **Bei Zeitgeber** hinterlegt haben. Dazu legen Sie für diese Eigenschaft den Wert **[Ereignisprozedur]** fest und klicken auf die Schaltfläche mit den drei Punkten (siehe Bild 1). Im VBA-Editor finden Sie dann direkt die entsprechende Ereignisprozedur **Form_Timer** vor:

```
Private Sub Form_Timer()

End Sub
```

Wenn Sie hier etwa die folgende Anweisung einfügen, erscheint **60** Sekunden nach dem Öffnen das entsprechende Meldungsfenster:

```
MsgBox "Zeitgeber"
```

Zeitgeber ausschalten

Es gibt zwei Möglichkeiten, den Zeitgeber auszuschalten:

- Sie schließen das Formular oder
- Sie stellen den Wert der Eigenschaft **Zeitgeberintervall** auf **0** ein.

Letzteres werden Sie nicht mehr im Eigenschaftsfenster durchführen, da dies ja in der Regel zur Laufzeit geschehen soll. Also erledigen Sie das per VBA-Code. In unserem Beispiel soll der Zeitgeber mit Erscheinen des Meldungsfensters deaktiviert werden. Die **Form_Timer**-Prozedur sieht dann so aus (stellen Sie zuvor die Eigenschaft **Zeitgeberintervall** auf **1.000** ein, damit Sie nicht jedes Mal eine Minute warten müssen):

```
Private Sub Form_Timer()
    If MsgBox("Zeitgeber fortsetzen?", 4 +
                vbYesNo) = vbNo Then
        Me.TimerInterval = 0
    End If
End Sub
```

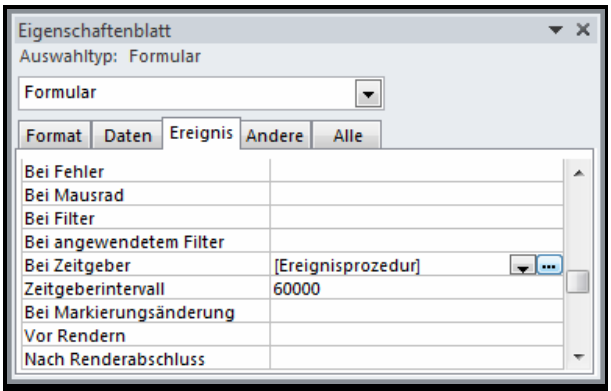


Bild 1: Anlegen eines Zeitgebers

In diesem Beispiel wird der Zeitgeber angehalten, während das Meldungsfenster erscheint – es wird also nicht etwa jede Sekunde ein neues Meldungsfenster eingeblendet. Wenn Sie im Meldungsfenster **Nein** anklicken, wird der Zeitgeber angehalten, indem **TimerInterval** auf **0** eingestellt wird.

Zeitgeber neu starten

Wenn Sie den Zeitgeber beispielsweise durch einen Mausklick auf eine Schaltfläche neu starten möchten, müssen Sie einfach nur das Zeitgeberintervall über die VBA-Eigenschaft **TimerInterval** auf den gewünschten Wert einstellen:

```
Private Sub cmdZeitgeberNeuStarten_
                                Click()
    Me.TimerInterval = 1000
End Sub
```

Beispiele zu Anwendungen finden Sie in den zukünftigen Ausgaben von **Access [basics]**.

Detailformulare aufrufen

Access-Anwendungen enthalten oft Übersichtsformulare, die alle Datensätze (oder eine Auswahl davon) in Listenform anzeigen, also beispielsweise in einem Unterformular in der Datenblattansicht oder in einem Listenfeld. Dieses ist dann üblicherweise mit einer Schaltfläche ausgestattet, um die Details zu einem solchen Datensatz in einem entsprechend gestalteten Formular anzuzeigen. Dieser Artikel zeigt, wie Sie das Öffnen eines solchen Formulars programmieren.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1406_DetailformulareAufrufen.mdb**.

Übersichts- und Detailformular

Bei den folgenden Beispielen gehen wir davon aus, dass Sie ein Formular verwenden, dass die Daten einer Tabelle entweder in einem Unterformular, einem Listenfeld oder auch in der Endlosansicht anzeigt.

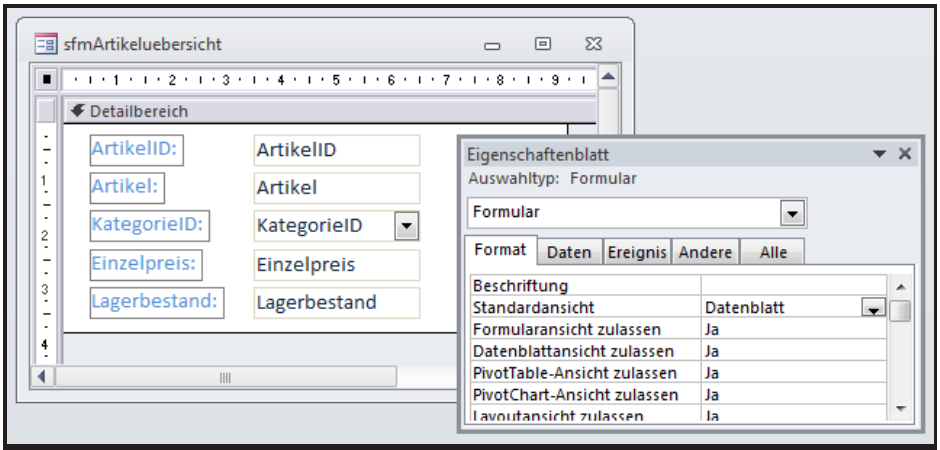


Bild 1: Unterformular mit der Liste der Artikel

Wenn der Benutzer einen der Datensätze auswählt und dann etwa auf eine Schaltfläche mit der Beschriftung **Details anzeigen** klickt, soll der aktuelle Datensatz in einem Detailformular angezeigt werden.

Das Detailformular besteht aus einem Hauptformular namens **frmArtikeluebersicht** und dem Unterformular **sfmArtikeluebersicht**.

Das Unterformular verwendet die Tabelle **tblArtikel** als Datenherkunft und zeigt seine Daten in der Datenblattansicht an (siehe Bild 1).

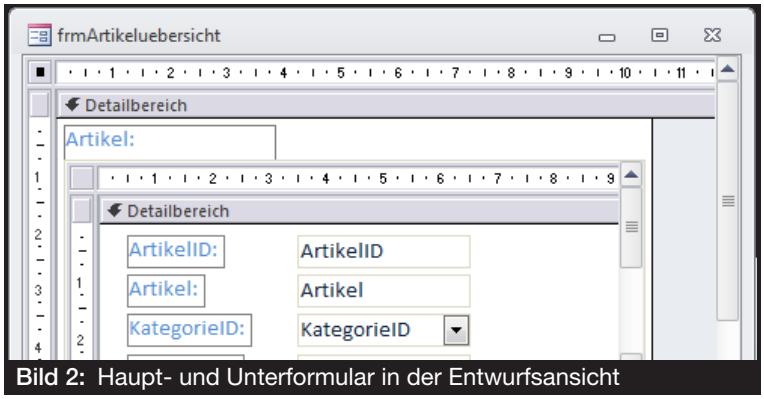


Bild 2: Haupt- und Unterformular in der Entwurfsansicht

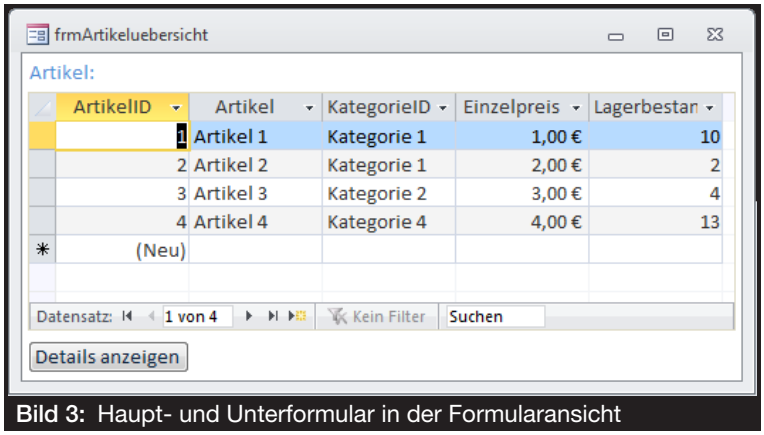


Bild 3: Haupt- und Unterformular in der Formularansicht

Das Hauptformular **frmArtikeluebersicht** nimmt das Unterformular auf und zunächst eine Schaltfläche namens **cmdDetailsAnzeigen** (siehe Bild 2). Wenn Sie die Eigenschaften **Navigationsschaltflächen**, **Datensatzmarkierer** und **Bildlaufleisten** des Hauptformulars auf **Nein** und **Automatisch zentrieren** auf **Ja** einstellen, erscheint das Formular in der Formularansicht wie in Bild 3.

Das Formular zur Anzeige der Artikeldetails heißt **frmArtikeldetails** und verwendet die Tabelle **tblArtikel** als

Datenherkunft – genau wie das Unterformular **sfmArtikeluebersicht**. Es enthält alle Felder der Tabelle **tblArtikel** sowie eine Schaltfläche namens **cmdOK**, um das Formular wieder zu schließen (siehe Bild 4).

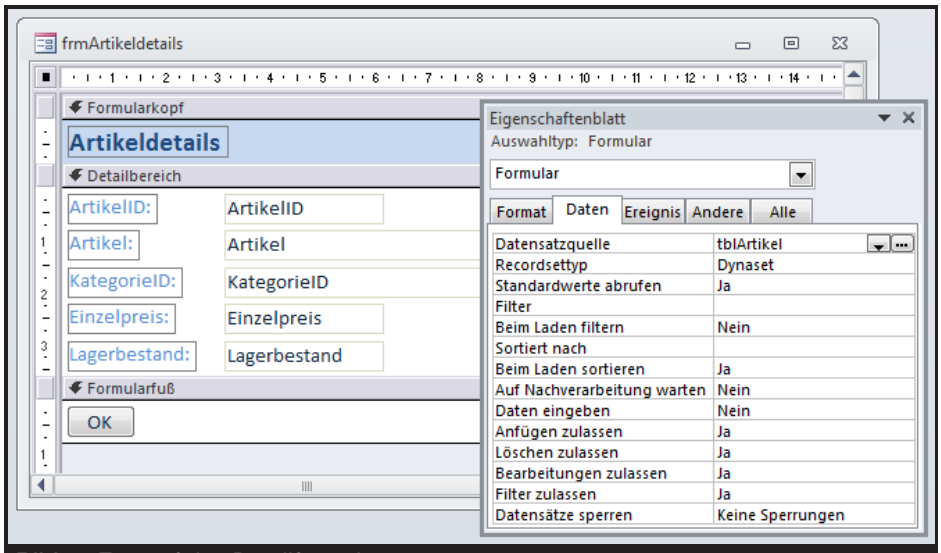


Bild 4: Entwurf des Detailformulars

Datensatz-ID per WhereCondition

Eine Art, eine Information von einem Formular zum nächsten zu übergeben, ist die Übergabe einer Bedingung per **WhereCondition**. Wenn Sie einen Datensatz im Unterformular des Formulars **frmArtikeluebersicht** markieren und dann auf die Schaltfläche mit dem Text **Details anzeigen** klicken, soll dies die Prozedur Listing 1 auslösen.

```
Private Sub cmdDetailsAnzeigen_Click()
    Dim lngArtikelID As Long
    lngArtikelID = Nz(Me!sfmArtikeluebersicht.Form!ArtikelID)
    If Not lngArtikelID = 0 Then
        DoCmd.OpenForm "frmArtikeldetails", WhereCondition:="ArtikelID = " & lngArtikelID
    Else
        MsgBox "Markieren Sie zunächst einen Artikel."
    End If
End Sub
```

Listing 1: Aufrufen des Detailformular mit entsprechendem Filter

Diese Prozedur speichert zunächst den Primärschlüsselwert des aktuell im Unterformular markierten Datensatzes in der Variablen **lngArtikelID**. Diesen ermittelt sie über den Ausdruck **Me!sfmArtikeluebersicht.Form!ArtikelID**. **Me** referenziert das aktuelle Formular, **sfmArtikeluebersicht** das Unterformular-Steuerelement, die Eigenschaft **Form** das darin enthaltene Unterformular (das ebenfalls **sfmArtikeluebersicht** heißt) und **ArtikelID** das Feld mit dem betreffenden Wert.

Befindet der Datensatzzeiger sich aktuell auf einem leeren, neuen Datensatz, liefert dieser Ausdruck den Wert **Null**. Wenn wir versuchen, diesen Wert in einer **Long**-Variablen wie **lngArtikelID** zu speichern, löst dies einen Fehler aus.

Also wandeln wir den Ausdruck zuvor vorsichtshalber mit der Funktion **Nz** in den Zahlenwert **0** um. Dies geschieht nur in dem Fall, dass der als erster Parameter der Funktion angegebene Ausdruck den Wert **Null** enthält – andernfalls liefert **Nz** den Wert aus dem ersten Ausdruck.

Die folgende **If...Then**-Bedingung prüft, ob **lngArtikelID** den Wert **0** enthält. Falls nicht, ruft sie das Formular **frmArtikeldetails** mit dem Wert **"ArtikelID = " & lngArtikelID** für den Parameter **WhereCondition** auf (siehe Bild 5).

Im Falle des Wertes **0** für die Variable **lngArtikelID** erscheint lediglich eine Meldung, dass der Benutzer zunächst einen der Einträge im Unterformular markieren soll.

Wenn der Benutzer nun zum Formular **frmArtikeluebersicht** zurückkehrt, einen anderen Datensatz markiert und erneut auf die Schaltfläche **cmdDetailsAnzeigen** klickt, wird das Formular **frmArtikeldetails** nicht erneut geöffnet. Stattdessen wird nur der Filter aktualisiert, was sich in der Anzeige des gewünschten Datensatzes äußert.

Filter auslesen

Wie sich die Übergabe eines Filterausdrucks mit dem Parameter **WhereCondition** auswirkt, haben wir ja nun gesehen. Was aber geschieht intern? Dies ermitteln wir, indem wir einmal über das Direktfenster auf die Eigenschaften des Formulars **frmArtikeldetails** zugreifen. Den Direktbe-

Datenbankereignis beim Schließen

Access bietet die Möglichkeit, beim Start ein Makro namens **AutoExec** auszuführen oder ein als Startformular angegebenes Startformular anzuzeigen. Damit lassen sich gleich bei Start notwendige Aktionen durchführen. Beim Schließen einer Datenbank sieht es anders aus: Hier stellt Microsoft keine eingebaute Möglichkeit bereit, um abschließende Arbeiten durchzuführen. Dazu gehört beispielsweise das Speichern von während der Benutzung geänderten Konfigurationsdaten. Dieser Artikel zeigt, wie Sie diese Funktion nachrüsten.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1405_DatenbankereignisBeimSchliessen.mdb**.

Dass beim Öffnen einer Datenbank Aktionen ausgeführt werden, ist selbstverständlich. Aber manchmal ist es auch nötig, das Schließen einer Datenbank abzufangen – und wenn Sie auch nur aus irgendeinem Grund fragen möchten: **Wollen Sie die Anwendung wirklich beenden?**

Vielleicht haben Sie aber auch einen besseren Grund – beispielsweise könnten Sie so Informationen, die Sie während der Anwendung in globalen Variablen oder in TempVars speichern, um nicht ständig auf die Optionentabelle zugreifen zu müssen, wieder zurückschreiben.

Wie auch immer: Zunächst einmal müssen herausfinden, wie man den Zeitpunkt des Schließens der Anwendung ermittelt. Tatsache ist nämlich, dass Access keine solche Funktion wie etwa beim Starten einer Anwendung bereitstellt.

AutoClose über Umwege

Da es keine eingebaute Möglichkeit gibt, prüfen wir einmal, welchen Weg wir wählen können. Gibt es überhaupt andere Ereignisse, die man etwa mit dem Access-Anwendungsobjekt assoziieren könnte? Also so etwas wie Access_Close? Nein: Access bietet keine solchen Möglichkeiten.

Aber wer sagt denn, dass wir uns in diesem Fall an Access selbst halten müssen? Vielleicht löst Access ja ein Ereignis eines anderen Objekts aus, wenn man es schließt ... und das ist unser Ansatzpunkt: Wenn beispielsweise bei Schließen von Access noch ein Formular geöffnet ist, dass wird dieses natürlich auch geschlossen. Außer, Sie haben Änderungen am Entwurf vorgenommen – dann fragt Access noch, ob Sie die Änderungen speichern wollen. Wir wollen uns aber ohnehin eher auf die Ereignisse konzentrieren, und dort bieten Formulare beim Schließen gleich mehrere Ereignisse, zum Beispiel

- **Beim Schließen** und
- **Beim Entladen**.

Legen wir also für beide Ereignisse testweise je eine Ereignisprozedur an, indem wir die entsprechenden Eigenschaften **Beim Schließen** und **Beim Entladen** auf **[Ereignisprozedur]** einstellen und dann auf die Schaltfläche mit den drei Punkten auf der rechten Seite klicken (siehe Bild 1).

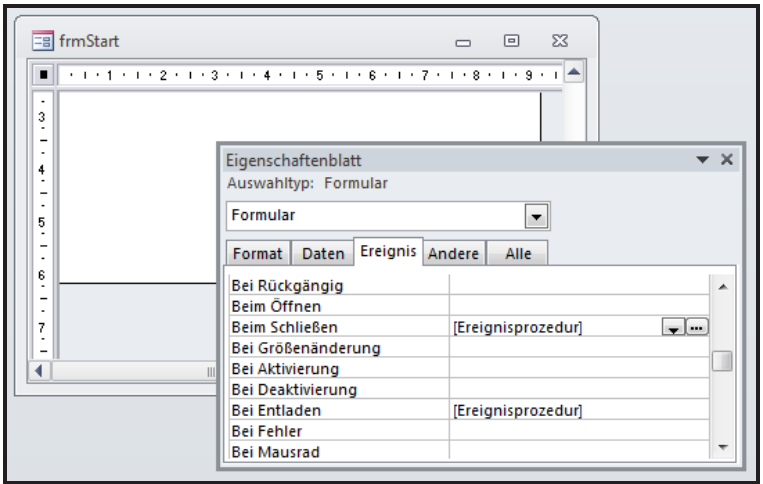


Bild 1: Ereignisse anlegen, die beim Schließen eines Formulars ausgelöst werden

Zunächst wollen wir testen, in welcher Reihenfolge die beiden Ereignisse ausgelöst werden. Dazu legen Sie für beide Prozeduren je einen Haltepunkt im VBA-Editor fest (Sie können auch Meldungsfenster einfügen, wenn Sie möchten). Dann öffnen Sie das Formular in der Formularansicht und schließen es. Schnell

wird deutlich, dass **Beim Entladen** zuerst aufgerufen wird, erst dann folgt **Beim Schließen** (siehe Bild 2).

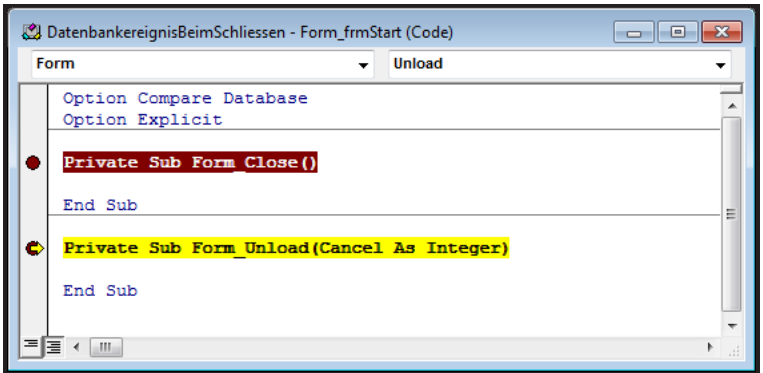


Bild 2: Haltepunkte der beiden beim Schließen ausgelösten Prozeduren im Test

```
Private Sub Form_Unload(Cancel As Integer)
    If MsgBox("Wirklich schließen?", vbYesNo) = vbNo Then
        Cancel = True
    End If
End Sub
```

Listing 1: Diese Prozedur wird beim Schließen des Formulars ausgelöst.

Wenn wir uns die beiden Ereignisse ansehen, fällt ein für unsere Anforderungen gravierender Unterschied auf: Die Ereignisprozedur **Form_Unload** (**Beim Entladen**) besitzt einen Parameter namens **Cancel**. Wenn Sie diesen auf True einstellen, wird das Schließen des Formulars abgebrochen! Das ist wichtig für unser Vorhaben, wenn wir die Möglichkeit haben wollen, das Schließen der Anwendung zu verhindern.

Probieren wir dies aus, indem wir die die Prozedur wie in Listing 1 erweitern. Beim Schließen des Formulars erscheint das entsprechende Meldungsfenster. Wenn Sie dort auf **Abbrechen** klicken, wird das Formular nicht geschlossen. Klicken Sie auf **OK**, wird der Vorgang wie üblich mit dem Schließen des Formulars abgeschlossen. Damit steht nun fest, welches Ereignis uns in diesem Fall hilft.

Aber funktioniert dies auch, wenn wir nicht das Formular schließen, sondern die Anwendung – wobei das Formular natürlich geöffnet sein muss? Probieren wir es aus, stellen wir schnell fest, dass auch dies gelingt. Nun haben wir nur noch ein Problem: Wir müssen dafür sorgen, dass das Formular gleich beim Öffnen der Anwendung geöffnet wird und dass der Benutzer es während der Arbeit mit der Anwendung nicht schließen kann. Dies stellen wir am einfachsen sicher, indem wir das Formular einfach unsichtbar machen!

Sicherungsformular beim Start öffnen

Also legen wir einen Mechanismus an, der das Formular bei Öffnen der Anwendung gleich mit öffnet – und dieses möglichst direkt unsichtbar schaltet. Das gelingt auf eine der folgenden Arten:

- Indem wir das Formular als Startformular festlegen und das Formular mit einem entsprechenden Ereignis bei Öffnen direkt unsichtbar machen,
- indem wir das Formular von dem standardmäßig beim Starten geöffneten Startformular unsichtbar öffnen lassen oder
- indem wir das Formular über das **AutoExec**-Makro öffnen – ebenfalls unsichtbar.

Formular beim Öffnen unsichtbar machen

Eigentlich können wir für alle drei Varianten eine Vorgehensweise definieren, um das Formular unsichtbar zu machen – wir werden diese aber nicht für alle Varianten nutzen. Formulare haben die Eigenschaft **Visible**, die Sie auf den Wert **False** einstellen können, um das Formular auszublenden. Diese sollten wir eigentlich in einer der Ereignisprozeduren, die beim Öffnen des Formulars ausgeführt werden, nutzen können. Also legen wir direkt eine Ereignisprozedur für das Ereignis **Beim Öffnen** des Formulars an. Diese sieht wie folgt aus:

```
Private Sub Form_Open(Cancel As Integer)
    Me.Visible = False
End Sub
```

Der erste Test verläuft allerdings wenig vielversprechend: Das Formular bleibt schlicht sichtbar. Funktioniert die **Visible**-Eigenschaft vielleicht gar nicht? Probieren wir dies doch mit der folgenden Anweisung im Direktbereich aus:

```
Forms!frmStart.Visible = False
```

Und siehe da: Das Formular verschwindet. Woran liegt es also, dass

Datenmakros III: Ereignisse

Mit der Einführung von Datenmakros mit in Access 2010 bietet sich nun auch für Access-Benutzer die Möglichkeit, über eine Art Trigger Aktionen auszuführen, die in Zusammenhang mit dem Anlegen, Ändern oder Löschen von Datensätzen stehen. Dieser Artikel stellt die verschiedenen Ereignisse vor und zeigt, wie Sie diese programmieren.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1406_Makros.accdb**.

Datenereignisse

Access bietet seit Access 2010 die folgenden Tabellenereignisse an:

- **Vor Änderung**
- **Vor Löschung**
- **Nach Einfügung**
- **Nach Aktualisierung**
- **Nach Löschung**

Wir schauen uns in den folgenden Abschnitten einfache Beispiele für diese Ereignisse an und zeigen Ihnen, welche Besonderheiten diese jeweils aufweisen. Die Ereignisse können Sie auf zwei verschiedenen Wegen festlegen:

- Über die Ribbon-Einträge unter **Tabelle|Vorabereignisse** und **Tabelle|Nachfolgeereignisse** der Datenblattansicht der Tabelle (siehe Bild 2) und
- über die Ribbon-Einträge unter **Entwurf|Feld-, Datensatz- und Tabellenereignisse|Datenmakros erstellen** der Entwurfsansicht einer Tabelle (siehe Bild 3).

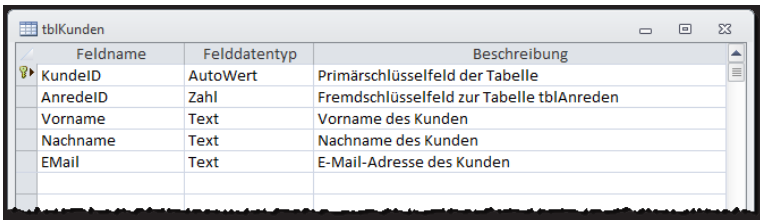


Bild 1: Tabelle zum Darstellen der Möglichkeiten von Tabellenereignissen

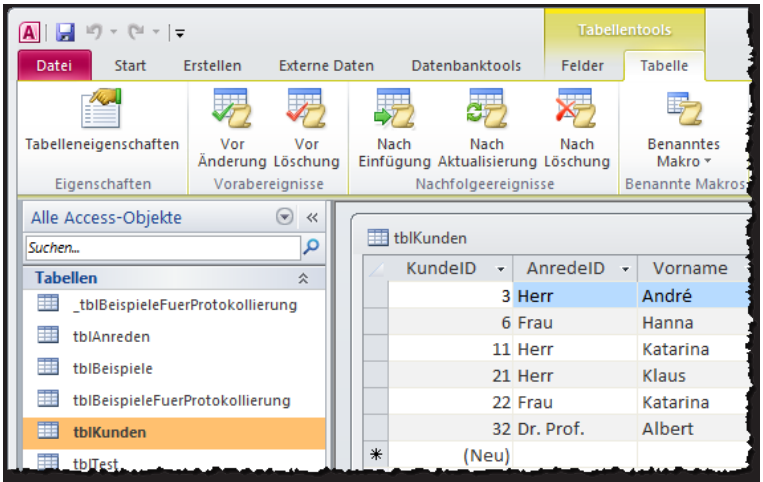


Bild 2: Schaltflächen zum Hinzufügen von Tabellenereignissen in der Datenblattansicht

Die Beispieldatenbank enthält eine einfache Tabelle namens **tblKunden**, die wie in Bild 1 aussieht. An dieser Tabelle demonstrieren wir die nachfolgenden Beispiele.

Mögliche Makrobefehle

Je nachdem, welches Datenmakro Sie verwenden, stehen Ihnen unterschiedliche Makrobefehle zur Verfügung.

Einen sehr eingeschränkten Vorrat an Makrobefehlen finden Sie für die beiden Datenmakros **Vor Änderung** und **Vor Löschung** vor:

- **Gruppieren**: Fasst mehrere Befehle zu einer Gruppe zusammen.

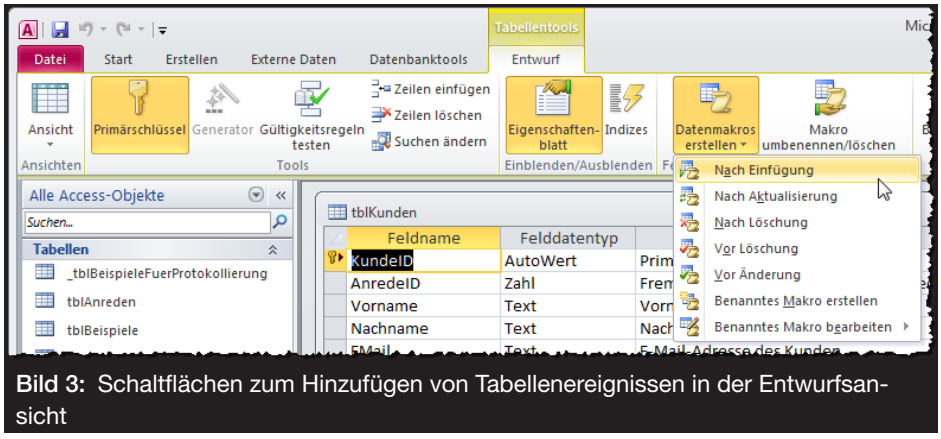


Bild 3: Schaltflächen zum Hinzufügen von Tabellenereignissen in der Entwurfsansicht

- **Kommentar**: Erlaubt die Eingabe eines Kommentars.
- **Wenn**: Bedingungsblock
- **AuslösenFehler**: Löst einen Fehler aus, der allerdings nur in der Protokolltabelle landet.
- **BeiFehler**: Aktiviert die Fehlerbehandlung.
- **FestlegenFeld** (nur in **Vor Änderung**, nicht in **Vor Löschung**): Legt den Wert eines Feldes der aktuellen Tabelle fest, die das Datenmakro ausgelöst hat. In den übrigen Tabellenereignissen nur über **DatensatzBearbeiten** und **DatensatzErstellen** verfügbar.
- **FestlegenLokaleVar**: Legt eine lokale Variable fest.
- **LöschenMakroFehler**: Setzt das **MakroError**-Objekt zurück.
- **NachschlagenDatensatz**: Wie **DLookup**.
- **StoppMakro**: Beendet das Makro.

Für die drei Datenmakros **Nach Einfügung**, **Nach Aktualisierung** und **Nach Löschung** stehen noch einige weitere Makrobefehle zur Verfügung:

- **AusführenDatenmakro**: Ruft ein weiteres Datenmakro auf.
- **BeendenFürJedenDatensatz**: Beendet eine **FürJedenDatensatz**-Schleife.
- **DatensatzBearbeiten**: Bearbeitet einen Datensatz. Darin Zuweisung der Werte mit **FestlegenFeld**.
- **DatensatzErstellen**: Erstellt einen neuen Datensatz. Darin Zuweisung der Werte mit **FestlegenFeld**.
- **DatensatzLöschen**: Löscht einen Datensatz.
- **FürJedenDatensatz**: Wie **Do While Not rst.EOF**-Schleife.
- **ProtokollierenEreignis**: Trägt ein Ereignis in die Tabelle **USysApplicationLog** ein.
- **SendenEMail**: Sendet eine E-Mail.
- **StoppAlleMakros**: Beendet alle Makros, auch diejenigen, die das aktuelle Makro aufgerufen haben.

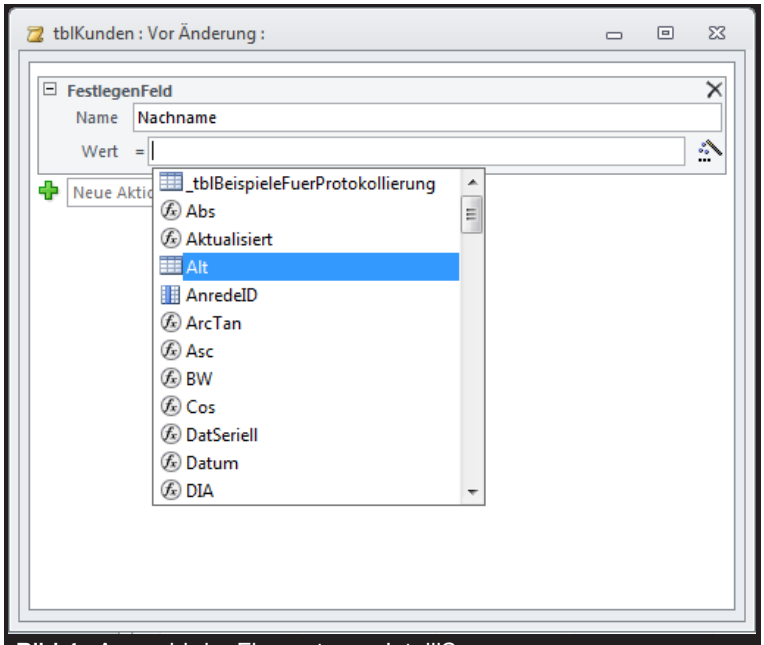


Bild 4: Auswahl der Elemente per IntelliSense

In den drei Makros **Nach Einfügung**, **Nach Aktualisierung** und **Nach Löschung** gibt es keinen direkten Zugriff auf den Makrobefehl **FestlegenFeld**.

Kein Wunder: Dieser steht im Datenmakro **Vor Änderung** auch nur bereit, weil er sich auf die Felder der aktuellen Tabelle bezieht.

Aus einem anderen Grund fehlt der Befehl im Datenmakro **Vor Löschung** komplett: Es würde ja keinen Sinn machen, einen Feldwert vor dem Löschen noch zu ändern.

Spezielle Tabellen, Felder und Funktionen

Wenn Sie Werte in die Eigenschaften der Makroaktionen eintragen wollen und den ersten Buchstaben eintippen, erhalten Sie gleich per IntelliSense alle möglichen Vorschläge.

Noch besser gelingt dies, wenn Sie innerhalb eines Eigenschaftsfeldes die Tastenkombination **Strg + Leertaste** betätigen. Der Makroeditor zeigt dann direkt alle zur Verfügung stehenden Elemente an (siehe Bild 4).

Nachfolgend finden Sie eine Auflistung spezieller Elemente inklusive der Tabellenereignisse, welche diese zur Verfügung stellen:

- **Aktualisiert** (**Vor Änderung**, **Nach Einfügung**, **Nach Aktua-**

lisierung, **Vor Löschung**, **Nach Löschung**): Gibt an, ob der Wert des als Parameter angegebenen Feldes geändert wurde. Das dieser auch in den beiden Ereignissen **Vor Löschung** und **Nach Löschung** angeboten wird, ist nicht wirklich sinnvoll – was spielt es für eine Rolle, ob ein zu löschender Datensatz zuvor geändert wurde?

- **Alt** (**Vor Änderung**, **Vor Löschung**, **Nach Einfügung**, **Nach Aktualisierung**, **Nach Löschung**): Ermöglicht den Zugriff auf den Datensatz vor der Änderung. Dabei verwenden Sie statt des Tabellennamens einfach den Namen **Alt**, zum Beispiel **Alt!KundeID**.

- **IstEingefügt** (**Vor Änderung**): Prüft, ob der geänderte Datensatz soeben geändert oder eingefügt wurde.

- **LetztesErstellenDatensatzID** (**Nach Einfügung**, **Nach Aktualisierung**, **Nach Löschung**): Damit können Sie, wenn Sie von einem Makro aus einen Datensatz anlegen, gleich den Primärschlüsselwert dieses Datensatzes ermitteln (sofern es sich um einen Autowert handelt).

- **MakroErneutGestartet** (**Nach Einfügung**, **Nach Aktualisierung**, **Nach Löschung**): Liefert eine Information, ob das Makro durch das gleiche Makro neu gestartet wurde. Wenn Sie beispielsweise in einem Nach Aktualisierung-Makro wieder ein Feld des gleichen Datensatzes aktualisieren, löst dies wieder das gleiche Datenmakro aus. Wenn dies nicht geschehen soll, können Sie dies durch das Makro abfangen.

- **MakroError** (**Vor Änderung**, **Vor Löschung**, **Nach Einfügung**, **Nach Aktualisierung**, **Nach Löschung**): Liefert Informationen über den zuletzt aufgetretenen Fehler.

- **Rückgabevariablen** (**Nach Einfügung**, **Nach Aktualisierung**, **Nach Löschung**): Wenn Sie ein weiteres Datenmakro aufrufen und dieses Rückgabevariablen liefert, können Sie diese über dieses Objekt auslesen.

Anwendung der fünf Tabellenereignisse

Beginnen wir mit dem Anlegen eines Datensatzes. In diesem Fall können Sie lediglich das Tabellenereignis **Nach Einfügung** nutzen. Dieses erlaubt es beispielsweise,

- eine E-Mail zu verschicken und jemanden über das Anlegen des Datensatzes zu informieren,

- das Anlegedatum und den Benutzer, der den Datensatz angelegt hat, zu speichern,

- ein Feld mit einem Standardwert zu füllen, der nicht über die dafür vorgesehene Eigenschaft ermittelt werden kann.

Aber warum gibt es überhaupt kein Tabellenereignis, das vor dem Anlegen eines Datensatzes ausgelöst wird? In der Tat gibt es dieses sehr wohl, allerdings etwas versteckt: Das Tabellenereignis **Vor Änderung** wird nämlich auch beim Anlegen eines Datensatzes ausgelöst. Mit der Eigenschaft **IstEingefügt** können Sie dann ermitteln, ob ein neuer Datensatz eingefügt oder ein bestehender Datensatz geändert werden soll.

Beim Ändern eines Datensatzes können Sie zwei Ereignisse programmieren: **Vor Änderung** und **Nach Aktualisierung** (nur Microsoft weiß, warum dies nicht **Vor Änderung** und **Nach Änderung** oder **Vor Aktualisierung** und **Nach Aktualisierung** heißt ...).

Im Ereignis **Vor Änderung** prüfen Sie, ob eine Änderung überhaupt zulässig ist oder nicht. Falls ein Feld eine von Ihnen definierte Bedingung nicht erfüllt, rufen Sie einfach die Makroaktion **AuslösenFehler** auf und geben damit – nebst Abbruch der Änderung – eine entsprechende Meldung aus.

In diesem Ereignis können Sie ebenfalls bereits Feldwerte der aktuellen Tabelle anpassen.

Das Löschen eines Datensatzes löst die beiden Ereignisse **Vor Löschung** und **Nach Löschung** aus. Das Ereignis **Vor Löschung** können Sie nutzen, um den Löschvorgang zu unterbinden – beispielsweise, wenn in einer Tabelle gar keine Datensätze gelöscht werden dürfen.

Das Ereignis **Nach Löschung** verwenden Sie beispielsweise, um die Daten eines gelöschten Datensatzes zu archivieren oder um Felder mit aggregierten Werten, die sich auch

auf den gelöschten Datensatz beziehen, zu aktualisieren.

Die Eigenschaft IstEingefügt

Mithilfe dieser Eigenschaft, die nur im Tabellenereignis **Vor Änderung** zur Verfügung steht, können Sie prüfen, ob ein zu ändernder Datensatz soeben angelegt wurde oder bereits vorhanden war.

Das Beispielmakro aus Bild 5 prüft, ob der Wert dieser Eigenschaft nicht wahr ist und führt nur dann die in der **Wenn**-Bedingung enthaltene Anweisung aus – also nur dann, wenn der Datensatz bereits vorhanden war und nicht neu angelegt wurde.

In diesem Fall soll eine Meldung angezeigt werden, die den Benutzer darauf hinweist, dass vorhandene Datensätze nicht geändert werden können – siehe Bild 6.

Der Datensatz kann nach dem Ausblenden der Meldung erst gespeichert werden, wenn die Änderungen etwa mit der Escape-Taste rückgängig gemacht wurden (siehe Tabelle **tblKunden_Beispiel_IstEingefuegt** der Beispieldatenbank).

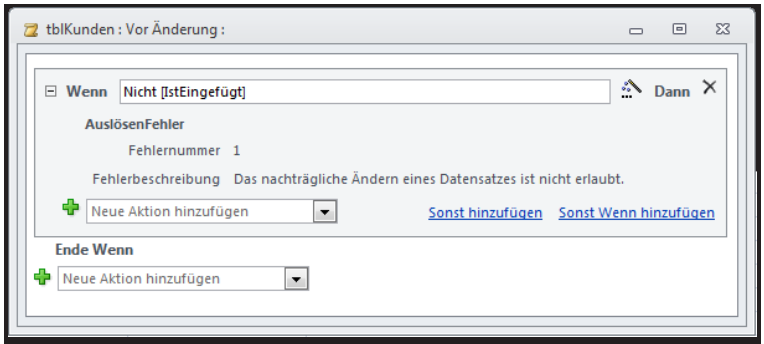


Bild 5: Auslösen eines Fehlers beim Versuch, einen Datensatz zu ändern

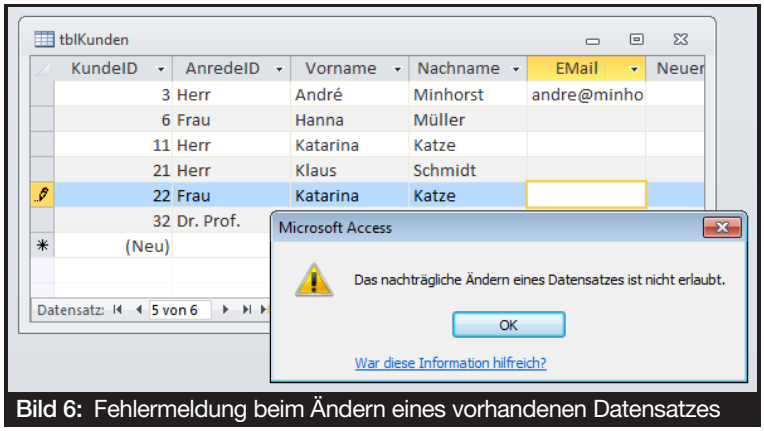


Bild 6: Fehlermeldung beim Ändern eines vorhandenen Datensatzes

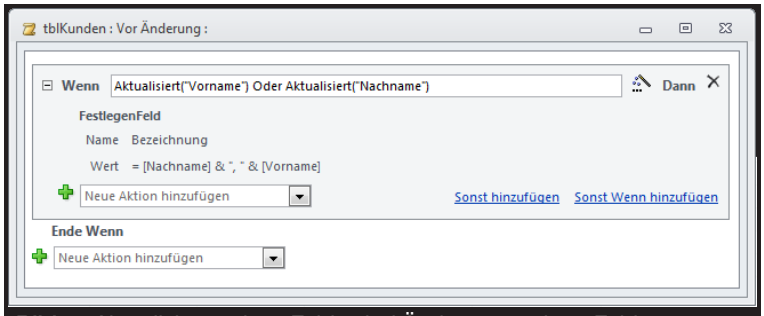


Bild 7: Aktualisieren eines Feldes bei Änderung anderer Felder

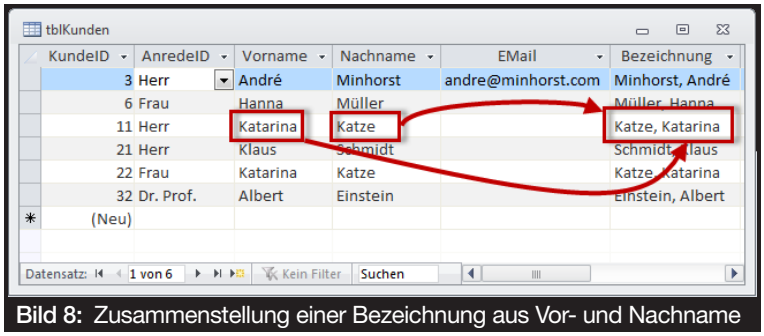


Bild 8: Zusammenstellung einer Bezeichnung aus Vor- und Nachname

Die Funktion Aktualisiert

Eine weitere sehr interessante Funktion heißt **Aktualisiert** und erwartet die Angabe eines Feldnamens als Parameter.

Wir schauen uns dies anhand der Tabelle **tblKunden_Beispiel_Aktualisiert** der Beispieldatenbank an. Diese Tabelle enthält wiederum ein Makro, das durch das Tabellenereignis **Vor Änderung** ausgelöst wird. Es prüft in seiner Wenn-Bedingung den folgenden Ausdruck:

```
Aktualisiert("Vorname")
Oder
Aktualisiert("Nachname")
```

Dieser ist wahr, wenn der Inhalt eines der Felder **Vorname** oder **Nachname** geändert wurde – was durch die Funktion **Aktualisiert** geprüft wird. In diesem Fall soll das Makro in das Feld **Bezeichnung** der Tabelle einen Ausdruck einfügen, der aus dem Inhalt des Feldes **Nachname**, einem Komma und dem Inhalt des Feldes **Vorname** des Datensatzes zusammengesetzt ist (siehe Bild 7).

Das Ergebnis sieht dann beispielsweise wie in Bild 8 aus.

Achtung: Sie müssen den als Parameter verwendeten Feldnamen unbedingt in Anführungszeichen einfassen.

DLookup und Co. in Datenmakros

Wer sich die zur Verfügung stehenden Funktionen in Datenmakros ansieht, stellt schnell fest, dass dort einige Funktionen fehlen – zum Beispiel **DLookup**. Das ist in diesem Fall aber kein Problem, denn die **DLookup**-Funktion können Sie durch die Makroaktion **DatensatzNachschlagen** ersetzen.

Aber wie sieht es mit den übrigen Domänenfunktionen wie **DMin**, **DMax**, **DSum** et cetera aus? Hier ist ein kleiner Trick nötig. Sie müssen den gesuchten Wert mithilfe einer Abfrage ermitteln, die sie in der Datenbank speichern. Dann können Sie mit **DatensatzNachschlagen** auf die entsprechenden Daten zugreifen.

Wir schauen uns dies am Beispiel der beiden Tabellen **tblBestellungen** und **tblBestelldetails** an. Der Tabelle **tblBestellungen** haben wir ein Feld namens Bestellsumme hinzugefügt. Dieses soll die Summe der Bestellpositionen aufnehmen, damit diese nicht immer neu dynamisch ermittelt werden muss. Damit erhalten wir redundante Daten, da diese Summe ja bereits in der Tabelle **tblBestelldetails** abgelegt ist. Damit daraus keine Inkonstistenzen entstehen, müssen wir uns absichern – mithilfe geeigneter Tabellenereignisse:

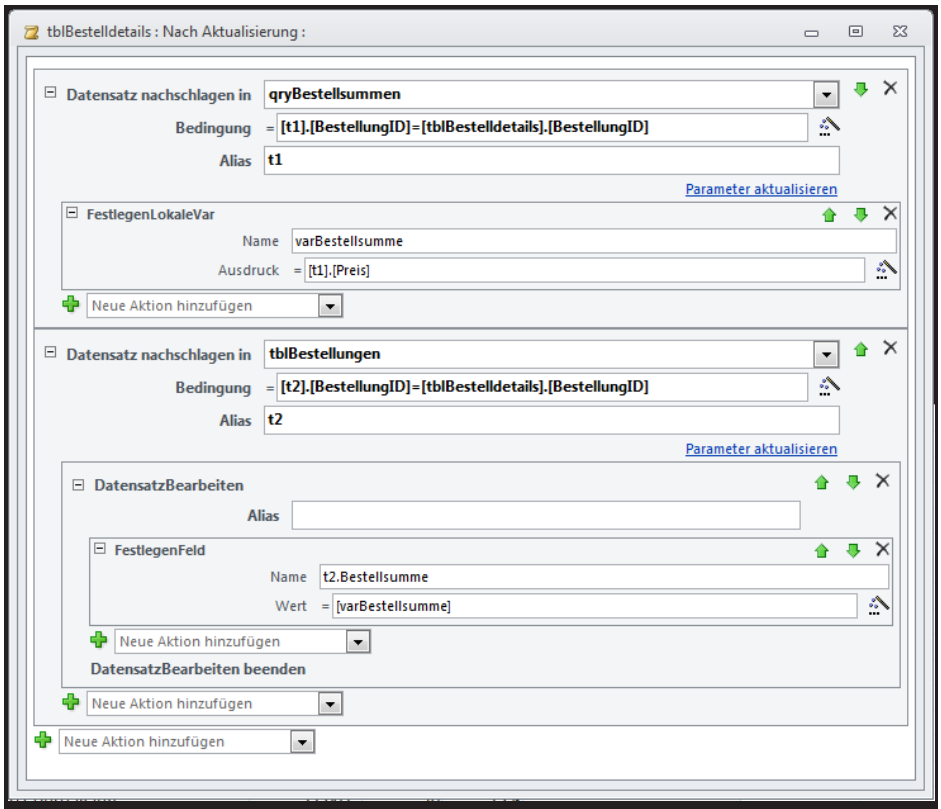


Bild 9: Dieses Makro speichert die aktuelle Bestellsumme in der Tabelle **tblBestellungen**.

- Das erste Ereignis **Nach Aktualisierung** der Tabelle **tblBestelldetails** soll Summe der Preise der einzelnen Bestellpositionen dieser Bestellung in das Feld **Bestellsumme** der Tabelle **tblBestellungen** eintragen.
- Das Gleiche müssen wir natürlich auch beim Löschen einer Bestellposition erledigen.
- Schließlich müssen wir sicherstellen, dass der Benutzer den Inhalt des Feldes **Bestellsumme** der Tabelle **tblBestellungen** nicht von Hand ändern kann.

Schauen wir uns zunächst das Makro an, dass durch das Ereignis **Nach Aktualisierung** der Tabelle **tblBestelldetails** ausgelöst wird. Dieses sieht wie in Bild 9 aus und ermittelt zunächst einen Wert aus einer Abfrage namens **qryBestellsummen**.

Den Entwurf dieser Abfrage finden Sie in Bild 10. Sie liefert zu allen Bestellungen in einem Feld namens **Preis** die Summe des folgenden berechneten Ausdrucks:

```
Summe([Einzelpreis]*[Anzahl]*
(1-[Rabatt]))
```

Außerdem liefert sie das Feld **BestellungID**, damit das Makro nur die Summe für die Bestellpositionen der jeweiligen Bestellung ermitteln muss.

Auf Basis dieser Abfrage führt das Makro die Makroaktion **DatensatzNachschlagenIn** aus und verwendet als Kriterium den Wert des Feldes

BestellungID des geänderten Datensatzes. Die Makroaktion **FestlegenLokaleVar** schreibt den Wert des Feldes **Preis** in die lokale Variable **varBestellsumme**. Nun folgt ein erneuter Aufruf der Makroaktion **DatensatzNachschlagenIn**. Diesmal ermittelt es den Datensatz der Tabelle **tblBestellungen**, mit dem der aktuelle Datensatz der Tabelle **tblBestelldetails** über das Feld **BestellungID** verknüpft ist. Die Makroaktion **DatensatzBearbeiten** bereitet den Datensatz für eine Bearbeitung vor und **FestlegenFeld** trägt den Wert der lokalen Variablen **varBestellsumme** in das Feld **Bestellsumme** der Tabelle **tblBestellungen** ein.

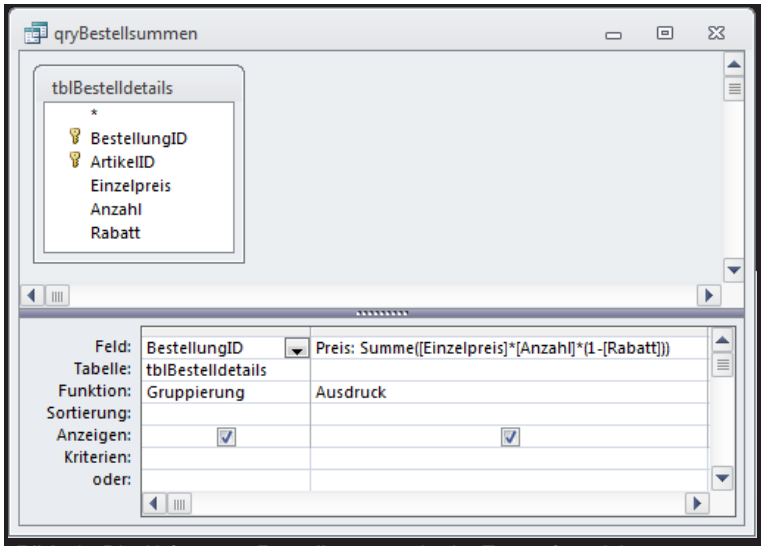


Bild 10: Die Abfrage **qryBestellsummen** in der Entwurfsansicht

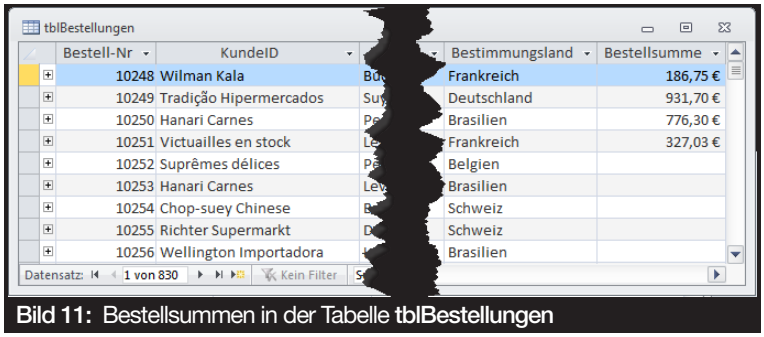


Bild 11: Bestellsummen in der Tabelle **tblBestellungen**

Es funktioniert: Die Tabelle **tblBestellungen** zeigt nach dem Ändern einer Bestellposition die aktuelle **Bestellsumme** im Feld **Bestellsumme** an (siehe Bild 11).

Nun müssen wir dafür sorgen, dass auch das Löschen eines Datensatzes aus der Tabelle **tblBestelldetails** zur Aktualisierung der **Bestellsumme** in der Tabelle **tblBestellungen** führt.

Dies wird allerdings etwas problematisch: Wir können ja gar nicht mehr auf den gelöschten Datensatz zugreifen, um dessen **BestellungID** zu ermitteln! Doch, können wir wohl: Zwar nicht wie in dem obigen Makro über den Bezug **tblBestelldetails**, dafür aber über die während der Ausführung dieses Makros zur Verfügung stehenden Tabelle **Alt**. Diese liefert die gleichen Felder wie die Originaltabelle, aber nur einen Datensatz – nämlich den soeben gelöschten.

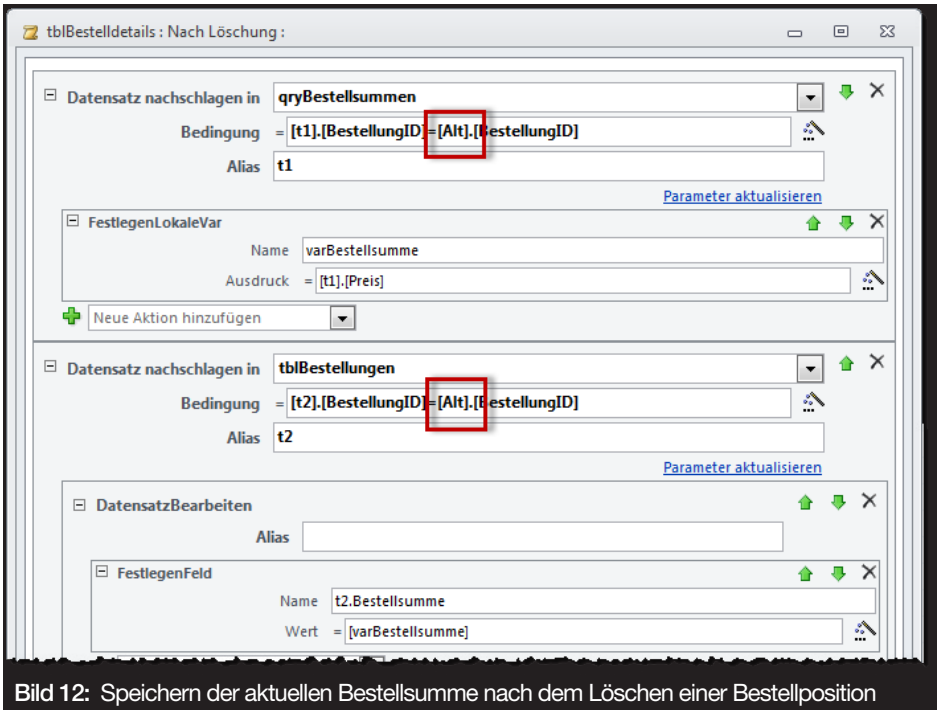


Bild 12: Speichern der aktuellen Bestellsumme nach dem Löschen einer Bestellposition

Legen wir also für das Ereignis **Nach Löschung** das gleiche Makro wie zuvor an, nehmen aber zwei Änderungen wie in Bild 12 vor: Dort ersetzen Sie jeweils den Bezug auf **tblBestelldetails** durch **Alt**. Übrigens: Um das Makro mit den gleichen Befehlen zu füllen wie das durch das Ereignis

Datensätze filtern per Kombinationsfeld

Wenn Sie ein Kombinationsfeld etwa zum Filtern von Artikeln nach einer bestimmten Kategorie verwenden, können Sie jeweils eine Gruppe von Artikeln anzeigen. Was aber, wenn Sie anschließend wieder alle Artikel anzeigen möchten? Für diesen Fall können Sie dem Kombinationsfeld einen Eintrag etwa mit dem Wert **<Alle anzeigen>** hinzufügen. Dessen Auswahl führt dann zur Aufhebung des jeweiligen Filters. Wir zeigen Ihnen, wie dies funktioniert.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1406_DatensaetzeFilternPerKombinationsfeld.mdb**.

Wenn Sie in einem Unterformular oder in einem Listenfeld die Übersicht über die Daten einer Tabelle oder Abfrage darstellen, die sich nach verschiedenen Kriterien gruppieren lassen, kann es sinnvoll sein, diese nach diesem Kriterium zu filtern. Im Beispiel schauen wir uns eine Übersicht aller Artikel einer Datenbank an, die nach der Kategorie gefiltert werden können.

Das Formular **frmArtikeluebersicht** zeigt die Artikel im Unterformular **sfmArtikeluebersicht** in der Datenblattansicht an.

Die Tabelle **tblArtikel**, die dort als Datenherkunft dient, verwendet ein Nachschlagefeld namens **KategorieID** zur Auswahl eines der Datensätze der verknüpften Tabelle **tblKategorien** (weitere Informationen zum Aufbau dieses Formulars erhalten Sie im Artikel **Datenübergabe zwischen Formularen**).

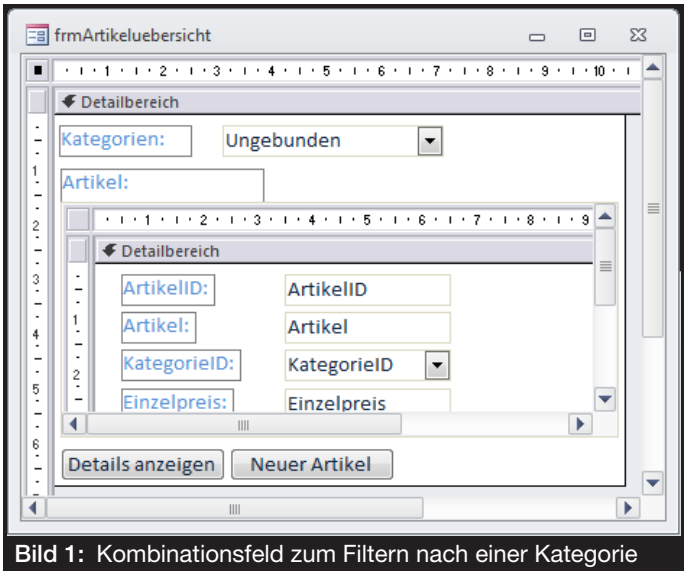


Bild 1: Kombinationsfeld zum Filtern nach einer Kategorie

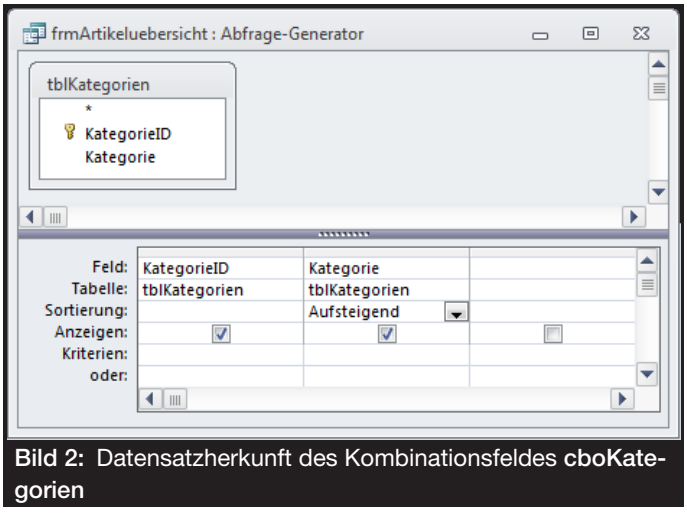


Bild 2: Datensatzherkunft des Kombinationsfeldes **cboKategorien**

Kombinationsfeld zum Filtern

Genau nach den Daten dieses Feldes wollen wir nun über ein Kombinationsfeld im Hauptformular filtern. Das bedeutet, dass wir im Kombinationsfeld alle Datensätze der Tabelle **tblKategorien** anbieten. Dieses Kombinationsfeld platzieren wir wie in Bild 1 oben im Formular und legen die Bezeichnung **cboKategorien** fest.

Damit dieses alle Kategorien anzeigt, stellen wir die Eigenschaft **Datensatzherkunft** auf eine entsprechende Abfrage ein. Diese sieht wie in Bild 2 aus. Die Abfrage liefert alle Datensätze der Tabelle **tblKategorien**, wobei diese nach dem Wert des Feldes **Kategorie** aufsteigend sortiert werden.

Damit das Kombinationsfeld nur das Feld mit den Bezeichnungen der Kategorien anzeigt, aber nicht den Inhalt des Primärschlüsselfeldes, stellen Sie außerdem die Eigenschaft **Spaltenanzahl** auf den Wert **2** und **Spaltenbreiten** auf **0cm** ein. Dadurch ist das Feld **KategorieID** zwar Bestandteil der Datensatzherkunft des Kombinationsfeldes und liefert auch den Wert der gebunde-

nen Spalte des ausgewählten Datensatzes, aber es ist nicht sichtbar. Das Kombinationsfeld zeigt seine Daten nun wie in Bild 3 an.

Unterformular filtern

Nun soll das Unterformular nach der Auswahl nur die Artikel anzeigen, die zur ausgewählten Kategorie gehören.

Zum Verständnis sind zwei Hinweise notwendig: Sowohl das Kombinationsfeld zur Auswahl der als Filter zu verwendenden Kategorie als auch das Feld mit der jeweiligen Kategorie der Artikel im Unterformular enthalten nicht die Kategorien, sondern lediglich die Primärschlüsselfelder der Werte der Tabelle **tblKategorien**.

Wir müssen also den Filter für die Datenherkunft des Unterformulars so einstellen, dass das Feld **KategorieID** als Vergleichswert den Zahlenwert aus dem Kombinationsfeld **cboKategorien** erhält.

Die Kurzfassung erfordert lediglich zwei Befehle (siehe Listing 1). Der erste stellt die Eigenschaft **Filter** des im Unterformularsteuerelements (**sfmArtikeluebersicht**) enthaltenen Formulars (**Form**) im aktuellen Formular (**Me**) auf den folgenden Ausdruck ein:

```
"KategorieID = " & Me!cboKategorien
```

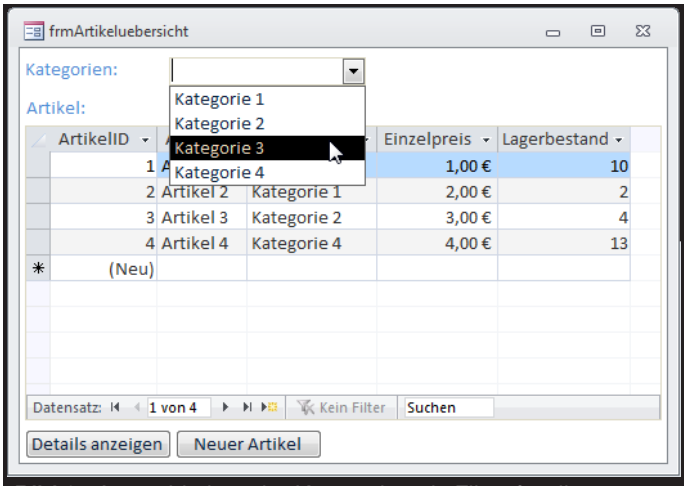


Bild 3: Auswahl einer der Kategorien als Filter für die angezeigten Datensätze im Unterformular

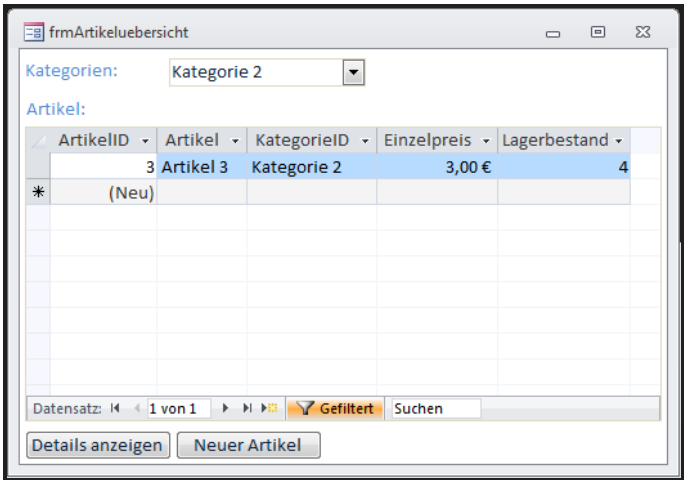


Bild 4: Filtern nach einer der Kategorien

Wenn das Kombinationsfeld (**Me!cboKategorien**) beispielsweise den Wert **1** enthält, sieht der Filterausdruck so aus:

```
KategorieID = 1
```

Der zweite Befehl aktiviert den Filter, indem er die Eigenschaft **FilterOn** auf den Wert **True** einstellt.

Dies ließe sich noch etwas übersichtlicher formulieren (siehe Listing 2). Wenn Sie beginnen, in VBA mit Ausdrücken wie mit diesem Filterausdruck zu arbeiten, sollten Sie den Filterausdruck bei Bedarf im Direktfenster ausgeben können. Dazu verwenden wir hier die Variable **strFilter**. Diese füllt die Prozedur zunächst mit dem ausgewerteten Ausdruck **"Kategorie = " & Me!cboKategorien** und gibt dann den Inhalt der Variab-

```
Private Sub cboKategorien_AfterUpdate()
    Me!sfmArtikeluebersicht.Form.Filter = "KategorieID = " & Me!cboKategorien
    Me!sfmArtikeluebersicht.Form.FilterOn = True
End Sub
```

Listing 1: Unterformular nach einem Wert eines Steuerelements im Hauptformular filtern

```
Private Sub cboKategorien_AfterUpdate()
    Dim strFilter As String
    strFilter = "KategorieID = " & Me!cboKategorien
    Debug.Print strFilter
    With Me!sfmArtikeluebersicht.Form
        .Filter = strFilter
        .FilterOn = True
    End With
End Sub
```

Listing 2: Übersichtlichere Variante mit der Möglichkeit, den Filterausdruck im Direktfenster auszugeben

len im Direktfenster aus (zum Beispiel **Kategorie = 1**).

Anschließend haben wir die beiden doch eher länglichen Ausdrücke zum Zuweisen und Aktivieren des Filters entzerrt, indem wir mit der **With**-Anweisung das Objekt benennen, auf das sich die folgenden Eigenschaften beziehen. Wie Bild 4 zeigt, funktioniert das Filtern bereits wie gewünscht.

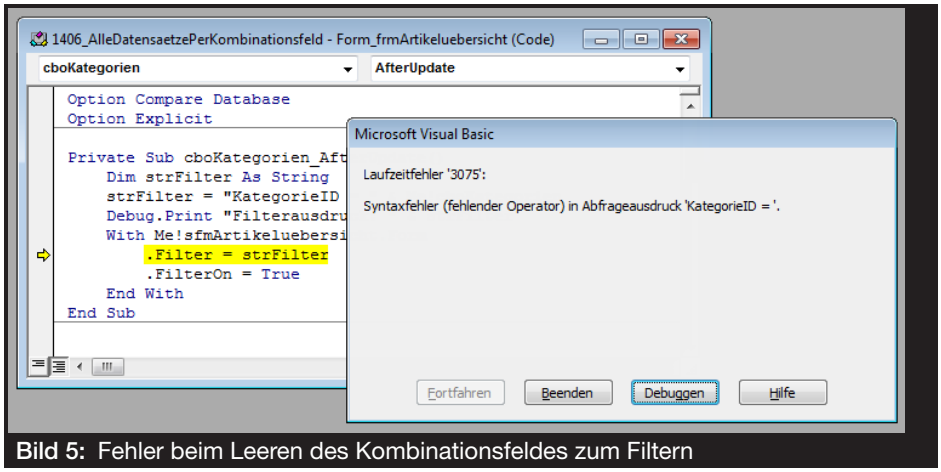


Bild 5: Fehler beim Leeren des Kombinationsfeldes zum Filtern

Filter aufheben

Nun kann es allerdings geschehen, dass der Benutzer den Filter wieder aufheben möchte. Aus Mangel an entsprechenden Einträgen wird er vermutlich den Inhalt des Kombinationsfeldes **cboKategorien** von Hand leeren und die Eingabetaste betätigen.

```
Private Sub cboKategorien_AfterUpdate()
    Dim strFilter As String
    If Not IsNull(Me!cboKategorien) Then
        strFilter = "KategorieID = " & Me!cboKategorien
        Debug.Print strFilter
        With Me!sfmArtikeluebersicht.Form
            .Filter = strFilter
            .FilterOn = True
        End With
    Else
        Me!sfmArtikeluebersicht.Form.Filter = ""
    End If
End Sub
```

Listing 3: Diese Version der Prozedur erlaubt auch das Leeren des Kombinationsfeldes

Dies führt dummerweise zu dem Fehler aus Bild 5. Was ist hier passiert? Nun: Das Kombinationsfeld weist keinen Eintrag auf, dessen Wert in der zweiten Spalte der Datensatzherkunft eine leere Zeichenkette enthält. Daher leert Access das Kombinationsfeld schlicht und einfach. Für die Zusammenstellung des Filterkriteriums in der Variablen **strFilter** ist das noch kein Problem – dieses sieht dann so aus:

```
KategorieID =
```

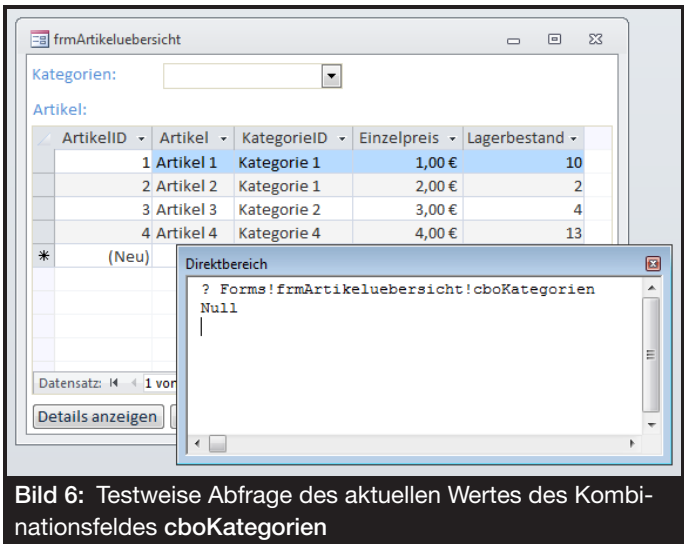


Bild 6: Testweise Abfrage des aktuellen Wertes des Kombinationsfeldes **cboKategorien**

Wenn Sie diesen Ausdruck allerdings der Eigenschaft **Filter** des Unterformulars zuweisen, löst dies den in der Abbildung dargestellten Syntaxfehler aus. Diese erwartet nämlich einen Vergleichswert, der hier allerdings nicht vorliegt.

Leeres Kombinationsfeld abfangen

Also müssen wir uns etwas überlegen, um diesen Fehler zu verhindern. Die einfachste Methode wäre es, gleich eingangs der Prozedur zu prüfen, ob der Benutzer überhaupt einen Wert für das Kombinationsfeld **cboKategorien** ausgewählt hat (siehe Listing 3).

Das Kombinationsfeld zu leeren bedeutet nämlich, dass dieses den Wert **Null** aufweist – dies ist gleichbedeutend mit "leer".

Von Formular zu Formular mit OpenArgs

Formulare zeigen üblicherweise die Daten aus Tabellen oder Abfragen an. Beim Aufruf von einem anderen Formular teilt man gegebenenfalls mit, welcher Datensatz angezeigt werden soll. In manchen Fällen ist es aber nötig, weitere Daten zu übergeben oder auch Daten von einem geöffneten Formular auszulesen. Wie dies in den verschiedenen Konstellationen gelingt, zeigt der vorliegende Artikel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1406_VonFormularZuFormular.mdb**.

Wert an Steuerelement übergeben

Im ersten Beispiel wollen wir einfach einen Wert in ein Textfeld in ein erstes Formular eingeben und diesen Wert per Schaltfläche in ein Textfeld eines anderen Formulars übertragen. Die Protagonisten dieses Beispiels heißen **frm1** und **frm2** (siehe Bild 1).

frm1 enthält ein Textfeld namens **txtZuUebergeben** und eine Schaltfläche namens **cmdWertUebergeben**, **frm2** enthält ein Textfeld namens **txtUebergebenerWert**.

Damit die Schaltfläche beim Anklicken die gewünschte Ereignisprozedur ausführt, wählen Sie für die Ereigniseigenschaft **Beim Klicken** des Steuerelements den Eintrag **[Ereignisprozedur]** aus und klicken dann auf die Schaltfläche rechts von der Eigenschaft. Es erscheint die leere Ereignisprozedur im VBA-Editor, die Sie wie in Listing 1 ergänzen. Die einzige Anweisung liest den Wert des Textfeldes **txtZuUebergeben** des Formulars **frm1** aus. **frm1** kann über den Ausdruck **Me** referenziert werden, da es sich im gleichen Formular befindet, zu dem das Klassenmodul mit dieser Prozedur gehört.

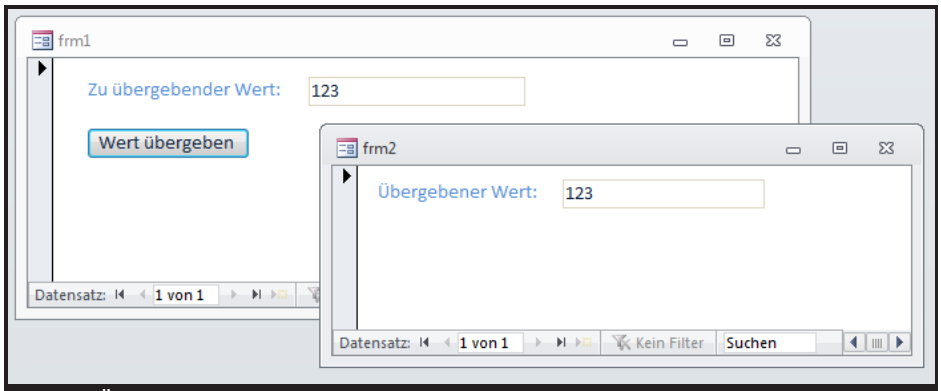


Bild 1: Übergeben eines Wertes von einem Formular zum nächsten

```
Private Sub cmdWertUebergeben_Click()
    Forms!frm2!txtUebergebenerWert = Me!txtZuUebergeben
End Sub
```

Listing 1: Übergeben eines Wertes zwischen zwei Formularen

Wenn Sie den Wert des Ausdrucks **Me!txtZuUebergeben** nun an ein Textfeld in einem anderen Formular übergeben möchten, können Sie nicht mit dem Ausdruck **Me** auf das Zielformular zugreifen – schließlich handelt es sich um ein anderes Formular. Also referenzieren wir dieses Formular über die Auflistung **Forms** und dessen Element **frm2** (**Forms!frm2**). Das Zielsteuerelement hängen wir, durch ein weiteres Ausrufezeichen angeführt, an diesen Ausdruck an (**Forms!frm2!txtUebergebenerWert**). Die Zuweisung sieht nun so aus:

```
Forms!frm2!txtUebergebenerWert = _
    Me!txtZuUebergeben
```

Es gibt noch eine ganze Menge weiterer Schreibweisen, um das Zieltextfeld zu referenzieren. Die folgende ist noch relativ einfach:

```
Forms("frm2")("txtUebergebenerWert")
```

Diese Version ist die ausführlichste:

```
Forms.Item("frm2").Controls.Item(_
    "txtUebergebenerWert").Value
```

Hier identifizieren wir das Zielformular über die **Item**-Eigenschaft der **Forms**-Auflistung. Dieses wiederum hat eine **Controls**-Auflistung, die ebenfalls eine **Item**-Eigenschaft für den Zugriff auf das gewünschte Element bietet.

Schließlich verwenden wir die Eigenschaft **Value**, um auf den Wert des Steuerelements zuzugreifen. Dabei handelt es sich um die Standard-

eigenschaft, die Sie normalerweise nicht anzugeben müssen.

Sie können das Zielformular auch vorher mit einer Variable des Typs **Form** referenzieren, was den Ausdruck etwas übersichtlicher macht – hier für eine zweite Schaltfläche namens **cmdWertUebergebenII** hinterlegt. Die Variable heißt **frmZiel** und wird mit dem Verweis auf **Forms!frm2** gefüllt (alternativ **Forms("frm2")** oder **Forms.Item("frm2")**):

```
Private Sub cmdWertUebergebenII_Click()
    Dim frmZiel As Form
    Set frmZiel = Forms!frm2
    frmZiel!txtUebergebenerWert = _
                Me!txtZuUebergeben
End Sub
```

Wert beim Öffnen übergeben

Normalerweise werden Sie den Wert gleich beim Öffnen eines Formulars vom öffnenden Formular an das geöffnete Formular übergeben wollen. Der Versuchsaufbau ist dem aus dem vorherigen Beispiel sehr ähnlich. Die Formulare heißen nun **frm3** und **frm4** (siehe Bild 2).

Der Unterschied liegt in der Zuständigkeit der verschiedenen Aufgaben. Während im ersten Beispiel der Wert innerhalb einer einzigen Anweisung von Formular zu Formular übergeben wurde, ist hier etwas mehr Aufwand erforderlich.

Die Schaltfläche **cmdOeffnenUndWertUebergeben** des Formulars **frm3** soll diesmal das Formular **frm4** öffnen und diesem gleich den entsprechenden Wert übergeben. Kein Problem, wie die Prozedur aus

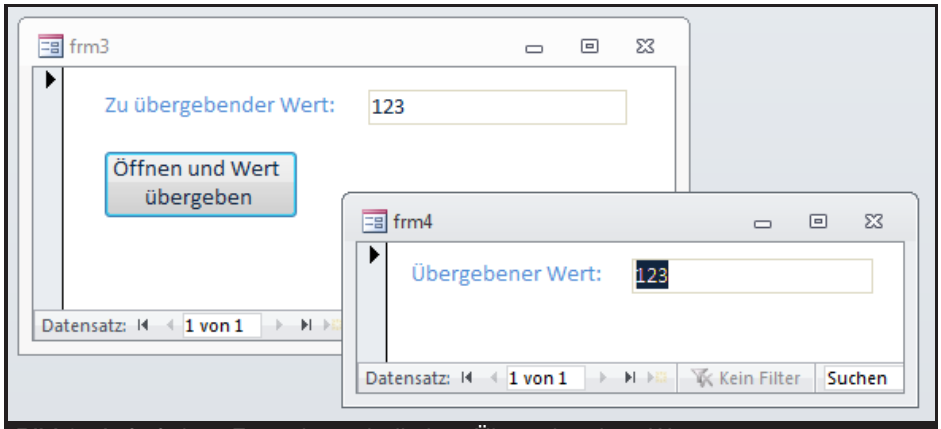


Bild 2: Aufruf eines Formulars mit direkter Übergabe eines Wertes

```
Private Sub cmdOeffnenUndWertUebergeben_Click()
    DoCmd.OpenForm "frm4"
    Forms!frm4!txtUebergebenerWert = Me!txtZuUebergeben
End Sub
```

Listing 2: Aufruf eines weiteren Formulars mit Übergabe eines Wertes

Listing 2 zeigt: Wir öffnen einfach das Formular **frm4** mit dem Aufruf der Methode **DoCmd.OpenForm** mit dem Namen des zu öffnenden Formulars als Parameter, also **frm4**.

Direkt nach dieser Anweisung übergeben wir wie im vorherigen Beispiel den Wert aus dem Textfeld an das entsprechende Textfeld des aufgerufenen Formulars.

Übergabe an modalen Dialog

Nun kann es ja sein, dass Sie das Formular **frm4** als modalen Dialog öffnen, damit der Benutzer zunächst die Eingaben in dieses Formular vornimmt und erst nach dem Schließen von **frm4** mit der Bearbeitung in **frm3** fortfahren kann.

In diesem Fall ändern wir die Parameter der **DoCmd.OpenForm**-Methode wie folgt:

```
DoCmd.OpenForm "frm4", _
                WindowMode:=acDialog
```

Allerdings zeigt dies nicht die gewünschte Wirkung: Das Formular **frm4** wird zwar geöffnet, das Textfeld aber nicht mit dem Wert aus

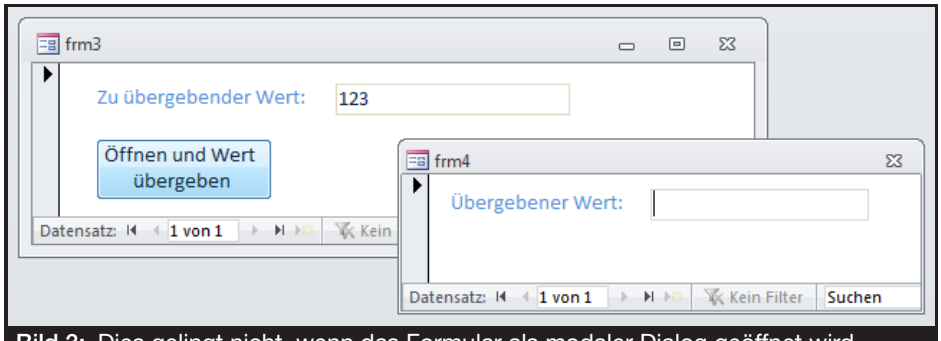


Bild 3: Dies gelingt nicht, wenn das Formular als modaler Dialog geöffnet wird.

dem Textfeld von **frm3** gefüllt (siehe Bild 3).

Darüberhinaus tritt nach dem Schließen von **frm4** auch noch ein Fehler auf (siehe Bild 4) – und zwar in der Zeile, welche die Zuweisung durchführen soll. Dies zeigt auch gleich, worin das Problem bei dieser Vorgehensweise liegt: Wenn die Prozedur das Formular **frm4** mit dem Parameter **WindowMode:=acDialog** öffnet, wird die aufrufende Prozedur an dieser Stelle unterbrochen und erst dann fortgeführt, wenn der Benutzer das Formular **frm4** wieder schließt.

In diesem Moment führt die Prozedur aus **frm3** die Anweisung aus, welche das Textfeld in **frm4** mit dem Wert des Textfeldes aus **frm3** füllen soll. Dies führt dann zu einem Fehler, da **frm4** ja gar nicht mehr vorhanden ist!

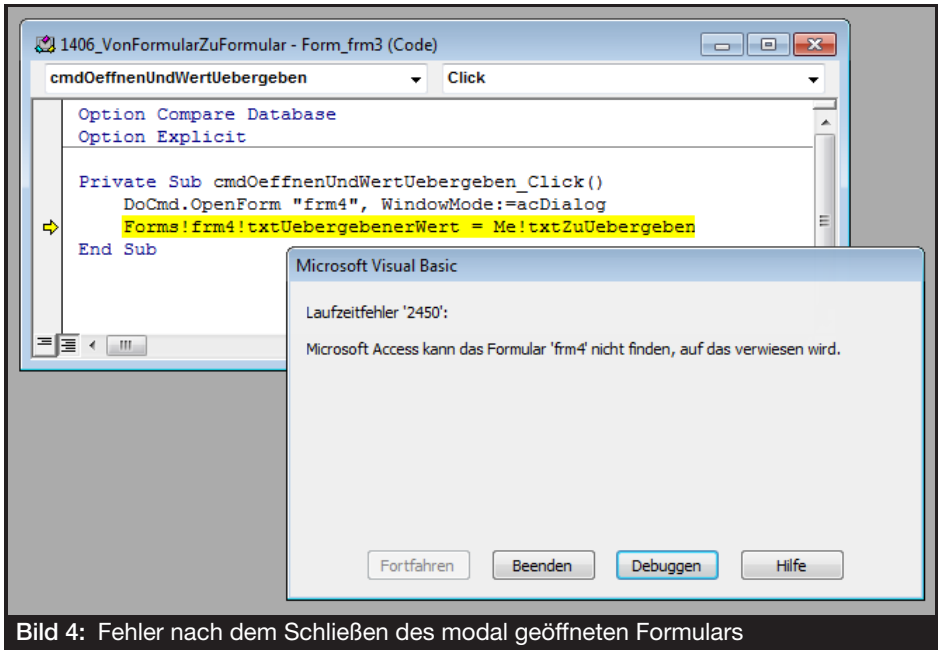


Bild 4: Fehler nach dem Schließen des modal geöffneten Formulars

```
Private Sub cmdOeffnenUndWertUebergeben_Click()
    DoCmd.OpenForm "frm4", WindowMode:=acDialog, OpenArgs:=Me!txtZuUebergeben
End Sub
```

Listing 3: Aufruf als modaler Dialog und Übergabe eines Wertes als Öffnungsargument

Wertübergabe per Öffnungsargument

Auch für diese Konstellation gibt es eine Lösung, die aber ein wenig aufwendiger zu programmieren ist. Sie sollten sich die notwendigen Schritte dennoch merken, denn Sie werden diese vermutlich sehr oft benötigen.

In diesem Falle ändern wir die Prozedur **cmdOeffnenUndWertUebergeben** erneut. Dabei entfernen wir die Anweisung, welche das Textfeld in **frm4** direkt mit dem Inhalt des Textfeldes aus **frm3** füllte. Dafür versehen wir die **DoCmd.OpenForm**-Anweisung mit einem weiteren Parameter, nämlich **OpenArgs**. Für diesen legen wir den Inhalt des Textfeldes als als Parameterwert fest (siehe Listing 3).

```
Private Sub Form_Open(Cancel As Integer)
    Dim varOpenArgs As Variant
    varOpenArgs = Me.OpenArgs
    Me!txtUebergebenerWert = varOpenArgs
End Sub
```

Listing 4: Entgegennahme des übergebenen Wertes vom geöffneten Formular

Solange Sie nun am aufgerufenen Formular keine Änderungen vornehmen, tut sich dort nach dem Öffnen durch eine Klick auf die Schaltfläche **cmdOeffnenUndWertUebergeben** zunächst einmal nichts.

Die Änderung spielt sich eher im Hintergrund ab, denn der für den Parameter **OpenArgs** angegebene Wert wird im Hintergrund an das aufgerufene Formular übergeben. Abrufen können Sie diesen über die Eigenschaft **OpenArgs** des Formulars. Dies machen wir uns in einer Ereignisprozedur zunutze, die durch das Ereignis **Beim Öffnen** des Formulars **frm4** ausgelöst wird. Diese legen Sie an, indem Sie das Formular **frm4** in der Entwurfsansicht öffnen, für die Ereigniseigenschaft **Beim Öffnen** den Wert **[Ereignisprozedur]** auswählen und dann die Prozedur mit einem Klick auf die Schaltfläche mit den drei Punkten im VBA-Editor öffnen. Diese Prozedur ergänzen Sie nun wie in Listing 4. Sie deklariert zunächst eine Variable namens **varOpenArgs**. Diese füllt die Prozedur dann mit dem Wert der Eigenschaft **OpenArgs**

Arrays mit Split und Join

Für die Arbeit mit Arrays gibt es zwei interessante Funktionen, die Ihnen möglicherweise einige manuelle zu programmierende Schritte abnehmen. Join erlaubt es, die Inhalte eines eindimensionalen Arrays in einer einzigen Zeichenkette zusammenzusetzen, wobei Sie noch angeben können, ob die einzelnen Elemente durch ein oder mehrere Zeichen getrennt werden sollen. Und die Split-Funktion erstellt ein neues Array auf Basis einer in einer Zeichenketten enthaltenen Auflistung von Texten. Einzige Bedingung: Es muss ein einheitliches Trennzeichen zwischen den Einträgen vorliegen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1406_Arrays-MitSplitUndJoin.mdb**.

Die Join-Funktion

Die **Join**-Funktion erwartet zwei Parameter: den Variablennamen des Arrays sowie als optionalen Parameter die Zeichenkette, die hinter jedem Element angehängt werden soll. Sie können diese Funktion auf die Schnelle im Direktfenster des VBA-Editors testen (zu aktivieren mit der Tastenkombination **Strg + G**):

```
? Join(Array("Wert1","Wert2","Wert3"),"|")
Wert1|Wert2|Wert3
```

In diesem Beispiel setzt die **Array**-Funktion aus den drei Zeichenketten **Wert1**, **Wert2** und **Wert3** ein Array zusammen. Dieses nimmt die Join-Funktion gleich wieder auseinander und fügt die einzelnen Elemente zu einer einzigen Zeichenkette zusammen. Dabei verwendet sie die mit dem zweiten Parameter übergebene Zeichenkette, hier das Pipe-Zeichen (|) als Trennzeichen, das zwischen die einzelnen Elemente geschrieben wird.

Ein aufwendigeres Beispiel finden Sie in Listing 1. Hier sollen alle Artikelnamen durch Kommata voneinander getrennt im Direktfenster ausgegeben werden. Dazu öffnet die Prozedur zunächst eine Recordset-Gruppe auf Basis der Tabelle **tblArtikel**, wobei jedoch nur das Feld **Artikel** berücksichtigt wird

```
Public Sub Beispiel_Join()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim strArtikel() As String
    Dim strArtikelliste As String
    Dim lngAnzahl As Long
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT Artikel FROM tblArtikel", dbOpenDynaset)
    rst.MoveLast
    lngAnzahl = rst.RecordCount
    ReDim strArtikel(lngAnzahl - 1)
    rst.MoveFirst
    Do While Not rst.EOF
        strArtikel(rst.AbsolutePosition) = rst!Artikel
        rst.MoveNext
    Loop
    strArtikelliste = Join(strArtikel, ", ")
    MsgBox strArtikelliste
End Sub
```

Listing 1: Zeichenkette aus den Elementen eines Array erstellen

Dann springt die Prozedur mit der Methode **MoveLast** zum letzten Datensatz des Recordsets und liest mit der **Recordcount**-Eigenschaft die Anzahl der enthaltenen Datensätze aus.

Das Ergebnis landet in der Variablen **lngAnzahl** – und zwar um eins verkleinert, da der Index 0-basiert ist. Das Springen auf den letzten Datensatz ist nötig, da sonst die Anzahl der Datensätze möglicherweise nicht korrekt bestimmt wird.

Danach springt der Datensatzzeiger durch den Aufruf der Methode **MoveFirst** wieder zum Start der Datensatzgruppe. Die **Redim**-Methode dimensioniert die Array-Variable **strArtikel** auf die Anzahl der Artikel, die das Array im Folgenden aufnehmen soll. Nun durchläuft die Prozedur in einer **Do While**-Schleife alle Datensätze des Recordsets und füllt für jeden Datensatz das entsprechende Element des Arrays mit dem Namen des Artikels. Damit ist

Finetuning für Steuerelemente

Wenn Sie Formulare und Steuerelemente in der Entwurfsansicht bearbeiten, ist oft Handarbeit gefragt. Steuerelemente wollen angelegt, formatiert oder ausgerichtet werden. Manche erfordern eine Beschriftung mit einer speziellen Schriftart oder -formatierung. Viele dieser Aufgaben lassen sich vereinfachen, wenn Sie ein paar Tricks kennen.

Mehrere Steuerelemente markieren

Wenn Sie einmal mehrere Steuerelemente gleichzeitig verschieben oder gemeinsame Eigenschaften anpassen möchten, müssen Sie diese zunächst markieren. Dazu gibt es verschiedene Möglichkeiten:

Halten Sie die **Strg**- oder die **Umschalt**-Taste gedrückt und klicken Sie die zu markierenden Steuerelemente an. Diese werden dann als markiert gekennzeichnet. Auf die gleiche Weise können Sie die Markierung Steuerelement für Steuerelement wieder entfernen – beispielsweise, wenn Sie versehentlich das falsche Steuerelement markiert haben.

Wenn die zu markierenden Steuerelemente beieinander liegen, können Sie auch einfach einen Rahmen aufziehen, der alle betroffenen Steuerelemente streift. Dabei drücken Sie die linke Maustaste, während sich der Mauszeiger außerhalb jeglicher Steuerelemente und ziehen dann den Rahmen auf.

Beide Methoden lassen sich auch gut kombinieren: Wenn Sie beispielsweise alle Steuerelemente eines Formulars bis auf einige wenige markieren möchten, ziehen Sie einen Rahmen um alle Steuerelemente auf und entfernen dann bei gedrückter **Strg**- oder **Umschalt**-Taste die Markierung für die nicht benötigten Steuerelemente.

Das Eigenschaftsfenster zeigt nun übrigens nur die Eigenschaften an, die allen markierten Steuerelementen gemein sind.

Steuerelemente ausrichten

Wenn Sie Steuerelemente links, oben, unten oder rechts gleich ausrichten möchten, markieren Sie die betroffenen Steuerelemente und klicken dann mit der rechten Maustaste darauf. Das Kontextmenü bietet dann unter **Ausrichten** die Befehle **Linksbündig**, **Rechtsbündig**, **Oben** und **Unten** sowie **Am Raster** an.

Ein Klick auf Linksbündig etwa verschiebt die markierten Steuerelemente so, dass deren linker Rand vertikal auf einer Linie liegt – und zwar gemessen am linken Rand des Steuerelements, das dem linken Rand des Formulars/Berichts am nächsten ist.

Eine Ausrichtung am Raster bedeutet, dass das Steuerelement horizontal und vertikal an den nächstliegenden Rasterlinien ausgerichtet wird. Das Raster blenden Sie mit dem Kontextmenübefehl **Raster** eines der Bereichsköpfe der Entwurfsansicht ein.

Nun erscheint dieses relativ grob (siehe Bild 1), was aber nicht über die wahre Gestalt des Rasters hinwegtäuschen darf: Tatsächlich ist das Raster üblicherweise feiner als dargestellt. Wie fein, entnehmen Sie den beiden Eigenschaften **Raster X** und **Raster Y**. Steht dort jeweils der Wert **10**, befinden sich zwischen jeder sichtbaren Rasterlinie zehn weitere unsichtbare Rasterlinien. An

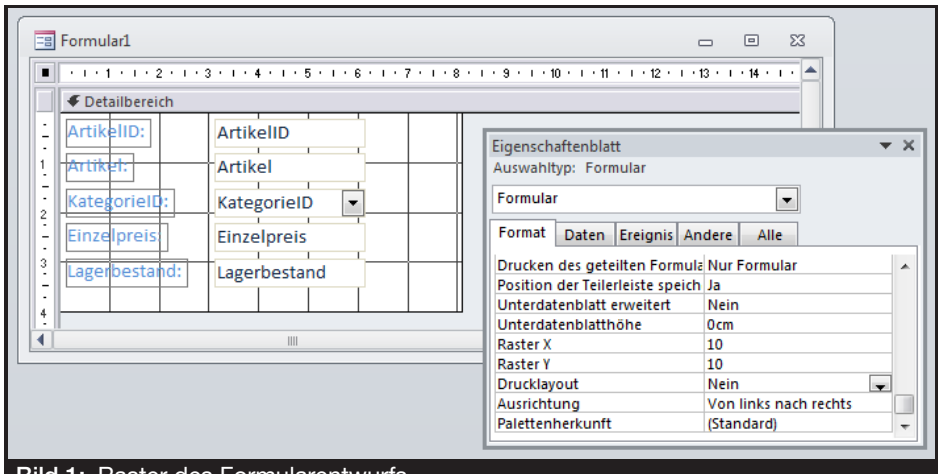


Bild 1: Raster des Formularentwurfs

diesen werden die Steuerelemente dann bei Auswahl von **Ausrichten|Am Raster** ausgerichtet. Manchmal zeigt Access das feinere Raster durch Punkte an – siehe Bild 2.

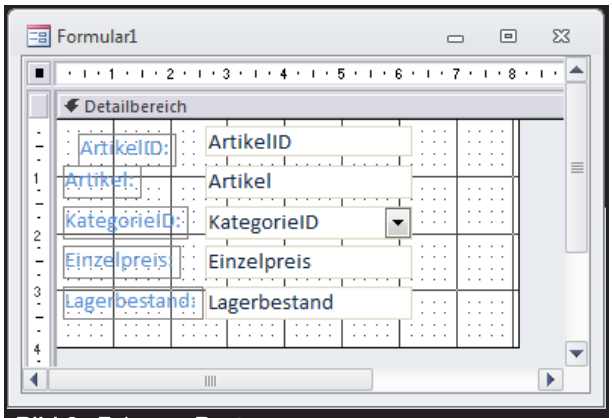


Bild 2: Feineres Raster

Tatsächlich ist eine Ausrichtung nur mit der Maus oder mit den Pfeiltasten nur an den Rasterlinien möglich. Wenn Sie eine feinere Ausrichtung wünschen, müssen Sie das auszurichtende Steuerelement markieren und dieses dann bei gedrückter **Strg**-Taste mit den Pfeiltasten verschieben.

Steuerelementgröße anpassen

Auch für das Anpassen der Größe der Steuerelemente gibt es verschiedene Möglichkeiten. Sie können mit der Maus an den einzelnen Rändern eines Steuerelements ziehen, um die Größe anzupassen. Dabei ändern Sie die Größe wiederum nur in Schritten, die das Raster zulässt. Wenn Sie die Größe mit den Pfeiltasten ändern möchten, drücken Sie gleichzeitig die **Umschalt**-Taste.

Die **Nach rechts**-Taste bewirkt dann etwa die Änderung der Breite bis zur nächsten Rasterlinie.

Aber auch hier können Sie eine feinere Skalierung wählen: Sie müssen wiederum nur zusätzlich die **Strg**-Taste gedrückt halten, egal, ob Sie die Größe mit der Maus oder mit den Pfeiltasten ändern.

Formatierung übertragen

Einer der wichtigsten Tricks ist das Übertragen einer Formatierung von einem Steuerelement auf ein oder mehrere andere Steuerelemente.

Dabei gehen wir davon aus, dass Sie ein bereits ein Steuerelement nach Ihren Wünschen formatiert haben. Wenn Sie nun die Formatierung der übrigen Steuerelemente an dieses anpassen möchten, brauchen Sie nur wenige Mausklicks:

- Markieren Sie das Steuerelemente mit der gewünschten Formatierung.
- Klicken Sie im Ribbon auf den Eintrag **Start|Zwischenablage|Format übertragen** (hier Access 2010 – siehe Bild 3).
- Klicken Sie auf das Steuerelement, das die Formatierung des zuvor markierten Steuerelements übernehmen soll. Fertig!

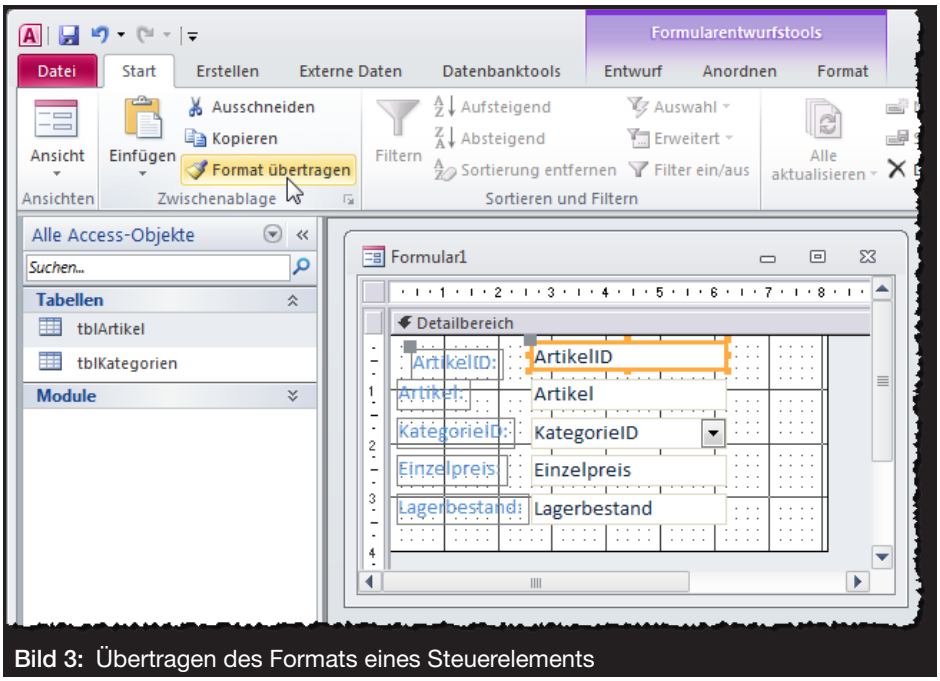


Bild 3: Übertragen des Formats eines Steuerelements

Wenn Sie gleich mehrere Steuerelemente mit dieser Formatierung aussstatten wollen, klicken Sie im zweiten Schritt doppelt auf **Format übertragen**. Der Eintrag im Ribbon bleibt dann hervorgehoben, bis Sie ihn erneut anklicken oder die **Escape**-Taste betätigen.

In der Zwischenzeit können Sie die zu formatierenden Steuerelemente nacheinander anklicken.

Access-Berichte drucken per VBA

Häufig unterschätzt man als Entwickler den Umfang von Daten, welche die Anwender der Datenbank zu Papier bringen möchten. Sollen diese Ausdrucke ansprechend gestaltet sein, so kommen Sie nicht umhin, dafür Berichte zu erstellen - der Ausdruck von Formularen sieht selten gut aus. Auch wenn Access über seine Oberfläche für den Druck alles mitbringt, stoßen Sie spätestens dann auf Probleme, wenn Sie auf die eingebauten Menüelemente verzichten und den Druck per VBA steuern wollen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1408_Drucken.mdb**.

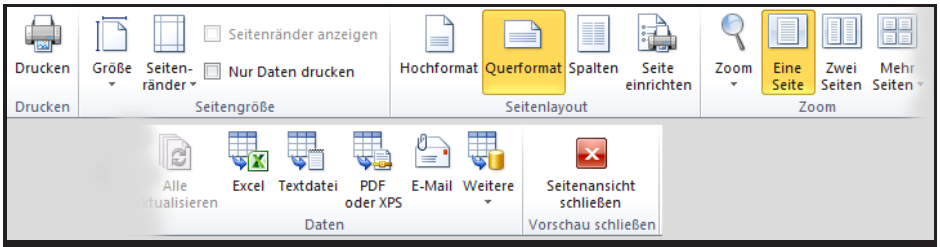


Bild 1: Ribbon bei Seitenansicht eines Berichts in Access 2010

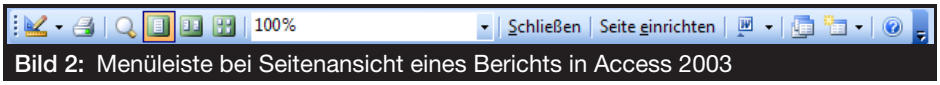


Bild 2: Menüleiste bei Seitenansicht eines Berichts in Access 2003

Drucken über Menüelemente

Öffnen Sie einen Bericht in der Seitenansicht, so präsentiert Ihnen Access ab Version 2007 ein Ribbon wie in Bild 1.

In früheren Versionen zeigt Access eine zusätzliche Menüleiste wie in Bild 2 an. Beide enthalten ein Symbol für das Drucken des Berichts. Die Funktionen unterscheiden sich indessen voneinander.

Während sich nach Klick auf das **Drucken**-Symbol im Ribbon ein Dialog zur Auswahl des gewünschten Druckers und weiterer Einstellungen öffnet (siehe Bild 3), führt dieselbe Aktion unter Access 2003 und früher unweigerlich dazu, dass der Bericht sofort auf den im System vorgegebenen Standarddrucker ausgegeben wird, oder, soweit angegeben, auf den im Berichtsentwurf über **Seite einrichten** eingestellten.

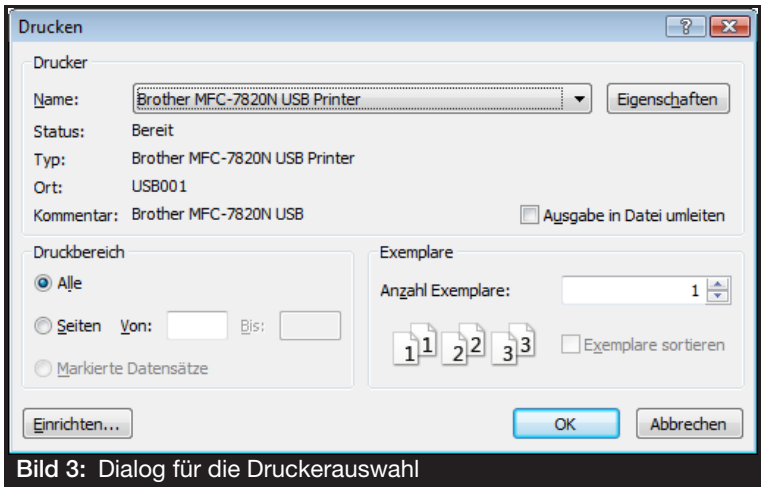


Bild 3: Dialog für die Druckerauswahl

Dabei ist es Access egal, ob dieser Drucker gerade offline ist oder etwa über das Netzwerk nicht erreichbar. Normal landet der Druckauftrag dann in der Druckerwarteschlange und wird kommentarlos ausgeführt, nachdem der Drucker online geschaltet wurde. Das jedoch bemerken Ihre Anwender unter Umständen nicht sofort und führen die Drucken-Aktion mehrmals durch. Besser wäre es, wenn Access hier eine entsprechende Mitteilung über den Status des Druckers machen würde.

Aber auch das Drucken über den Auswahldialog hindert Access nicht daran, einen nicht verfügbaren Drucker anzusprechen. Das Ergebnis sähe dann etwa aus, wie in Bild 5. Dennoch wird der Druckauftrag, je nach Druckereinrichtung, auch hier für gewöhnlich in die Warteschlange gestellt, was zu verspäteter Ausführung des Auftrags führt und die Anwender abermals verwirrt.

Sollte Ihren von merkwürdigem Druckverhalten Ihrer Datenbank berichtet werden, so lohnt sich übrigens zur Analyse ein Blick in die Ereignisanzeige von Windows. Dort werden nämlich fehlgeschlagene Druckvorgänge mit Angabe von Gründen protokolliert.

Das muss allerdings unter Windows auch korrekt eingestellt sein. Sie können das überprüfen, indem sie die Eigenschaften des Druckservers inspizieren (siehe Bild 4). Sie gelangen dorthin über den Ordner **Drucker** und dessen Menü **Datei|Servereigenschaften...**.

Ohne Administratorrechte haben Sie nur passiven Zugriff auf dessen Einstellungen. Ab Windows Vista können Sie die Häkchen für die Spooler-Einstellungen nur setzen, wenn Sie die **Druckverwaltung** über das **Startmenü|Verwaltung** öffnen. Gehen Sie dort über das Kontextmenü des lokalen Druckservers auf **Eigenschaften...|Erweitert**.

Praktisch also wäre, wenn Ihre Datenbank den Druck nur dann ausführt, wenn der Zieldrucker wirklich erreichbar ist, oder über einen Dialog nur eben diese Drucker anböte. Wie eine solche Lösung, realisiert über VBA, aussehen könnte, zeigen wir Ihnen später.

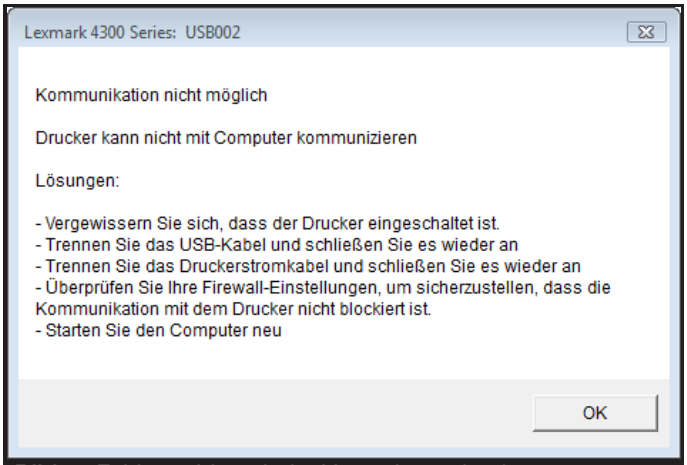


Bild 4: Fehlermeldung beim Versuch, zu drucken

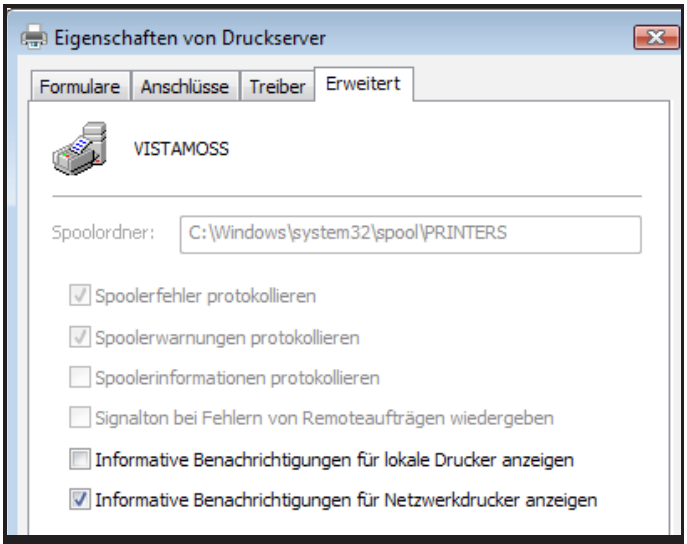


Bild 5: Systemeinstellungen für den Druckauftrag-Spooler von Windows

Abgesehen von den angesprochenen Umständen haben Sie ihre Datenbank möglicherweise aber ohnehin mit eigenen Menüleisten oder Ribbon-Definitionen versehen und möchten gar nicht, dass die eingebauten Elemente in der Seitenansicht auftauchen.

Neben den durchaus nützlichen Elementen zur Seiteneinrichtung – Format, Orientierung, Seitenränder – enthält das Ribbon auch noch weitere Symbole für den Export des Berichts nach Excel, Word und ähnliche.

Es ist keine schlechte Idee, dem Anwender diese Funktionen vorzuenthalten, denn das Layout des Berichts unter Word oder Excel entspricht keinesfalls dem Original und führt regelmäßig zu Frustrationen.

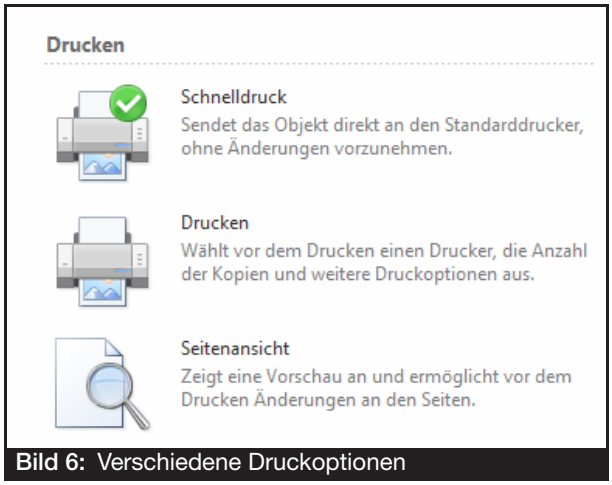


Bild 6: Verschiedene Druckoptionen

Blenden Sie also das Ribbon oder die Menüleiste zur Seitenansicht besser aus und steuern das Drucken über eigene Oberflächenelemente, wie Buttons oder eigene Menüleisten, die VBA-Code auslösen, wie im Folgenden beschrieben.

Auch wenn Sie vom direkten Ausdruck von Formularen Abstand nehmen sollten – Da-

tenblätter wären da eine Ausnahme –, sollen hier ein paar Anmerkungen dazu nicht fehlen. Öffnen Sie etwa das Formular **frmKunden** in der Beispieldatenbank.

Unter Access 2003 finden Sie in der dann erscheinenden Menüleiste ein Symbol zum Drucken. Analog zum Bericht erfolgt bei Klick eine Ausgabe auf den Standarddrucker. Den Druckerauswahl-Dialog bekommen Sie nur, wenn Sie über das Hauptmenü **Datei|Drucken...** gehen.

Ab Access 2007 gibt es das **Drucken**-Symbol im Ribbon bei geöffnetem Formular jedoch nicht mehr. Drucken können Sie das Formular unter Access 2010 etwa nur über den länglichen Umweg **Datei|Drucken**.

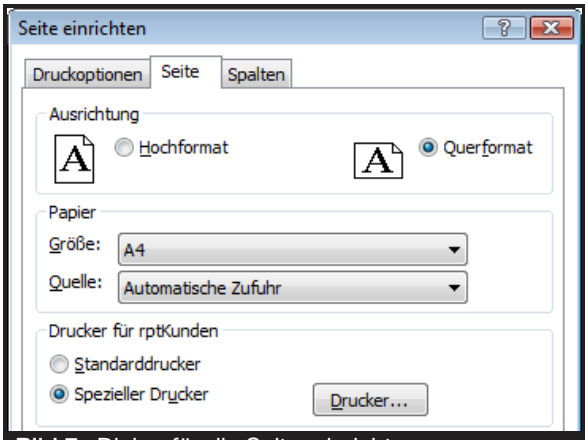


Bild 7: Dialog für die Seiteneinrichtung

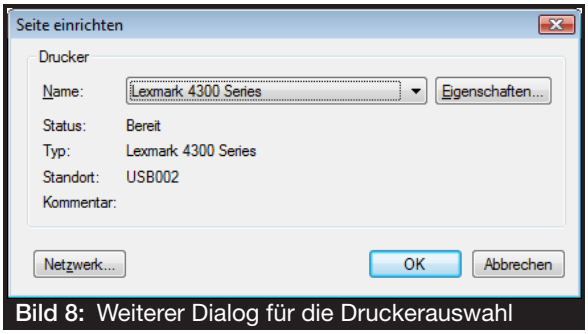


Bild 8: Weiterer Dialog für die Druckerauswahl

Im **Backstage**-Bereich sehen Sie dann die Elemente in Bild 6. Erst, wenn Sie den Eintrag zur Seitenansicht anklicken, öffnet sich das Formular in eben dieser. Im Ribbon finden sich nun die gleichen Elemente, wie beim Bericht. Die Seitenansicht kann in der Formularansicht nicht direkt aufgerufen werden.

Der Auswahl-Button zur Ansicht lässt unverständlicherweise nur die Entscheidung über Formular- oder Entwurfsansicht zu.

Druckeinstellungen im Bericht

In allen Versionen von Access kann für einen Bericht eingestellt werden, auf welchen Drucker dieser ausgegeben werden soll und wie dabei das Format und die Seiteneinstellungen aussehen sollen. All dies kann auch über VBA erfolgen, wobei dies vor Access 2003 noch sehr umständlich zu bewerkstelligen war.

Ab Access 2003 ist das Objektmodell weitgehend konsistent geblieben und bietet nun über die **Printers**-Auflistung und das **Printer**-Objekt komfortablen Zugriff auf die Druckfunktionen.

Manuell nimmt man die Einstellungen vor, indem man den Bericht entweder im Entwurf oder in der Seitenansicht lädt. Unter Access 2003 muss im Entwurf der Einstellungsdialog über das **Hauptmenü|Seite einrichten...** aufgerufen werden, in der Seitenansicht über die entsprechende Menüschaltfläche der Menüleiste **Seitenansicht**.

Ab Access 2007 ist der Aufruf selbsterklärend. Der Dialog an sich sieht in allen Access-Versionen gleich aus (siehe Bild 7). Nachdem die Einrichtung vorgenommen wurde, muss der Bericht noch mit den neuen Angaben gespeichert werden, was ebenfalls in der Seitenansicht möglich ist und nicht nur im Entwurf.

Neben den Einstellungen zu den Seitenrändern können Sie über den Reiter **Seite** das Ausgabeformat festlegen und außerdem dezidiert einen bestimmten Drucker angeben, auf den der Bericht standardmäßig gedruckt werden soll.

Diesen erreichen Sie über die Schaltfläche **Drucker...**, die den weiteren Dialog in Bild 8 öffnet. Das Festlegen eines bestimmten Druckers sollten Sie jedoch vermeiden, weil sonst unter Umständen jene Fehlfunktion wie oben dargestellt auftreten kann.

Belassen Sie es beim Standarddrucker, der gewöhnlich auf jedem System eingerichtet ist. Als Entwickler werden Sie wahrscheinlich ohnehin nicht über genau die Druckerumgebung verfügen, mit denen Ihre Anwender arbeiten.

Druckeinstellungen per VBA

Die Mühe, für jeden Bericht manuell Seiteneinstellungen einzurichten, brauchen Sie sich nicht unbedingt zu machen, da alles, was über den

Anwendungseinstellungen abspeichern

Sie haben eine umfangreiche Datenbank entwickelt und möchten Ihren Anwendern die Möglichkeit bieten bestimmte Einstellungen etwa der Benutzeroberfläche selbst festlegen zu können – so, wie sie das in anderen Anwendungen von den Einstellungen- und Optionen-Menüs her kennen. Damit lassen sich etwa das Erscheinungsbild steuern oder Pfade für Auswahldialoge vorgeben. Wie sich das sinnvoll per VBA umsetzen lässt möchten wir in diesem Beitrag beleuchten.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1408_Einstellungen.mdb**.

Benutzerdefinierte Einstellungen der Datenbank
Viele Einstellungen einer Datenbank verwaltet Access selbst. Sie können bei der Entwicklung etwa über das Optionen-Menü verschiedene Einstellungen fest vorgeben. Dort finden sich etwa Vorgaben für das Layout von Datenblättern, die Darstellung von Menüs oder die Ansicht von Steuerelementen, aber auch technische Vorgaben, wie das Verhalten bei Datensatzsperrung.

Unter Access 2003 war noch nicht ersichtlich, ob sich diese Vorgaben auf den Arbeitsplatz, also Access generell, beziehen, oder auf die eben geöffnete Datenbank. Unter Access 2007 wurde im Optionen-Dialog dann der Bereich **Aktuelle Datenbank** eingeführt, der deutlich macht, was genau in der geöffneten Datenbank festgelegt wird und nur in dieser.

Doch auch der Anwender kann manche Änderungen dauerhaft abspeichern. Änderungen an Spaltenbreiten von Datenblättern oder an der Größe von Formularen werden verewigt, wenn anschließend über Menü oder Symbolleiste die Speichern-Aktion ausgeführt wird. Die Schriftart der Steuerelemente eines Formulars etwa kann er hingegen nicht beeinflussen.

Wollten Sie dem Anwender diese Möglichkeit einräumen, so müssten neben dem Code, der die Schriften der Textfelder beim Laden des Formulars steuert, und einem Dialog, der die Schriften anböte, auch noch das Feature hinzukommen, die Schrifteinstellungen irgendwo abzuspeichern, damit sie beim nächsten Laden wieder zur Anwendung gelangten. Oft ist die Aufgabe jedoch trivialer. Besteht beispielsweise in der Anwendung die Notwendigkeit, öfter Dateien per Auswahldialog auszuwählen, dann ist es sicher nützlich, wenn man jeweils im zuvor geöffneten Ordner landet und nicht immer neu dorthin navigieren muss.

Der zuletzt ausgewählte Pfad könnte hier abgespeichert und bei erneutem Öffnen des Auswahldialogs gleich angeboten werden. Oder nehmen wir den selbstgebauten Drucker-Dialog aus dem Artikel **Access-Berichte drucken**: Günstig wäre, wenn hier der zuletzt verwendete Drucker im Kombinationsfeld voreingestellt wäre und nicht jedes Mal neu in der Liste gesucht werden müsste.

Nun stellt sich die Frage, wo genau die Parameter dieser Einstellungen abgespeichert werden sollten?

Entwicklereinstellungen und Anwendereinstellungen
Diese beiden Bereiche wären zu trennen. Es gibt Einstellungen, die der Anwender nicht beeinflussen können sollte. Nehmen wir an, sie hätten eine Schnittstelle zu einem Webservice in Ihrer Datenbank implementiert. Die URL zu diesem Webservice kann sich ändern. Statt die URL als String hart in ein VBA-Modul zu kodieren, eignet sich eine benutzerdefinierte abspeicherbare Variable für diesem Zweck besser. So müssen Sie nur den Wert der Speicherstelle ändern und kein Modul modifizieren. Der Anwender sollte von alledem nichts mitbekommen.

Umgekehrt braucht Sie als Entwickler nicht zu interessieren, welchen Drucker der Anwender verwendet oder einstellt. Je nach Anwender- oder Entwicklerbereich unterscheiden sich dann auch die Speicherorte für die Einstellungsvariablen. Ein gewichtiger Grund dafür : Wenn Sie ein Update Ihrer Datenbank verteilen, die Einstellungen aber in der Datenbank selbst gespeichert sind, so gehen alle vom Anwender vorgenommenen Änderungen flöten und er wird mit

Ihren Entwicklereinstellungen konfrontiert. Deshalb müssen Anwendereinstellungen zwingend extern abgespeichert werden.

Speicherorte für Einstellungen

Möglichkeiten bieten sich dafür viele an. Früher gab es einmal INI-Dateien, die Einstellungen aufnahmen. Heute verwendet man stattdessen die Registry – dazu mehr in einem anderen Beitrag dieser Ausgabe. Statt INI-Dateien kommen auch einfache Textdateien infrage, in die per VBA und dessen Open-Anweisung Variablenwerte geschrieben würden.

Doch was liegt in einer Datenbank näher, als solche Daten in Tabellen zu schreiben? Für die Entwicklereinstellungen nimmt man eine Tabelle in der Anwendungsdatenbank selbst, für die Anwender eine externe verknüpfte Datenbank, die nur die eine Tabelle mit den Einstellungen enthält. Als weitere Alternative kämen auch andere Datenbanksysteme infrage, wie etwa SQLite, das für diesen Zweck von diversen Applikationen verwendet wird – der Firefox Browser ist ein Beispiel dafür. Doch warum in die Ferne schweifen...
Schließlich gibt es noch eine weitere Alternative, derer sich auch Access selbst bedient: Die Datenbank-Properties.

Datenbank-Properties

Access speichert die ganzen Optionen, die Sie über das entsprechende Menü

```
Sub ListProperties()
    Dim dbs As Database
    Dim oPrp As DAO.Property
    Dim n As Long
    Set dbs = CurrentDb
    For n = 0 To dbs.Properties.Count - 1
        Set oPrp = dbs.Properties(n)
        If oPrp.Name <> "Connection" Then
            Debug.Print n, oPrp.Name, _
                TypeName(oPrp.Value), CStr(oPrp.Value)
        End If
        Set oPrp = Nothing
    Next n
End Sub
```

Listing 1: Alle Datenbank-Properties im VBA-Direktfenster ausgeben

öffnen, in den Properties der Datenbank. Über VBA und DAO erhalten Sie Zugriff auf die Datenbank über das Zauberwort **CurrentDb**. Dessen Eigenschaften sind die

```
CurrentDB.Properties
```

Es handelt sich dabei um ein Auflistungsobjekt, das, wie so oft, entweder über eine Ordinalzahl oder einen String abgefragt werden kann.

```
CurrentDb.Properties(33).Value
```

und

```
CurrentDb.Properties
        ("AllowSpecialKeys").Value
```

sind dasselbe und stellen den Wert der Eigenschaft "Access-Spezialtasten zulassen" dar. Diese Auflistung ist nicht statisch, sondern kann selbst erweitert werden. Heißt: Neue Elemente können hinzugefügt, geändert, oder gelöscht werden. So auch Variablen für die eigenen Einstellungen.

Übrigens speichert Access diese Properties auch wieder in einer Tabelle – wo auch sonst? In diesem Fall ist es die ausgeblendete Systemtabelle **MSysObjects** und dort das Feld **LvProp** des Datensatzes **MSysDb** (siehe Ausschnitt in Bild 1). Die Binärdaten darin haben einen proprietären Aufbau.

Sie können sich alle Datenbank-Properties ansehen, indem Sie eine Routine wie in Listing 1 aus dem Modul **mdlProperties** verwenden. In dieser werden alle Properties ausgelesen und die Namen, Datentypen und Werte im VBA-Direktfenster ausgegeben. Komfortabler geht es mit dem Formular **frmProps** der Beispieldatenbank, die eine ähnliche Prozedur nutzt, um die Properties in einem Listenfeld darzustellen (siehe Bild 2).

Die Prozedur durchläuft alle Elemente der **Properties**-Auflistung in einer Schleife, und setzt sie jeweils auf **oPrp**, eine **DAO.Property**-Variable. Dieser Objekttyp hat die wesentlichen Eigenschaften **Name**, **Type** und **Value**. Als Value kann ein beliebiger **Variant**-Wert gesetzt sein, **Type** ist auf den Datentyp der Eigenschaft eingestellt. Die Schleife einhält eine Ausnahme: Die Standardeigenschaft **Connection** muss außen vor bleiben, weil sie einen

LvModule	LvProp	Name
	Long binary-Daten	MSysDb

Bild 1: Properties in der Tabelle **MSysObjects**

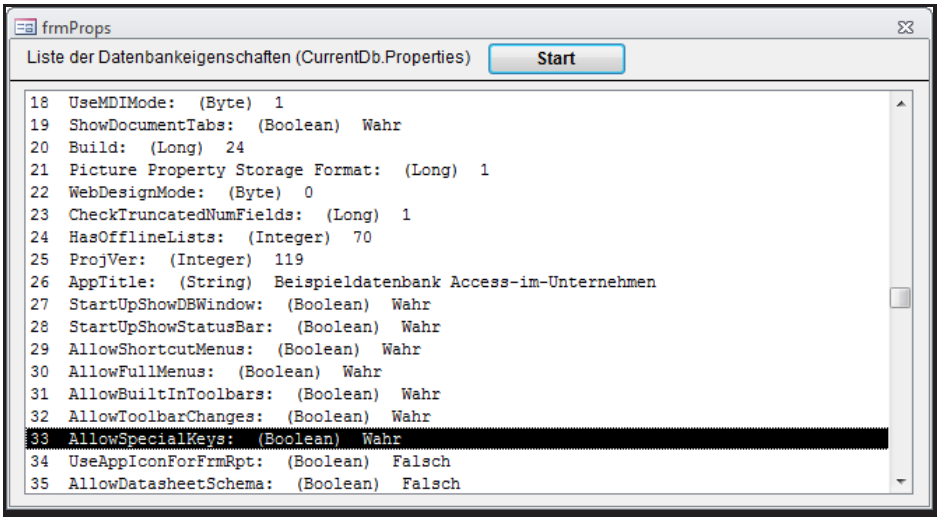


Bild 2: Eigenschaften der Datenbank im Formula **frmProps** der Beispieldatenbank

Datentyp aufweist, den VBA nicht verarbeiten kann. Wundern Sie sich nicht, dass die Routine an manchen Stellen stockt. Für das Auslesen mancher eingebauter Eigenschaften braucht Access offenbar etwas Bedenkzeit.

Mehr als das Auslesen, interessiert uns die Anlage neuer Properties. Auch hierzu finden Sie eine allgemeinverwendbare Routine in der Beispieldatenbank, die Prozedur **SetProperty** (Listing 2). Als Parameter übergeben Sie ihr den Namen der Eigenschaft sowie ihren Wert. Es spielt dabei keine Rolle, ob die Eigenschaft bereits existiert oder nicht. Denn zunächst versucht die Prozedur das **Property**-Objekt auf eine existierende Eigenschaft zu setzen. Gibt es diese unter dem übergebenen Namen nicht, so löst dies normal einen Fehler aus. Die Fehlerbehandlung wurde jedoch außer Kraft gesetzt. Bei Erfolg ist das **Property**-Objekt gültig, bei Misserfolg entspricht es **Nothing**. Davon hängt ab, wie der Code weiter verzweigt. Ist das Objekt leer, so fährt er mit dem Erzeugen der Eigenschaft per **CreateProperty** fort. Dieser Methode der **Properties**-Auflistung ist neben dem Namen und Wert der Eigenschaft auch noch der Datentyp in Form einer Konstanten zu übergeben.

Leider handelt es sich hier um eine DAO-Konstante, nicht um eine VBA-Konstante, denn sonst könnte der Typ des Werts **Value** einfach über die VBA-Funktion **VarType**() ermittelt werden. Deshalb ist im Modul eine Hilfsfunktion **VbTypeToDAO** eingebaut, die die Übersetzung von VBA-Datentyp zu DAO-Datentyp gewährleistet. In Kurzform enthält es Zeilen wie diese:

```
If VarType (Value) = vbDate Then
        Result = DAO.dbDate
```

Properties verwenden

Machen wir die Probe und legen mithilfe dieser **SetProperty**-Prozedur eine neue Eigenschaft an:

```
SetProperty "Version", "1.0"
```

Das Auslesen gibt uns indessen Ratsel auf:

```
? CurrentDb.Properties("Version")
```

Wir erhalten seltsamerweise "4.0". Der Grund dafür ist, dass diese Eigenschaft von Access aus bereits existiert, aber schreibgeschützt ist. Sie müssen also darauf achten, dass sie bei eigenen Properties nicht die Namen verwenden, die Access intern verwendet. Die reservierten Namen erhalten Sie ja über das Formular (Bild 2). Nehmen Sie im Beispiel stattdessen den Namen **DBVersion**,

```
Sub SetProperty(sName, Value As Variant)
    Dim dbs As Database
    Dim oPrp As DAO.Property
    Set dbs = CurrentDb
    On Error Resume Next
    Set oPrp = dbs.Properties(sName)
    If oPrp Is Nothing Then
        On Error GoTo 0
        Set oPrp = dbs.CreateProperty(sName, VbTypeToDAO(Value), Value)
        dbs.Properties.Append oPrp
    Else
        On Error GoTo 0
        oPrp.Value = Value
    End If
End Sub
```

Listing 2: Code zum Anlegen neuer Properties

Registry-Zugriff per VBA

Die Windows-Registrierungsdatenbank, kurz Registry, ist die zentrale Sammelstelle für Einstellungsdaten aller Art. Die Konfiguration des gesamten Systems und fast aller Anwendungen hängt von ihr ab. Ob lesend oder schreibend, auch als Access-Entwickler hätte man gern eine Programmierschnittstelle für den Zugriff auf die Registry. Weil VBA hier nur eingeschränkte Möglichkeiten bietet, ist dem Thema dieser Beitrag gewidmet.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1408_Registry.mdb**.

Eingebaute Zugriffsmethoden

Im Beitrag **Anwendungseinstellungen abspeichern** dieser Ausgabe wurde deutlich, dass die Werte von Eigenschaftsvariablen, die über die aktuelle Access-Sitzung hinaus Bestand haben sollen, außerhalb der Datenbank selbst gespeichert werden müssen. Statt dafür eine externe verknüpfte Tabelle zu verwenden, könnte man ebenso Variablen in der Registry unterbringen.

Genau dieser Zweck schwebte den VBA-Entwicklern wohl vor, als sie die vier Methoden zum Zugriff auf die Registry in die VBA-Bibliothek einbauten. Sie lauten

```
GetSetting
GetAllSettings
SaveSetting
DeleteSetting
```

GetSetting liest einen Registry-Wert aus, **GetAllSettings** liest alle Werte eines Schlüssels auf einmal, **SaveSetting** schreibt einen Wert in die Registry, und **DeleteSetting** löscht einen solchen wieder. Der Haken an der Sache ist, dass all diese Methoden nur einen ganz kleinen reservierten Bereich der Registry ansprechen können, nämlich den Schlüssel

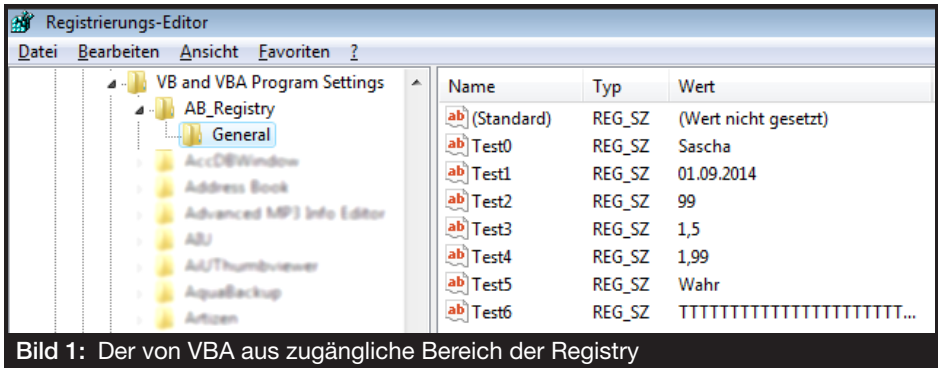


Bild 1: Der von VBA aus zugängliche Bereich der Registry

```
HKEY_CURRENT_USER\Software\
  VB and VBA Program Settings
```

Unterhalb dieses Schlüssels kann man walten, wie man will, aber von allen anderen Bereichen der Registry ist man ausgesperrt. Dass dieser Schlüssel unter **HKEY_CURRENT_USER** liegt, ist nicht verwunderlich, denn für alle anderen Root-Schlüssel benötigt man schreibend Administratorrechte – und Access wird normalerweise nicht mit diesen Rechten gestartet.

Die Schlüsselstruktur unterhalb dieses Hauptschlüssels ist immer dieselbe. Sie wird in Bild 1 ersichtlich. Dort ist ein Zweig abgebildet, der über die Routinen der Beispieldatenbank erstellt wurde. **AB_Registry** als Unterschlüssel ist quasi der Generalschlüssel und Anwendungsname, und darunter kann ein oder können mehrere Bereichsschlüssel liegen. Im Beispiel ist es nur einer mit dem Namen **General**. Werte können außerdem nur im untersten Schlüssel gespeichert werden, also nicht etwa unter **AB_Registry**. Sie können eine beliebige Anzahl von Werten unterbringen. Bei größerem Umfang ist der Übersicht halber aber die Anlage mehrerer Unterschlüssel überlegenswert. Zum **General**-Schlüssel könnte etwa noch ein **Printer**-Schlüssel kommen, der Werte für die Steuerung von Druckern aufnähme.

Wie kommt nun der Test-Eintrag zustande? Die Syntax der Code-Zeile sieht so aus:

```
SaveSetting "AB_Registry", "General",
"Test0", "Sascha"
```

Als Parameter werden also Hauptschlüssel, Unterschlüssel, Wertname und Wert übergeben. Die Schlüssel müssen zuvor nicht erst angelegt worden sein, wie das etwa bei Routinen, die das Windows-API benutzen, der Fall ist. Die **SaveSetting**-Anweisung macht alles auf einen Schlag: Hauptschlüssel, Unterschlüssel anlegen, Wert speichern. Als Wert können Sie einen beliebigen Variant-Typ übergeben und das selbstredend auch über eine Variable. Erlaubt sind alle Standarddatentypen, nicht jedoch Arrays oder Objekte. Es spielt keine Rolle, welchen Datentyp Sie verwenden, denn alle werden automatisch in Strings verwandelt. In Bild 1 wird das aus der Typ-Spalte des Editors ersichtlich, wo alle Werte als **REG_SZ** aufgeführt sind, dem Pendant zu **vbString**.

Mit **GetSetting** lesen Sie den Wert über diese Syntax aus:

```
? GetSetting ("AB_Registry", "General", "Test0", "nix")
```

Die Parameter sind also Hauptschlüssel, Unterschlüssel, Wertname, und die Rückgabe ist immer vom Typ **String**. Ganz hinten können Sie optional noch einen weiteren Parameter anschließen. Das ist ein Wert, der immer dann als Resultat der Funktion zurückgegeben wird, wenn entweder der Registry-Wert leer ist, der Wertname nicht existiert, oder es den Schlüssel selbst nicht gibt. VBA löst nämlich keine Fehler aus, wenn die Namensparameter ungültig sind!

Im Modul **mdlRegistryBeispiele** der Beispieldatenbank finden Sie mehrere Prozeduren, die den Umgang mit den vier VBA-Methoden vereinfachen. Einen Ausschnitt daraus zeigt Listing 1. Statt etwa **GetSetting** immer mit allen fünf Parametern bestücken zu müssen, nehmen Sie einfach die kürzere Variante **SaveVarToRegistry**, welche nur einen Parameter erwartet: den Wertnamen.

Die Prozedur enthält nur eine Zeile mit der **GetSetting**-Funktion, der ihrerseits als Parameter aber zwei Konstanten übergeben werden, die oben im Kopf des Moduls global deklariert wurden: die Namen der Haupt- und Unterschlüssel. Denn diese dürften für die Datenbank konstant bleiben. Wenn Sie das Modul in eine andere Datenbank importieren, so ändern Sie dort nur diese zwei Konstanten. Das erspart Ihnen, überall im Code die Schlüsselnamen neu manuell einzufügen.

Die weiteren drei Prozeduren des Moduls arbeiten ähnlich und sind wohl selbsterklärend. Lediglich der **GetVarsFromRegistry** wäre gesonderte Aufmerksamkeit zu schenken. Sie verwendet die **GetAllSettings**-Funktion von VBA. Die Rückgabe ist ein Variant. Was sich hinter dieser Rückgabe verbirgt, macht die Prozedur **TestGetAllVars** deutlich:

```
Sub TestGetAllVars()
Dim i As Long
Dim V As Variant
V = GetVarsFromRegistry
For i = 0 To UBound(V, 1)
  Debug.Print _
        V(i, 0) & " = " & V(i, 1)
Next i
End Sub
```

Die Schleife zeigt, dass es sich bei der Variant-Rückgabe um ein zweidimensionales Array handelt. Die erste Dimension entspricht der Zahl der zurückgegebenen Registry-Werte. Die zweite hat immer zwei Elemente und enthält entweder den Namen oder den Wert des Registry-Eintrags.

```
Public Const cMainRegKey = "AB_Registry"
Public Const cSubRegKey = "General"

Sub SaveVarToRegistry(ByVal KeyName As String, Value As Variant)
    SaveSetting cMainRegKey, cSubRegKey, KeyName, Value
End Sub

Function GetVarFromRegistry(ByVal KeyName) As Variant
    GetVarFromRegistry = GetSetting(cMainRegKey, _
        cSubRegKey, KeyName, "LEER")
End Function

Sub RemoveVarFromRegistry(ByVal KeyName As String)
    DeleteSetting cMainRegKey, cSubRegKey, KeyName
End Sub

Function GetVarsFromRegistry() As Variant
    GetVarsFromRegistry = GetAllSettings(cMainRegKey, _
        cSubRegKey)
End Function
```

Listing 1: Ausschnitt aus dem Modul zum Bearbeiten von Registry-Werten

V(0,0) wäre der Name des ersten Eintrag, **V(0,1)** sein Wert. und so weiter.

Probleme mit den eingebauten Methoden

Wie erwähnt wandelt VBA jeglichen Datentyp beim Speichern in die Registry in einen **String** um. Beim Auslesen erhält man infolgedessen immer einen **String**. Das kann zu Problemen führen, wenn etwa ein **Boolean**-Wert abgespeichert wurde, in der Registry dann der String **Wahr** steht und beim Auslesen wieder einer **Boolean**-Variablen zugewiesen werden soll. Zwar ist VBA in der Regel intelligent genug, auch hier die Rückkonvertierung zu gewährleisten. Doch möchten Sie den **Boolean**-Wert einer **Long**-Variablen zuweisen, so geht die Sache schief. VBA kann den **String**-Ausdruck **Wahr** nicht in einen **Long**-Wert verwandeln, während der direkte Weg von **Boolean** zu **Long** (**-1**) problemlos funktioniert. Ähnliche Konvertierungsprobleme stehen mit dem Datumstyp an. Kann man normal einen **vbDate**-Typ einer **Double**-Variablen übergeben, funktioniert dies nicht, wenn der Ausgangswert ein String der Form **1.09.2014** ist.

Daraus folgt, dass beim Auslesen von Registry-Werten grundsätzlich der gleiche Variablentyp verwendet werden sollte, wie beim Speichern. Nur dann kommt VB mit der zweifachen Konvertierung klar.

Komplettzugriff auf die Registry

Wenn Sie mehr wollen, als nur den beschränkten Bereich, den Ihnen VBA anbietet, dann brauchen Sie entweder aufwändige API-Lösungen, oder zusätzliche Bibliotheken, die in die Verweise des Projekts aufgenommen werden. Die API-Variante folgt im nächsten Abschnitt. Hier zunächst die Lösung per zusätzlicher Bibliothek. Davor jedoch noch der Hinweis, dass mit beiden Varianten zwar der lesende Zugriff auf fast alle Bereiche der Registry möglich ist, der schreibende aber nur auf die Schlüssel unter **HKEY_CURRENT_USER**.

Die wahrscheinlich einfachste Bibliothek, welche Methoden zum Zugriff auf die Registry ermöglicht, ist das **Windows Scripting Host Object Model**, kurz **WSHOM**. Die DLL dazu findet sich auf allen Windows-Plattformen im Systemverzeichnis und lautet **wshom.ocx**. Nachdem Sie die Bibliothek in die Verweise des VBA-Projekts aufgenommen haben, finden Sie in der Objektklasse **WshShell** die drei auf die Registry bezogenen Methoden **RegDelete**, **RegRead** und **RegWrite** (siehe Bild 2). Deren Syntax ist verblüffend simpel. So erwartet die **RegRead**-Funktion lediglich die Angabe des Schlüssels/Wertnamen-Strings:

```
? RegRead ("HKEY_CURRENT_USER\
Software\VB and VBA Program Settings\
AB_Registry\General\Test0")
```

Im Unterschied zu den VBA-Methoden ereignet sich ein Fehler, wenn Schlüssel- oder Wertnamen nicht gültig sind. Im Modul **mdlWSHOM** der Beispieldatenbank ist deshalb eine Fehlerbehandlung in die entsprechenden Prozeduren zum Lesen und Schreiben eingebaut. Bei ungültigen Namensparametern zeigt sich eine **MsgBox** mit der Meldung "Ungültiger Registry Key". Die Prozedur **SetVarWSHOM** des Moduls zum Schreiben von Registry-Werten ist etwas aufwändiger gestaltet. Sie macht sich zunutze, dass die **RegWrite**-Methode des **WshShell**-Objekts neben Schlüssel/Wertname-Kombination und dem Wert auch noch den Datentyp erwartet. Allerdings nicht VB-Datentypen, mit denen die Registry nichts anfangen kann, sondern deren eigene Typen. Für uns sind dabei nur die beiden Typen **REG_SZ** und **REG_DWORD** von Belang. Ein **DWORD** entspricht dem VB-Typ **Long**. Damit lassen sich hier immerhin die VB-Typen **Long**, **Integer** und **Boolean** von den restlichen absetzen und als **DWORD** speichern. Beim Auslesen kommt dann eben auch ein

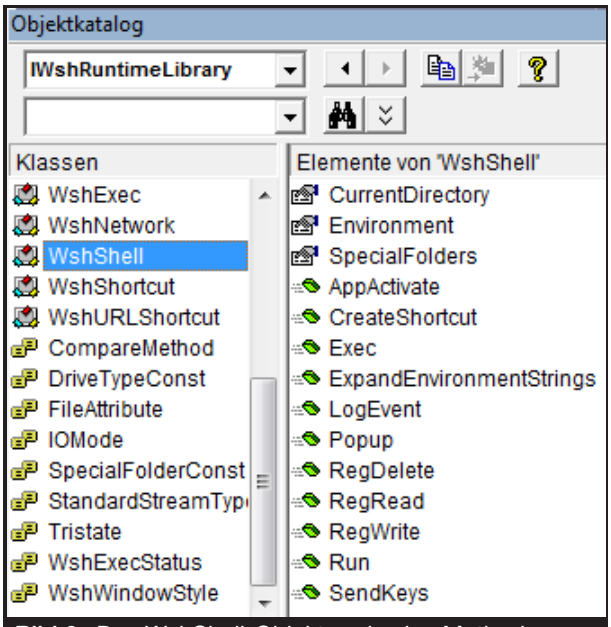


Bild 2: Das WshShell-Objekt und seine Methoden zum Zugriff auf die Registy

Filtern und Suchen

Sobald Daten in größerem Umfang anfallen, sind Such- und Filtermechanismen in Datenbanken gefragt. Ohne Filtern geht der Überblick verloren und für Auswertungen sind Sie das A und O. Abseits der Möglichkeiten, die Abfragen und VBA bereitstellen, hat Access über seine Benutzeroberfläche mehr zu bieten, als auf den ersten Blick offenbar wird.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1409_FilternUndSuchen.mdb**.

Indexname	Feldname	Sortierreihenfolge
PrimaryKey	ID	Aufsteigend
idxNat	Nat	Aufsteigend
idxGeburtsdatum	Geburtsdatum	Aufsteigend
idxAnrede	IDAnrede	Aufsteigend
idxNachname	Nachname	Aufsteigend
idxVorname	Vorname	Aufsteigend
idxNameVorname	Nachname	Aufsteigend
	Vorname	Aufsteigend
idxStrasse	Strasse	Aufsteigend
idxNr	NrStrasse	Aufsteigend
idxPLZ	PLZ	Aufsteigend
idxOrt	IDOrt	Aufsteigend
idxLand	IDLand	Aufsteigend

Bild 1: Indizes der Tabelle tblStammdaten

Individuelle Suchfunktion

Wenn Sie eine Datenbank für andere Benutzer bereitstellen, so ist es möglicherweise sinnvoller, ihnen über Formulare programmierte Such- und Filterfunktionen anzubieten, weil die über die Oberfläche von Access verfügbaren nicht so ohne weiteres auffallen und etwas der Schulung bedürfen.

In Formularen etwa wird erst über das Kontextmenü deutlich, dass Filter aufrufbar sind oder sich Datensätze durchsuchen lassen.

Bei Datenblattansichten ist sind die Funktionen schon deutlicher sichtbar, weil sich hier bei Klick auf die Spaltenköpfe – zumindest unter Access 2007 und neuer – ein entsprechendes Menü öffnet.

Arbeiten Sie selbst mit der Datenbank, dann sollten Sie jedoch die Filterfunktionen der Oberfläche einmal gründlicher durchforsten.

Anforderungen an zu filternde Tabellen

Egal, ob Filtern oder Suchen im Fokus steht: Indizierung ist das Zauberwort. Das Indizieren von Tabellenfeldern wurde eben für diesen Zweck geschaffen. Es beschleunigt Suchvorgänge ungemein. Nehmen Sie eine Tabelle mit Zigtausend Datensätze und einem Feld für den **Nachnamen**. Ohne Indizierung müsste die **Access Database Engine** jeden einzelnen Datensatz aufrufen und den Nachnamen mit einem gesuchten Begriff vergleichen. Das würde sehr viel Zeit kosten. Ist ein Index auf das Feld gesetzt, so legt Access intern eine Datenstruktur an, ein Indextabelle, die über die Oberfläche von Access allerdings nicht sichtbar wird. In diesem Index sind die Datensatzinhalte etwa alphabetisch angeordnet und mit Sprungadressen versehen.

Der Name "Sacher" wird gefunden, indem nur der Block zu "S" analysiert wird, der an einer bestimmten Adresse liegt. Zumindest ist das das Prinzip – in der Realität sieht die Sache weitaus komplizierter aus und die eingesetzten BTree-Algorithmen sind auf Performance optimiert.

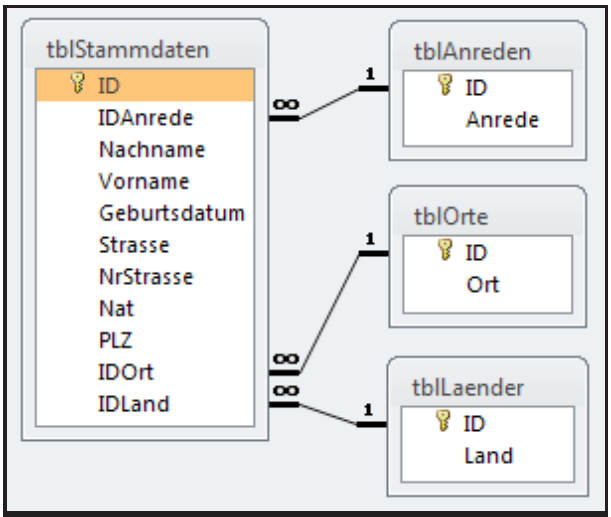


Bild 2: Beziehungslayout der Beispieltdatenbank

In Bild 2 sehen Sie den Tabellenaufbau der Beispieldatenbank mit der Haupttabelle **tblStammdaten**. Bild 1 sind deren Indizes aus der Entwurfsansicht abgebildet. Es ist auf fast alle Felder ein Index gesetzt. Das bedeutet, dass für jedes dieser Felder Such-, Filter, Sortierungs- und Gruppierungsfunktionen schneller ablaufen.

Alle Tabellenfelder also, bei denen davon auszugehen ist, dass sie später Suchvorgängen unterzogen werden, sollten mit einem Index versehen werden. Beim Feld für die Hausnummer der Stammdatentabelle ist dies nicht anzunehmen.

Warum legt Access nicht einfach automatisch für jedes Feld einen Index an? Ganz einfach: die Indexstrukturen belegen viel Speicherplatz. Der Speicherbedarf der Indizes einer Tabelle kann sogar größer werden als deren Inhalte selbst!

Und außerdem bedarf es wesentlich mehr Rechenaufwand, um nicht nur etwa einen Namen in der Tabelle zu speichern, sondern diesen zusätzlich in eine Indexstruktur zu überführen und die in der Indextabelle an der richtigen Stelle unterzubringen.

Letzteres vollzieht Access übrigens beim Speichern eines Datensatzes aus Performancegründen nicht optimal. Sonst müssten die gesamten Indexstrukturen jedes Mal erneut aufgebaut werden.

Genau das passiert aber, wenn Sie die Anweisung **Komprimieren und Reparieren** für die Datenbank über

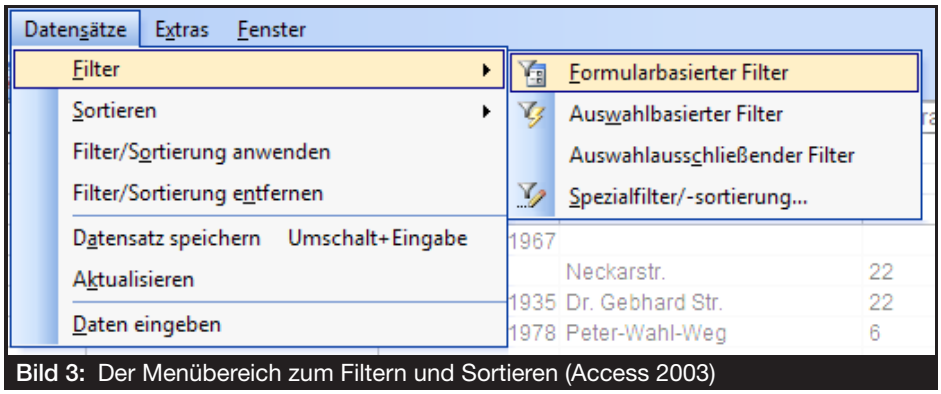


Bild 3: Der Menübereich zum Filtern und Sortieren (Access 2003)

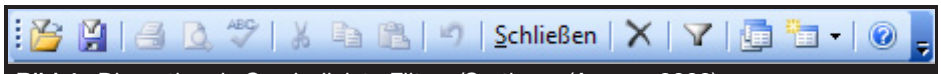


Bild 4: Die optionale Symbolleiste Filtern/Sortieren (Access 2003)

das entsprechende Menü oder den Ribbon aufrufen. Deshalb sollte dieser Vorgang nach häufigen Speicher- oder Löschvorgängen in der Datenbank hin und wieder ausgeführt werden.

Filtern über das Menü und die Symbolleiste

Unter Access 2003 und früher finden sich die Funktionen zum Suchen und Filtern über den Hauptmenüeintrag **Datensätze**. Was sich hier ab Access 2007 geändert hat, erfahren Sie später.

Öffnen Sie die Tabelle **tblStammdaten** in der Datenblattansicht. Die Menüleiste erweitert sich dann unter anderem um den Popup-Button **Datensätze**, der bei Klick das verschachtelte Untermenü in Bild 3 öffnet. Der obere Teil davon ist den Filter- und Sortierfunktionen spendiert.

Gleichzeitig blendet sich eine zusätzliche Symbolleiste, die den Namen **Tabellendatenblatt** trägt. Ein Ausschnitt mit den für das Filtern relevanten Buttons ist in Bild 5 zu sehen. Wenn Sie deren Symbole mit denen des Menüs vergleichen, wird klar, was sie bedeuten. Es spielt auch keine Rolle, ob Sie eine Funktion über das Menü oder einen Symbolleisten-Button aufrufen.

Bild 5: Die Menüleiste mit den Symbolen zum Filtern und Sortieren (A2003)

Rätsel gibt indessen die Symbolleiste **Filtern/Sortieren** auf, die Sie optional einblenden können, nachdem Sie sie über **Extras > Anpassen...** in der Liste der verfügbaren Leisten aktiviert haben (Bild 4). Seltsamerweise finden sich hier weder Buttons zum Sortieren, noch die spezielleren für das Filtern.

Nur das eine Trichtersymbol verweist auf den Filter. Es ist allerdings deaktiviert, wenn das geöffnete Datenblatt keinen gespeicherten Filter besitzt – dazu später mehr. Sie können diese Leiste also zunächst vergessen.

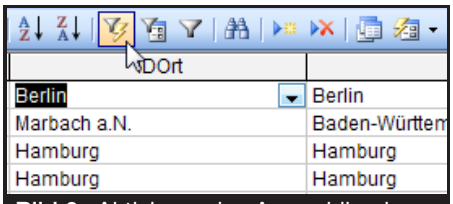


Bild 6: Aktivieren des Auswahlbasierten Filters

An der Funktion zum Sortieren gibt es nichts zu erläutern. Die aufsteigende Sortierung A-Z sortiert die aktive Spalte, also jene, in der sich der Cursor befindet, alphabetisch oder bei numerischem Inhalt nach dem Zahlenwert. Liegt der Cursor hingegen in einem OLE-Feld, wie bei der Spalte **Bild** der Stammdatentabelle, so deaktivieren sich die Sortierschaltflächen und -menüeinträge.

Die Suchfunktionen (Fernglassymbol) sollen später noch beleuchtet werden.

Auswahlbasierter Filter

Die Funktion dieses Filters setzt voraus, dass Sie eine beliebige Zelle des offenen Datenblatts markiert haben, wie in Bild 6 für den Ort "Berlin".

Sobald Sie den Blitz-Button aktivieren, verschwinden alle Datensätze, bei denen Berlin nicht als Ort eingetragen war. Gleichzeitig wird der generelle Filter-Button (Trichter) markiert, um zu verdeutlichen, dass gerade ein Filter auf das Datenblatt gesetzt ist. Wenn Sie diesen markierten Button anklicken, wird der Filter wieder entfernt und alle Datensätze angezeigt.

Sie können aber auch die Zelle einer anderen Spalte markieren und erneut den **Auswahlbasierten Filter** aktivieren. Versuchen Sie das etwa für den Eintrag "Herr" in der Spalte **Anrede** der Stammdatentabelle. Nun werden alle Datensätze angezeigt, die Kunden mit der Anrede "Herr" aufweisen **und** in Berlin wohnen.

#Der Filter arbeitet also kumulativ. Diese Schritte können für eine beliebige Anzahl von Felder wiederholen und den Filter immer weiter kumulieren. Eine Rückgängig-Funktion dafür existiert allerdings nicht. Sie können die gesetzten Filter nur auf einen Schlag entfernen.

Auswahlausschließender Filter

Dieser Filter wirkt umgekehrt, wie der zuvor besprochene. Hätten Sie ihn im Beispiel von Bild 6 benutzt, so kämen alle Datensätze **ohne** den Ort Berlin auf den Schirm. Auch dieser Filter arbeitet kumulativ. Sie können so alle Herren ausschließen, und außerdem alle Kunden, die in Berlin wohnen.

Da eine derartige Filterung nicht so trivial ist und wohl auch seltener Anwendung findet, hat Microsoft ihr auch keinen eigenen Button in der Symbolleiste spendiert. Der Auswahlausschließende Filter lässt sich nur über das Menü oder Kontextmenü aufrufen.

Filter speichern

Es spielt keine Rolle, ob Sie im angezeigten Datenblatt gerade einen Filter aktiviert haben, oder ihn über **Filter/Sortierung entfernen** deaktiviert haben: Beim Schließen des Datenblatts über das Kreuzchen werden Sie gefragt, ob die Änderungen gespeichert werden sollen. Bejahen Sie dies. Nun sollte man erwarten, dass beim nächsten Öffnen der Filter wieder angewendet ist, wenn er zuvor aktiv war. Das ist jedoch nicht der Fall. Wohl aber tritt er wieder in Aktion, sobald Sie den Trichter anklicken. Wenn im Datenblatt kein Filter gespeichert wurde, so ist der Trichter-Button auch gar nicht erst aktivierbar. Damit erklärt sich auch, weshalb Access auch dann nach Speicherung fragt, wenn der Filter in der Ansicht "entfernt" wurde. Der Begriff ist schlicht irreführend. Korrekter wäre die Bezeichnung "Filter deaktivieren" gewesen.

Wenn sich der Filter aber nicht wirklich entfernen lässt, sitzt er dann auf alle Ewigkeit im Datenblatt fest? Nein! Öffnen Sie die Tabelle **tblStammdaten** im Entwurf und klicken auf **Ansicht > Eigenschaften**. im Eigenschaftendialog finden Sie eine Zeile **Filter**, die gewöhnlich leer ist, nun aber diesen Inhalt zeigt:

```
(((Lookup_IDOrt.Ort="Berlin"))) AND
((Lookup_IDAnrede.Anrede="Herr"))
```

Sie brauchen nur den Inhalt dieser Zeile zu leeren, um den Filter endgültig aus dem Datenblatt zu löschen.

So praktisch die automatische Speicherung des zuletzt aktivierten Filters ist, so unpraktisch ist leider sein Entfernen. Über die Oberfläche von Access lässt sich das nicht bewerkstelligen.

Formularbasierter Filter

Diese Bezeichnung führt etwas in die Irre, weil dieser Filter nichts mit Access-Formularen zu tun hat.

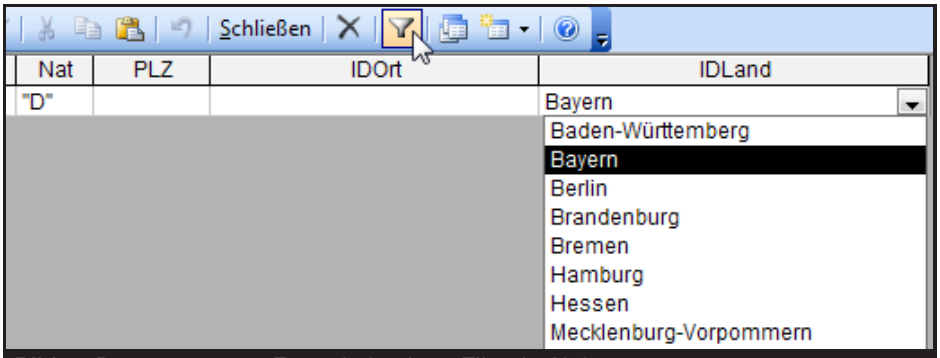


Bild 7: Der sogenannte Formularbasierte FIlter in Aktion

Öffnen Sie das Datenblatt **tblStammdaten** und aktivieren Sie den Filter über den Button mit dem kombinierten Trichter-Dialogfenster-Symbol. Ein Formular tritt nun nicht in Erscheinung. Stattdessen reduziert sich das Datenblatt auf nur eine Zeile ohne Inhalt, die so wirkt, als könne man hier einen neuen Datensatz eingeben – ein Beispiel, dass der Otto-Normal-Benutzer Ihrer Datenbank hier ohne Schulung überfordert ist.

Wie in Bild 7 zu sehen, haben sich in dieser Filteransicht die Felder in Kombinationssteuerelemente verwandelt, aus denen man Inhalte der zugrundeliegenden Datensätze auswählen kann – etwa die Bundesländer. Im Beispiel ist außerdem als Nation Deutschland ausgewählt. In dieser Ansicht blendet sich übrigens die Symbolleiste **Filtern/Sortieren** aus Bild 4 ein, deren Schaltflächen nun einen Sinn ergeben.

Das Kreuzchen entfernt die Auswahl in der Entwurfszeile, der **Schließen**-Button verlässt die Entwurfsansicht und kehrt zum normalen Datenblatt zurück... ohne den eingestellten Filter zu speichern! Das nämlich passiert erst, nachdem Sie den Trichter aktiviert haben. Die Entwurfsansicht wird dann ebenso verlassen, in der Datenblattansicht sind nun aber nur die ihren Filterregeln entsprechenden Zeilen zu sehen, im Beispiel also alle Kunden aus Bayern.

Worin soll hier nun der Unterschied zum Auswahlbasierten Filter bestehen, werden Sie fragen? Auch über jenen ließe sich ja diese Filterung vornehmen. Tatsächlich aber ist dieser Filter hier intelligenter. In den Kombinationsfeldern des Entwurfs zeigt er nur für solche Felder die Datenauswahl an, bei denen sich die Datenmenge in überschaubaren Grenzen hält. In Feldern, wie der Postleitzahl (**PLZ**), bei denen die Unmenge von Einträgen keine ergonomische Auswahl zuließe, finden sich stattdessen die Einträge **Ist Null** und **Ist Nicht Null**. Man kann hier also die Unterscheidung zwischen gefülltem und leerem Feldinhalt filtern, mehr nicht.

Erst bei genauerem Hinsehen wird man im Entwurf des formularbasierten Filters des veränderten Fußbereichs gewahr. Dort haben sich Laschen gebildet (Bild 8). Bei Klick auf ein **Oder** wird ein neue leere Zeile im Datenblatt eingeblendet, in der man eine zusätzliche Filterkombination angeben kann. Ein **Und** ergäbe keinen Sinn, weil dies bereits mit einem Filter möglich ist, da sich die Bedingungen hier ja per **AND** kumulieren.

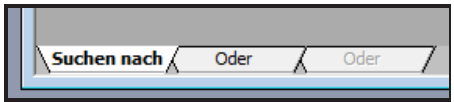


Bild 8: Fußbereich des Datenblatts beim Formularbasierten Filter

Haben Sie auf diese Weise mehrere Filter ODER-verknüpft, so können Sie das Ergebnis im Eigenschaftenblatt des Tabellenentwurfs begutachten, etwa:

```
((Lookup_IDAnrede.Anrede="Herr") AND
(tblStammdaten.Nat="CH"))
OR
((Lookup_IDAnrede.Anrede="Frau") AND
(Lookup_IDLand.Land="Schleswig-Holstein"))
```

Ob der Mechanismus des Formularbasierten Filters und seine Umsetzung wirklich nützlich und ergonomisch sind, ist fraglich. Als Entwickler würde man eher gleich eine SQL-Abfrage auf die Tabelle formulieren und speichern, statt über den etwas kryptischen Filterentwurf zu gehen. Als Datenbankanwender muss man erst über die entsprechenden Skills verfügen, um ihn bedienen zu können. Und die sind in der Regel nicht vorhanden.

Spezialfilter

Um es gleich vorweg zu nehmen: Das eben Resümierte gilt genauso für den Spezialfilter, der über das Menü und sein Eintrag **Spezialfilter/-sortierung...** aufzurufen wäre. Dabei

Zufallszahlen generieren und verwenden

Ein Würfelspiel in Access realisiert? Oder ein Roulette-Tisch in einem Formular? Das wären denkbare Einsatzgebiete für generierte Zufallszahlen. Aber gibt es über solche Späße hinaus sinnvolle Verwendung für Zufälle in Datenbanken? Beleuchten wir in diesem Beitrag einmal, wie Zufallszahlen erzeugt und wofür sie zum Einsatz kommen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1410_Zufallszahlen.mdb**.

Zufallszahlen erzeugen

Rein mathematisch gesehen ist das Berechnen von Zufallszahlen eine Wissenschaft für sich, und selbst die Definition des Ausdrucks **Zufall** ist Thema philosophischer Abhandlungen. Ein Prozessor kennt im Prinzip nun mal nur exakte Berechnungen und kann sich nicht einfach etwas ausdenken.

Tatsächlich finden Sie nicht einmal im Windows-API eine Funktion für Zufallszahlen. High-Level-Sprachen, wie VBA, müssen sich daher etwas einfallen lassen. Gerne werden aktuelles Datum und Uhrzeit als zufällige Werte angenommen und davon Zahlen abgeleitet.

Der Systemzeitgeber spielt also häufig eine Rolle für zufällige Werte. Lange Rede, kurzer Sinn: Access oder die Datenbank-Engine selbst enthalten keine Zufallszahlengeneratoren, wohl aber VBA mit der einzigen Methode **Rnd()**.

Die VBA-Funktion Rnd()

Wollten Sie in der Variablen x einen Zufallswert haben, so geschähe das in der kürzesten Form mit dieser Zeile:

```
Dim x As Double
x = Rnd()
```

Rnd erzeugt einen zufälligen Double-Wert im Bereich von 0 bis 1. Die Funktion basiert, wie schon angedeutet, auf dem Systemzeitgeber von Windows. Wenn Sie es schaffen, auf zwei Rechnern zur gleichen Zeit diese Funktion aufzurufen, dann wird das Ergebnis der "Zufallszahl" dasselbe sein. Versuchen Sie es aber lieber erst gar nicht, denn hier werden sogar Nanosekunden ausgewertet.

Für ein Würfelspiel reicht dieser Wertebereich nicht aus. Um aus ihm den Bereich 1 bis 6 zu erzeugen, reicht eine simple Multiplikation aus:

```
Dim x As Integer
x = 1 + 5 * Rnd()
```

Der Mindestwert von 1 muss addiert werden, sonst kämen auch Würfel ohne Punkt heraus. Aber auch die angegebene Zeile führt zu gezinkten Würfeln, denn die 6 käme seltener zum Vorschein, als ihr gebührt. Dasselbe gilt für die 1.

Der Knackpunkt nämlich ist die Zuweisung eines Double-Werts zu einer Integer-Variablen. VBA muss hier runden: Double-Werte größer als 0.5 führen zu einer 1, die anderen zu 0. Folglich erscheinen mit der Code-Zeile die Werte 1 und 6 genau halb so oft, wie die Werte 2, 3, 4, 5. Wegen dieses Rundungsverhaltens wäre die Zeile dergestalt zu korrigieren:

```
Dim x As Integer
x = 0.5 + 6 * Rnd()
```

Die 1 erscheint nun bei Zufallswerten von etwa 0.51 bis 1.5, die 6 bei Werten von 5.51 bis 6.5.

Diesen Umstand sollten Sie immer im Auge behalten, wenn Sie die Rückgabe von **Rnd** in Integer-Werte konvertieren.

Eigentlich wäre nun bereits alles gesagt, würde **Rnd** nicht noch mit einem Parameter aufwarten, der die Steuerung der Rückgabewerte ermöglicht.

Startwerte von Rnd()

Der optionale Parameter von **Rnd** kann unterschiedliche Bedeutungen einnehmen, je nachdem, wie groß er ist. Ein möglicher Wert ist die Null:

```
Dim x As Double
x = Rnd(0)
```

Wenn Sie diese Zeile ausführen, erhalten Sie als Ergebnis jedes Mal den gleichen Wert. Um das zu erläutern, muss etwas weiter ausgeholt werden, wie **Rnd** arbeitet. Beim aller-

ersten Aufruf der Funktion nämlich initialisiert VBA den Zufallsgenerator mit einem Startwert, der sich vom Zeitgeber ableitet. Bei allen weiteren Aufrufen aber spielt dieser keine Rolle mehr. Er wird nicht mehr abgefragt, sondern die weiteren Zufallszahlen einer Serie ausgegeben, die VBA intern anlegt beziehungsweise berechnet.

```
Sub FillTableRandom()
    Dim rs As DAO.Recordset
    Dim i As Long
    Dim n As Long

    CurrentDb.Execute "DELETE FROM tblZufall"
    Set rs = CurrentDb.OpenRecordset("tblZufall", dbOpenDynaset)
    n = 20000
    For i = 0 To n
        If (i Mod 200) = 0 Then Randomize
        rs.AddNew
'        rs!Zahl.Value = 100 * Rnd   'Falsch!
        rs!Zahl.Value = 101 * Rnd - 0.5
        rs.Update
    Next i
    rs.Close
End Sub
```

Listing 1: Code zum Befüllen der Tabelle tblZufall mit 20000 Zufallszahlen

Sie können sich das wie eine Tabelle vorstellen, in der zufällige Zahlen gespeichert sind. Beim ersten Aufruf wird der Datensatzzeiger auf einen zufälligen Datensatz gesetzt und anschließend alle weiteren Datensätze ab diesem ausgegeben. **Rnd(0)** entspricht nun genau dem Datensatz mit dem ersten Zeiger. Ein weiterer Aufruf von **Rnd** ohne Parameter ändert diesen Zeiger aber wieder auf Basis des Systemzeitgebers. **Rnd(0)** gibt also immer den zuletzt generierten Wert zurück.

Ein anderer Parameterwert wäre eine Zahl (**Single**) größer als **0**:

```
Dim x As Double
x = Rnd(0.3)
x = Rnd(99.5)
```

Obwohl Sie hier beliebige Single-Zahlen angeben können, spielt ihr Wert für das Ergebnis keine wesentliche Rolle. Die Zahl setzt den Zeiger auf die Zufallsserie lediglich um mehr, als eine Position, nach vorn. Eine Zufallszahl kann aber nicht zufälliger sein, als eine andere! Die Verwendung dieses Parameters kommt damit dem Aufruf der Funktion **ohne** Parameter gleich. Sie können Parameterwerte größer, als 0, demnach ignorieren.

Anders sieht es aus, wenn ein Parameterwert kleiner 0 gesetzt wird:

```
Dim x As Double
x = Rnd(-14)
```

x wird damit immer mit derselben Zufallszahl gefüllt. Setzen Sie statt der -14 eine -13 ein, so ändert sich die zurückgelieferte Zahl.

Der Parameter kommt damit der absoluten Positionierung des Zeigers auf die Zufallsserie gleich, während die positiven Parameterwerte relative Positionierung bedeuteten.

Sie können Zufallszahlen so immer wiederholen. Wofür ist das gut? Tja, außer einem manipulierten Casinospiel fällt uns dazu auch nicht viel ein. Sie werden dieses Verhalten sehr selten benötigen.

Randomize

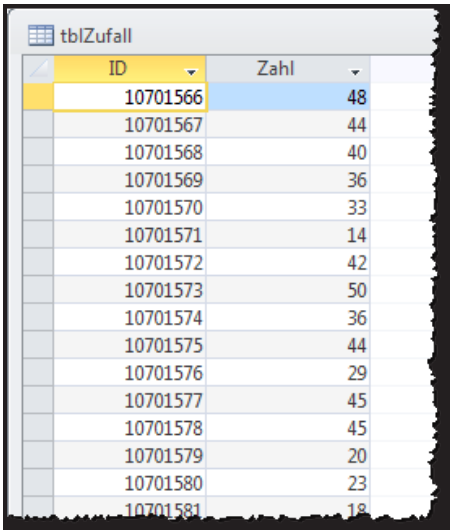
Eng verbündet mit der **Rnd**-Funktion ist die VBA-Anweisung **Randomize**. VBA erzeugt ja, ausgehend von einem auf dem Zeitgeber basierenden Ausgangspunkt, die Zufallsserie immer nach dem gleichen Schema, einem Algorithmus, der keinen wirklichen Zufall mehr enthält. Auf Rechner A sähe die Serie exakt gleich aus, wie auf Rechner B, so der erste Aufruf von **Rnd** zum selben Zeitpunkt erfolgte. Sie können die Zufälligkeit gewissermaßen erhöhen, indem Sie Randomize aufrufen:

```
Dim x As Double
x = Rnd()
Randomize
x = Rnd()
```

Randomize zwingt den Zufallsgenerator erneut den Systemzeitgeber abzufragen, einen neuen Startwert zu setzen, und damit die Serie neu zu belegen. Auch hier stellt sich wieder die philosophische Frage nach der Zufälligkeit eines Zufalls. Streuen Sie einfach Randomize gelegentlich ein, wenn sie lange Serien von Zufallszahlen erzeugen, um Ihr Gewissen zu beruhigen.

Verteilung des Zufalls

Machen wir uns an die statistische Auswertung der von **Rnd()** generierten Zufallszahlen. Man könnte erwarten, dass bei einer großen Menge von Zufallszahlen alle genau gleich häufig auftreten. Um das


tblZufall

ID	Zahl
10701566	48
10701567	44
10701568	40
10701569	36
10701570	33
10701571	14
10701572	42
10701573	50
10701574	36
10701575	44
10701576	29
10701577	45
10701578	45
10701579	20
10701580	23
10701581	18

Bild 1: Inhalt der Tabelle tblZufall

zu überprüfen, rufen Sie die Prozedur **FillTableRandom** in der Beispieldatenbank, Modul **mdlZufall**, auf. Sie füllt die Tabelle **tblZufall** mit 20.000 zufälligen Werten im Bereich von 0 bis 100 (siehe Listing 1).

Das Feld Zahl der Tabelle ist als Long definiert, was zu den besprochenen Rundungsproblemen führt, weshalb die korrekte Zeile für die Wertzuweisung so lautet:

```
rs!Zahl.Value = 101 * Rnd - 0.5
```

Jeden 200ten Durchlauf der Schleife wird zusätzlich die **Randomize**-Anweisung aufgerufen. Im Ergebnis sieht die Tabelle dann etwa aus, wie in Bild 1. Um die statistische Verteilung der Zufallszahlen zu ermitteln wird die Abfrage aus Bild 2 auf die Tabelle angesetzt. Zu jeder Zahl von 0 bis 100 wird deren Anzahl berechnet. Diese Abfrage wiederum ist Grundlage für ein **PivotChart**-Formular, das die Auswertung grafisch aufbereitet. Dazu erzeugen Sie ein neues Formular **frmStatistik** in der Datenbank, spendieren ihm als Datenherkunft die Abfrage **qry_ZufallStatistik**, und stellen die Ansicht auf **PivotChart**.

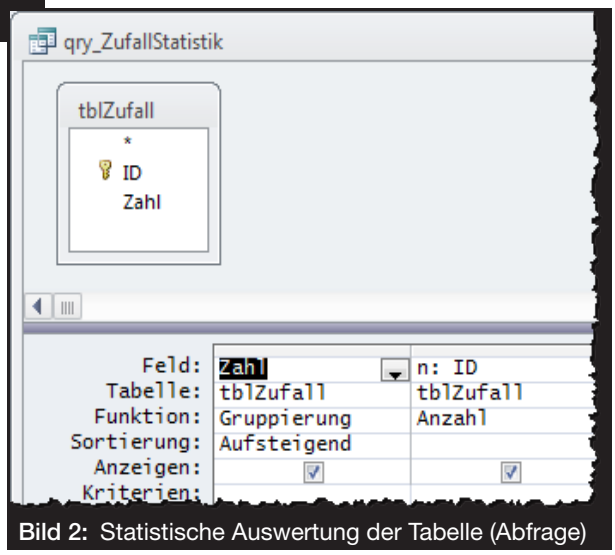


Bild 2: Statistische Auswertung der Tabelle (Abfrage)

Der Entwurf des Formulars ist, ähnlich, wie bei einem Datenblatt, sehr einfach gestaltet (Bild 3). Die Formatierung des Charts erfolgt erst, wenn Sie es in der Formularansicht öffnen. Sie können dann die Wertebereiche der Achsen bestimmen, den Typ des Diagramms ändern (hier: Balkendiagramm), oder die Farben der Elemente einstellen, indem Sie auf die jeweiligen Elemente klicken und im daraufhin sich öffnenden Dialogfenster Angaben in den entsprechenden Steuerelementen machen.

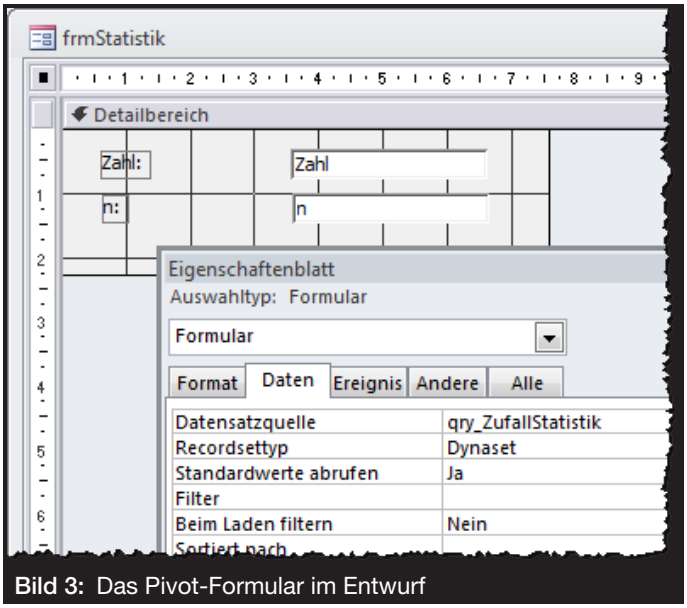


Bild 3: Das Pivot-Formular im Entwurf

Heraus kommt eine Ansicht wie in Bild 4. Trotz der relativ hohen Anzahl an Stichproben – 200 je Wert – variiert die Verteilung doch deutlich. Nicht alle Werte sind gleichmäßig vertreten. Ohne ins Detail zu gehen, lässt sich sagen, dass dies dennoch normal ist. In der Stochastik gibt es sogar Berechnungsmethoden für die Größe der durchschnittlichen Abweichung bei einer bestimmten Ausgangsmenge von Zufallszahlen.

Sie können das leicht selbst überprüfen: Setzen Sie das Schleifenlimit in der Prozedur einfach herauf. Sie werden feststellen, dass sich der obere Rand der Balken immer mehr einer Line annähert. Bei einer Million Datensätzen sieht das dann schon recht gleichmäßig aus.

Interessant ist nun, ob die Anweisung **Randomize** hierauf einen Einfluss hat. Spielen Sie mit dem Wert des **Mod**-Operators, der

Tabellen bearbeiten mit dem TableDef-Objekt

Wenn Sie über VBA auf die Objekte Ihrer Datenbank zugreifen, dann hat sicher der Datenzugriff über das Recordset-Objekt die größte Bedeutung. Seltener werden Sie die Definitionen der Tabellen auslesen, bearbeiten oder gar anlegen müssen. Wenn doch, so ist das TableDef-Objekt von DAO die zentrale Anlaufstelle. Sehen Sie selbst, wie der Umgang damit aussieht und was möglich ist.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1409_Tabledefs.mdb**.

Die richtige Bibliothek

Die VBA-Zugriffsmethoden für die Datenbank sind nicht Teil von Access selbst, sondern in eine zusätzliche Daten-Engine ausgelagert. Das führt dazu, dass Sie in Ihrem VBA-Projekt einen zusätzlichen Verweis auf diese Engine benötigen. Wenn Sie eine neue Datenbank anlegen, so setzt Access diesen Verweis automatisch, doch leider war man sich bei Microsoft nicht einig, welche Bibliothek hierfür die geeignete ist.

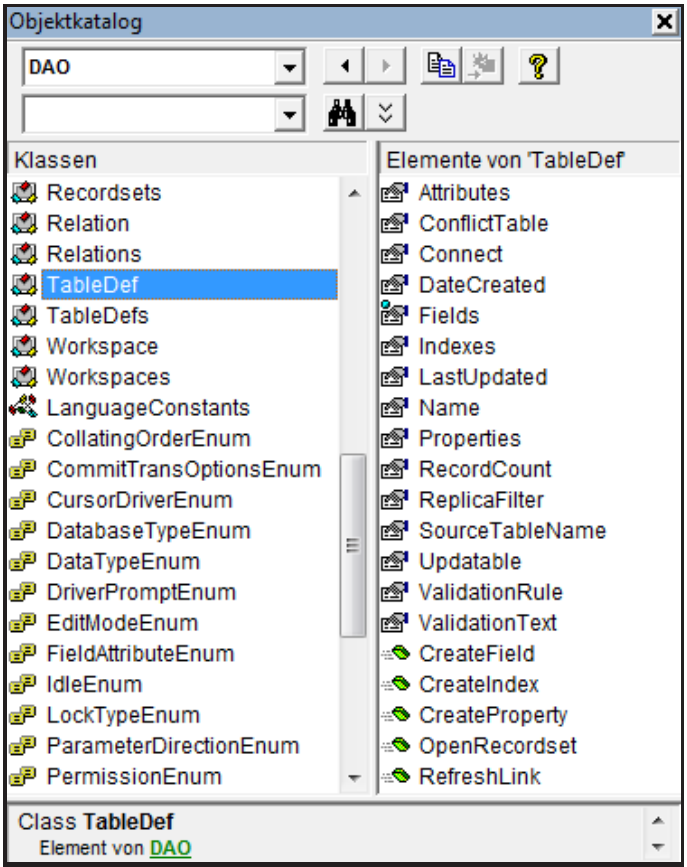


Bild 1: Die Klasse DAO.TableDef im VBA-Objektkatalog

So war man einige Zeit der Meinung, dass **ADODB** die bessere Zugriffsbibliothek darstellt, und folglich wurde beim Anlegen einer Datenbank darauf ein Verweis gesetzt. Erst ab Access 2007 schwenkte man wieder um und manifestierte **DAO** als Standard.

Sie können daher nicht sicher sein, dass in Ihrer oder einer fremden Datenbank bereits ein Verweis auf die Bibliothek DAO gesetzt ist, welche die Klasse **TableDef** enthält, von der hier die Rede sein soll.

Im Zweifel schauen Sie im VBA-Objektkatalog nach, ob dort der Eintrag **DAO** vorhanden ist, oder rufen alternativ den Verweise-Dialog über das Menü **Extras|Verweise...** auf. DAO hieß zwar früher schon DAO, der Beschreibungstext des Verweises änderte sich allerdings. So ist bis Access 2003 ein Verweis auf die **Microsoft DAO 3.6 Objekt Library** zu setzen und für die Folgeversionen einer auf die **Microsoft Office 12.0 Access database engine Object Library**.

Machen Sie für Access 2010 aus der 12 eine 14 und für Access 2013 eine 15. In allen Fällen aber finden Sie anschließend im Objektkatalog den Eintrag **DAO** in der linken oberen Combobox. Wird er ausgewählt, so erscheinen die Datenzugriffsklassen der Bibliothek auf der linken Seite, wie in Bild 1, wo auch schon die **TableDef-Klasse** markiert wurde und rechts ihre Methoden wiedergibt.

TableDefs-Auflistung

Alle Tabellen Ihrer Datenbank sind in einer Art Katalog über die **TableDefs**-Klasse zu erhalten. Wie in allen Auflistungsklassen können deren **TableDef**-Elemente in einer **For..Each**-Schleife durchlaufen werden, die Zahl der Elemente über die Eigenschaft **Count** ermittelt oder

Elemente gezielt mit der (versteckten) **Item**-Eigenschaft abgefragt werden. Woher jedoch weiß DAO, wenn es gar nicht Teil des Access-Objektmodells ist, welche Tabellen ihn der Access-Datenbank stecken? Dazu muss über ein **Database**-Objekt, welches selbst die TableDefs-Methode ausweist, eine Verbindung zu ihr geschaffen werden.

Dieses Bindeglied gibt es in Access entweder über die Eigenschaft **Application.CurrentDb** oder über die Eigenschaft **Application.DBEngine**. Sobald eine Datenbank in Access geöffnet wird, tritt die Datenbank-Engine in Aktion und lädt in ihr **Workspace** ein neues **Database**-Objekt. Eine Kopie davon lässt sich dann über **CurrentDb** erhalten. Im Prinzip sollte man über beide Ansätze zum gleichen Ergebnis kommen:

```
1.
Dim dbs As DAO.Database
Set dbs = Application.CurrentDb
2.
Dim dbs As DAO.Database
Dim wrk As DAO.Workspace
Set wrk = Application. _
  DBEngine.Workspaces(0)
Set dbs = wrk.Databases(0)
Abgekürzt:
Set dbs = DBEngine(0)(0)
```

Dass es leider nicht egal ist, für welche der beiden Methoden man sich entscheidet, werden wir noch sehen.

Versuchen wir zunächst alle Tabellen der aktuellen Datenbank namentlich im VBA-Direktfenster auszugeben. Rufen Sie dazu die erste der drei Routinen in Listing 1 auf. **ListTableDefs** definiert erst eine TableDef-Variable **tdf** und füllt diese in der folgenden **For..Each**-Schleife jeweils mit dem nächsten Element der TableDefs-Auflistung von **CurrentDb**. Die Eigenschaft **Name** des TableDef-Objekt wird ins Direktfenster geschrieben.

```
Sub ListTableDefs()
    Dim tdf As TableDef
    For Each tdf In CurrentDb.TableDefs
        Debug.Print tdf.Name
    Next tdf
End Sub

Sub ListSysTableDefs()
    Dim tdf As TableDef
    For Each tdf In CurrentDb.TableDefs
        If (tdf.Attributes And dbSystemObject) <> 0 Then Debug.Print tdf.Name
    Next tdf
End Sub

Sub ListNonSysTableDefs()
    Dim tdf As TableDef
    For Each tdf In CurrentDb.TableDefs
        If (tdf.Attributes And dbSystemObject) = 0 Then Debug.Print tdf.Name
    Next tdf
End Sub
```

Listing 1: Ausgeben aller Tabellen der Datenbank über die TableDefs-Auflistung

Auf diese Weise entsteht eine Liste wirklich aller Tabellen der Datenbank, zu der auch die Systemtabellen gehören, welche allesamt das Präfix **MSys** tragen. Wollten Sie sie ausfiltern, so könnten Sie dies über eine Textfunktion tun, die die ersten vier Zeichen des Namen auswertet.

Da ein **TableDef**-Objekt aber eine Methode besitzt, welche die Systemeigenschaft direkt anzeigen kann, ist der Gebrauch von **Attributes** verlässlicher. **Attributes** ist ein Long-Wert, der verschiedene Eigenschaften der Tabelle wiedergibt und als Zahlen binär addiert. Sie finden diese in den Enumerationskonstanten der Klasse **TableDefAttributeEnum** von DAO im Objektkatalog. Dort gibt es etwa für Systemtabellen die Konstante **dbSystemObject** oder für verknüpfte Tabellen das Attribut **dbAttachedTable**.

Um zu ermitteln, ob **dbSystemObjekt** im **Attributes**-Wert der Tabelle steckt, muss diese Eigenschaftszahl per booleschem **AND** verglichen werden. Genau das geschieht auch in den beiden weiteren Routinen aus Listing 1. **ListSysTableDefs** gibt alle Systemtabellen aus und **ListNonSysTableDefs** die von Ihnen angelegten. Wobei hier zu erwähnen wäre, dass über die Attributes-Eigenschaft eine jede Tabelle auch auf den Status Systemobjekt eingestellt werden kann! Wollten Sie etwa der Tabelle **tblStammdaten** der Beispieldatenbank den Systemstatus verleihen, so genügt dafür eine Zeile:

```
CurrentDB.TableDefs("tblStammdaten").
Attributes = dbSystemObject
```

Ebenso könnten Sie die Tabelle verstecken, indem sie **dbHiddenObject** zuwiesen. In beiden Fällen würde die Tabelle im Datenbankfenster oder Navigationsbereich nicht mehr auftauchen, so Sie nicht in deren Optionen eingestellt hatten, dass auch versteckte und System-Objekte angezeigt werden sollen.

Tabelleneigenschaften auslesen

Attributes als Eigenschaftswert einer Tabelle wurde eben schon angeführt. Die weiteren Standardeigenschaften einer Tabelle entsprechend jenen, die rechts in Bild 1 zu sehen sind. Listing 2 zeigt beispielhaft, wie das geht.

Der Prozedur **ReadTableDef** können Sie optional den Namen einer Tabelle als Parameter übergeben. Falls Sie dies unterlassen, wird einfach die Stammdatentabelle verwendet. Im **With-Block** der Routine werden alle verfügbaren Eigenschaften des TableDef-Objekts **tdf** ausgegeben.

Interessant sind nun die im oberen Bereich auskommentierten Zeilen. Die erste versucht, die TableDef-Variable auf

```
Currentdb.TableDefs(sTable)
```

zu setzen. Das wird anstandslos durchgeführt, doch bei ersten Zugriff auf eine Eigenschaft der TableDef-Variablen (**tdf.Name**) erfolgt eine Fehlermeldung, dass das Objekt nicht mehr gültig sei. Grund dafür ist, dass **CurrentDb** nur eine temporäre Instanz des Database-Objekts widerspiegelt, das keinen Bestand hat. Sobald Sie das temporäre Objekt

```
Sub ReadTabledef(Optional sTable As String = "tblStammdaten")
    Dim dbs As DAO.Database
    Dim tdf As DAO.TableDef

'    Set tdf = CurrentDb.TableDefs(sTable)
'    Set dbs = CurrentDb
'    Set tdf = dbs.TableDefs(sTable)

    With DBEngine(0)(0)
        .TableDefs.Refresh
        Set tdf = .TableDefs(sTable)
    End With

    With tdf
        Debug.Print "Name", tdf.Name
        Debug.Print "Connect", .Connect
        Debug.Print "DateCreated", .DateCreated
        Debug.Print "LastUpdated", .LastUpdated
        Debug.Print "SourceTableName", .SourceTableName
        Debug.Print "Updatable", .Updatable
        Debug.Print "ValidationRule", .ValidationRule
        Debug.Print "ValidationText", .ValidationText
        Debug.Print "Attribute Text:", StrTableAttrib(.Attributes)
        Debug.Print "ReplicaFilter", .ReplicaFilter
        Debug.Print "ConflictTable", .ConflictTable
        Debug.Print "RecordCount", .RecordCount
        Debug.Print "Fields.Count", .Fields.Count
        Debug.Print "Indexes.Count", .Indexes.Count
        Debug.Print "Properties.Count", .Properties.Count
    End With
End Sub
```

Listing 2: Eigenschaften eines TableDef-Objekts ausgeben

aber einer **Database**-Variablen **dbs** zuweisen, wie in den nächsten auskommentierten Zeilen, funktioniert die Routine.

Warum trat dann der Fehler nicht bereits in den Prozeduren aus Listing 1 auf? Ganz einfach: Der Schleifen-Enumerator fragt das **CurrentDb**-Objekt bei jedem Durchlauf neu ab!

Auf die Bedeutung der einzelnen Eigenschaften des TableDef-Objekts soll hier nicht weiter eingegangen werden. Ziehen Sie dazu die Hilfe zu Rate, indem Sie im Objektkatalog eine Methode markieren und F1 drücken. Zum gleichen Ziel führt auch, wenn Sie einen Eigenschaftsnamen im Quellcode markieren und F1 drücken.

Falls Sie über den Ausdruck **StrTableAttrib** im Code stolpern: Das ist der Name einer Hilfsfunktion im Modul **mdlTDFRead**, die aus dem numerischen **Attributes**-Wert einen aussagekräftigeren Begriff macht. Denn was sollte Ihnen der Wert -2147483646 sagen? "SystemObjekt" hilft da schon eher weiter.

Die Methoden des **TableDef**-Objekts geben nur die Standardeigenschaften einer Tabelle zurück. Zwar ist nicht so recht nachzuvollziehen, warum dazu etwa auch **RecordCount** gehört, ein berechneter Wert, der die

Etiketten drucken per Assistent

Weihnachten steht vor der und Tür, und Sie möchten an Verwandtschaft und Freunde oder Geschäftspartner Grußbriefe versenden. Die Adressaten hatten Sie bereits in einer Access-Tabelle verewigt. Einfacher, als Couverts direkt mit Adressen zu beschriften, lassen sie diese auf Etiketten aufbringen. Zum Glück stellt Access einen Etikettenassistenten bereit, der Ihnen das manuelle Erstellen eines Etikettenberichts abnimmt. Doch leider legt der Ihnen einige Stolpersteine in den Weg...

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1410_Etiketten.mdb**.

Im Online-Handel haben Sie ein passendes Produkt erworben, das für Adressen geeignet erscheint, wie auch die Information **Geeignet für Internetmarke** auf der Verpackung suggeriert (Bild 1).

Breite und Höhe der Etiketten sind groß genug, um bei lesbaren Schriftgrößen auch längliche Adressdaten aufzunehmen. Machen Sie sich an die Aufgabe, die Daten aus der Tabelle auf die Etiketten zu transferieren.

Der Etiketten Assistent ist ein Tool, das dialoggeführt einen Access-Bericht erstellt, welcher die Daten einer Tabelle oder Abfrage so gruppiert und in ein Spaltenlayout bringt, dass beliebige Etiketten damit bedruckt werden können. Sie finden ihn unter Access 2003 über **Beri**

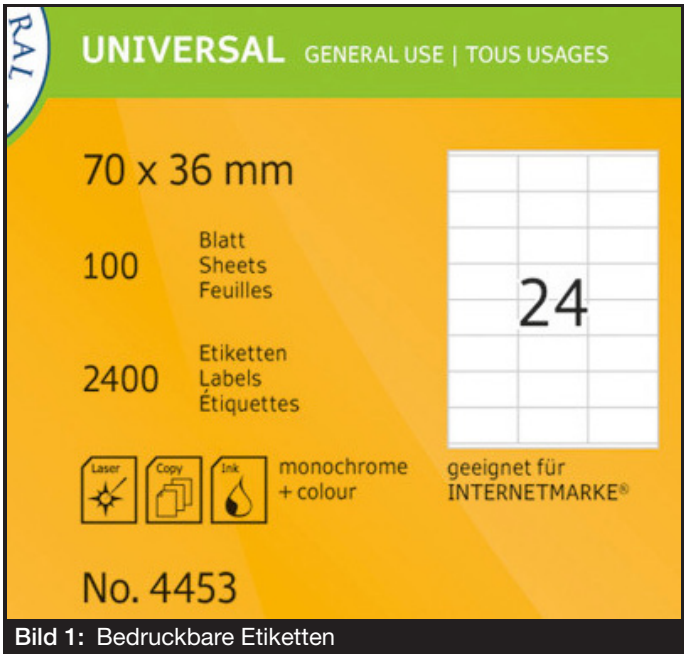


Bild 1: Bedruckbare Etiketten

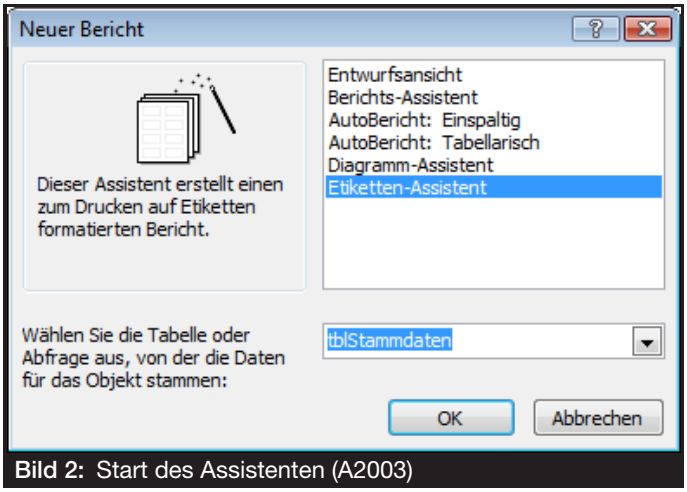


Bild 2: Start des Assistenten (A2003)

cht|Neu...|Etikettenassistent und ab Access 2007 im Ribbon über **Erstellen|Etikettenassistent**. Im ersten Fall wählen Sie aus dem Kombinationsfeld den Namen einer Tabelle oder Abfrage aus (siehe Bild 2), deren Daten im Bericht erscheinen sollen, im zweiten muss vor dem Klick auf die Ribbon-Schaltfläche bereits eine Tabelle oder Abfrage im Navigationsbereich markiert worden sein.

Der weitere Verlauf der Dialogführung entspricht sich dann in allen Access-Versionen.

Zunächst werden Sie aufgefordert aus der Liste verfügbarer Etikettvorlagen dasjenige Produkt auszuwählen, welches Sie erworben hatten, wobei als Erstes der Hersteller angegeben wird.

Danach filtert sich die Liste der Formate alphabetisch mit den Namen der Formate des Herstellers. Mit der Vorahnung, dass Microsoft die Liste wohl kaum immer der aktuellen Marktlage anpasst, machen Sie sich auf die Suche nach dem Produkt **Herma 4453**

(Bild 3). Fehlanzeige! Was nun? Entweder treten Sie ihr erworbenes Produkt in die Tonne, oder Sie suchen in der Liste auch unter anderen Herstellern nach Etiketten, die die gleichen Abmessungen aufweisen.

Dies indessen ist ein mühseliges und von wenig Erfolg gekröntes Unterfangen. Denn neben Breite und Höhe der Etiketten spielen auch der Abstand zwischen den Etiketten, sowie der Seitenrand der Vorlage, eine Rolle. Gottlob hat Microsoft diesen Umstand ins Kalkül gezogen und stellt die Möglichkeit bereit, ein neues Format zu fabrizieren. Versuchen wir das im Folgenden.

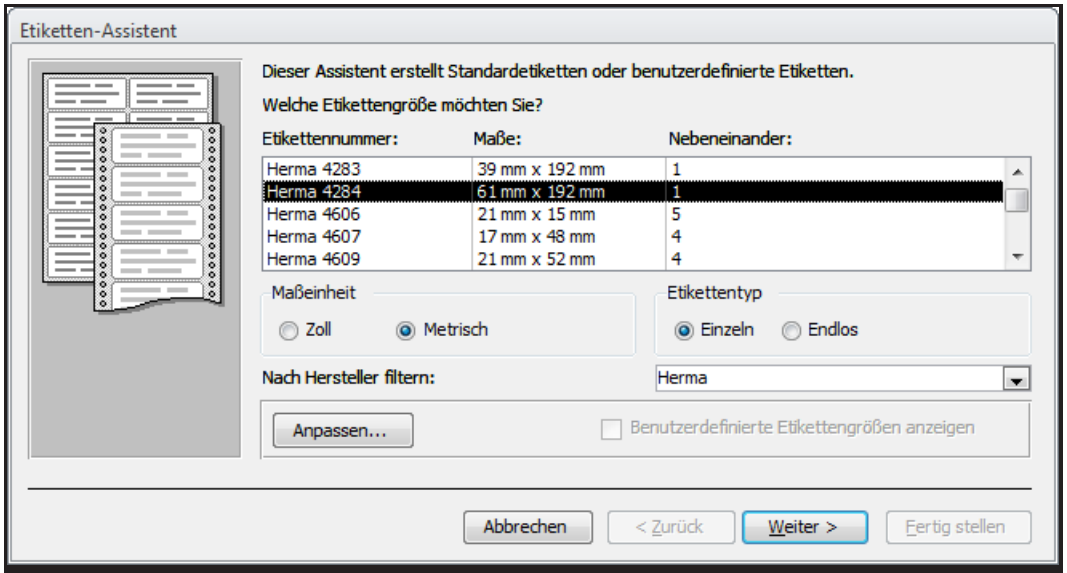


Bild 3: Auswahl eines Etikettenprodukts im Assistenten

Etiketten anpassen

Es ist ist im Assistenten scheinbar keine Möglichkeit vorgesehen, ein Etikett von Grund auf neu zu erstellen. Stattdessen muss erst ein bestehendes Format ausgewählt und anschließend verändert werden.

Das neue Format kann man dann unter eigenem Namen abspeichern. Bevor sie nun alle verfügbaren Formate nach ihrem Produkt möglichst ähnlichen

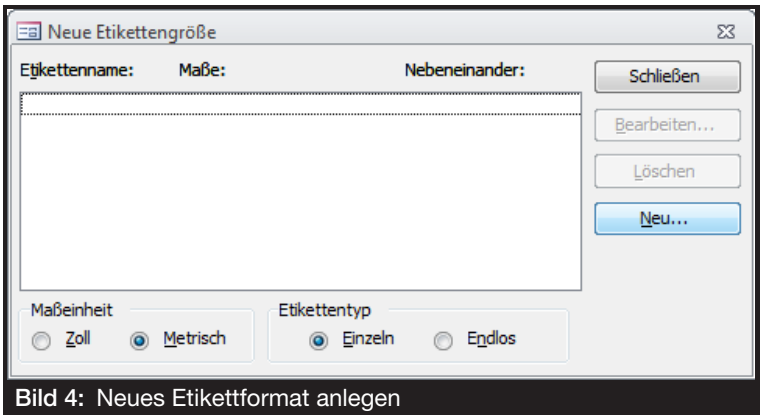


Bild 4: Neues Etikettformat anlegen

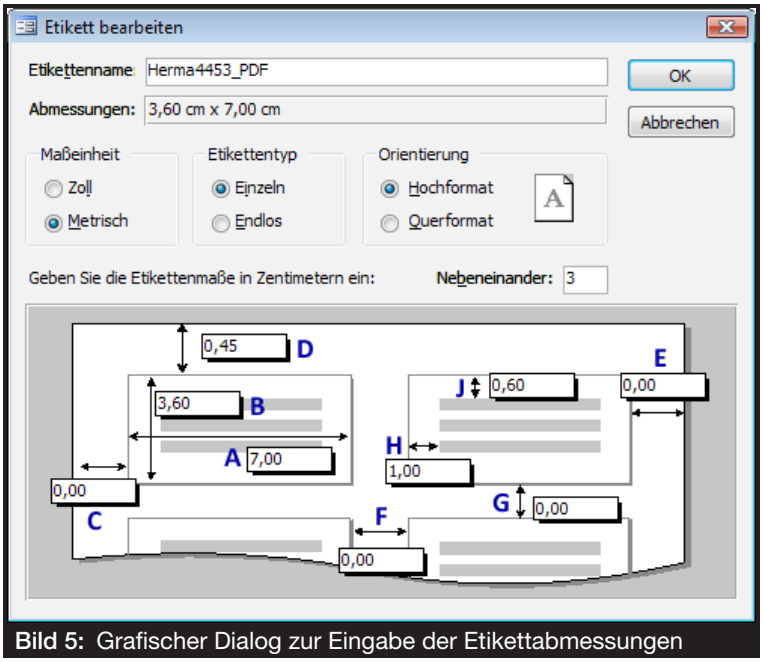


Bild 5: Grafischer Dialog zur Eingabe der Etikettabmessungen

Abmessungen scannen, hier gleich der Hinweis, dass es völlig irrelevant ist, was Sie nehmen – die Formatabmessungen müssen eh alle komplett neu eingestellt werden. Der Klick auf den Button **Anpassen...** führt zum Dialog in Bild 4. Nun präsentiert sich doch noch ein Button mit der Aufschrift **Neu...**, der zum Fenster in Bild 5 führt, in das in Textfelder – hier mit A bis H beschriftet – die neuen Abmessungen eingetragen werden können.

Fehlen nur noch die korrekten Zahlen für Ihr Format. In der Regel müssen Sie nicht Ihr Geodreieck hervorkramen, um von einem Etikettenblatt des Produkts alle Abmessungen

abzulesen, denn auf den Seiten der Hersteller lassen sich Vorlagen mit den gesuchten Angaben herunterladen. Teilweise liegen den Paketen auch Einleger bei, auf denen die Maße aufgedruckt sind.

Für unsere Herma-Etiketten gibt es ein downloadbares PDF, welches ausweist, dass die Breite der Etiketten 7 cm, ihre Höhe 3,6 cm und die Zellabstände jeweils 0 betragen. Oben und unten existiert ein Rand von 4,5 mm und links und rechts fehlen Seitenränder völlig. In das Dialogfenster tragen Sie nun alle Angaben in cm ein, vergeben dem Etikett einen Namen und beenden es dann über OK.

Die Welt wäre schön, wenn es damit getan wäre! Denn Access konfrontiert Sie nun mit der Nachfrage, ob es die Seitenränder korrigieren soll, weil das Format so nicht gedruckt werden könne. Bejahen Sie diese Nachfrage, so setzt Access die Seitenränder links und rechts, sowie oben (C, D, E), auf Werte, die es von Ihrem Standarddrucker erhalten hat – kein einfacher gängiger Drucker kann ohne Seitenränder drucken.

So klug Access hier zu sein scheint, so sinnlos ist dieses Unternehmen zugleich. Denn mit den so modifizierten Rändern werden die Adressen nun mal nicht korrekt in die Etikettzellen wandern. Das Layout wäre komplett korrumpiert. Erste Regel wäre hier also, die Nachfrage von Access mit **Nein** zu beantworten.

Es speichert Ihre Angaben dann unverändert und kehrt zum Dialog

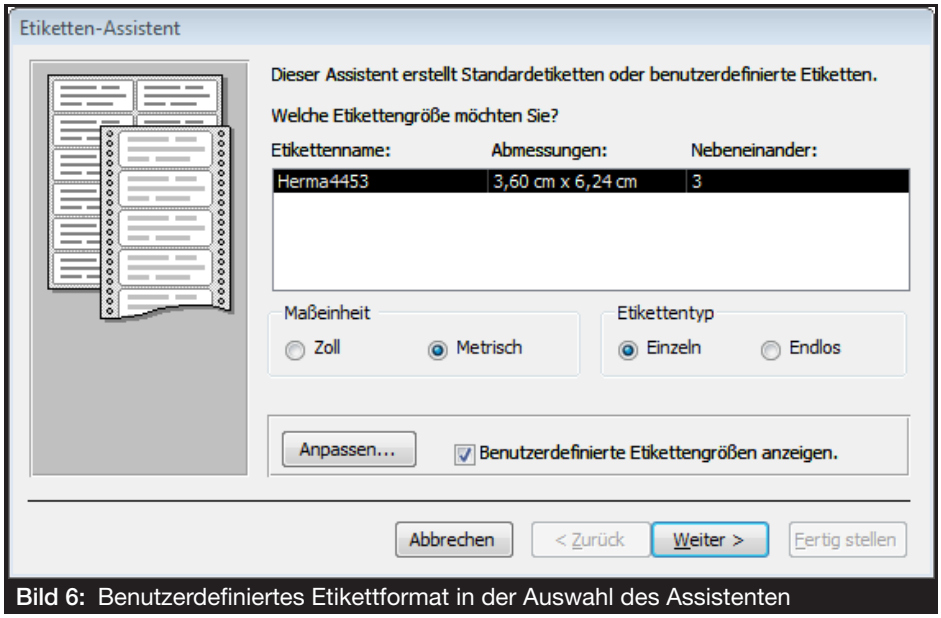


Bild 6: Benutzerdefiniertes Etikettformat in der Auswahl des Assistenten

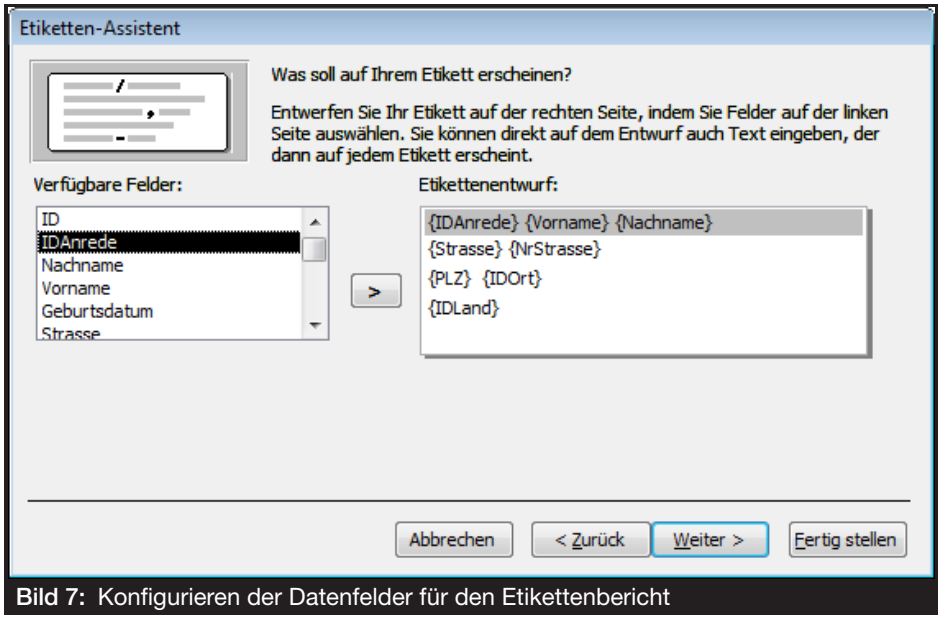


Bild 7: Konfigurieren der Datenfelder für den Etikettenbericht

aus Bild 4 zurück. in dem nun der Name der neuen Vorlage erscheint. Nach Schließen des Fensters ist das Format in die Liste der Etiketten aufgenommen, wird dort allerdings zunächst nicht angezeigt, da es keinem Hersteller zugeordnet ist. Sie müssen erst ein Häkchen auf **Benutzerdefinierte Etikettgrößen anzeigen** setzen, um es für die nächsten Schritte auswählen zu können (Bild 6).

Datenquelle einstellen

Über die Schaltfläche **Weiter >** nähern wir uns der Auswahl der Daten für die Etiketten. Zuvor möchte Access jedoch noch wissen, welche Schrift zur Anwendung kommen soll. Sie können sich hier nach Belieben austoben, jedoch im Hinterkopf behalten, dass für ein Adressetikett eine Schriftgröße von mehr als 12 Punkt ein Risiko darstellt. Straßen- oder

Ortsnamen samt Nummern oder Postleitzahlen dürfen nicht breiter, als maximal 6 cm werden, wenn, wie in Bild 5, als linker Rand 1 cm eingestellt ist.

Wenn Sie nun **Weiter** schalten, befinden Sie sich bereits in der Dialogmaske zum Anlegen der Datenfelder (Bild 7). In der Beispieldatenbank wurde als Datenquelle des Assistenten die Tabelle **tblStammdaten** verwendet, die, so sie sie in Augenschein nehmen, offenbar alle benötigten Felder enthält.

Der Dialog funktioniert so: Markieren Sie links ein Feld und klicken auf den Pfeil-Button rechts. Das Feld wird in das Layout-Fenster rechts übernommen. Wiederholen Sie das für alle benötigten Felder.

Der Clou ist, dass das Layout-Feld ähnlich wie eine Textverarbeitung arbeitet. Zwischen die Datenfelder in geschweiften Klammern können Leerzeichen oder andere Zeichen

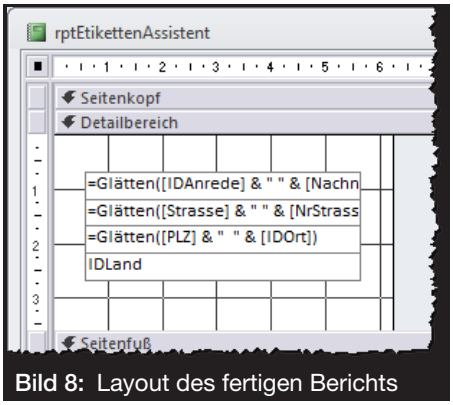


Bild 8: Layout des fertigen Berichts

untergebracht und Zeilenumbrüche gemacht werden. Selbst zusätzliche Zeilen mit beliebigem Text lassen sich einfügen.

Das Layout in Bild 7 entspricht wohl dem Standard für Adressen. Anrede, Vorname und Nachname sind durch Leerzeichen getrennt. Ebenso Straße und Nummer, sowie Postleitzahl und Ort.

Was Access letztendlich aus diesen Angaben macht, sehen Sie schon Mal in Bild 8. Es generiert für jede Zeile jeweils ein Textfeld im Bericht, das als Inhalt eine String-verknüpfte Formel mit den beteiligten Datenfeldern enthält. Es kümmert sich über die Funktion **Glätten** (in VBA: **Trim**) sogar darum, dass überflüssige Leerzeichen entfernt werden.

Doch bevor Sie zum fertigen Berichtsentwurf kommen, möchte der Assistent noch wissen, in welcher Weise die Daten sortiert und gruppiert werden sollen. Sie können ein oder mehrere Datenfelder bestimmen, nach denen die Etiketten später sortiert ausgegeben werden (Bild 9).

Eigentlich würde es ja reichen, wenn diese Angaben im Eigenschaftenfeld **Sortiert nach** des Berichts erscheinen würden, da es für eine Gruppierung keinen Grund gibt – die Etiketten erscheinen ja seriell.

Da eine Gruppierung aber automatisch auch eine Sortierung nach sich zieht, ist Microsoft diesen Weg gegangen, der später die Möglichkeit offen lässt, etwa für Postleitzahlenbereiche neue Etikettenseiten zu erzeugen (Gruppierungsfunktion **Neue Seite vor Bereich**).

Der Dialog zur Sortierung ist der letzte des Etikettenassistenten. Sie werden zum Abschluss nur noch gefragt, ob Sie den generierten Bericht in der Entwurfsansicht oder der Seitenansicht öffnen wollen.

Der Name des Berichts lautet automatisch immer **Etiketten [Name der Datenherkunft]**, also im Beispiel **Etiketten tblStammdaten**. Der Bericht wird auch dann unter diesem Namen gespeichert, wenn Sie ihn zunächst im Entwurf öffnen.

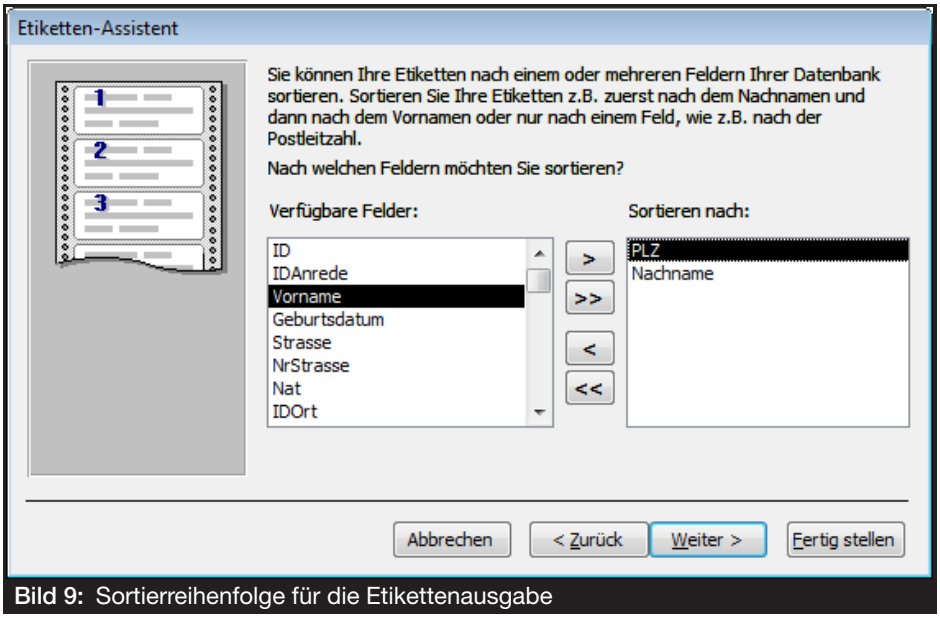


Bild 9: Sortierreihenfolge für die Etikettenausgabe

Ex- und Import zu Microsoft Word

Der Austausch von Daten zu einer Excel-Mappe ist sicher eine häufige Aufgabe unter Access. Seltener hingegen findet eine Anbindung an Word statt. Dabei ist gerade das eine Anforderung, die durchaus oft gewünscht wird, sei es, um Berichte manuell weiter zu verarbeiten oder Seriendokumente mit einer Access-Datenquelle zu versehen. Wir zeigen, wie Ex- und Import zu Word in beide Richtungen aussehen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1410_WordIm-Export.mdb**.

Schnittstellen zu Word

Access zeigt seine Bereitschaft zur Kommunikation mit Word bereits in der Oberfläche an. Unter Access 2003 über die Menü- oder Symbolleisten (siehe Bild 1), in den Folgeversionen direkt im Ribbon unter dem Reiter **Externe Daten** (Bild 2), erkennbar jeweils an den Word-Symbolschaltflächen. Diese Schaltflächen sind manchmal aktiv, manchmal deaktiviert, je nachdem, welches Objekt im Datenbankfenster oder Navigationsbereich markiert ist, oder welches Objekt in der Datenbank gerade angezeigt wird. Es lässt sich feststellen, dass der Export-Button mit der Aufschrift **Veröffentlichen mit Microsoft Word** bei folgenden Objektarten aktiv wird: Tabellen, Abfragen, Formulare und Berichte. Dabei spielt es keine Rolle, ob das Objekt geöffnet, oder nur markiert ist.

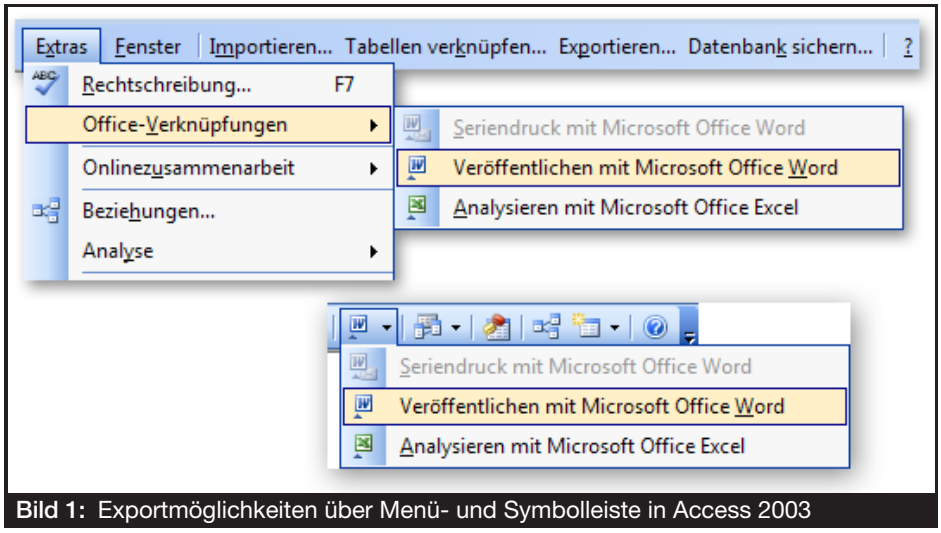


Bild 1: Exportmöglichkeiten über Menü- und Symbolleiste in Access 2003

Export von Tabellen und Abfragen

Markieren oder öffnen Sie in der Beispieldatenbank die Abfrage **qryStammdaten**. Sie zeigt alle Felder der Tabelle **tblStammdaten**, filtert die Datensatzgruppe jedoch nach allen Adressaten, die in Hamburg wohnen. Klicken Sie dann auf die Menüschaltfläche für den Word-Export Veröffentlichen in Microsoft Word. Sie werden nun nach dem Bestimmungsort und -namen für die anzulegende Word-Datei gefragt. Wählen Sie einfach das Verzeichnis der Datenbank selbst. Ganz passend ist der Ausdruck Word-Datei dabei nicht, denn Access lässt hier keine andere Wahl, als den Export in eine RTF-Datei. Mehr lässt sich zu diesem Vorgang nicht sagen. Access schaltet keine weiteren Dialoge mehr dazwischen und beginnt mit der Ausgabe, wobei ein ähnlicher Fortschrittsdialog, wie beim Drucken, erscheint. Schließlich öffnet sich Microsoft Word mit dem angelegten Dokument, falls Sie unter Access 2010 anfangs die Option **Öffnen der Zieldatei nach dem Exportvorgang** aktiviert hatten – unter Access 2003 passiert dies automatisch. Als Ergebnis erhalten Sie ein Dokument mit einer Tabelle, die im Kopf die Feldnamen trägt (wie in Bild 3). Das Layout ist DIN-A4 Hoch-

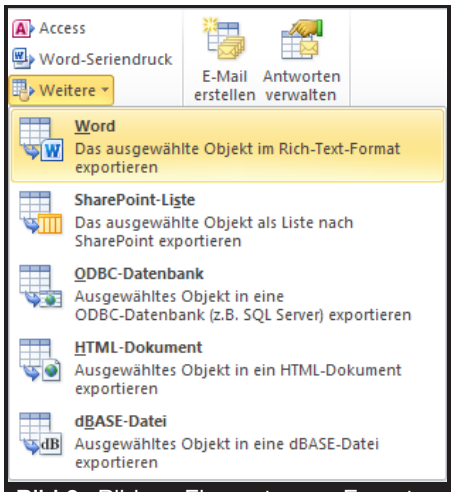


Bild 2: Ribbon-Elemente zum Export nach Word

format. Bei breiten Tabellen ragt die Word-Tabelle über den Rand hinaus, weshalb Sie sie besser markieren und über das Kontextmenü die Funktion **AutoAnpassen|Größe an Fenster anpassen** aufrufen. Der weiteren Formatierung des Dokuments steht nichts mehr im Wege.

IDAnrede	Vorname	Nachname	Strasse	NrStrasse	PLZ	IDOrt	IDLand
	Helmut	Dörr	Bautzener Allee	51	20095		
		Andrée	Wüstenrotstr.	12	20097		
		Schwarck			20099		
	Wilma	Teller	Schertlinstr.	11/144	20099		
	Anneliese	Bien	Im Holser Bruch	123	20144		
	Werner	Reitner	Brahmsallee	32	20144		
	Elisabeth	Trojok	Bundesstr.	86	20144		
	Bernhard	Bucher			20146		
		Lässig	Johann-Sebastian-Bach-Str.	3	20146		
	Thomas	Schultze	Nussbaumweg	13	20146		
	Renate	Stürhof			20146		
		Hielinger			20148		

Bild 3: Die nach Word transferierte Abfrage qryStammdaten

Wiederholen Sie nun den gesamten Vorgang für das Formular **frmStammdaten**. Der Ablauf ist identisch und... das Ergebnis ebenfalls! Die Hoffnung, Access würde hier eventuell das Layout des Formulars in Word nachbilden, wird nicht erfüllt. Es zieht lediglich die Datenherkunft des Formulars – hier ebenfalls die Abfrage **qryStammdaten** – heran und exportiert diese.

Export von Berichten

Anders sieht es bei Berichten aus. Machen Sie den Test mit dem Bericht **rptStammdaten** der Beispieldatenbank. Dort sind die Daten zusätzlich nach PLZ und Nachnamen gruppiert und mit einem abgestuften Layout versehen. Tatsächlich gleicht das exportierte Dokument dem Bericht weitgehend (Bild 4). Access analysiert hier also nicht nur die Daten, sondern auch die Positionierung der Ausgabefelder und einige Formatierungen. Dazu gehören Schriftarten und -größen, sowie Textfarben, nicht aber etwa Rahmen oder Hintergrundfarben. Linien oder Rechtecke, von eingebetteten Bildern ganz zu schweigen, bleiben außen vor. Für einfache Berichte ist dieser Export brauchbar, bei komplexen haben Sie oder Ihre Datenbankanwender das Nachsehen. Ein deutlicher Minuspunkt stellt etwa die Tatsache dar, dass die Positionierung der Texte Im Word-Dokument über Tabstopps geschieht. Das macht eine weitere Formatierung praktisch unmöglich. Geeigneter wäre ein Tabellenaufbau gewesen. Und Kopf- wie Fußbereiche des Berichts wandern auch nicht etwa in Kopf- und Fußzeilen des Dokuments, sondern sind normaler Bestandteil des Textes.

Stammdaten

Ort Hamburg

PLZ 20095

Nachname	Vorname	Anrede	Strasse	NrStrasse	Land
Dörr	Helmut	Frau	Bautzener Allee	51	Hamburg

PLZ 20097

Nachname	Vorname	Anrede	Strasse	NrStrasse	Land
Andrée		Herr Dr.	Wüstenrotstr.	12	Hamburg

PLZ 20099

Nachname	Vorname	Anrede	Strasse	NrStrasse	Land
Schwarck		Herr			Hamburg
Teller	Wilma	Frau	Schertlinstr.	11/144	Hamburg

PLZ 20144

Nachname	Vorname	Anrede	Strasse	NrStrasse	Land
Bien	Anneliese	Herr und	Im Holser Bruch	123	Hamburg
Reitner	Werner	Herr	Brahmsallee	32	Hamburg
Trojok	Elisabeth	Frau	Bundesstr.	86	Hamburg

Bild 4: Der nach Word exportierte Bericht rptStammdaten

Überlegen Sie also genau, ob Sie anderen Datenbankanwendern diese Exportmöglichkeit anbieten wollen.

Export per VBA

Der manuelle Exportvorgang lässt sich unter VBA mit einer einzigen Code-Zeile bewerkstelligen. Das Zauberwort ist hier die Anweisung OutputTo des DoCmd-Objekts. Über verschiedene Parameter lässt sich der Export steuern:

```
DoCmd.OutputTo acOutputQue-
ry, _
"qryStammdaten", _
acFormatRTF, _
CurrentProject.Path & _
"\stammdaten.rtf", True
```

Die Parameter im Einzelnen: Der erste gibt die Art des Objekts an, welches exportiert werden soll. Hier per Konstante **acQutputQuery** eine Abfrage. Darauf folgt der Name des Objekts in der Datenbank. Im Weiteren das Ausgabeformat, welches für den Word-Export immer **acFormatRTF** lauten muss. Und schließlich der Dateiname, unter dem das Dokument gespeichert wird. Im Beispiel ist das der Pfad der Datenbank **CurrentProject.Path** und **stammdaten.rtf** als Dateiname. Optional können Sie hinten noch angeben, ob das exportierte Dokument auch gleich in Word geöffnet werden soll und übergeben dafür **True** oder **False**.

qryStammdaten

Frau	Dörr	Helmut	Bautzener Allee	51	20095	Hamburg	Hamburg
Herr Dr.	Andrée		Wüstenrotstr.	12	20097	Hamburg	Hamburg
Herr	Schwarck				20099	Hamburg	Hamburg
Frau	Teller	Wilma	Schertlinstr.	11/144	20099	Hamburg	Hamburg
Herr und Frau	Bien	Anneliese	Im Holser Bruch	123	20144	Hamburg	Hamburg
Herr	Reitner	Werner	Brahmsallee	32	20144	Hamburg	Hamburg
Frau	Trojok	Elisabeth	Bundesstr.	86	20144	Hamburg	Hamburg
Frau	Bucher	Bernhard			20146	Hamburg	Hamburg

Bild 5: In Word geöffnete HTML-Datei als Resultat des HTML-Exports aus Access

qryStammdaten

Anrede	Nachname	Vorname	Strasse	NrStrasse	PLZ	Ort	Land
Frau	Dörr	Helmut	Bautzener Allee	51	20095	Hamburg	Hamburg
Herr Dr.	Andrée		Wüstenrotstr.	12	20097	Hamburg	Hamburg
Herr	Schwarck				20099	Hamburg	Hamburg
Frau	Teller	Wilma	Schertlinstr.	11/144	20099	Hamburg	Hamburg
Herr und Frau	Bien	Anneliese	Im Holser Bruch	123	20144	Hamburg	Hamburg
Herr	Reitner	Werner	Brahmsallee	32	20144	Hamburg	Hamburg

Bild 6: Mit Formatierung exportiertes HTML-Dokument

Export über HTML

Da Word auch HTML-Dateien öffnen kann, ist der Export eines Objekts aus der Datenbank in das HTML-Format ebenfalls eine Option. Dazu markieren Sie wieder das gefragte Objekt und stoßen den Export über **Externe Daten|Weitere|HTML-Dokument** an (Access 2007 ff.), oder über die Menüleiste **Datei|Exportieren... HTML-Documents** (Access 2003). Wenn Sie dabei nicht das Kontrollkästchen **Mit Formatierung** aktiviert hatten, sieht das Ergebnis unter Word aus, wie in Bild 5. Hier fehlen seltsamerweise die Spaltenüberschriften. Mit aktivierter Formatierung hingegen erhalten Sie das Ergebnis in Bild 6 mit Spaltenköpfen.

Import aus Word

Dass ein normaler Brief nicht in eine Datenbanktabelle importiert werden kann, bedarf wohl keiner Erläuterung. Sinn macht der Import nur aus einem Dokument, welches eine Tabelle enthält.

Access selbst sieht keine Möglichkeit vor, Daten aus einem Word- oder RTF-Dokument zu importieren. Der Workaround besteht darin, das Word-Dokument im Format **Nur-Text** zu speichern und dann unter Access mit dem Import-Assistenten zu transferieren. Beim Speichern eines Dokuments mit Tabelle wird allerdings eine Sequenz der Zellen generiert, die sich so nicht in Access importieren lässt. Vor dem Speichern steht

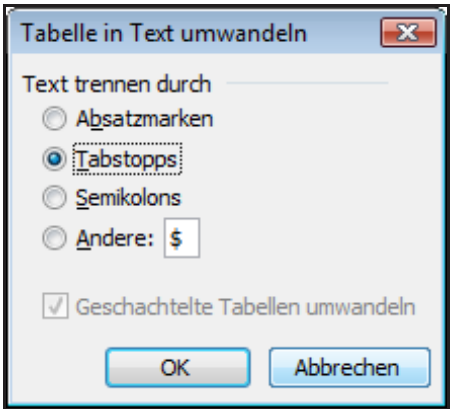


Bild 7: Die Word-Tabelle muss erst in Text umgewandelt werden

Anrede	Nachname	Vorname	Strasse	Nr	Strasse	PLZ	Ort	Land
Frau	Dörr	Helmut	Bautzener Allee	51		20095	Hamburg	Hamburg
Herr Dr.	Andrée		Wüstenrotstr.	12		20097	Hamburg	Hamburg
Herr	Schwarck					20099	Hamburg	Hamburg
Frau	Teller	Wilma	Schertlinstr.	11/144		20099	Hamburg	Hamburg
Herr und Frau	Bien	Anneliese	Im Holser Bruch	123				20144 Hamburg
Herr	Reitner	Werner	Brahmsallee	32		20144	Hamburg	Hamburg
Frau	Trojok	Elisabeth	Bundesstr.	86		20144	Hamburg	Hamburg
Frau	Bucher	Bernhard				20146	Hamburg	Hamburg
Frau	Lässig		Johann-Sebastian-Bach-Str.	3		20146		Hamburg
Frau	Schultze	Thomas	Nussbaumweg	13		20146	Hamburg	Hamburg

Bild 8: Durch Tabstopps getrennte Tabellenfelder eines Word-Dokuments

deshalb das Umwandeln der Tabelle in Text an. Markieren Sie die Tabelle und rufen Sie die entsprechende Funktion über Menüleiste oder Ribbon auf. Der Dialog aus Bild 7 fragt nun danach, wie die Zellinhalte getrennt werden soll.

Belassen Sie es bei der Voreinstellung **Tabstopps**. Nach Konvertierung hat das Dokument das Aussehen wie in Bild 8. Speichern Sie nun im Nur-Text-Format.

Zum Import in Access rufen Sie über das Menü **Datei|Externe Daten|Importieren...** auf, oder über das Ribbon **Externe Daten|Text-Datei**. Wählen Sie die soeben unter Word gespeicherte Textdatei aus und folgen Sie den Schritten des Importassistenten (Bild 9). Kontrollieren Sie dabei vor allem, ob Access die Felder richtig interpretiert und den Spalten die korrekten Datentypen zuweist. Access verwendet für die Analyse nur Stichproben der Tabelle und kommt so häufig zu falschen Entscheidungen. Die Spalte Hausnummer etwa wird gern als Integer-Wert angesehen, was den Import fehlschlagen lässt, wenn ein Datensatz mit einer Nr. 22A enthalten ist. Korrigieren Sie in diesem Fall den Felddatentyp manuell.

Außerdem achtet Access beim Import auf das sogenannte Dezimaltrennzeichen und kommt ins Schlingern, wenn es ein entsprechendes vorfindet. Ändern Sie das Zeichen für die Dezimaltrennung daher über den Dialog, der über den Button **Weitere...** erscheint in ein unwahrscheinliches Zeichen, wie etwa ALT-254 auf der numerischen Tastatur.

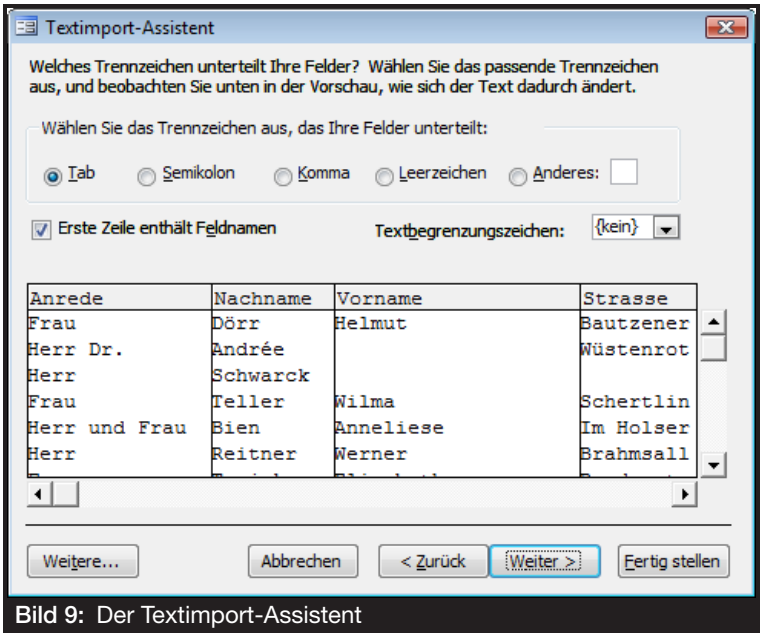


Bild 9: Der Textimport-Assistent

Nach Abschluss des Assistenten finden Sie eine Tabelle in der Datenbank vor, die dem Inhalt der Word-Tabelle entspricht. Fassen wir zusammen: Der Import aus Word ist stark eingeschränkt und lässt sich nur auf Tabellen anwenden, die zuvor in Tab-getrennten Text zu verwandeln sind. Als Speicherformat taugt ausschließlich das Nur-Text-Format.

Word-Seriendruck aus Access heraus

Bisher haben wir uns auf die einfache Ausgabe von Datenbankobjekten in Word-Tabellen beschränkt. Ausgefeilter ist dagegen die Seriendruckfunktion von Access, die Ihnen einige Arbeit beim Erstellen eines Seriendruckdokuments unter Word abnimmt.

Wieder einmal möchten wir Adressen der Tabelle **tblStammdaten** als Quelle für einen Serienbrief benutzen und filtern die Datensätze über die Abfrage **qryStammdaten**. Markieren Sie diese im Datenbankfenster oder im Navigationsbereich und klicken Sie auf das Menüelement zum Word-Seriendruck (Bild 1, Bild 2). Der nun erscheinende Seriendruck-Assistent besteht aus nur einem einzigen Dialogfenster, das lediglich die Auswahl zwischen dem Export in eine bereits bestehende

Rund um den VBA-Editor

Ob Sie schon ein alter Hase sind, oder gerade die ersten Schritte der Programmierung unter Access unternehmen, eine eingehendere Auseinandersetzung mit der Entwicklungsumgebung von Office und Visual Basic for Applications kann nicht schaden. Beleuchten wir deshalb in diesem Beitrag im Schnelldurchlauf deren Bestandteile, welche sich unter der Haube des VBA-Editors verbergen.

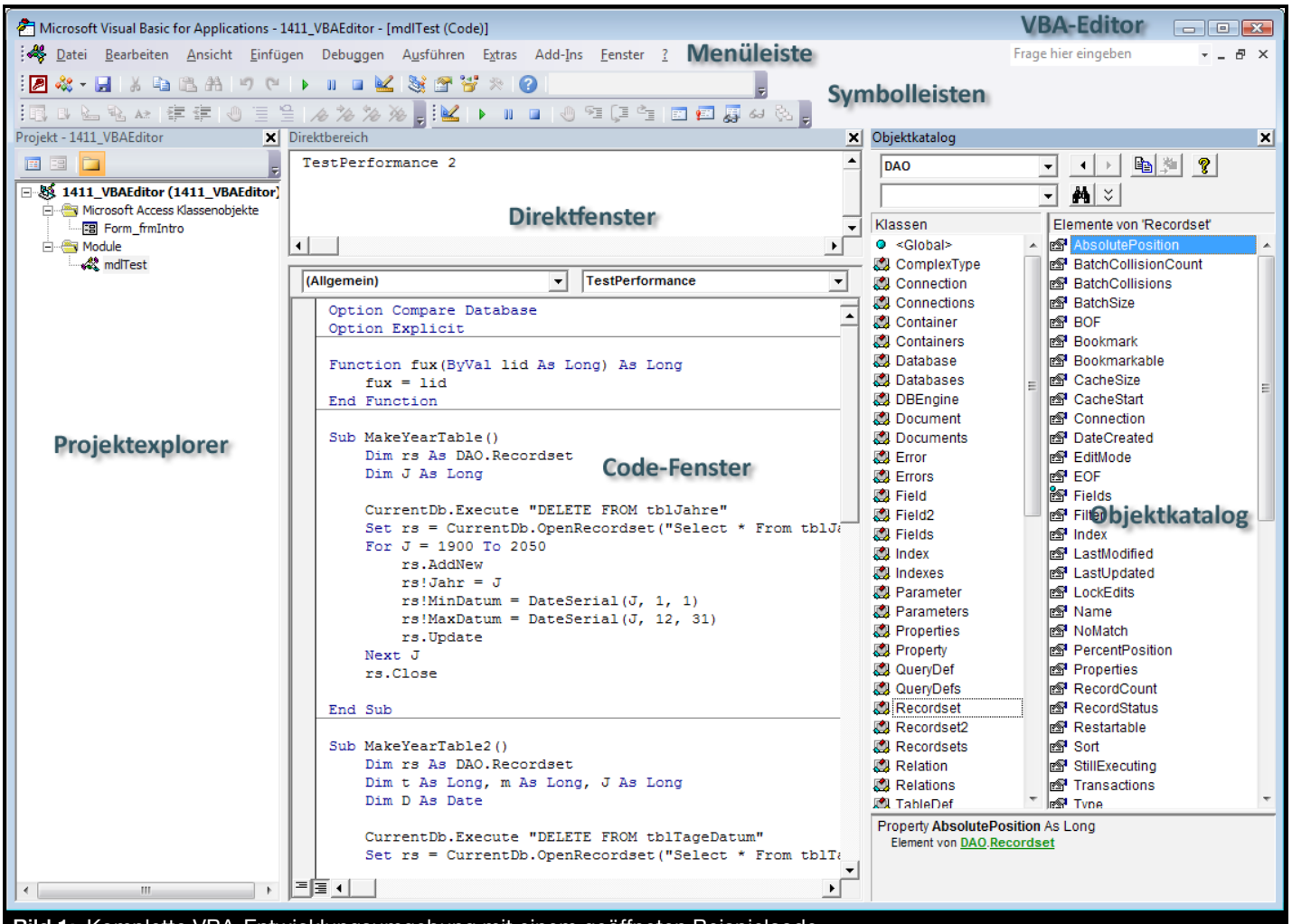


Bild 1: Komplette VBA-Entwicklungsumgebung mit einem geöffneten Beispielcode

Sie rufen die VBA-Entwicklungsumgebung von Access auf, indem Sie im Ribbon über den Reiter **Datenbanktools** die Schaltfläche **Visual Basic** anklicken, oder alternativ einfach die Tastenkombination **STRG-G** drücken. Ebenso gelangen Sie dorthin, wenn Sie den Button mit den drei Pünktchen betätigen, der sich hinter jeder Zeile eines Ereignisses im Formular- oder Berichtsentwurf befindet, wobei dann allerdings auch gleich der Rahmen einer Ereignisprozedur angelegt wird. Die VBA-Entwicklungsumgebung – kurz: VBA-Editor – sieht dann etwa so aus, wie in Bild 1 und stellt eine Sammlung von Einzelkomponenten dar, die sich ein- und ausblenden lassen. Sie unterscheidet sich in nichts von der VBA-Umgebung, die Sie auch unter Excel oder Word vorfinden, und sogar die hier vorgenommenen Einstellungen werden von allen Office-Anwendungen geteilt.

Die Komponenten sind in der Regel Unterfenster der Rahmenumgebung, die sich selbst an eine beliebige Stelle des Bildschirm platzieren lässt.

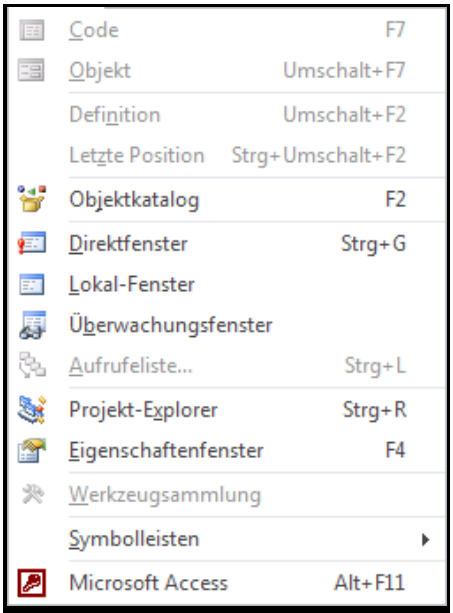


Bild 2: Menü Ansicht des VBA-Editors

Sollten Sie zwei Monitore zur Verfügung haben, dann macht es Sinn, den VBA-Editor auf den zweiten Bildschirm zu schieben, damit die Access-Oberfläche nicht von ihm überdeckt wird. Dort können Sie das Rahmenfenster auch maximieren. VBA merkt sich die Positionen und den Zustand aller Bestandteile und stellt sie beim nächsten Aufruf wieder her.

Der wichtigste Teil ist sicher das Code-Fenster, welches sich im Zentrum des Editors zeigt. Ein Code-Fenster weist kein eigenes Schließen-Kreuzchen auf, sondern wird über das rote Kreuzchen rechts oben unterhalb der Schließen-Schaltfläche der Umgebung selbst geschlossen. Welche weiteren Komponenten in der Umgebung eingeschaltet sind, steuert das Menü **Ansicht** (siehe Bild 2) und dessen Untermenü **Symbolleisten** (Bild 3).

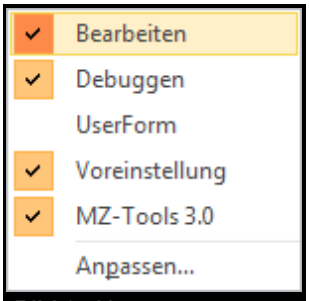


Bild 3: Untermenü zu Symbolleisten

Symbolleisten

Die Symbolleisten sind imgrunde nur für den schnelleren Zugriff auf bestimmte Funktionen gedacht, weisen aber keine Elemente auf, die nicht auch über das Menü des Editors zu erreichen wären. So finden sich etwa in der Symbolleiste **Voreinstellung** – in Bild 1 ganz oben – viele Elemente, die auch im Menü **Ansicht vorkommen**. Den Objektkatalog können Sie so über nur einen Button-Klick aufrufen, statt erst das Menü **Ansicht** aufzuklappen. Statt mehrere Symbolleisten anzuordnen, können Sie genauso gut über **Anpassen...** ihre eigene Zusammenstellung fabrizieren, und zwar auf die gleiche Weise, wie in früheren Office-Applikationen. Dabei ist es ratsam, nicht die eingebauten Leisten zu modifizieren, sondern eine neue Leiste zu erzeugen und die gewünschten Elemente in Sie zu kopieren, denn beim Zurücksetzen von durcheinander gekommenen eingebauten Leisten gingen sonst alle Anpassungen verloren. Vor allem sollten Sie Abstand davon nehmen, die Leisten von Add-Ins zu verändern, wie etwa die der **MZ-Tools** in Bild 3, auf die wir noch zu sprechen kommen.

Andockbare Fenster

Bis auf die Code-Fenster lassen sich alle weiteren VBA-Komponenten in die Umgebung eindocken oder als separate Fenster anzeigen. Diese Möglichkeit lässt sich über den Optionen-Dialog von VBA steuern (Bild 4). Standardmäßig ist das seltsamerweise für den Objektkatalog deaktiviert, was Sie besser ändern, um zu einer Anordnung, wie in Bild 1 kommen zu können. Sie lösen ein Fenster aus der Verankerung, indem Sie es an der Titelleiste solange in eine Richtung ziehen, bis es als separates Fenster erscheint. Sie verankern es wieder, indem Sie es an der Titelleiste in die gewünschte Umgebungsposition ziehen, was bei zahlreichen sichtbaren Komponenten nicht ganz einfach ist. Blenden Sie deshalb zunächst alle Kom-

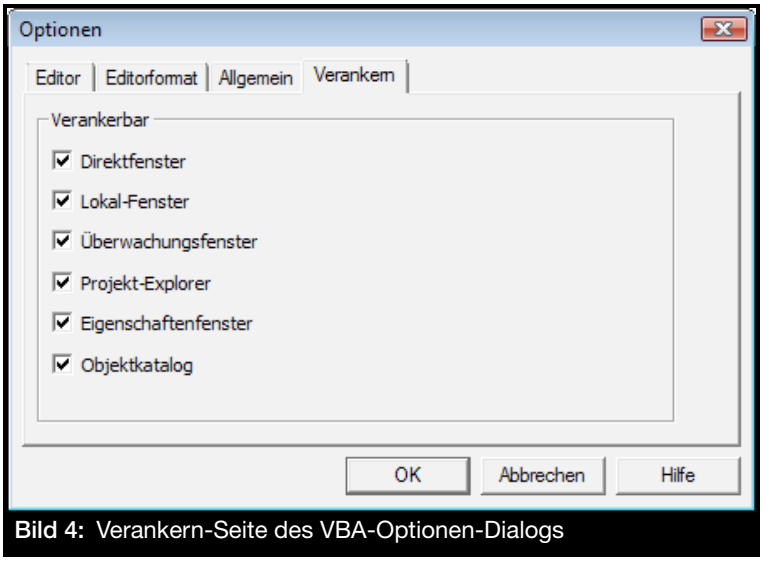


Bild 4: Verankern-Seite des VBA-Optionen-Dialogs

ponenten bis auf den Projektexplorer aus und fügen sie nach und nach die anderen Fenster hinzu, wobei Sie jeweils die Verankerung neu bestimmen. Welche Stellen Sie dabei bevorzugen, ist Geschmackssache.

Der Projektexplorer

Eine Datenbank kann ein VBA-Projekt enthalten, das alle seine Code-Teile beherbergt. Also globale Module, Klassenmodule, sowie Formular- und Berichtsmodule. Einen Spezialfall stellen die UserForm-Module dar, die jedoch in Access-Datenbanken, im Gegensatz zu Excel oder Word, praktisch irrelevant

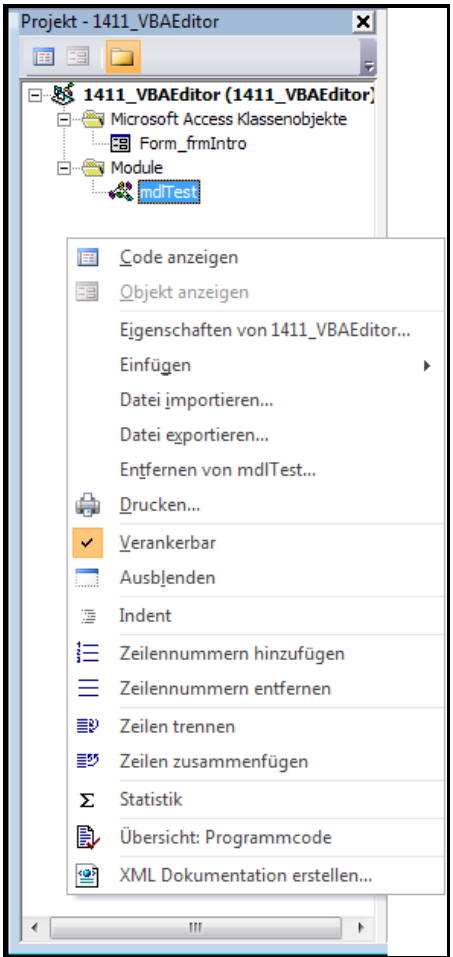


Bild 5: Der VBA-Projektexplorer mit seinem Kontextmenü

sind. Die Verwaltung der Module übernimmt der Projektexplorer, der mitsamt seinem Kontextmenü in Bild 5 zu sehen ist. Dieses Menü ist abermals eine Zusammenstellung von Anweisungen, die auch im Menü des VBA-Editors auftauchen, enthält jedoch zusätzliche Befehle, wie den zum Anlegen eines neuen Moduls per **Einfügen**. Manche Add-Ins legen darüber hinaus hier auch zusätzliche Einträge an – in Bild 5 etwa Anweisungen der **MZ-Tools**. Sie können den Projektexplorer ebenfalls als separates Fenster aus der Verankerung lösen. Es gibt allerdings keinen triftigen Grund, dies zu tun. In der Regel befindet er sich eingedockt an der linken Seite der VBA-Umgebung.

Code-Fenster

Den Code für ihre Module schreiben Sie in die Code-Fenster, die sich etwa öffnen, wenn Sie einen Eintrag im Projektexplorer doppelklicken. Mehrere Code-Fenster können parallel geöffnet sein, meist jedoch werden Sie nur eines im Zentrum der Umgebung angezeigt haben wollen. Mehrere zugleich braucht man eigentlich nur dann, wenn Code-Teile zu vergleichen sind oder Code von einem in ein anderes Modul zu kopieren ist. Die Ansicht der Code-Fenster wird über das Menü **Fenster** des VBA-Editors geregelt. Dort findet sich neben den üblichen Einträgen zu MDI-Anwendungen der spezielle Eintrag **Teilen**. Das aktive Code-Fenster wir damit horizontal unterteilt, wobei jeder Bereich zwar dasselbe Modul zeigt, jedoch unterschiedliche Stellen in ihm, die durch Scrollen in den sichtbaren Bereich gebracht werden. Auch diese Funktion wird man selten

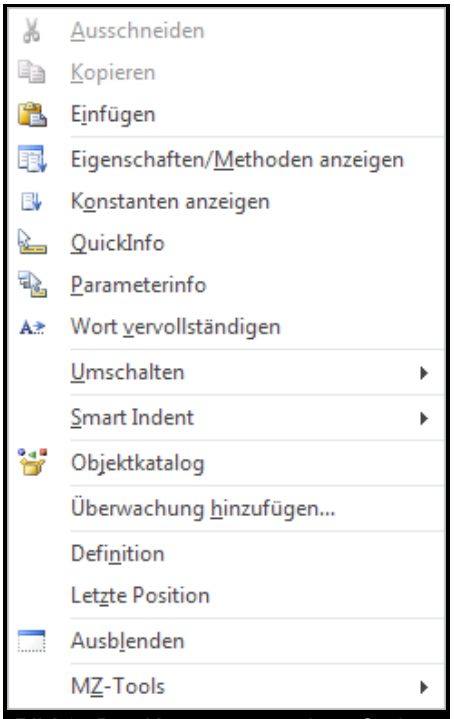


Bild 6: Das Kontextmenü eines Code-Fensters

benötigen. Sie ist etwa dann dienlich, wenn Sie in einer Prozedur den Aufruf einer Unterfunktion stehen haben, die Sie dann im unteren geteilten Bereich einblenden, um die Code-Übersicht zu verbessern.

Das Kontextmenü (siehe Bild 6) eines Code-Fensters weist etliche Einträge auf, die auch im Menü des VBA-Editors (**Bearbeiten**) vorkommen. Gehen wir hier jedoch auf einige Anweisungen ein, die nicht ganz selbsterklärend sind:

- **Definition** wirkt sich erst dann aus, wenn Sie einen Ausdruck im Code markiert haben. Es erfolgt in diesem Fall ein Sprung zu der Stelle, an der der Ausdruck definiert wurde. Das kann die Definition einer Variablen sein, einer API-Deklaration, einer eigenen Funktion, oder die Methode einer

Adressen konsolidieren

Der geradezu prototypische Anwendungsfall für eine Datenbank ist sicherlich die Speicherung von Kontakten. Ob Nordwind-, unsere Südwinddatenbank oder die hier oft verwendete Stammdatentabelle: im Zentrum stehen Adressdatensätze. Auf den ersten Blick gibt es dabei keinen Erklärungsnotstand, denn so ein Kontaktdatensatz scheint eine einfache Angelegenheit zu sein. Bei genauerem Hinsehen tauchen aber doch Fragen auf, die dieser Beitrag thematisiert.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1411_Adressen.mdb**.

Für die Beispieldatenbank zu diesem Beitrag wurde die Stammdatentabelle aufgebohrt und mit zusätzlichen Feldern und Daten versehen. Neben Namen und Anschrift kommen Telefonnummern und E-Mail-Adressen hinzu, so dass sie sich wie in Bild 1 gestaltet. In diese Tabelle könnten Sie bereits etwa Ihre Outlook-Kontakte importieren.

Bild 1: Aufbau der Adressentabelle

Man findet Adresstabellen in dieser Form recht häufig in Datenbanken. Dabei ist der Aufbau keineswegs optimal, obwohl es sich bereits um eine normalisiertere Form handelt, denn Anreden, Orte und Länder sind bereits mit **Lookup-Tabellen** verknüpft (siehe Bild 2).

Diese Felder verhindern auch einen einfachen Import aus anderen Datenquellen. Damit etwa im Feld **IDOrt** der Bezug zur Tabelle **tblOrte** gespeichert werden kann, muss diese erst mit Orten gefüllt worden sein. Natürlich könnte man diese Felder auch als Text anlegen, was in Hinsicht auf den Speicherbedarf jedoch suboptimal ist. Beim Import aus anderen Datenquellen kommt es in der Regel jedoch zu so einer Tabelle, die in der Beispieldatenbank als **tblAdressen-Import** zu finden ist (Bild 3).

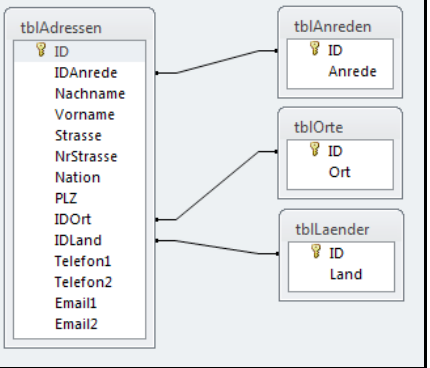


Bild 2: Lookup-Tabellen der Adressentabelle

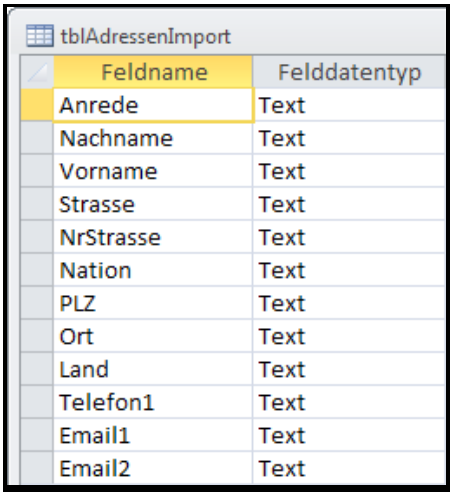


Bild 3: Aufbau der Importtabelle

Normalisieren des Imports
Im Folgenden wird Schritt für Schritt aufgezeigt, wie die Importtabelle in eine normalisierte Form überführt wird.

Legen Sie also zunächst die Lookup-Tabellen rechts in Bild 2 an, wobei für das Feld ID ein Autowert als Primärschlüssel zu verwenden ist und für das andere Feld – Anrede, Ort, Land – das Textformat. Jetzt kann die Tabelle **tblOrte** mit Daten gefüllt werden, wofür eine Anfügeabfrage zum Einsatz kommt:

```
INSERT INTO tblOrte (Ort)
SELECT DISTINCT Ort
FROM tblAdressenImport;
```

Nur die Orte der Ausgangstabelle werden hier in die Orte-Tabelle gespeichert, wobei Doppelungen infol-

ge der **DISTINCT**-Klausel ausbleiben. Analog die Anfügeabfragen für die Länder und Anreden:

```
INSERT INTO tblLaender (Land)
SELECT DISTINCT Land
FROM tblAdressenImport;

INSERT INTO tblAnreden (Anrede)
SELECT DISTINCT Anrede
FROM tblAdressenImport;
```

Sehen Sie sich die Inhalte der Tabellen an. Hier wird schon deutlich, worin der Vorteil der Normalisierung liegt. Waren in der Importtabelle noch 11.444 Felder mit Länderbezeichnungen gefüllt, sind es in der Ländertabelle nur noch 16 Einträge, auf die per ID verwiesen werden muss. Und da der Long-Wert von **IDLand** weniger Bytes benötigt, als ein Bezeichnungs-String, wird die Speicherersparnis offensichtlich.

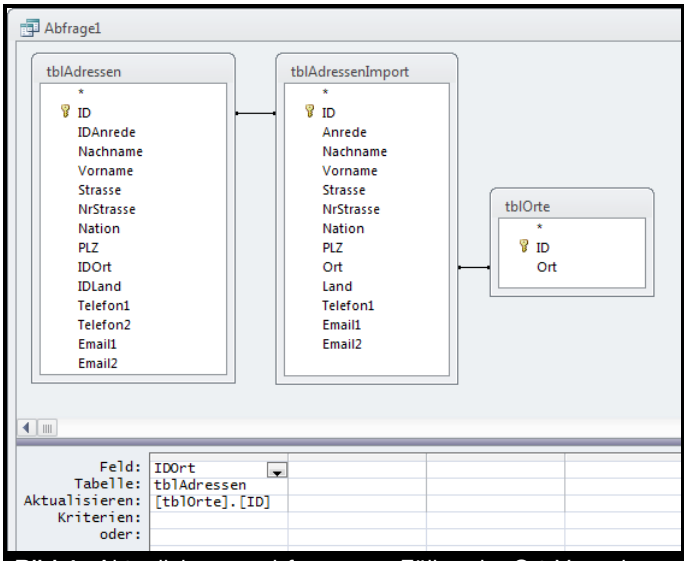


Bild 4: Aktualisierungsabfrage zum Füllen der Ort-Verweise

Diese Rechnung zeigt allerdings, dass nicht in jedem Fall eine solche Normalisierung nützlich ist. Nehmen Sie etwa das Feld **Nation** der Importtabelle. Auch hier könnte man eine Lookup-Tabelle **tblNationen** anlegen, die dann lediglich die zwei Einträge "D" und "CH" für Deutschland und Schweiz enthielte.

Ein Verweis auf die ID dieser Einträge kostet 4 Bytes und ist damit größer, als die ein beziehungsweise zwei Bytes der Strings. Eine Lookup-Tabelle wäre hier nutzlos und machte Abfragen auf die Tabellen nur komplizierter – es sei denn, Sie wollten die Lookup-Tabelle für die Auswahl in einem Kombinationsfeld verwenden.

Umgekehrt macht die Auslagerung in eine Lookup-Tabelle nur dann Sinn, wenn sich die Datenmenge dadurch reduzieren lässt. Würde man Postleitzahlen auslagern, wäre der Nutzen sehr gering, weil es eben so viele unterschiedliche Postleitzahlen gibt, dass man sie auch gleich in der Originaltabelle belassen kann.

Wenden wir uns dem Überführen der Daten aus der Importtabelle in die **tblAdressen** zu. Auch dafür eignet sich eine Anfügeabfrage, die verkürzt so daher kommt:

```
INSERT INTO tblAdressen ( Nachname, Vorname, Strasse, ...)
SELECT Nachname, Vorname, Strasse, ...
FROM tblAdressenImport;
```

Transferiert werden alle Daten mit Ausnahme der Felder für Anrede, Ort und Land, denn diese existieren in der Zieltabelle nicht als Textfelder. Sie enthält ja stattdessen Verweise auf die IDs der entsprechenden Lookup-Tabellen.

Wie kommen diese Lookup-Verweise nun aber in die Tabelle **tblAdressen**? Das ist nicht ganz so einfach. Dazu braucht es drei Ausgangstabellen, die miteinander verknüpft eine Aktualisierungsabfrage bilden, wie in Bild 4.

Die Importtabelle vergleicht über eine Beziehung das Ort-Feld mit den Inhalten der **tblOrte**. Die so ermittelte ID des Ortes wird dann benutzt, um das Feld **IDOrt** der Tabelle **tblAdressen** zu aktualisieren.

Auf ähnliche Weise verfahren Sie mit dem Aktualisieren der Felder für Anrede und Land.

Sie habe damit die neue Tabelle **tblAdressen** vollständig befüllt und können die Ausgangstabelle wieder löschen. Um ihre Daten anzuzeigen, bedienen Sie sich einer Auswahlabfrage, die den gleichen Aufbau hat, wie die Beziehungen in Bild 2.

Flexibilisieren des Adressenaufbaus

Eingangs war davon die Rede, dass der Aufbau der **tblAdressen** samt Lookup-Tabellen nicht wirklich optimal ist. So findet man für Telefonnummern und E-Mail-Adressen jeweils zwei Felder in ihr, was im

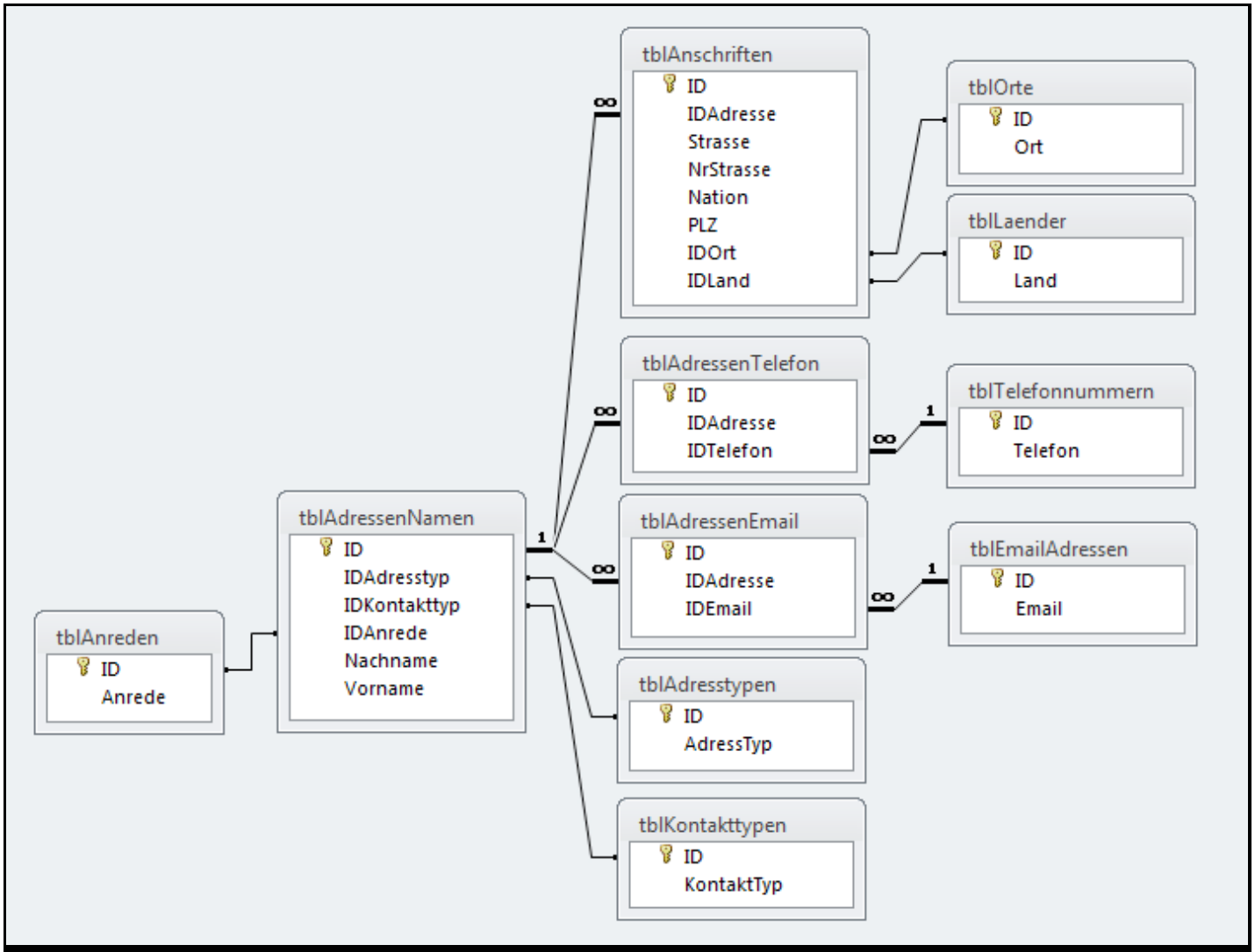


Bild 5: Das komplette Beziehungsmodell der Adressdatenbank, normalisiert und komsolidiert

einen Fall zu viel, im anderen Fall zu wenig sein kann. Ist zu einem Kontakt keine Telefonnummer bekannt – ein Umstand, der gerade in den Outlook-Kontakten nicht selten ist –, so bleiben die beiden Felder leer. Ebenso kann ein Kontakt aber auch mehr als zwei Telefonnummern aufweisen, etwa Privat-, Geschäfts- und Mobilnummer. Dann ist guter Rat teuer. Dasselbe gilt für E-Mail-Adressen.

Natürlich könnte man nun etwa die Zahl der Telefonnummernfelder in der Tabelle heraufsetzen, was jedoch zu weitgehend leeren Zellen führte wird und die Speicherverschwendung vorantriebe.

Für diesen Fall ist es daher günstiger, die Telefonnummern und E-Mail-Adressen in gesonderte Tabellen auszulagern und über die ID eines Adressdatensatzes zu verknüpfen.

Dabei entsteht allerdings keine Lookup-Tabelle, sondern eine n:m-Detailtabelle, die über eine Verknüpfungstabelle an die Adresse gekoppelt wird.

Legen Sie eine Tabelle **tblTelefonnummern** an, die den gleichen Aufbau hat, wie die zuvor beschriebenen Lookup-Tabellen und befüllen Sie sie auch analog.

Die Anfügeabfrage ist aber zweimal durchzuführen: je einmal für das Feld **Telefon1** und das Feld **Telefon2**:

```
INSERT INTO tblTelefonnummern (Telefon)
SELECT Telefon1 FROM tblAdressen;

INSERT INTO tblTelefonnummern (Telefon)
SELECT Telefon2 FROM tblAdressen;
```

VBA-Funktionen in Abfragen

Ob für die Berechnung von Feldern oder für Kriterien, um den Einsatz von VBA-Funktionen in Abfragen und SQL-Ausdrücken werden Sie häufig nicht herum kommen. Die Liste von originären Funktionen der Access-SQL-Engine ist ziemlich kurz, und für alle weitergehenden Rechenaufgaben verwendet Access nach Aussage von Microsoft den sogenannten VBA Expression Service. Betrachten wir in diesem Beitrag, wie beide zusammenspielen und auf was hier das Augenmerk liegen sollte.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1411_VBAIn-Abfragen.mdb**.

Abfragen-Komponenten

Wenn Sie in Ihrer Datenbank eine Abfrage zusammenbasteln, so kommen intern gleich drei Komponenten ins Spiel. Die offensichtlichste ist der Abfrageneditor von Access, der eine grafische Oberfläche zur Verfügung stellt, um per Maus und Tastatur, sowie die entsprechenden Hilfsdialoge, eine Abfrage visuell zu erstellen. In Bild 1 sehen Sie etwa den Entwurf einer ganz einfachen Abfrage auf die Tabelle **tblStammdaten**, um die sich auch in der Beispieldatenbank dieses Beitrags alles rankt. In ihr werden alle Felder und Datensätze der Tabelle ausgegeben, und das Ergebnis ist dasselbe, als würden Sie die Tabelle in der Datenblattansicht öffnen. Die Berechnung des Ergebnisses nimmt dabei aber nicht Access selbst vor, sondern die **Office Access Database Engine** (**ACE**), welche sich unter VBA auch über die **DAO**-Bibliothek ansprechen lässt. Access übersetzt lediglich den visuellen Entwurf in ein SQL-Statement und übergibt es der ACE-Engine.

Bild 1: Die Stammdatentabelle über eine Abfrage ausgeben

Das SQL-Statement können Sie ja auch über die Ansicht des Entwurfs einsehen oder modifizieren. Die ACE ist das Datenbank-Herzstück und quasi der SQL-Server von Access. Sie erwartet einen bestimmten SQL-Dialekt, reagiert also auf Anweisungen, die aus genau definierten Ausdrücken und Operatoren bestehen. Die Liste ihrer reservierten Ausdrücke finden Sie etwa unter **http://msdn.microsoft.com/en-us/library/bb208875%28v=office.12%29.aspx**.

Funktionen zur Weiterverarbeitung von Variablenwerten, seien es numerische oder alphanumerische, sind dort indessen so gut wie keine aufgeführt. Es handelt sich hier eher um Operatoren, die mit den Datenmengen selbst umgehen. Beispiele für originäre SQL-Funktionen wären etwa die **LIKE**- oder **BETWEEN**-Anweisungen. Sobald Sie jedoch darüber hinaus Funktionen auf Feldinhalte loslassen, klinkt sich der sogenannte **VBA Expression Service** als dritte Komponente ein. Eine Funktion, wie **Left()** zum Abschneiden von Strings – von Access im Abfragenentwurf überflüssigerweise in **Links**() eindeutscht – wird von VBA behandelt, weshalb deren Syntax und Parameter auch identisch sind mit dem, was Sie im VBA-Code angeben würden.

Sie können also fast beliebige VBA-Funktionen auf Feldinhalte anwenden. Der Umfang dieser Funktionen ergibt sich aus dem Objektkatalog des VBA-Editors, indem Sie dort die Bibliothek **VBA** auswählen. Diese Bibliothek ist kein Bestandteil von Access! Sie können etwa die **ACE** auf einem System auch ohne Access installieren und dennoch VBA-Funktionen verwenden, wenn Sie die Engine von einer anderen Programmiersprache aus benutzen.

Eine Besonderheit des **VBA Expression Service** ist jedoch, dass er benutzerdefinierte Funktionen ermöglicht. Die Verbindung zwischen eigenem Code und dem Service wird allerdings von Access vorgenommen.

Es sagt dem Service gewissermaßen, wo die Funktionen zu finden sind. Wie genau das funktioniert ist nicht dokumentiert, und es läuft ohne weiteres Zutun Ihrerseits ab.

Zusammengefasst lassen sich die an einer Abfrage beteiligten Komponenten im Überblick schematisch wie in Bild 2 darstellen. Access stellt damit ein recht leistungsfähiges System zur Verfügung, das mit sehr geringem Verwaltungsaufwand für die Entwicklung von Abfragen auskommt.

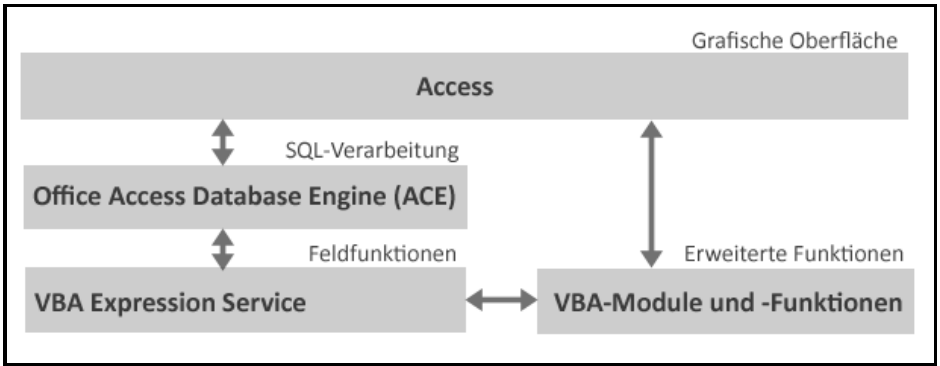


Bild 2: An einer Access-Abfrage beteiligte Komponenten (Schema)

VBA-Einsatz in Abfragen

Wo Licht ist, da ist auch Schatten. Bekannt etwa dürfte sein, dass VBA-Funktionen nicht unbedingt rasende Performance aufweisen. Das gilt im Besonderen für selbstprogrammierte Module, da VBA keinen richtigen Binär-Code erzeugt, sondern halbinterpretierend arbeitet. Eigene VBA-Funktionen sind also ziemlich langsam. Darauf muss geachtet werden, wenn solche Funktionen in Abfragen eingesetzt werden. Die mangelnde Performance von VBA allein ist jedoch noch nicht der einzige Bremsschuh. Um dies zu erläutern, betrachten wir ein Beispiel, welches die Abfrage aus Bild 1 um eine benutzerdefinierte Funktion erweitert (siehe Bild 3). Die VBA-Funktion **fux** ist im Modul **mdlTest** der Beispieldatenbank so angelegt:

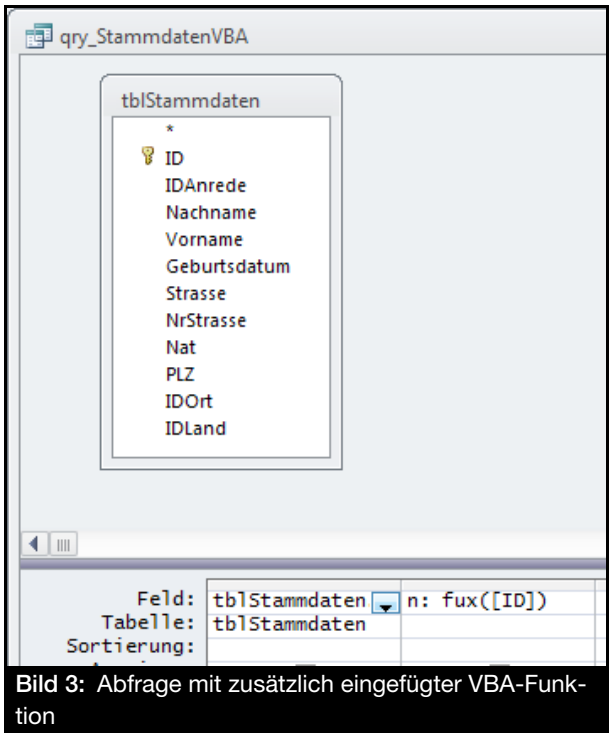


Bild 3: Abfrage mit zusätzlich eingefügter VBA-Funktion

```
Function fux(lId As Long) As Long
    fux = lId
End Function
```

Sie tut nichts anderes, als den ihr übergebenen Wert als Resultat wieder zurückzugeben. Genauso gut hätte man für die Abfragenspalte **n** auch auch schlicht das Feld **ID** direkt einsetzen können.

Um die Performance dieser Abfrage zu prüfen, existiert im Modul **mdlTest** eine Routine **TestPerformance**, die in einer Schleife hundertmal ein Recordset auf die Abfrage öffnet und alle Datensätze durchläuft.

Die dafür benötigte Zeit wird gemessen. Sie ist in Listing 1 zu finden und wird später noch gesondert beleuchtet. Im VBA-Direktfenster gibt sie schließlich die benötigte Zeit für ihre Ausführung aus. Rufen Sie sie etwa über diese Anweisung auf:

```
TestPerformance 0
```

Auf einem etwas betagten Rechner kommt für die in Bild 3 dargestellte Abfrage ein Wert von 3300 ms heraus. Ändern Sie die Abfrage ab, indem Sie die VBA-Funktion **fux** eliminieren und einfach durch den Inhalt das Feldes **ID** ersetzen, so gibt **TestPerformance** eine Zeit von 3360 ms aus. Das ist irritierend, denn zu erwarten wäre nach dem bisher Gesagten eher, dass die Ausführung nun schneller vonstatten gehen müsste. Der Grund liegt in der Code-Zeile verborgen, die einen Feldwert eines Datensatzes ermittelt:

```
V = rs.Fields(0).Value
```

Es wird der Wert der ersten Spalte des Recordsets abgefragt, und das

ist hier die ID des Datensatzes. Darum kommt die VBA-Funktion überhaupt nicht zur Geltung. Beweisen lässt sich dies, indem Sie einen Haltepunkt in die Funktion setzen und die Testroutine abermals starten: Ein Aufruf der Funktion **fux** findet nicht statt. Das Bild ändert sich, wenn Sie statt der Spalte 0 die elfte abfragen, also jene, die die Funktion **fux** auswertet.

Schon steigt die Ausführungszeit auf 8600 ms, also auf etwa das Zweieinhalbfache. Und das, obwohl die VBA-Funktion an sich sehr schnell ist und gar nichts berechnet. Das Ganze ist der Interaktion zwischen der Database Engine und dem VBA Expression Service im Verein mit dem VBA-Modul geschuldet.

Zusätzlich ändert die Engine ihren sogenannten **Query Plan**, den Ablaufplan der Abfrage, sobald benutzerdefinierte VBA-Funktionen eingebaut sind. Das zu erläutern führte an dieser Stelle zu weit.

Festgehalten werden sollte lediglich, dass Access bei Abfragen mit benutzerdefinierten VBA-Funktionen alle Datensätze eines Recordsets ermittelt, während ohne diese, je nach Gestaltung der Abfrage, meist nur die benötigte Teilmenge berechnet wird.

Weitere Beispiele finden Sie in Bild 4 und Bild 5, wo die Datensätze nach dem Wert der Spalte **ID** beziehungsweise der Funktion **fux** aufsteigend sortiert werden. Die Ausführungszeiten bei uns:

qry_Stammdaten_Sort	3343 ms
qry_StammdatenVBA_Sort	10875 ms

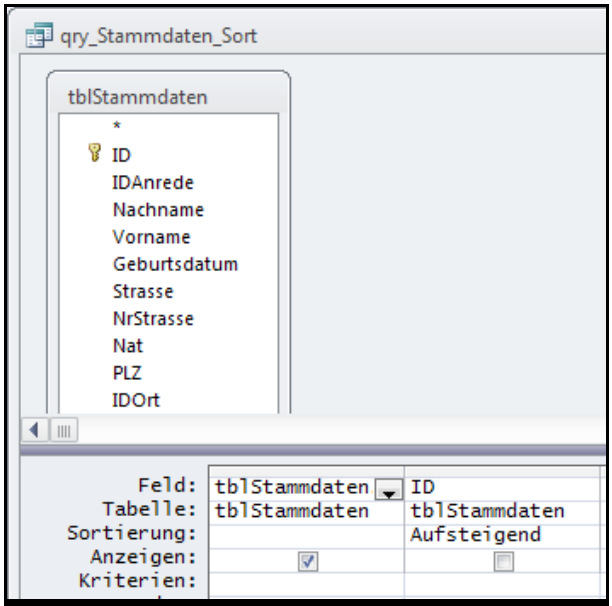


Bild 4: Datensätze nach der ID aufsteigend sortieren

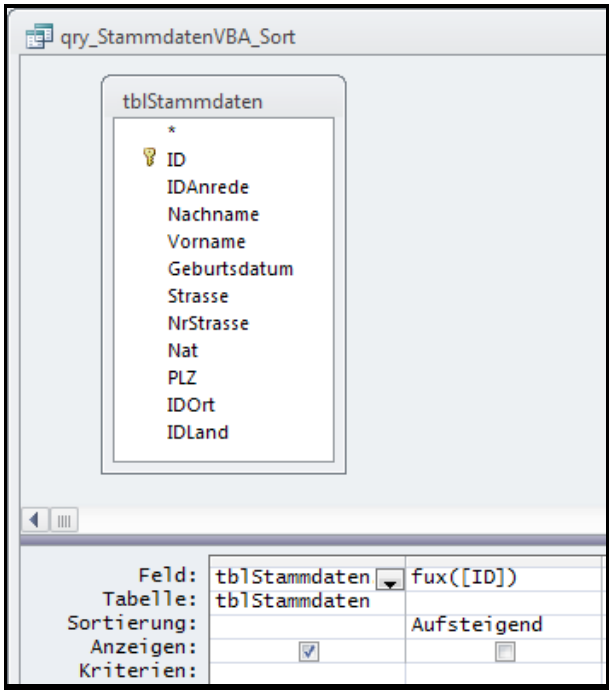


Bild 5: Datensätze nach fux() aufsteigend sortieren

Obwohl die Funktionalität bei beiden Abfragen dieselbe ist, unterscheidet sich die benötigte Zeit um den Faktor 3. Minimieren Sie also, wenn möglich, den Einsatz von VBA-Funktionen, so gut es geht.

VBA-String-Funktionen

Die bisherigen Beispiele hatten nur einen demonstrativen Zweck. Wenden wir uns im Folgenden sinnvolleren Beispielen zu. Sie wollten etwa alle Stammdaten ermitteln, deren Nachnamen mit "BE" beginnen. Über VBA ließe sich das per **Instr()** so lösen:

```
SELECT tblStammdaten.*
FROM tblStammdaten
WHERE
InStr(1,[Nachname],"BE"))=1
```

Wenn der Teil-String "BE" sich an der ersten Stelle befindet, ist die Bedingung erfüllt. Auf VBA kann aber in diesem Fall verzichtet werden, denn über den **LIKE**-Operator von SQL geht das Ganze auch so:

```
SELECT tblStammdaten.*
FROM tblStammdaten
WHERE [Nachname] Like "BE*"
```

Die Ausführungszeiten im Vergleich:

qry_Stammdaten_BE	437 ms
qry_Stammdaten_BEVBA	8230 ms

Die Performance unterscheidet sich krass! Das gute Abschneiden der SQL-Lösung beruht hier unter ande-

Access-Systemtabellen

Im Datenbankfenster oder im Navigationsbereich von Access mögen Ihnen die ausgegrauten Bezeichnungen von Tabellen mit dem Präfix MSys schon aufgefallen sein. Dabei handelt es sich um Systemtabellen, ohne die Access nicht funktionieren könnte. Reißen wir kurz an, was sich hinter diesen ominösen Tabellen verbirgt und wie Sie sie gar gewinnbringend für eigene Zwecke verwenden.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1412_Systemtabellen.accdb**.

Wenn Sie unter Access eine Datenbank entwickeln, so sammeln sich Unmengen von Definitionsdaten an. Ob es sich um die Entwürfe von Tabellen oder Formularen handelt, das Anlegen von Beziehungen, oder den VBA-Code, für all diese Objekte sind Eigenschaften und Einstellungen zu speichern. Und das macht Access natürlich selbst wieder in Tabellen, die sich in der Diktion von Microsoft eben Systemtabellen nennen.

Ihre genaue Bedeutung, Funktionsweise und ihr Aufbau sind jedoch nicht dokumentiert, weshalb sich eine Menge von Access-Spezialisten um deren Aufklärung bemühte. Viel davon liegt noch im Verborgenen, weshalb es bisher noch keine Software gibt, die Access-Datenbanken in allen Details in ein anderes Format konvertieren könnte. Während die Office-Dateiformate etwa von Word und Excel seitens Microsoft weitgehend offen gelegt wurden, verschließt man sich bisher einer Dokumentation für das Access-Dateiformat.

Systemtabellen anzeigen

Alle Systemtabellen haben eines gemeinsam: Sie sind mit dem Tabellen-Präfix **MSys** ausgestattet und haben das Attribut **System** und **Versteckt**. Zwar lassen sie sich damit öffnen, jedoch weder im Entwurf ändern, noch bearbeiten. Datensätze können in der Regel nicht gelöscht oder hinzugefügt werden – Ausnahmen bestätigen die Regel.

Damit versteckte Objekte im Datenbankfenster oder Navigationsbereich angezeigt werden, sind unter Umständen ein paar Handgriffe nötig. Klicken Sie etwa auf die Titelzeile des Navigationsbereichs rechts und betätigen **Navigationsoptionen...**.

Im nun erscheinenden Dialog haken Sie die Optionen **Systemobjekte anzeigen** und **Ausgeblendete Objekte anzeigen** an. Erst dann werden die Systemtabellen als graue Einträge aufgelistet, wie in Bild 1, wo Sie der Übersicht halber in eine eigene Navigationsgruppe verfrachtet wurden.

Der Umfang dieser Systemtabellen hat sich über die Access-Versionen hinweg immer wieder geändert. Viele sind erst in den neueren Versionen hinzugekommen, andere wieder sind inzwischen weggefallen, weil sich die Funktionalität von Access umgekrempelt hat.

Manche sind grundsätzlich in allen Datenbanken zu finden, andere entstehen erst, wenn bestimmte Vorgänge in der Datenbank angestoßen werden. Auch die Relevanz ist unterschiedlich zu bewerten. Im Folgenden geht es daher um jene Tabellen, die besonders wichtig sind und außerdem jene, die sich für eigene Zwecke – sprich: Abfragen – verwenden lassen.

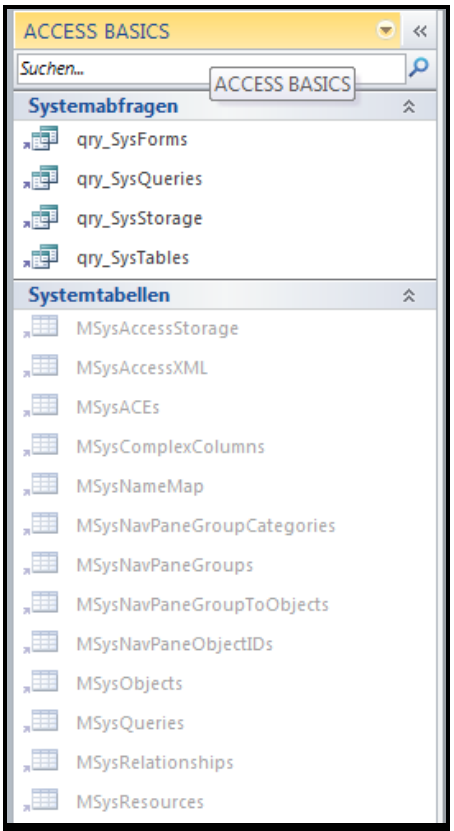


Bild 1: MSys-Systemtabellen

MSysObjects

Hier handelt es sich um eine immer präsente Systemtabelle, die ganz entscheidend ist, weil sie den Katalog der Objekte einer Datenbank enthält. Ganz egal, was Sie in Ihre Datenbank einbauen oder importie-

qry_SysObjects

Connect	DateCreate	DateUpdate	Flags	ForeignName	Id	LvExtra	LvProp	Name	ParentId	Type
	17.11.2014 09:34:55	17.11.2014 09:34:55	0		-2147483648			DataAccessPages	251658240	3
	17.11.2014 09:34:55	17.11.2014 09:34:55	0		-2147483647			Forms	251658240	3
	17.11.2014 09:34:55	17.11.2014 09:34:55	0		-2147483646			Modules	251658240	3
	17.11.2014 09:34:55	17.11.2014 09:34:55	0		-2147483645			Reports	251658240	3
	17.11.2014 09:34:55	17.11.2014 09:34:55	0		-2147483644			Scripts	251658240	3
	17.11.2014 09:34:55	17.11.2014 09:34:55	0		-2147483643			SysRel	251658240	3
	17.11.2014 11:23:17	18.11.2014 12:06:56	0		-2147483642	ıg binary-Daten	ı binary-Daten	AccessLayout	251658242	-32757
	17.11.2014 09:34:55	17.11.2014 09:34:55	0		-2147483641		ı binary-Daten	SummaryInfo	251658242	-32757
	17.11.2014 09:34:55	17.11.2014 09:34:55	0		-2147483640		ı binary-Daten	UserDefined	251658242	-32757
	09.01.2015 09:22:23	09.01.2015 09:22:23	0		-2147483639		ı binary-Daten	frmBestellungen	-2147483647	-32768
	09.01.2015 09:23:17	09.01.2015 09:23:17	0		-2147483638		ı binary-Daten	frmBestellungenChart	-2147483647	-32768
	09.01.2015 10:17:42	09.01.2015 10:17:42	0		-2147483637		ı binary-Daten	frmBestellungenDiagram	-2147483647	-32768
	21.11.2014 14:28:31	21.11.2014 14:28:31	0		-2147483636		ı binary-Daten	frmIntro	-2147483647	-32768
	09.01.2015 10:14:06	09.01.2015 10:14:06	0		-2147483635			mdlTest	-2147483646	-32761
	12.01.2015 13:44:26	12.01.2015 13:44:26	0		-2147483634		ı binary-Daten	rptBeziehungen	-2147483645	-32764
	09.01.2015 08:52:10	12.01.2015 14:24:12	0		-2147483633	ıg binary-Daten		Admin	-2147483643	-32758
	11.01.2015 19:02:27	11.01.2015 19:02:27	3		-2147483632		ı binary-Daten	~sq_cfrmBestellungen~s	251658241	5

Datensatz: 11 von 65 | Kein Filter | Suchen

Bild 3: Datenblattansicht der Systemtabelle MSysObjects; einige irrelevante Spalten sind ausgeblendet

MSysObjects

Feldname	Felddatentyp
Connect	Memo
Database	Memo
DateCreate	Datum/Uhrzeit
DateUpdate	Datum/Uhrzeit
Flags	Zahl
ForeignName	Text
Id	Zahl
Lv	OLE-Objekt
LvExtra	OLE-Objekt
LvModule	OLE-Objekt
LvProp	OLE-Objekt
Name	Text
Owner	Binär
ParentId	Zahl
RmtInfoLong	OLE-Objekt
RmtInfoShort	Binär
Type	Zahl

Bild 2: Systemtabelle MSysObjects

ren, es dazu wird immer als Datensatz in der Tabelle erscheinen. Den Aufbau sehen Sie in Bild 2, und in Bild 3 ausschnittsweise die Anzeige in der Datenblattansicht (Beispieldatenbank). Jeder Datensatz beschreibt ein einzelnes Objekt. Die Erklärung der Felder in der Reihenfolge ihrer Bedeutung:

- **Name** ist die Bezeichnung des Objekts, wie Sie oder Access selbst sie für ein Objekt vergeben haben. In der Abbildung ist etwa das Formular **frmBestellungen-Chart** markiert.

- **Id** ist der Primärschlüssel des Datensatzes, der das Objekt damit eindeutiger kennzeichnet, als der Name.

- **Type** charakterisiert den Typ des Objekts, wobei hier Zahlenwerte verwendet werden. Der Wert **-32768** etwa bedeutet immer **Formular**, eine **1 Tabelle**, oder **4** eine **verlinkte Tabelle**. In der Tabelle **tblObjektTypen** der Beispieldatenbank finden Sie eine Aufstellung der wichtigsten Objekte und deren Typwerte.

- **DateCreate**, **DateUpdate**. Hier sind Erstell- und Änderungsdatum des Objekts festgehalten.

- **Connect** ist ein String, der nur dann angegeben ist, wenn es sich beim Objekt um eine verknüpfte Tabelle handelt. Für Verknüpfungen zu Access-Datenbanken steht hier der Pfad zur Access-Datei, bei ODBC-Tabellen die Datenquelle oder deren Definition.

- **ForeignName** füllt sich ebenfalls nur bei verlinkten Tabellen. Dann steht hier der Tabellenname der Fremddatenbank, der ja vom Verknüpfungsnamen (**Name**) abweichen kann.

- **Flags** braucht eigentlich nicht weiter zu interessieren. Es ist eine Kombination von Attributkonstanten für ein Objekt, deren Anteile noch nicht aufgeklärt sind. Ob ein Objekt etwa versteckt ist, wird jedenfalls hier indiziert.

Bei den **Lv**-Feldern geht es um binäre Daten, die eine proprietäre Satzbeschreibung haben, deren Aufbau weitgehend unbekannt ist und deren Bedeutung je nach Objekttyp variiert.

Meist sind hier zusätzliche Interna und Eigenschaften des Objekts gespeichert. **LvProp** aber beschreibt die angelegten Eigenschaften eines Objekts, welche auch über die

Properties-Auflistung eines Objekts in VBA zu erhalten sind.

ParentId ist ein Verweis auf einen anderen Datensatz der Tabelle. Imgrunde ist sie rekursiv angelegt. Betrachten Sie die **ParentId** des Formulars **frmBestellungenChart**, so verweist sie auf den Datensatz mit der gleichwertigen **Id** und dem Namen **Forms**. Hier handelt es sich nicht um ein reales Objekt, sondern um ein Auflistungsobjekt – quasi die Kategorie des Objekts. Mithilfe der **ParentID** kann also im Verein mit **Type** die Zugehörigkeit eines Datenbankobjekts zu einer Objektkategorie festgestellt werden.

Die weiteren Felder der Systemtabelle haben nur Bedeutung im Zusammenhang mit ODBC-Datenbanken; wir schenken ihnen an dieser Stelle keine weitere Beachtung.

MSysObjects auswerten

Die Daten der Systemtabelle können unter keinen Umständen direkt verändert werden, aber als Grundlage für lesende Abfragen ist sie selbstverständlich nutzbar.

Normalerweise gelangen Sie an die Liste der in der Datenbank enthaltenen Tabellen nur über VBA und die Bibliothek **DAO**, indem entweder die **TableDefs**- oder die **Containers**-Auflistung durchlaufen wird. Durch **MSysObjects** wird das viel einfacher. Legen Sie eine Abfrage an, fügen die Systemtabelle ein und filtern Sie sie nach dem Wert von **Type**, wobei als Kriterium **=1** einzutragen wäre. Oder listen Sie alle Formulare der Datenbank auf, indem Sie

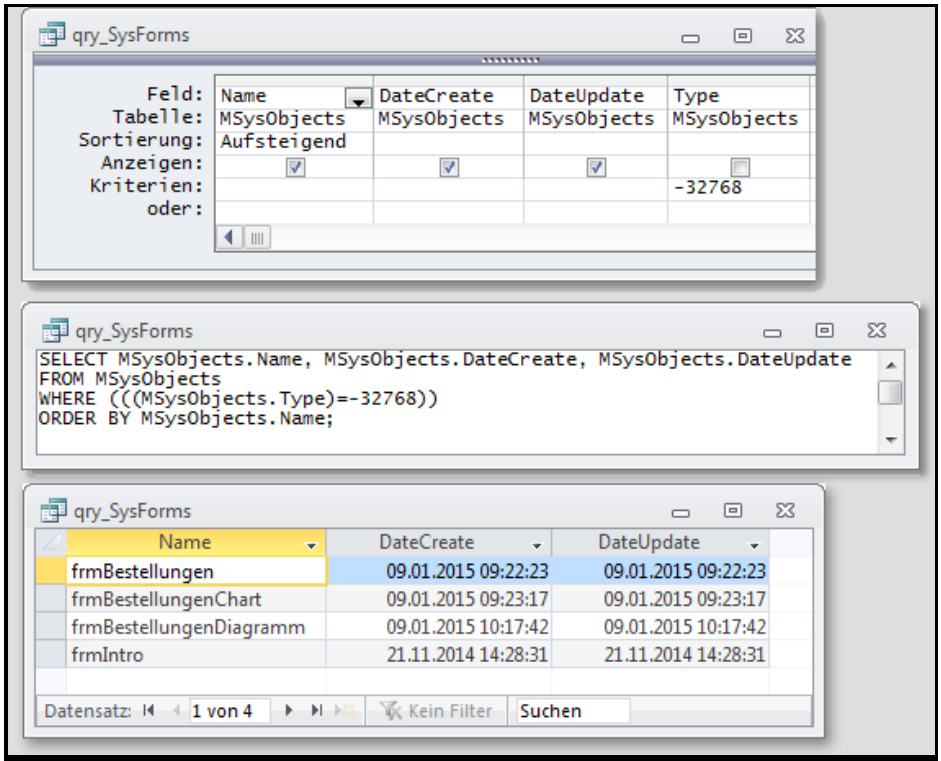


Bild 4: Formulare der Datenbank über MSysObjects auswerten und anzeigen

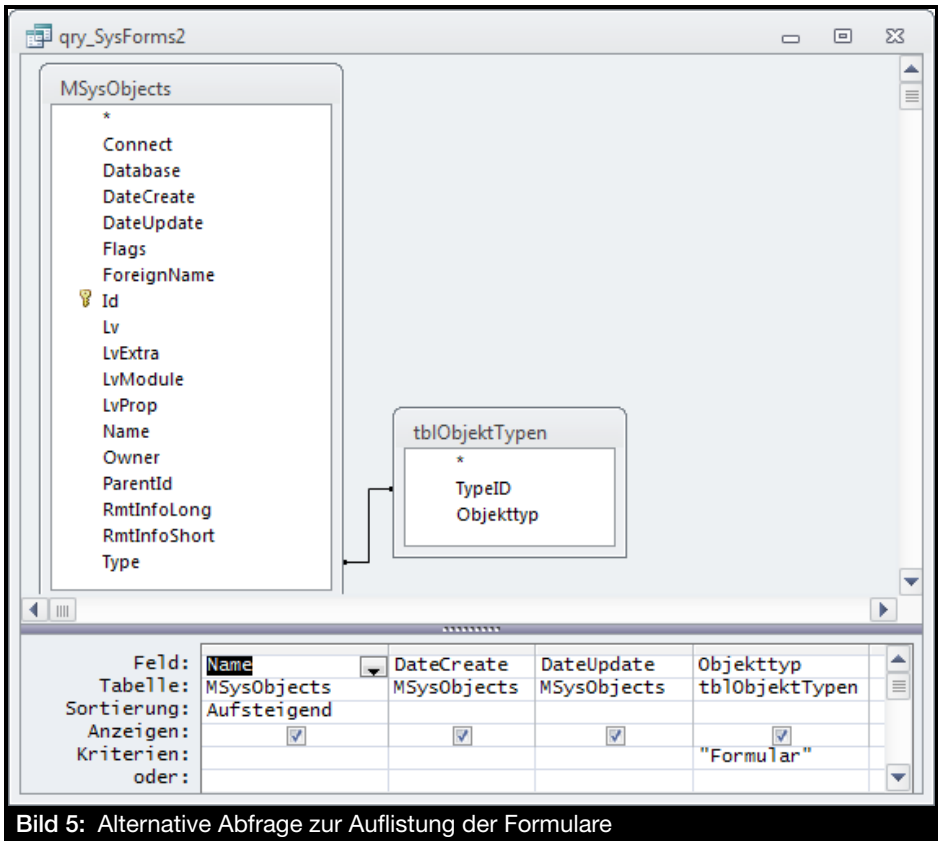


Bild 5: Alternative Abfrage zur Auflistung der Formulare

als Kriterium für **Type** die **-32.768** angeben. In Bild 4 sehen Sie Entwurf und SQL-Ausdruck dieser Abfrage, sowie das Ergebnis. Eine alternative Abfrage, die zum gleichen Ergebnis führt, zeigt Bild 5, wo die Hilfstabelle **tblObjektTypen** herangezogen wird. Auf ähnliche Weise können Sie mit MSysObjects alle anderen Objekttypen abfragen.

Interessant ist auch die Reihenfolge, in der die Objekte einer Datenbank angelegt wurden. Sortieren Sie MSysObjects einfach aufsteigend nach **DateCreate** und **Id**. Sie werden finden, dass in allen Datenbanken MSysObjects an oberster Stelle steht, gefolgt anderen Systemtabellen und den Objekt-Containern, die alle dasselbe Datum aufweisen.

MSysAccessStorage

Das ist die zweite wichtige Systemtabelle jeder Datenbank, wobei diese mindestens im Access 2003-Format vorliegen muss. In früheren Versionen existiert sie noch nicht. Das frühere Pendant wäre die **MSysAccessObjects**, deren Aufbau jedoch grundlegend anders und komplizierter ist und deshalb hier nicht erläutert wird.

In **MSysAccessStorage** (siehe Bild 6) werden alle Objekte selbst und deren Eigenschaften gespeichert – mit Ausnahme der Tabellen und Abfragen. Ihr Inhalt ist abermals rekursiv, was bedeutet, das ein Datensatz über **ParentId** auf einen anderen verweisen kann.

Die Verschachtelung ist allerdings tiefer, als bei **MSysObjects**. Ein Datensatz in ihr kann entweder ein Objekt selbst darstellen – dann hat **Type** den Wert 2 –, oder eine Art Verzeichnis – **Type** zeigt 1. Die Objekteinträge enthalten in der Spalte **Lv** in der Regel einen Binärwert, und das ist das eigentliche Objekt. Sein Inhalt hängt vom Objekttyp ab, ist von proprietärem Aufbau und bisher nirgendwo dokumentiert.

Auch in dieser Tabelle finden sich Angaben zu Erstell- und Änderungsdatum des Objekts. Die Spalte **Name** bedarf der Erläuterung: Hier sind die Objekte nicht so benannt, wie in der Datenbank selbst, sondern über Zahlenwerte indiziert. Eine Zuordnung vom Namen in **MSysObjects** zu diesen Werten gibt es nicht direkt.

MSysAccessStorage

DateCreate	DateUpdate	Id	Lv	Name	ParentId	Type
17.11.2014 09:34:55	21.11.2014 14:28:00	1		MSysAccessStorage_ROOT	1	1
17.11.2014 09:34:55	17.11.2014 09:34:55	2		MSysAccessStorage_SCRATCH	2	1
17.11.2014 09:34:55	21.11.2014 14:28:00	3	ıng binary-Daten	PropData	1	2
17.11.2014 09:34:55	12.01.2015 13:41:36	4		Forms	1	1
17.11.2014 09:34:55	12.01.2015 13:45:05	5		Reports	1	1
17.11.2014 09:34:55	09.01.2015 10:14:28	6		Modules	1	1
17.11.2014 09:34:55	13.01.2015 10:56:11	7		Scripts	1	1
17.11.2014 09:34:55	12.01.2015 14:29:22	8		VBA	1	1
17.11.2014 09:34:55	17.11.2014 09:34:55	9		Cmdbars	1	1
17.11.2014 09:34:55	17.11.2014 09:34:55	10		DataAccessPages	1	1
17.11.2014 09:34:55	17.11.2014 09:34:55	11		Databases	1	1
17.11.2014 09:34:55	17.11.2014 09:34:55	12		0	11	1
17.11.2014 09:34:55	17.11.2014 09:34:55	13		Blob	12	2
17.11.2014 09:34:55	17.11.2014 09:34:55	14		CustomGroups	1	1
17.11.2014 09:34:55	17.11.2014 09:34:55	15	ıng binary-Daten	└DirData	11	2

Datensatz: 1 von 76 | Kein Filter | Suchen

Bild 6: Die Systemtabelle MSysAccessStorage: Binär-Container für Objekte

Eigenschaftenblatt
Auswahltyp: Abfrageeigenschaften
Allgemein

Beschreibung	
Standardansicht	Datenblatt
Alle Felder ausgeben	Nein
Spitzenwerte	Alle
Keine Duplikate	Nein
Eindeutige Datensätze	Nein
Quelldatenbank	(aktuell)
Quellverbindung	
Datensätze sperren	Bearbeiteter Datensatz
Recordsettyp	Dynaset
ODBC-Wartezeit	60
Filter	([qry_SysStorage].[Name]="MSysAccessStorage_ROOT")
Sortiert nach	
Max Datensätze	
Ausrichtung	Von links nach rechts
Unterdatenblattname	Abfrage.qry_SysStorage
Verknüpfen von	ParentId
Verknüpfen nach	Id
Unterdatenblatthöhe	0cm
Unterdatenblatt erweitert	Nein
Beim Laden filtern	Nein
Beim Laden sortieren	Ja

Bild 7: Abfrage qry_SysStorage zur Anzeige der Rekursion in MSysAccessStorage

Diagrammformulare

Für die Auswertung von Daten unter Access reichen die Datenblattansichten von Abfragen oft nicht aus. Statistische Erhebungen lassen sich mit Diagrammen wesentlich anschaulicher auswerten. Werfen wir einen Blick auf die Möglichkeiten, die uns Access hier von Haus aus anbietet, ohne Umwege über Excel gehen zu müssen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1412_Pivotdiagramme.mdb**.

Es ist bei vielen Entwicklern gang und gäbe für die grafische Auswertung von Daten Excel-Diagramme heranzuziehen. Das ist indessen ein umständliches Unterfangen, denn Excel benötigt für Diagramme als Datenquelle ein Excel-Tabellenblatt.

Deshalb müssen Ergebnisse von Access-Abfragen erst nach Excel exportiert und dort weiterverarbeitet werden. Zwar lässt sich unter Excel auch eine direkte Verknüpfung zu einer Access-Datenbank anlegen, die Anwender Ihrer Datenbank werden jedoch froh darüber sein, wenn Sie Ihnen das Umschalten zwischen den beiden Office-Applikationen ersparen und Diagramme direkt in die Oberfläche von Access einbinden.

Microsoft Graph

Grundsätzlich besteht in allen Access-Versionen die Möglichkeit, ein Formular mit einem Diagramm zu bestücken. Bis **Office XP** kam hierfür ausschließlich **Microsoft Graph** zur Anwendung, das bei der Office-Installation ausdrücklich mitinstalliert werden musste. Es handelt sich dabei um eine ActiveX-Komponente, die sogar im Standalone-Modus ausgeführt werden kann, unter Access aber in Formulare oder Berichte eingebettet wird. **Graph** ist imgrunde ein Abkömmling des Excel-Diagramms und wird auf ähnliche Weise gehandhabt.

Damit Graph nicht über mehrere Schritte als ActiveX-Objekt einfügt werden muss, stellt Microsoft direkt über das Menü oder die Symbolleiste des Formularentwurfs das **Diagramm-Objekt** bereit. Beim Einfügen desselben startet ein Assistent, der Sie die Datenquelle auswählen und filtern lässt, sowie den gewünschten Typ des Diagramms festlegt. Die eigentliche Formatierung des Diagramms erfolgt erst nach dem Einfügen per Rechtsklick und **Bearbeiten**.

Die Anbindung von **Graph** an Access war jedoch stets undurchsichtig und leidet bis heute an OLE-Einbettungsproblemen. So lassen sich die Schriftgrößen der Diagrammbeschriftungen nicht sauber einstellen. Auch dann, wenn Sie in den Optionen eines Beschriftungselements die **Automatische Skalierung** deaktivieren, werden die Schriften nicht in der eingestellten Größe formatiert.

Bei 8-Punkt erscheint etwa eine Schrift, die eher nach 14-Punkt Fett aussieht. Diesem Verhalten ist nicht wirklich beizukommen – wahrscheinlich mit ein Grund, weshalb das Diagramm unter Access ein Nischendasein fristet.

Die grafische Qualität des Diagramms lässt darüber hinaus auch sonst zu wünschen übrig, wie etwa Bild 1 dokumentiert. Es zeigt das Formular **frmBestellungenDiagramm** der von Nordwind abgeleiteten Beispieldatenbank und wertet die monatliche Anzahl der Bestellungen über einer Zeitachse aus.

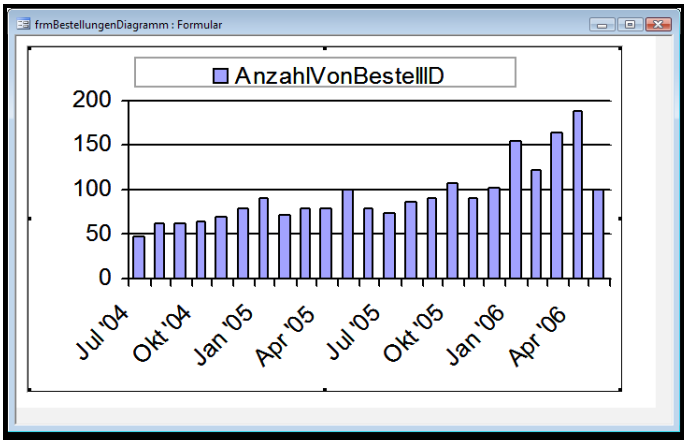


Bild 1: MS Graph-Diagramm der Beispieldatenbank

Pivot-Charts

Ab Access XP hat sich Microsoft einsichtig gezeigt und das Objektmodell von Formularen überarbeitet. Nun besteht die Möglichkeit das ganze Formular in den Modus **PivotChart** zu versetzen. Neben Formular-,

Entwurfs- und Datenblattansicht ist das also ein ganz neuer Ansichtsmodus. Dabei wird das Formular scheinbar selbst zu einen Diagramm, was bedeutet, dass weder Kopf- und Fußbereich, noch Navigationssteuerelemente angezeigt werden können. Wenn Sie diese brauchen, dann verwenden Sie das Pivot-Chart-Formular als eingebettetes Unterformular in einem entsprechend gestalteten Hauptformular.

Scheinbar deshalb, weil es sich abermals um einen Trick handelt: Das Pivot-Chart ist nämlich mitnichten ein Access-Original, sondern lediglich ein weiteres ActiveX-Steuerelement, das die ganze Fläche des Formulars einnimmt. Microsoft griff diesmal zu den **Office Web Components** die eigentlich für die Darstellung von Diagrammen im Internet Explorer entwickelt wurden.

Die Datenanbindung fällt nun allerdings erheblich komfortabler aus und die Qualität der Diagramme liegt deutlich über der von **Microsoft Graph**. Ein Minuspunkt ist die mäßige Performance der Web Components, denn diese verwenden intern nicht das DAO-Datenmodell, sondern ADODB. Unter Access geht deshalb im Hintergrund eine Konvertierung der Daten vonstatten, die bei größerem Datenumfang zu spürbaren Denkpausen bei der Berechnung des Diagramms führt.

Auch wenn Microsoft versucht hat, die Chart-Komponente in den Formularentwurf zu integrieren, ist die Handhabung nicht selbsterklärend, weshalb wir die grundlegenden Verfahrensweisen im Folgenden Schritt für Schritt aufzeigen.

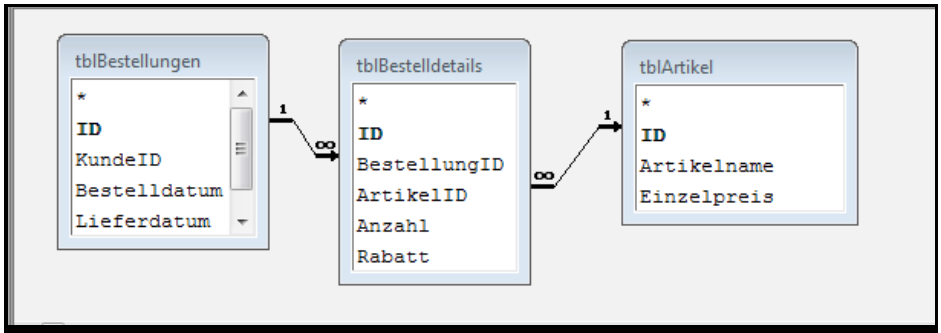


Bild 2: Tabellenbeziehungen der Beispieldatenbank

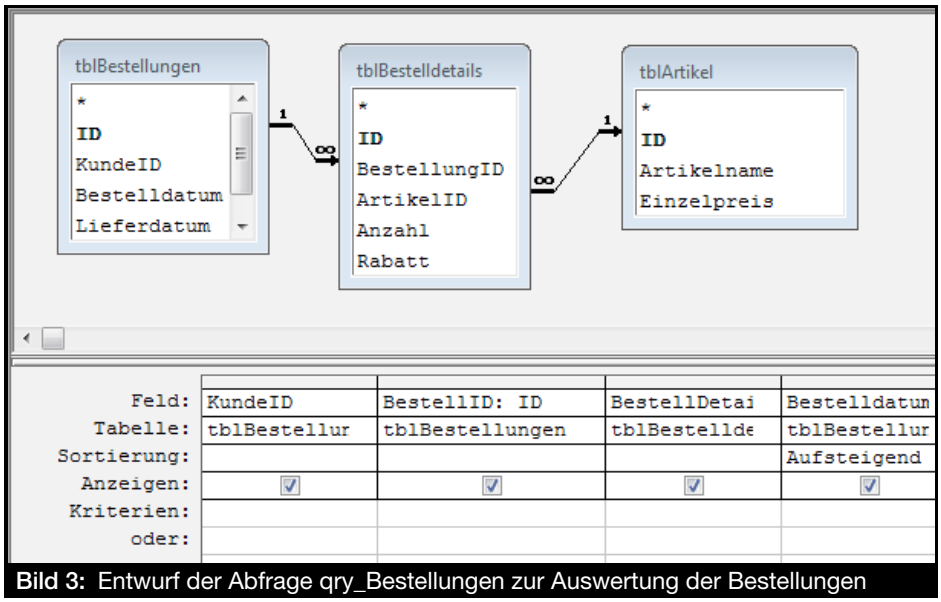


Bild 3: Entwurf der Abfrage qry_Bestellungen zur Auswertung der Bestellungen

Bestellungen auswerten

Die Beispieldatenbank zum Beitrag wurde aus der Nordwind-Datenbank abgeleitet, die Kunden, Artikel und Bestellungen verwaltet. Ziel ist es, die Anzahl von Bestellungen und den Umsatz über einen Zeitraum in einem Diagramm auszuwerten.

Für diese Auswertung wird die Tabelle Kunden selbst nicht benötigt, so dass im Datenmodell nur drei Tabellen vorkommen (siehe Bild 2). Als Datenherkunft für das Pivot-Formular **frmBestellungenChart** wird die Abfrage **qry_Bestellungen** herangezogen (Bild 3), welche die Tabellen in ähnlicher Weise verknüpft. Eine Gruppierung der Daten wird in der Abfrage nicht benötigt, denn die nimmt das Pivot-Chart später selbst vor.

Fehlanzeige unter Access 2013

Sollten Sie Access 2013 verwenden, so müssen wir Sie enttäuschen. Denn in dieser Version sind Pivot-Charts dem Rotstift zum Opfer gefallen. Über die Gründe kann nur spekuliert werden. Wahrscheinlich möchte sich Microsoft prinzipiell von ActiveX-Komponenten verabschieden. Unter Access 2013 kann

nur das unzureichende Graph-Diagramm benutzt werden, es sein denn, Sie installierten auf Umwegen die **Office Web Components** separat und fügten das Chart-Diagramm als ActiveX-Objekt in ein Formular ein.

Wie das geht, beschreiben wir später. Ist Ihnen die Installation der Web Components jedoch verbaut, so sollten Sie auf Diagramme unter Access eher verzichten und auf Excel ausweichen. Genau das empfiehlt Microsoft leider auch in der **Beschreibung der Unterschiede zwischen Access 2013** und den Vorversionen...

Pivot-Chart-Entwurf

Der Entwurf des Formulars ist zunächst sehr einfach. Wie für ein Datenblatt geben Sie einfach die Datenherkunft an und füllen den Detailbereich mit allen für das Diagramm benötigten Datensteuerelementen (Textfelder). Tatsächlich ist es der kürzeste Weg, erst ein **Autoformular Datenblatt** auf Grundlage der Abfrage vom Access-Formularassistenten erstellen zu lassen (Bild 4) und dann im Formularentwurf schlicht die Eigenschaft **Standardansicht** auf **PivotChart** zu stellen. Über das Diagramm können Sie zu diesem Zeitpunkt noch keinerlei Angaben machen.

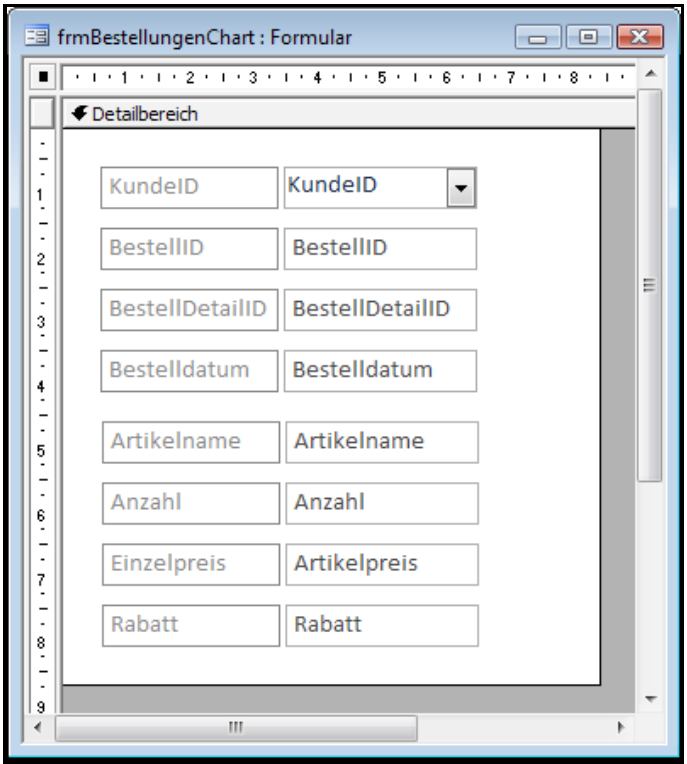


Bild 4: Das Pivot-Chart-Formular in der Entwurfsansicht

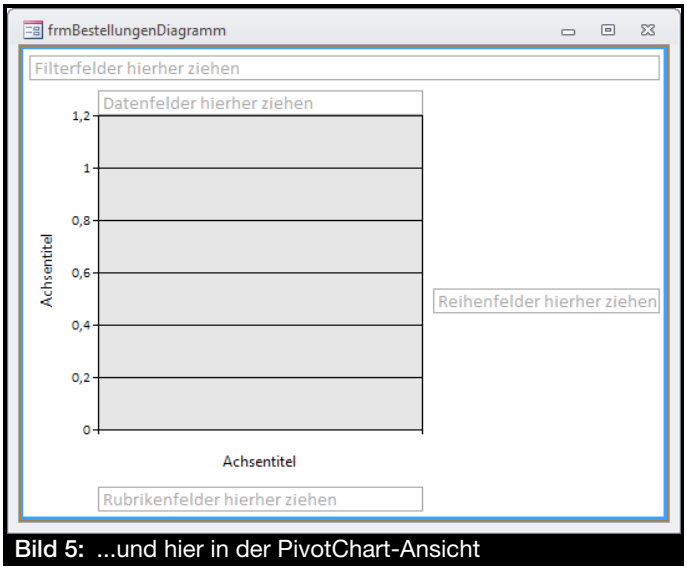


Bild 5: ...und hier in der PivotChart-Ansicht

Der eigentliche Entwurf des Charts erfolgt erst in der Formularansicht, die sich nun aber in Menü, Symbolleisten oder Ribbon **PivotChart-Ansicht** nennt.

Schalten Sie sie ein, so ändert sich das Layout wie in Bild 5. Von den zugrundeliegenden Daten ist noch nichts zu sehen. Diese blenden Sie ein, indem Sie den Dialog der sogenannten **Feldliste** aufrufen.

Das können Sie über das Kontextmenü des Diagramms oder über die in dieser Ansicht neu erschienene Menüleiste (Access 2003), beziehungsweise das Ribbon (Access 2007 ff.), bewerkstelligen – beide in Bild 6 dargestellt. Aus der Feldliste können Sie

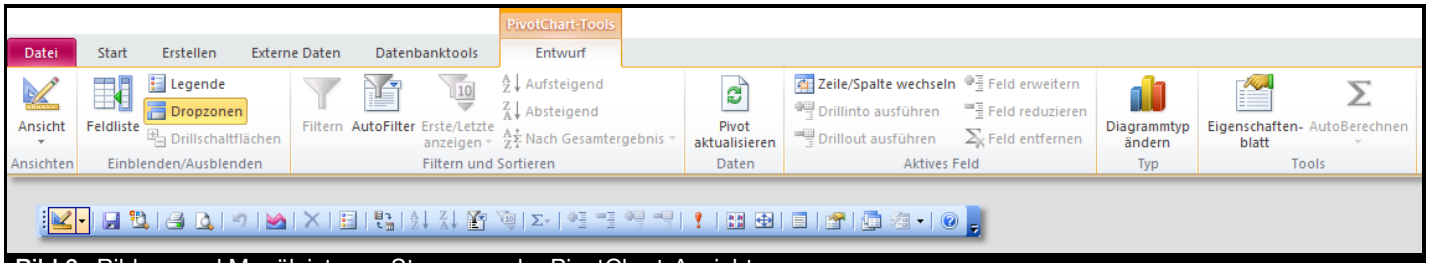


Bild 6: Ribbon und Menüleiste zur Steuerung der PivotChart-Ansicht

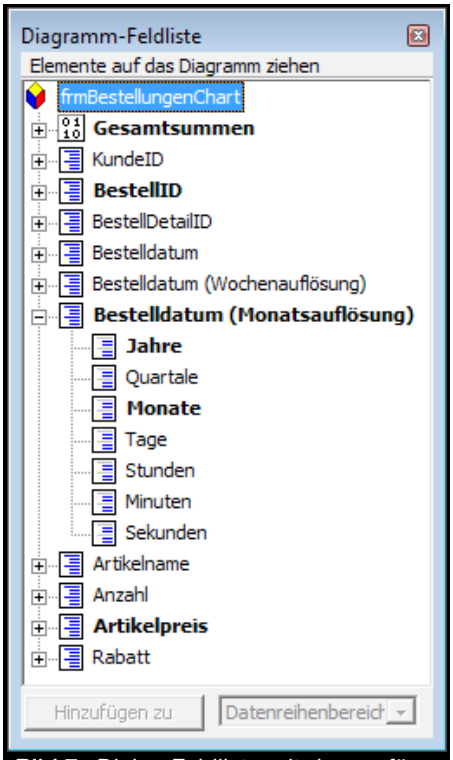


Bild 7: Dialog Feldliste mit den verfügbaren Datenfeldern der Datenquelle

dann einfach per **Drag And Drop** die gewünschten Abfragedaten in eine der vier **Drop-Zones** ziehen, von denen im Entwurf erst nur die Bereiche **Rubrikenfelder** und **Datenfelder** interessieren. Sollten die Dropbereiche nicht sichtbar sein, so blenden Sie sie ebenfalls über Ribbon oder Menüleiste ein.

Das Ribbon zum PivotChart ist dabei deutlich übersichtlicher, als das Menü von Access 2003. Dort wurden nämlich die Einträge auf verschiedene Stellen verteilt, so dass etwa das Item **Dropbereiche** im Menü **Ansicht** zu finden ist, die Formatierungen des Diagramms im Menü **PivotChart**, oder alle vereint in der neuen Symbolleiste.

Die **Feldliste** (Bild 7) zeigt also alle Datenfelder der im Formularentwurf zuvor eingestellten Datenherkunft an, hat jedoch bereits selbständig über die Intentionen des Entwicklers nachgedacht und schlägt deshalb zum **Bestelldatum** zusätzlich die Gruppierung auf Wochen- oder Monatsebene vor.

Damit wird deutlich, dass das Chart die Daten selbst verwaltet und Gruppierungen oder Filterungen eigenständig vornehmen kann, was die Definition der Formulardatenquelle erleichtert. Leider arbeitet es dabei nicht sonderlich performant, was sich bei zahlreichen Datensätzen deutlich niederschlägt. Sie sollten deshalb, soweit möglich, die Datenherkunft in einer Abfrage vorfiltern und die Zahl der Datensätze minimieren.

Unsere Absicht war gewesen, die Bestellungen der Nordwind-Datenbank über einen Zeitraum auszuwerten. Also muss das Bestelldatum auf der X-Achse angeordnet sein, welche im Pivot-Chart mit **Rubrikenfelder** bezeichnet ist.

Zögen Sie allerdings den Eintrag **Bestelldatum** mit der Maus auf den Dropbereich der Rubrikenfelder, so würde sich die X-Achse in Hunderte von Daten unterteilen und könnte nachträglich nicht mehr nach Monaten und Jahren gruppiert werden. Umgekehrt funktioniert das sehr wohl: Öffnen Sie den Zweig **Bestelldatum (Monatsauflösung)** und ziehen Sie den Untereintrag **Jahre** auf die Rubrikenachse.

Schon findet sich unten eine Beschriftung und die Achse ist mit den Jahren 2004 bis 2006 unterteilt – das ist der Zeitraum aller Bestelldaten in der Beispieldatenbank. Der Bereich der Achse stellt sich also automatisch nach den Daten ein, was sich jedoch auch ändern lässt, wie wir später sehen werden. Die Achsenbezeichnung **Jahre** weist außerdem einen Dropdown-Pfeil auf, der ein Dialogfenster wie in Bild 8 öffnet. Über den Dialog kann der Bereich der Achse gesteuert, also eine Filterung auf Jahresebene durchgeführt werden.

Nun reicht uns die Unterteilung nach Jahren für die Statistik nicht aus. Wir

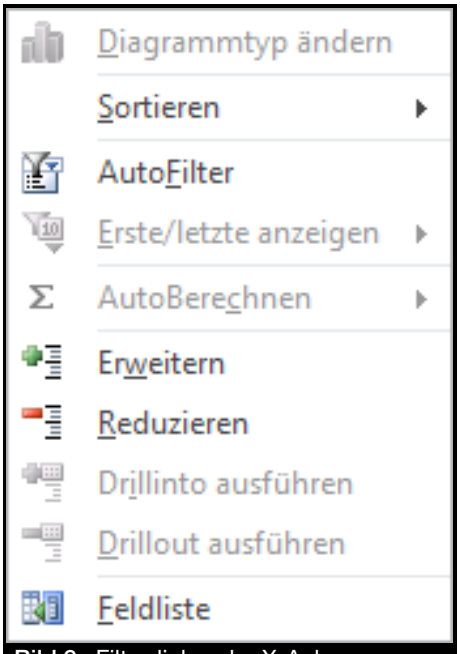


Bild 9: Filterdialog der X-Achse

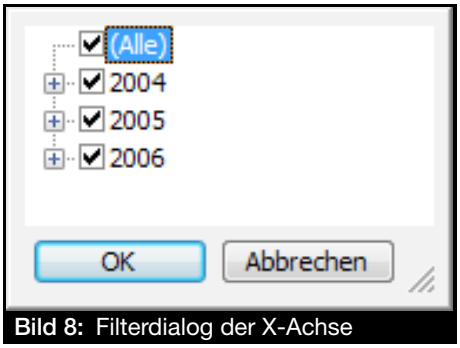


Bild 8: Filterdialog der X-Achse

hätten gleich den Untereintrag **Monate** der Feldliste in den Dropbereich ziehen können, es gibt aber einen Trick, um die Monatsgruppierung auf anderem Wege nachzuholen. Rechtsklicken Sie dazu auf die Bezeichnungsfläche **Jahre** und aktivieren im Kontextmenü (Bild 9) den Eintrag **Erweitern**. Dadurch entsteht eine weitere Achsenunterteilung in Quartale.

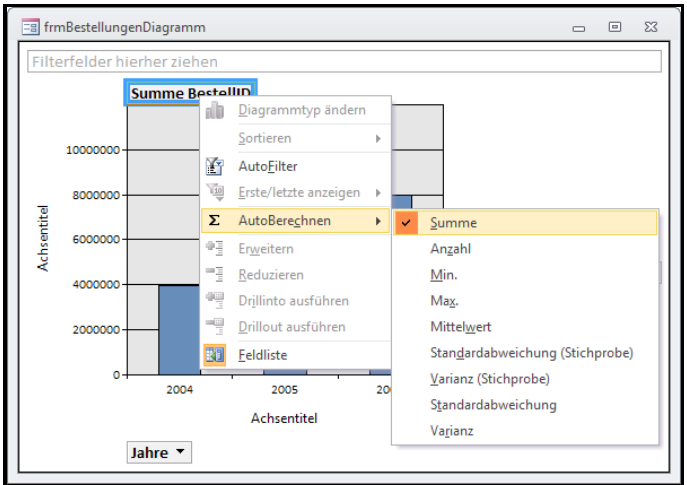


Bild 10: Ändern der Aggregatfunktion für die Datenreihe

Wiederholen Sie den Vorgang, so tauchen auch die Monate auf. Das ließe sich bis auf Sekundenebene fortsetzen. Für unseren Zweck benötigen wir nur Jahre und Monate, nicht aber die Quartale. Diese lassen sich schnell entfernen: Entweder markieren Sie die Bezeichnungsfläche mit der Maus und drücken die ENTF-Taste, oder Sie ziehen sie mit der Maus soweit in irgendeine Richtung, bis das Löschen-Symbol erscheint.

Nachdem die Unterteilung der X-Achse festgelegt ist, schreiten wir zur Bestimmung der im Diagramm anzuzeigenden Daten, was sich durch Bestücken des Dropbereichs **Datenfelder** vollzieht. Um die Anzahl von Bestellungen zu ermitteln, kann etwa die ID jedes Datensatzes verwendet und anschließend gezählt werden. Ziehen Sie den Eintrag **BestellID** aus der Feldliste in den Dropbereich **Datenfelder** und haben damit – voilá! – schon ein vollständiges Diagramm fabriziert. Allerdings zeigt es noch nicht das, was beabsichtigt war: Statt der Anzahl von Datensätzen ermittelt es die Summe der ID-Werte.

Korrigieren Sie das, indem Sie auf die Datenbezeichnung **Summe BestellID** rechtsklicken und aus dem Kontextmenü die **AutoBerechnen**-Funktion **Anzahl** auswählen (siehe Bild 10). Sie erhalten nun das gewünschte Ergebnis.

Pivot-Charts formatieren

Das Diagramm zeigt jetzt bereits die korrekte Statistik, lässt aber im Layout noch zu wünschen übrig. Natürlich lassen sich fast alle Elemente des Diagramms, ähnlich, wie unter Excel, in ihrem Aussehen modifizieren. Dafür gibt es einen eigenen Dialog, das Eigenschaftenfenster. Bringen Sie es über das Menü **Ansicht|Eigenschaften** oder die Ribbon-Schaltfläche **Eigenschaftenblatt** in den Vordergrund (Bild 11).

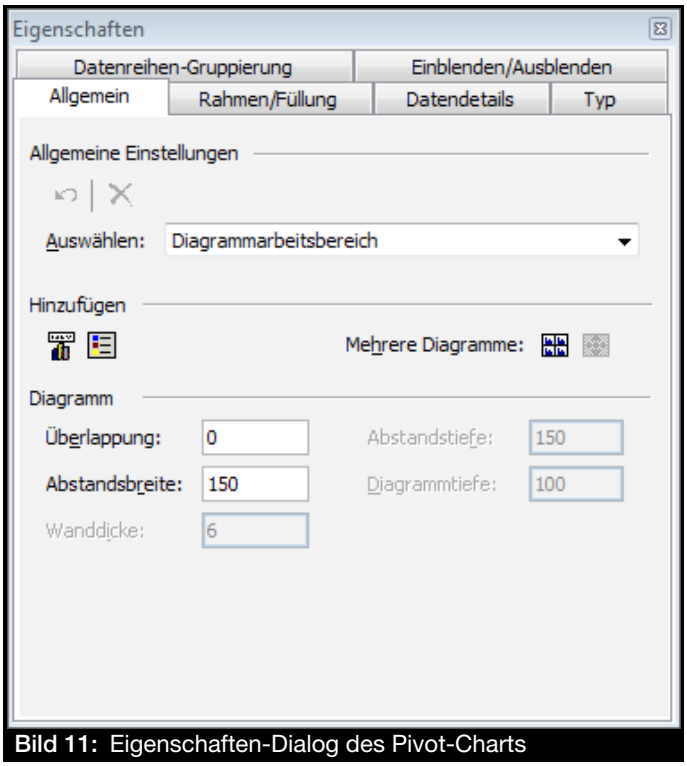


Bild 11: Eigenschaften-Dialog des Pivot-Charts

Nachdem es einmal offen ist, lassen sich die Diagrammelemente auf zweierlei Arten formatieren. Entweder klicken Sie mit der Maus auf das gefragte Element/den Bereich, oder spezifizieren Sie es über die Combobox **Auswählen** auf der Registerseite **Allgemein**. Dabei ändern sich Zahl und Bezeichnung der weiteren Registerkarten des Dialogs. Über die Seiten des Dialogs lassen sich nun alle möglichen Einstellungen und Formatierungen zum gewählten Element vornehmen, seien es Linienformen, Farben, Füllungen, Schriftarten, oder auch Angaben zu Achsenbereichen.

Datenbanken und Objekte sichern

Während der Entwicklung Ihrer Datenbank kann einiges schief gehen. Sei es, dass Sie sich vertan haben und einen Schritt zurückgehen müssen, versehentlich Objekte oder Code löschten, oder die Datenbankdatei selbst aus unerfindlichen Gründen korrupt wird. Wohl dem, der da regelmäßig Sicherungen anlegt, auf die später zurückgegriffen werden kann!

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1501_Sicherung.mdb**.

Sichern über die Benutzeroberfläche
Die einfachste Art, um Scherereien aus dem Weg zu gehen, ist das regelmäßige Backup Ihrer Datenbankdatei. Hier können Sie auf die Backup-Funktionen des Betriebssystems zurückgreifen, oder auf die Hilfe und Software von Drittanbietern, die häufig etwas ausgefeiltere Möglichkeiten aufweisen. Grundsätzlich ist dabei ratsam, die Backup-Dateien nicht immer mit der aktuellsten Version zu überschreiben, sondern sie mit einem Datum im Dateinamen zu versehen, damit verschiedene frühere Versionen und Entwicklungsstände zur Verfügung stehen.

Denn häufig wird man erst nach einiger Zeit gewahr, dass ein ehemals verworfenes Modul doch noch benötigt wird. Die Arbeit, es dann neu zu schreiben, können Sie sich bei regelmäßiger Sicherung ersparen.

Backup-Möglichkeiten außerhalb von Access sollen aber nicht Thema dieses Beitrags sein. Schauen wir einmal, welche Möglichkeiten es selbst anbietet.

Seit Access-Version 2003 sieht es eine Funktion vor, mit der sich die aktuell geöffnete Datenbank in eine neue Datei speichern lässt. Sie nennt sich schlicht **Datenbank sichern**. Unter Access 2003 findet sie sich über das Menü **Datei**, in den Folgenversionen im sogenannten **Backstage**, der sich bei Access 2007 über den **Office-Button**, unter Access 2010 über den Reiter **Datei** öffnen lässt (siehe Bild 1). Doppelklicken Sie auf die Fläche zum Sichern, dann fragt sie Access in einem Dateiauswahldialog, wohin es die Sicherung speichern soll und schlägt auch gleich einen Namen für die Backup-Datei vor. Dieser bildet sich in der

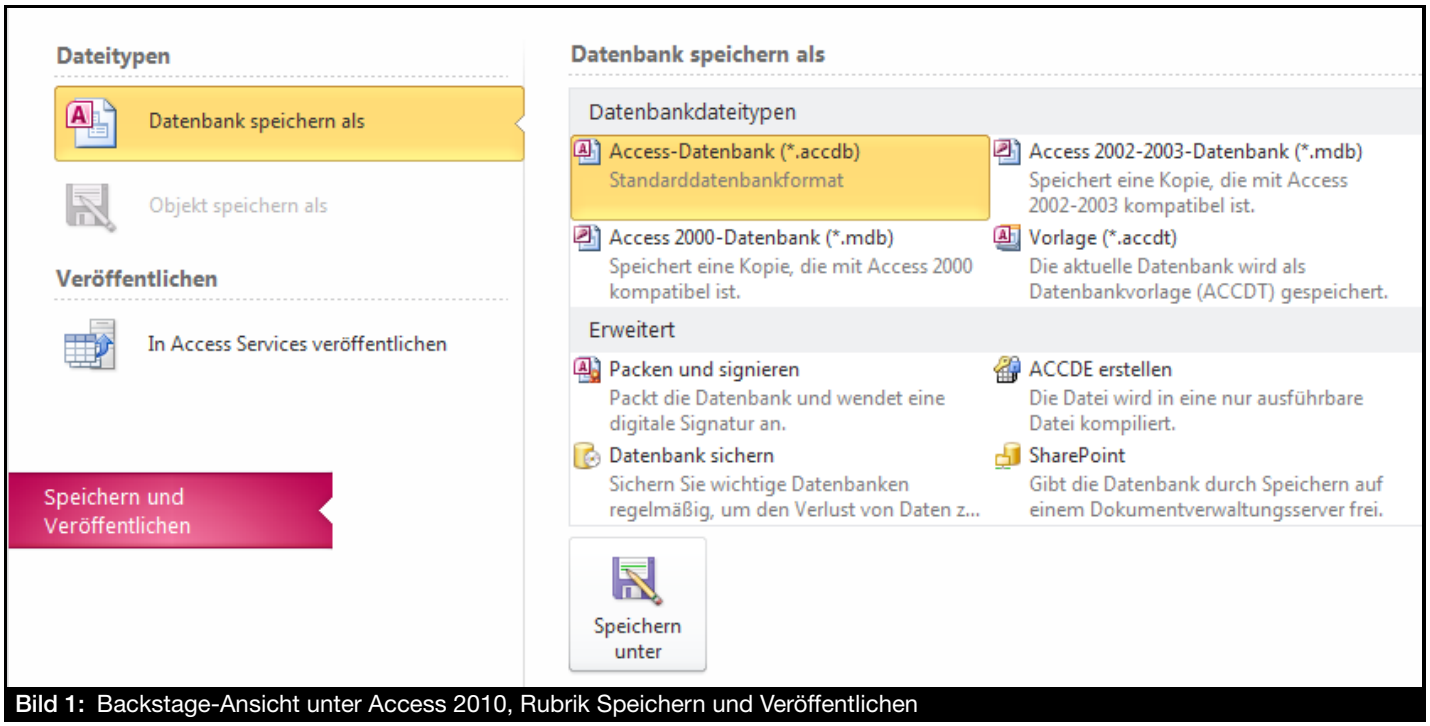


Bild 1: Backstage-Ansicht unter Access 2010, Rubrik Speichern und Veröffentlichen

Regel aus dem Originalnamen der Datenbank und einem Datum-String, der das aktuelle Datum enthält. Die Uhrzeit bleibt außen vor. Sichern Sie eine Datenbank mehrmals am Tag, so wird zusätzlich nach dem Datum automatisch ein Zähler angehängt. Das ist unproblematisch, da Sie die Erstellzeit ja aus dem Explorer ersehen können.

Access merkt sich nun, welchen Ordner Sie auswählten und schlägt diesen in der Folge immer wieder vor. Es handelt sich dabei um eine Eigenschaft der Datenbank, die in ihr automatisch benutzerdefiniert gesetzt wird:

```
? CurrentDb.Properties _
("DefaultBackupLocation")
```

Nach der Lektüre des Beitrags zu **DAO-TableDef**-Objekten dieser Ausgabe werden Sie wissen, dass sich diese Eigenschaft per Code auch ändern oder neu anlegen lässt. Sie ist nämlich nur dann vorhanden, wenn die Sicherung nicht am gleichen Ort erfolgt, in dem die Datenbank selbst liegt. Gleiches gilt für das übergeordnete Verzeichnis. Den Ort für die Sicherung können Sie also per VBA bestimmen, die Sicherung jedoch nicht programmatisch anstoßen, wie wir noch sehen werden.

Empfehlenswert ist jedenfalls die Anlage eines gesonderten Ordners für die Backups.

Wie in Bild 1 zu sehen, sieht Access 2010 noch weitere Funktionen im Backstage vor, die sich für ein Backup missbrauchen ließen.

Da gibt es einmal direkt die Anweisung **Datenbank speichern als**, welche ebenfalls einen Dateiauswahldialog hervorbringt, über den Sie die aktuelle Datenbank in einem Verzeichnis Ihrer Wahl kopieren können. Unter **Speichern und Veröffentlichen** findet sich diese Funktion abermals, zusätzlich aber auch die erwähnte Sicherungsfunktion und die Möglichkeit, die Datenbank in einem anderen Format abzuspeichern. Das ist für eine Sicherung wahrscheinlich nicht unbedingt sinnvoll, aber die Speicherung einer ACCDB- in eine MDB-Datenbank bringt unter Umständen auch Vorteile mit sich. So können Sie bei Parallelinstallationen auch von

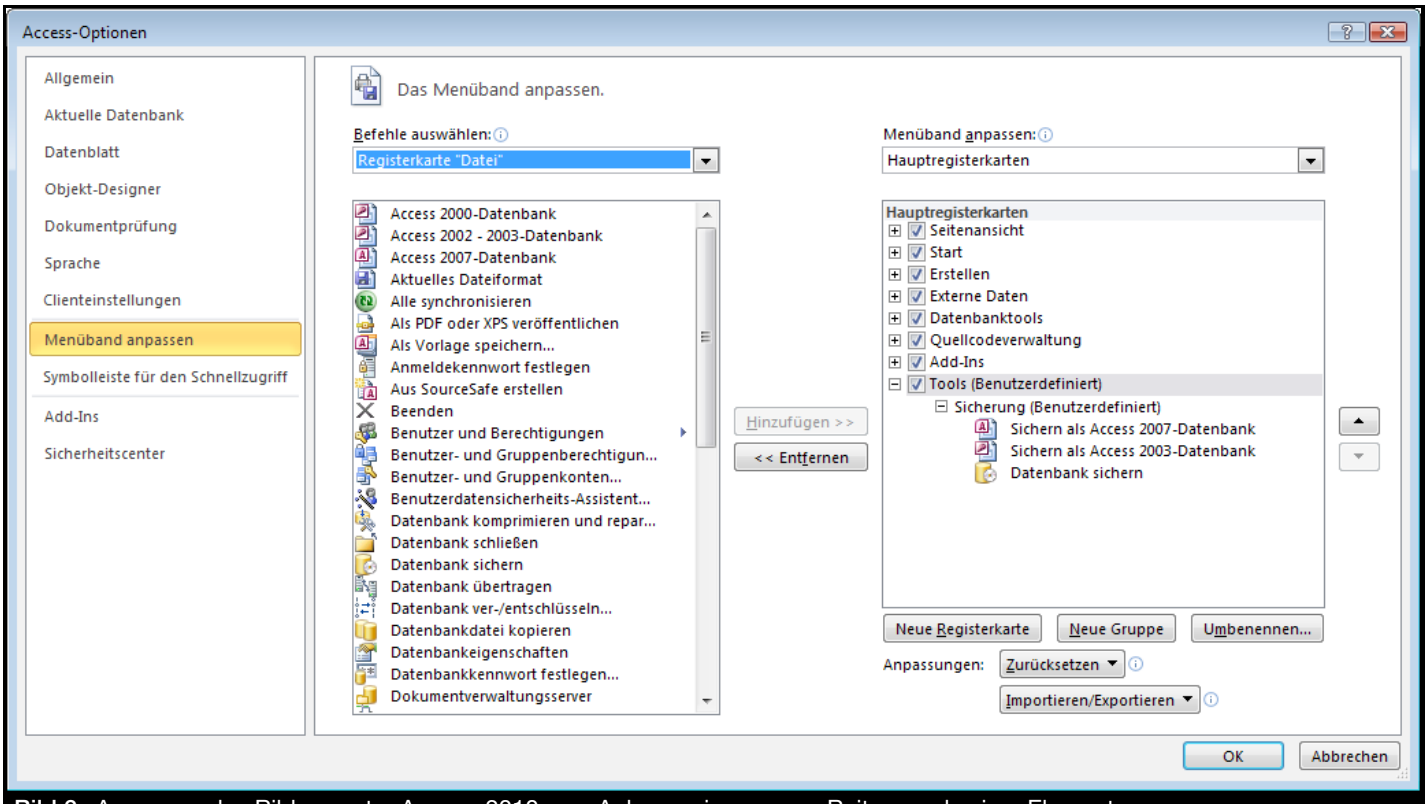


Bild 2: Anpassen des Ribbon unter Access 2010 zum Anlegen eines neuen Reiters und seiner Elemente

Access 2003 aus auf diese Backups zugreifen.

Wenn Sie hart an einer Neuentwicklung arbeiten, so ist ein Backup in Stundenabständen anzuraten. Dabei ist der Umstand, dass Sie immer den Weg über den schwerfälligen Backstage gehen müssen, etwas hinderlich. Aber stellen Sie sich doch einfach eine neue Gruppe von Sicherungsanweisungen im Ribbon zusammen, die Sie dann mit nur einem Klick ausführen können! Öffnen Sie dazu den Ribbon-Editor (Bild 2) über Rechtsklick auf den Ribbon (**Menüband anpassen...**) oder die Optionen von Access. Wählen Sie aus dem Kombinationsfeld

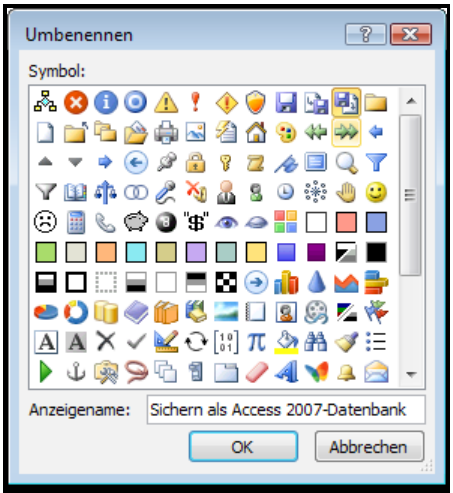


Bild 3: Umbenennen eines Eintrags und Auswahl eines Icons für die Anweisung

oben links die Anweisungsgruppe **Registerkarte "Datei"**. Hier finden Sie schnell jene Befehle, die auch im Backstage vorkommen. Klicken Sie auf den Button **Neue Registerkarte**. Es entsteht nun im Baum oben ein neuer Eintrag, den Sie erst mal etwa in **FileTools** umbenennen. Markieren Sie dazu den Eintrag und

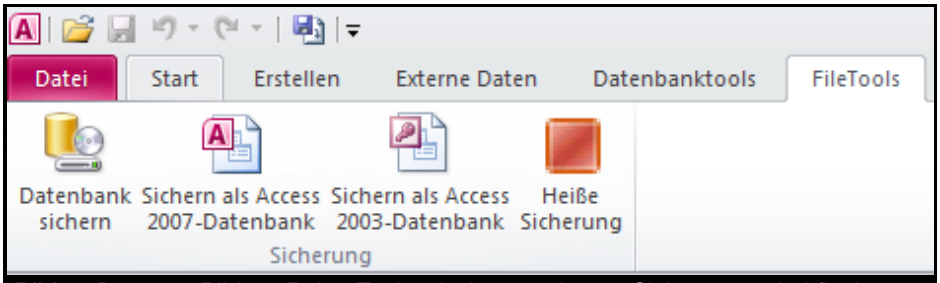


Bild 4: Der neue Ribbon-Reiter Tools mit den angelegten Sicherungsschaltflächen

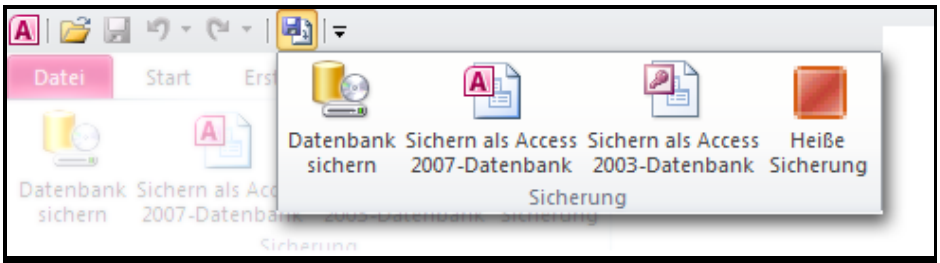


Bild 5: Die modifizierte Schnellzugriffsleiste mit den Sicherungsbefehlen

klicken auf den Button **Umbenennen...**. Wiederholen Sie dies für die ebenfalls neue Untergruppe, der Sie die Bezeichnung **Sicherung** geben. Hier allerdings öffnet sich nicht nur ein Texteingabedialog, sondern der in Bild 3 gezeigte, welcher außer der Benennung auch noch die Auswahl eines Icons für die Gruppe zulässt.

Im Folgenden markieren Sie aus der Liste der Befehle links nacheinander jene, die für die Sicherungen in Betracht kommen. Klicken Sie dabei jeweils auf den Button **Hinzufügen**, der den Eintrag in die markierte Gruppe **Sicherung** verfrachtet. Wenn Ihnen die von Access vorgesehene Bezeichnung, wie **Access 2002-2003-Datenbank**, nicht zusagt, so können Sie anschließend über die **Umbenennen**-Funktion der Anweisung einen anderen Namen und ein anderes Icon verabreichen, etwa **Sichern als Access 2003-Datenbank**.Schlussendlich lässt sich der neue Reiter auch noch über die Pfeil-Buttons nach oben oder unten verschieben, so dass er nach Beenden des Ribbon-Editors ganz rechts positioniert erscheint, wie in Bild 4. Was es mit dem Button für die **Heiße Sicherung** auf sich hat, welcher das Makro **makBackup** ausführt, kommt später noch zur Sprache.

Soll der Zugriff besonders schnell gehen, so legen Sie die Sicherungsanweisungen zusätzlich in der Schnellzugriffsleiste an; in Bild 4 ganz oben zu finden. Rechts steht dort ein neuer Auswahlbutton zur Verfügung, welcher, so betätigt, die Sicherungsgruppe (Bild 5) an die Oberfläche bringt.

Die Schnellzugriffsleiste lässt sich ähnlich einrichten, wie das Ribbon. Klicken Sie rechts auf sie und wählen den Eintrag **Symbolleiste für den Schnellzugriff anpassen...**. Der Editor für die Anpassungen (Bild 6) funktioniert weitgehend genau so, wie der Menüband-Editor. Statt nun aber wie

Tabellen und Abfragen über DAO verwalten

Der Umgang mit den Datensätzen einer Tabelle über das Recordset-Objekt der DAO-Bibliothek ist das A und O der Programmierung unter Access und dürfte Ihnen deshalb wahrscheinlich geläufig sein. Nicht selten aber wird auch der Zugriff auf die Eigenschaften der Tabellen und Abfragen selbst benötigt – und hier kommt abermals DAO mit seinen TableDef- und QueryDef-Objekten ins Spiel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1501_DAO.mdb**.

Tabellen und Abfragen im Objektmodell von DAO

Access legt beim Erstellen einer Datenbankdatei automatisch einen Verweis auf die **DAO-Bibliothek** im VBA-Projekt an, weil ohne sie keine Möglichkeit besteht, an die Datenobjekte heranzukommen. Zwar gibt es auch eine weitere Bibliothek namens **ADODB**, die ersatzweise verwendet werden kann, doch diese wird inzwischen von Microsoft weder weiterentwickelt, noch empfohlen. Sie war einst in erster Linie für den erweiterten Zugriff auf den SQL-Server gedacht, als Access 2000 mit den Access-Projekten (**ADP**) aufwartete. Doch da diese etwa von Access 2013 gar nicht mehr unterstützt werden, gibt es auch keinen triftigen Grund mehr, diese Bibliothek einzusetzen.

DAO als Abkürzung für **Data Access Objects** ist nur der Modellname der Bibliothek. In der Liste der Verweise finden Sie sie jedoch unter der monströsen Bezeichnung **Microsoft Office 14.0 Access database engine Object Library**, wobei sich das je nach Version von Access geringfügig unterscheidet. Vor Access 2007 lautete die Bezeichnung allerdings tatsächlich **Microsoft DAO 3.6 Object Library**.

Öffnen Sie im VBA-Editor über das Menü **Ansicht** den Objektkatalog und wählen aus dem Kombinationsfeld oben links den Eintrag **DAO** aus, wie in Bild 1. Der Katalog füllt sich dann mit allen Objektklassen, die die Datenzugriffsbibliothek aufweist, bei denen die Klasse **Database** von zentraler Bedeutung ist und quasi das Ebenbild der Datenbank ausmacht. Beim Klicken auf diese Klasse zeigen sich im rechten Abteil die Methoden und Eigenschaften des Database-Objekts. Da uns für das Thema dieses Beitrags nun besonders die Tabellen und Abfragen interessieren, die in einer Datenbank enthalten sind, müsste das **Database**-Objekt wohl irgendwelche Methoden aufweisen, um auf diese zugreifen zu können. Tatsächlich ist das mit den beiden Auflistungseigenschaften **TableDefs** und **QueryDefs** gegeben, die in Bild 1 markiert wurden.

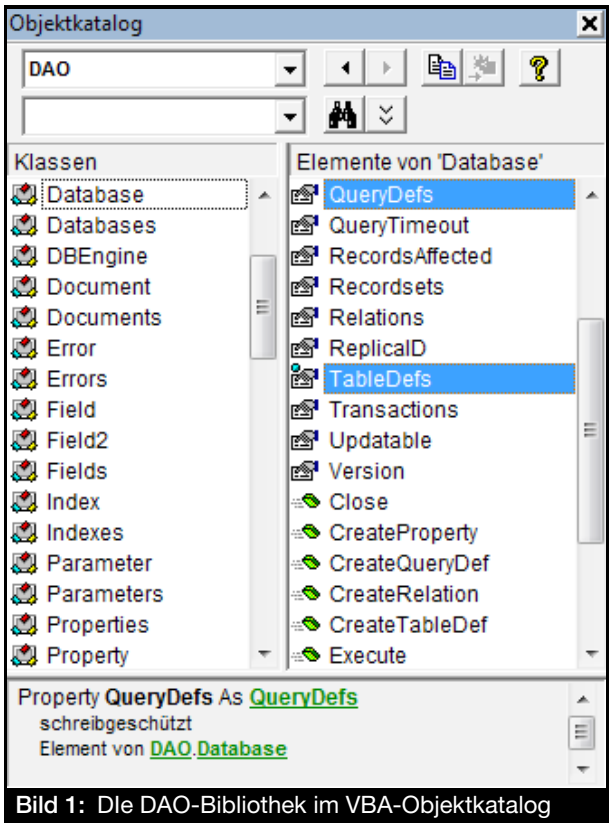


Bild 1: Die DAO-Bibliothek im VBA-Objektkatalog

Sie sind beide Container für die einzelnen Tabellen- und Abfragen-Objekte. Dabei steht **TableDef** für "Tabellendefinition" und **QueryDef** für "Abfragedefinition".

Nehmen wir uns zunächst die Aufgabe vor, über diese DAO-Klassen alle Tabellen aufzulisten, die die aktuelle Datenbank enthält. Als Erstes braucht es dafür ein **Database**-Objekt, von dem sich dann die **TableDefs**-Auflistung ableitet. Allerdings ist **Database** ja nur eine Modellklasse und noch kein Objekt – eine Klasse muss erst in Form einer Objektvariablen instanziiert werden. Und da die DAO-Bibliothek eine externe Komponente ist, die lediglich als Modell in das VBA-Projekt

eingeklinkt ist, weiß sie eigentlich auch gar nichts von der Access-Datenbank, in der sie untergebracht wurde. Es muss also ein Bezug zu einem **Database**-Objekt hergestellt werden, das Access bereitstellt. Das tut es auch mit der Eigenschaft **CurrentDb** des Access-Application-Objekts:

```
Dim dbs As Database
Set dbs = Application.CurrentDb
```

Das **Application** kann auch weggelassen werden, da es sich um eine globale Eigenschaft von Access handelt.

Auch das Übertragen auf eine Objektvariable **dbs** ist nicht zwingend notwendig, da **CurrentDb** jederzeit erneut abgefragt werden kann. Neben **CurrentDb** gibt es noch eine weitere Variante der Zuweisung:

```
Set dbs = Application.DbEngine(0)(0)
```

Die Nullen in Klammern sind eine zulässig abgekürzte Form von

```
Dbengine.Workspaces(0).Databases(0)
```

Nachdem nun ein gültiges **Database**-Objekt vorliegt, kann es auch an die Liste der Tabellen gehen:

```
? CurrentDb.TableDefs.Count
```

Das gibt die Anzahl der in der Datenbank enthaltenen Tabellen im Direktfenster aus. Analog geht das für die Abfragen mit

```
? CurrentDb.QueryDefs.Count
```

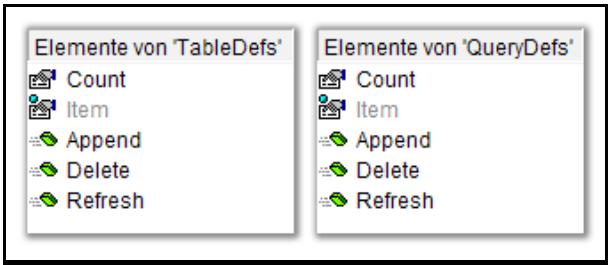


Bild 2: Methoden der TableDefs- und QueryDefs-Auflistungsklassen

Damit Sie nicht spekulieren müssen, wie mit diesen Auflistungen weiter verfahren wird, sind in Bild 2 deren Methoden aus dem Objektkatalog dargestellt.

Count ermittelt die Anzahl der jeweiligen Objekte. **Delete** löscht ein Element. Da ein **TableDef**-Objekt direkt an die zugehörige Tabelle gekoppelt ist, bedeutet Löschen hier, dass auch die Tabelle aus der Datenbank entfernt wird:

```
Currentdb.TablesDefs.Delete "Tabelle1"
```

Tabelle1 wandert damit ins Nirwana. **Append** würde eine Tabelle zur Datenbank hinzufügen. Das Vorgehen ist hier allerdings nicht so einfach, wie beim Löschen, denn **Append** erwartet ein fertiges **TableDef**-Objekt, das erst über die **Database**-Methode **CreateTableDef** erzeugt und mit allerlei weiteren Eigenschaften, wie Feldern, versehen sein muss. Tabellen über VBA anlegen zu müssen, dürfte jedoch ziemlich selten vorkommen, so dass dies hier nicht weiter besprochen wird.

Refresh ist eine Methode, die die Auflistung auf den neuesten Stand bringt. Löschen Sie etwa eine Tabelle per **Delete**, so findet sie sich zunächst trotzdem noch in der Auflistung. Erst ein **Refresh** aktualisiert die Liste wieder.

Item schließlich ist der Bezug zu einem einzelnen **TableDef**-Objekt, das über den Index der Auflistung angesprochen wird:

```
CurrentDb.TablesDefs.Item(0)
```

Hier erhält man das **TableDef**-Objekt, welches auf die erste Tabelle der Datenbank verweist. Weil **Item** die sogenannte **Default**-Eigenschaft der Klasse ist, erkennbar am blauen Punkt im Objektkatalog, kann der Methodenname auch weggelassen werden:

```
Dim tdf As TableDef
Set tdf = CurrentDb.TableDefs(0)
```

Außerdem kann auch direkt der Name des Objekts als Index statt der Ordinalzahl verwendet werden:

```
CurrentDb.TableDefs("Tabelle1")
```

Mit diesem Wissen könnten Sie bereits die Routine zum Ausgeben aller Tabellen der Datenbank anfertigen. Es muss nur die Zahl der **TableDefs** mit **Count** ermittelt und dann in einer Schleife über eine Zählervariable **n** jedes Objekt mit **Item(n)** angesprochen werden. Name ist die Eigenschaft des **TableDef**-Objekts, welches auch dem Namen der Tabelle entspricht.

Es geht aber noch einfacher. Weil die Auflistungen COM-Enumeration unterstützen, kann man auf das Ansprechen per Index verzichten und eine **For..Each**-Schleife einsetzen.

Der Beispielcode steht in Listing 1. Im VBA-Direktfenster wird erst die Zahl der Tabelle ausgegeben, danach jedes Element der Auflistung durchlaufen und der Name der entsprechenden Tabelle angezeigt.

Wie aus Bild 2 hervorgeht, gleichen sich die Auflistungen für **TableDefs** und **QueryDefs** vollständig. Deshalb ist der Code zur Ausgabe aller Abfragen ebenfalls identisch (Listing 2).

Das TableDef-Objekt

Hat man erst einmal ein **TableDef**-Objekt, so lässt sich damit natürlich noch mehr anstellen, als nur die Ausgabe des Tabellennamens. Eine Übersicht aller Methoden sowohl der **TableDef**- als auch der **QueryDef**-Klassen zeigt der kombinierte Screenshot des Objektkatalogs in Bild 3. Einige davon sind nur für sehr spezielle Aufgaben von Belang, etwa für Tabellen, die über **ODBC** in die Datenbank verknüpft sind, oder **PassThrough-Abfragen**, die ebenfalls auf einen SQL-Server aufsetzen. Gleiches gilt für Tabellen, die in Datenbanken mit **Replikation** vorkommen - teilweise Features, die die aktuellen Versionen von Access gar nicht mehr unterstützen! Diese Methoden sind der Kompatibilität wegen noch

```
Sub ListTables()
    Dim tdf As DAO.TableDef

    Debug.Print CurrentDb.TableDefs.Count & " Tabellen"
    For Each tdf In CurrentDb.TableDefs
        Debug.Print tdf.Name
    Next tdf

End Sub
```

Listing 1: Ausgeben aller Tabellen der Datenbank über die TableDefs-Auflistung

```
Sub ListQueries()
    Dim qdf As DAO.QueryDef

    Debug.Print CurrentDb.QueryDefs.Count & " Abfragen"
    For Each qdf In CurrentDb.QueryDefs
        Debug.Print qdf.Name
    Next qdf

End Sub
```

Listing 2: Ausgeben aller Abfragen der Datenbank über die QueryDefs-Auflistung

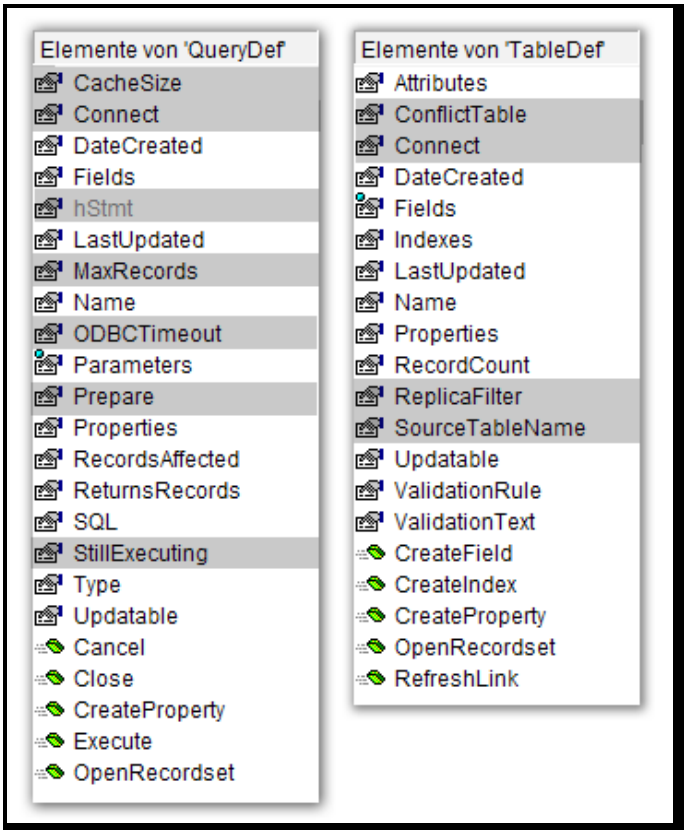


Bild 3: TableDef- und QueryDef-Methoden

im Objektmodell vorhanden, geben meist aber gar keine Werte mehr zurück. In Bild 3 sind alle Methoden, die Sie kaum interessieren dürften, grau hinterlegt.

Im Folgenden werden die relevantesten Methoden der **TableDef**-Klasse angeführt. Alles Weitere erfahren Sie in der Hilfe zu DAO, die allerdings leider nicht automatisch mit den neueren Access-Versionen installiert wird. Es gibt auch keine aktualisierte Hilfedatei zu DAO seit Access 2003. Deshalb gelangen Sie auch nicht zur Hilfe, wenn Sie etwa in Listing 2 den Ausdruck **QueryDef** markieren und **F1** drücken. Denn in das Objektmodell ist nach wie vor ein Bezug zur Hilfedatei **DAO360.chm** gespeichert, der ins Leere führt, wenn sich diese Datei nicht auf ihrem System befindet.

Wenn Sie die kontextsensitive Hilfe verwenden wollen, so besorgen Sie sich diese Hilfedatei im Internet und speichern sie im Windows-Verzeichnis im Unterordner **Help**. Ansonsten sind Sie darauf angewiesen, die Hilfe zu Access-VBA über das Menü des VBA-Editors aufzurufen und dann in der **Access Entwicklerreferenz** den Zweig **Microsoft-Datenzugriffsobjekte (DAO)**-

Referenz zu erforschen. Zurück zu den **TableDef**-Methoden:

DateCreated gibt zurück, wann eine Tabelle erstellt, **LastUpdated** den Zeitpunkt, wann sie zuletzt im Entwurf aktualisiert wurde. **Fields**, **Indexes** und **Properties** sind Auflistungseigenschaften, auf die wir noch gesondert zu sprechen kommen.

RecordCount ist eine interessante Eigenschaft: Sie zeigt die Zahl der in der Tabelle gespeicherten Datensätze an, soweit es sich um eine nichtverknüpfte Tabelle handelt.

Im Gegensatz zu den missverständlichen Erläuterungen der DAO-Hilfe stimmt diese Zahl immer, auch wenn nicht, wie dort angeführt, bereits erst auf alle Datensätze zugegriffen wurde. Dies gilt nur für ein Recordset-Objekt.

Updatable kennzeichnet, ob der Inhalt einer Tabelle schreibgeschützt ist, oder aktualisierbar. Für die Systemtabellen wird hier in der Regel ein **False** zurückgegeben.

ValidationRule und **ValidationText** sind Synonyme für die Gültigkeitsregel einer Tabelle und die gegebenenfalls ausgegebene Meldung. Das sind Eigenschaften, von denen selten Gebrauch gemacht wird, da Gültigkeitsbeschränkungen meist auf Feldebene definiert werden.

Die **Create**-Methoden korrelieren mit den entsprechenden Auflistungen zu Fields, Indexes und Properties. Hierüber können neue Felder in einer Tabelle erzeugt, neue Indizes oder neue benutzerdefinierte Eigenschaften hinzugefügt werden.

Connect und **RefreshLink** haben nur Bedeutung im Zusammenhang mit verknüpften Tabellen. In **Connect** wird der Pfad zu einer Backend-Datenbank angegeben und mit **RefreshLink** erst wirklich aktualisiert. Alle anderen Tabellen zeigen für **Connect** einen leeren String.

OpenRecordset öffnet eine Datensatzgruppe auf die Tabelle. Das dürfte Ihnen bekannt vorkommen. Hier handelt es sich um eine Alternative zur Syntax mit

```
CurrentDb.OpenRecordset("tblXY")
```

Dasselbe Ergebnis erhalten Sie mit

```
Dim tdf As TableDef
Set tdf = CurrentDb.TableDefs("tblXY")
tdf.OpenRecordset
```

Beide Varianten sind in Listing 3 aufgeführt. Im ersten Teil kommt die konventionelle Methode zum Einsatz, im zweiten das **TableDef**-Objekt. Eigentlich gibt es keinen Grund, zu dieser Variante zu greifen. Sie ist hier aber aufgeführt, weil sie so schön zwei VBA-Spezialitäten demonstriert: Einmal die **With-Anweisung** – in dieser Ausgabe bereits besprochen – und dann die Gültigkeit eines Objekts. Machen Sie den Versuch und schreiben zum Setzen der **TableDef**-Variablen diesen Code:

```
Set tdf = _
    CurrentDb.TableDefs("tblLaender")
Set rs = tdf.OpenRecordset()
```

Während die erste Zeile noch anstandslos ausgeführt wird, kommt es bei der zweiten Zeile zu der Fehlermeldung, dass ein Objekt nicht mehr festgelegt sei. Dass die **tdf**-Variable nicht der Übeltäter sein kann, erfah-

```
Sub OpenRecordSetTDF()
    Dim rs As DAO.Recordset
    Dim tdf As TableDef

    Set rs = CurrentDb.OpenRecordset("tblLaender", dbOpenDynaset)
    Do While Not rs.EOF
        Debug.Print rs(0).Value, rs(1).Value
        rs.MoveNext
    Loop
    rs.Close

    With CurrentDb
        Set tdf = .TableDefs!tblLaender
        Set rs = tdf.OpenRecordset(dbOpenDynaset)
    End With
    Do While Not rs.EOF
        Debug.Print rs(0).Value, rs(1).Value
        rs.MoveNext
    Loop
    rs.Close
End Sub
```

Listing 3: Öffnen eines Recordsets auf ein TableDef-Objekt zum Durchlaufen der Datensätze der Tabelle tblLaender

ren Sie über den Test im Direktfenster, nachdem Sie einen Haltepunkt auf die Zeile setzten:

```
? tdf.Name
```

gibt korrekt **tblLaender** zurück. Was aber sollte dann nicht festgelegt sein? Es bleibt nur noch das **CurrentDb**-Objekt. Tatsächlich handelt es sich bei diesem um ein nichtpersistentes Objekt, welches nur eine Momentaufnahme der aktuellen Datenbank versinnbildlicht. In der zweiten Code-Zeile ist es sozusagen bereits wieder verschwunden. Die übliche Lösung für diesen Umstand besteht im zusätzlichen Anlegen einer **Database**-Variablen:

```
Dim dbs As Database
Set dbs = CurrentDb
Set tdf = dbs.TableDefs("tblLaender")
```

Der Clou jedoch ist, dass dies mit der **With-Anweisung** umgangen werden kann. Da, wie im Beitrag zu **With** in dieser Ausgabe bereits erläutert, der Bezug zur mit **With** gesetzten Objektvariablen aufrechterhalten wird, bis ein **End With** auftritt, funktioniert die Zuweisung an **tdf** auch in Listing 3 – **CurrentDb** bleibt bis zum **End With** gültig. Das ist ein netter Beleg für den sinnvollen Einsatz der **With**-Anweisung.

Zur **TableDef**-Eigenschaft **Attributes** später mehr.

Das QueryDef-Objekt

In vielerlei Hinsicht ähneln sich **TableDef**- und **QueryDef**-Objekt.

```
Sub FindQuery()
    Dim qdf As DAO.QueryDef
    Const Expr = "tblOrte"
    For Each qdf In CurrentDb.QueryDefs
        If InStr(1, qdf.SQL, Expr) > 0 Then
            Debug.Print qdf.Name
            Debug.Print qdf.SQL
            Debug.Print
        End If
    Next qdf
End Sub
```

Listing 4: Ermitteln aller Abfragen der Datenbank, die den Tabellenverweis tblOrte enthalten

Dennoch ist eine Abfrage nun mal keine Tabelle. Die wichtigsten abweichenden Methoden der Klasse im Überblick:

SQL ist sicher die Eigenschaft, welche im Zentrum steht. Mit ihr lässt sich der SQL-Ausdruck einer Abfrage sowohl auslesen, wie auch bestimmen. Ein Beispiel, wofür sich der lesende Zugriff eignen kann, steht in Listing 4. In einer Schleife werden hier alle Abfragen der Datenbank durchlaufen, der SQL-String ausgelesen und dabei über **Instr()** überprüft, ob der Ausdruck **tblOrte** in ihm vorkommt, was darauf schließen lässt, dass diese Tabelle als Quelle in der Abfrage verwendet wird. Falls ja, so werden Name und SQL-String im Direktfenster ausgegeben. In umfangreichen Datenbanken ist so eine Routine etwa nützlich, wenn eine Tabelle umbenannt wird, die auf ihr aufbauenden Abfragen nun aber ebenfalls modifiziert werden müssen. Schnell haben Sie so aus Dutzenden von Abfragen ermittelt, wo Sie ansetzen müssen. Die Routine können Sie leicht so umbauen, dass statt der Ausgabe im Direktfenster der SQL-String der Abfrage automatisch ersetzt wird:

```
qdf.SQL = Replace(qdf.SQL, _
 "tblOrte","tblAndernorts")
```

Mit wenigen Zeilen Code ersparen Sie sich also eine Menge manueller Arbeit.

Execute ist eine Methode des **QueryDef**-Objekts, die nur für Aktionsabfragen gültig ist. Sie führen sie damit aus. (Auswahlabfragen können nicht direkt ausgeführt werden, sondern dienen als Vorlage für ein zu öffnendes Recordset.) Beliebtes Beispiel wäre das Leeren einer Tabelle über die **Delete**-Anweisung von SQL:

```
qdf.SQL = "DELETE * FROM tblOrte"
qdf.Execute
```

Aktualisierungs-, Definitions- und Anfügeabfragen sind weitere Typen, die die **Execute**-Methode erlauben.

Hier kommt eine weitere Eigenschaft des **QueryDef**-Objekts ins Spiel. Mit **ReturnsRecords** können Sie ermitteln, ob eine Abfrage Datensätze zurückgibt. Sobald Sie den SQL-String der Abfrage gesetzt haben, analysiert die **Access Database Engine** den Ausdruck und entscheidet, ob es sich um eine Auswahlabfrage handelt. In diesem Fall gibt **ReturnsRecords** den Wert **True** zurück. Sonst handelt es sich um eine Aktionsabfrage.

Bei letzterer lässt sich die Eigenschaft **RecordsAffected** verwenden, um auszulesen, wie viele Datensätze von der Aktion betroffen waren. Nach **Execute** ist hier ein Wert gesetzt.

Die VBA-With-Anweisung

Unter den Elementen, die den Programm-Code unter VBA strukturierter erscheinen lassen, findet die With-Anweisung oft wenig Berücksichtigung. Dabei spart man mit ihr nicht nur Programmtext ein, sondern erleichtert sich über IntelliSense zusätzlich das Schreiben von Objektmethoden.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1501_WithAnweisung.mdb**.

Grundlegende Syntax

Die **With**-Anweisung stellt einen Bezug zu einer Variablen her, der entweder ein benutzerdefinierter Datentyp (**UDT**) zugrunde liegt, oder ein Objekt. Ob diese in ihren eigenen Modulen deklariert sind, oder in einer Verweisbibliothek, spielt dabei keine Rolle. Die Blockanweisung besteht aus zwei Elementen, dem Blockstart und dem Blockende:

```
Dim Variable As ...
With [Variable]  'Blockstart
.[Methode oder]
.[Eigenschaft] = ...
End With  'Blockende
```

Nach dem reservierten Wort **With** folgt der Name einer Variablen, auf die sich die Methoden, Eigenschaften oder Typelemente im Folgenden beziehen sollen.

Wird in einer Prozedur die Variable nicht mehr weiter benötigt, so schließt man den Bezug mit der Anweisung **End With**.

With im Einsatz

Einen Code aus dem wahren Datenbankprogrammiererleben finden Sie im Listing 1 der Beispieldatenbank, wo auf eine Tabelle **tblTest** eine Datensatzgruppe (**Recordset**) geöffnet wird, um sie mit Datensätzen zu füllen und jene anschließend im VBA-Direktfenster auszugeben.

```
Sub OhneWITH()
    CurrentDb.Execute "DELETE * FROM tblTest"
    Set rsTest = CurrentDb.OpenRecordset("SELECT * FROM tblTest", dbOpenDynaset)
    For i = 1 To 100
        rsTest.AddNew
        rsTest!ID = i
        rsTest!Code = Chr$(65 + 28 * Rnd) & Chr$(65 + 28 * Rnd) & Chr$(65 + 28 * Rnd)
        rsTest.Update
    Next i

    rsTest.MoveFirst
    Do While Not rsTest.EOF
        Debug.Print rsTest!ID, rsTest!Code
        rsTest.MoveNext
    Loop
    rsTest.Close
End Sub
```

Listing 1: Durchlaufen der Datensätze einer Tabelle

```
Sub MitWITH()
    CurrentDb.Execute "DELETE * FROM tblTest"
    Set rsTest = CurrentDb.OpenRecordset("SELECT * FROM tblTest", dbOpenDynaset)
    With rsTest
        For i = 1 To 100
            .AddNew
            !ID = i
            !Code = Chr$(65 + 28 * Rnd) & Chr$(65 + 28 * Rnd) & Chr$(65 + 28 * Rnd)
            .Update
        Next i

        .MoveFirst
        Do While Not .EOF
            Debug.Print !ID, !Code
            .MoveNext
        Loop
        .Close
    End With
End Sub
```

Listing 2: Der gleiche Code unter Einsatz der With-Anweisung

In der ersten Zeile der Routine wird die Tabelle zunächst mit einer SQL-Anweisung per **Database.Execute** geleert. Dann erfolgt das Setzen des Recordsets **rsTest** auf die Tabelle.

Die Recordset-Variable **rsTest** wurde übrigens bereits im Modulkopf deklariert, ebenso die Zählervariable **i**:

```
Private rsTest As DAO.Recordset
Private i As Long
```

Im weiteren Verlauf werden in einer Schleife auf den Zähler **i** der Tabelle hundert Datensätze hinzugefügt, wobei für das Feld **Code** jeweils ein dreistelliger zufälliger Buchstabencode generiert wird. Zum Schluss werden alle Datensatzinhalte wieder in einer Schleife über **Debug.Print** in das VBA-Direktfenster ausgegeben. Das Ergebnis wäre etwa dieses:

```
 1            J\Q
 2            PEZ
 3            BWU
 4            H[N
 5            ULG
...
```

Auffallend ist, dass der Variablenname **rsTest** sehr häufig in der Routine auftaucht. Genau dies lässt sich mit der **With**-Anweisung umgehen, indem zum Blockstart der Verweis auf **rsTest** gesetzt wird und im Folgenden darauf nur noch indirekt Bezug genommen wird. Der abgewandelte Code steht in Listing 2.

Statt **rsTest.AddNew** zum Anlegen eines neuen Datensatzes wird nun nur noch der Ausdruck **.AddNew** benötigt – VBA denkt sich vor den Punkt den oben definierten Objektbezug.

Statt des Punktes ist auch der VBA-Ausrufezeichen-Operator für Auflistungen erlaubt. Deshalb lässt sich nun ein Feld des Recordsets statt über **rsTest!Code** einfach mit **!Code** ansprechen.

Kürzer wird die Routine damit nicht. Sie ist im Gegenteil wegen der **With**-Anweisungen sogar um zwei Zeilen länger. Der Zeichenumfang ist jedoch geringer. Das spielt allerdings weniger eine Rolle als die nun erhöhte Übersichtlichkeit des Codes. Die Methoden, Eigenschaften und Feldbezeichnungen des Recordsets treten nun optisch deutlicher zutage, weil sie isolierter dastehen.

Aber nicht nur die Gestalt des Programmcodes ändert sich damit, sondern auch die Art, wie VBA das Modul kompiliert. Denn bei jedem Ansprechen der Variablen **rsTest** in der ersten Variante sucht es sich gesondert einen Zeiger auf die Objektvariable, um dann deren Methoden auszuführen. Bei **With**-Blöcken wird intern nur ein Zeiger auf die Objektvariable angelegt und dann für die Methodenaufrufe mehrfach verwendet. Das führt zu einer geringfügig besseren Ausführungsgeschwindigkeit der Routine. Ehemals wurde die **With**-Anweisung deshalb auch in Sammlungen zu Performance-Tipps aufgenommen, spielt aber in dieser Hinsicht heute keine nennenswerte Rolle mehr – ein paar Nanosekunden mehr oder weniger sind unerheblich.

Objektvariablen überflüssig machen

Die Variante in Listing 1 benötigt für das Ansprechen der Recordset-Funktionen die Objektvariable **rsTest**. Durch die **With**-Anweisung können Sie darauf aber auch verzichten, und damit tatsächlich Zeilen einsparen, wie Listing 3 zeigt. Hier wird in der zweiten Zeile die **With**-Anweisung statt mit einer Variablen mit einem Ausdruck bestückt, dem Öffnen des Recordsets über **OpenRecordset**. Da das Ergebnis dieses Aufrufs einer Variablen gleichkommt, funktioniert dieser Bezug ebenfalls. Die Performance ist sogar nochmals um Micro-Prozent höher. Ob der Verzicht

```
Sub MitWITHKurz()
    CurrentDb.Execute "DELETE * FROM tblTest"
    With CurrentDb.OpenRecordset( _
        "SELECT * FROM tblTest", dbOpenDynaset)
        For i = 1 To 100
            .AddNew
            !ID = i
            !Code = Chr$(65 + 28 * Rnd) & Chr$(65 + 28 * Rnd) & Chr$(65 + 28 * Rnd)
            .Update
        Next i

        .MoveFirst
        Do While Not .EOF
            Debug.Print !ID, !Code
            .MoveNext
        Loop
        .Close
    End With
End Sub
```

Listing 3: Gekürzte Fassung ohne Recordset-Variable

auf die Objektvariable jedoch der Code-Verständlichkeit zugutekommt, ist hingegen eher fraglich.

With verschachteln

Es lassen sich **With**-Blöcke auch ineinander einfügen. Im Prinzip sieht das dann so aus:

```
With [Var1]  'Blockstart 1
  .[Methoden auf Var1]
  With [Var2]  'Blockstart 2
    .[Methoden auf Var2]
  End With  'Blockende 2
 .[Methoden auf Var1]
End With  'Blockende 1
```

Der Punkt-Operator von Objektmethoden wirkt hier immer nur auf den zuvor definierten **With**-Block. Ein **End With** setzt den Objektbezug wieder auf den äußeren Block, im Beispiel also auf **Var1**. Vermischen lassen sich beide nicht.

Für das vorangegangene Beispiel lässt sich eine solche Verschachtelung realisieren. Um den Bezug auf das Recordset **rsTest** wird in Listing 4 noch ein Bezug auf das **Database**-Objekt (**CurrentDb**) gebaut. Die **OpenRecordset**-Methode etwa bezieht sich dann auf **CurrentDb**, das **AddNew** auf **rsTest**. Innerhalb von Block 2 wiederum kann kein Bezug zu **CurrentDb** hergestellt werden.

Der Vollständigkeit halber ist in Listing 5 noch die gekürzte Fassung dargestellt, welche komplett ohne die Definition von Objektvariablen auskommt.

An diesen zwei Beispielen wird auch besonders klar, warum der Inhalt der **With**-Blöcke zusätzlich eingerückt werden sollte. Dadurch ist immer auf den ersten Blick zu erkennen, auf welches Objekt sich die Punkt-Methoden richten. Die Struktur der Routinen wird durch die **With**-Blöcke also deutlicher.

```
Sub MitDoppelWITH()
    With CurrentDb   'Block 1
        .Execute "DELETE * FROM tblTest"
        Set rsTest = .OpenRecordset( _
            "SELECT * FROM tblTest", dbOpenDynaset)
        With rsTest    'Block 2
            For i = 1 To 100
                .AddNew
                !ID = i
                !Code = Chr$(65 + 28 * Rnd) & Chr$(65 + 28 * Rnd) & Chr$(65 + 28 * Rnd)
                .Update
            Next i

            .MoveFirst
            Do While Not .EOF
                Debug.Print !ID, !Code
                .MoveNext
            Loop
            .Close
        End With    'Ende Block 2
    End With    'Ende Block 1
End Sub
```

Listing 4: With-Blöcke lassen sich ineinander verschachteln

```
Sub MitDoppelWITHKurz()
    With CurrentDb
        .Execute "DELETE * FROM tblTest"
        With .OpenRecordset( _
            "SELECT * FROM tblTest", dbOpenDynaset)
            For i = 1 To 100
                .AddNew
                !ID = i
                !Code = Chr$(65 + 28 * Rnd) & Chr$(65 + 28 * Rnd) & Chr$(65 + 28 * Rnd)
                .Update
            Next i

            .MoveFirst
            Do While Not .EOF
                Debug.Print !ID, !Code
                .MoveNext
            Loop
            .Close
        End With
    End With
End Sub
```

Listing 5: Derselbe Code in gekürzter Fassung ohne Objektvariablen

With auf benutzerdefinierte Typen

Imgrunde ist ein Objekt auch nur eine spezielle Art eines benutzerdefinierten Typs (**UDT** = **User Defined Type**). Also lässt sich die **With**-Anweisung auch auf solche Typvariablen setzen.

Zippen mit Bordmitteln

Das VBA-gesteuerte Packen und Archivieren von Dateien ist eine Aufgabe, der auch Access-Entwickler immer wieder begegnen. Dass man dabei keineswegs auf externe Komponenten oder Hilfsmittel angewiesen ist und mit Windows-Bordmittel auskommen kann, soll dieser Beitrag zeigen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1502_Zippen.mdb**.

Zip-Format

Es gibt zweifellos Alternativen zum altehrwürdigen Zip-Format, denn **RAR** oder **7z** etwa bieten erheblich bessere Kompressionsraten an. Doch diese Formate setzen voraus, dass auf dem System ein entsprechendes Archivierungsprogramm installiert ist, und bei Weitergabe einer solchen Archivdatei können Sie nicht einfach davon ausgehen, dass dies beim Adressaten der Fall ist. Mit dem Zip-Format sind Sie auf der sicheren Seite: Windows hat seit einigen Generationen einen entsprechenden Support eingebaut. Über den Explorer lassen sich Zip-Dateien wie Ordner behandeln, einsehen und bearbeiten. Und was sich manuell erledigen lässt, sollte sich über irgendeinen Weg auch programmtechnisch realisieren lassen.

Shell-Funktionen

Die übliche Herangehensweise, wenn Funktionen von Windows gefragt sind, ist die Suche im API-Katalog. In unserem Fall kann darauf verzichtet werden, da Windows mit der **Shell**-Bibliothek eine COM-Schnittstelle zur Verfügung stellt, die alles Nötige mitbringt. Es handelt sich um die **shell32.dll** im Systemverzeichnis, welche sich in der Liste der Verweise von VBA namentlich als **Microsoft Shell Controls And Automation** ausgibt. Nachdem Sie die Bibliothek ihrem Projekt über den Verweisedialog hinzugefügt haben, finden Sie deren Klassen im Objektkatalog, wenn Sie links oben den Eintrag **Shell32** auswählen (Bild 1). An den Klassenbezeichnungen, wie **Folder**, **FolderItem** und **ShellFolderView**, lässt sich bereits ablesen, dass es hier zuvörderst um Verzeichnisse und deren Inhalte geht. Da die Shell, wie erwähnt, eine Zip-Datei als Verzeichnis ansieht, wäre nun zu erläutern, wie man so ein Verzeichnis anspricht. Dazu gehen Sie vom Hauptobjekt der Bibliothek aus, der **Shell**-Klasse, welche in der Abbildung bereits markiert ist.

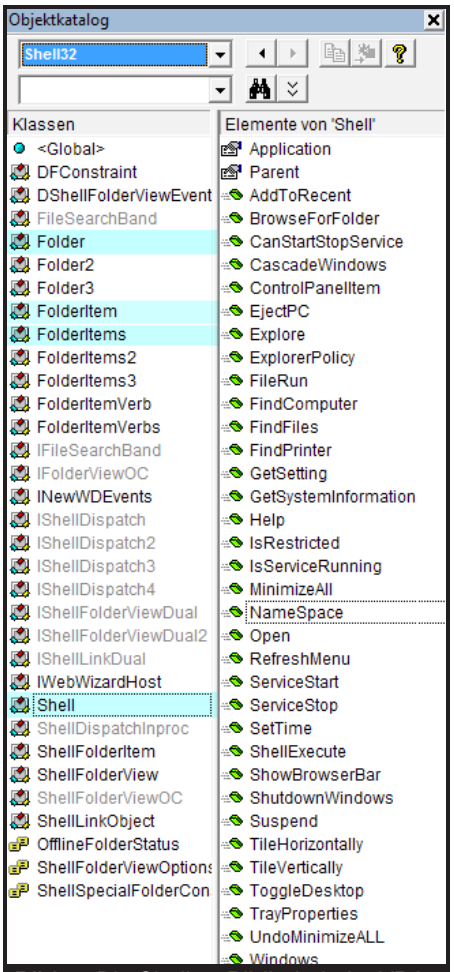


Bild 1: Die Shell32-Bibliothek, im VBA-Objektkatalog dargestellt

Eine Instanz dieser Klasse erhalten Sie einfach über die **New**-Zuweisung an eine Objektvariable:

```
Dim objShell As New Shell32.Shell
```

Hier sollte nicht unerwähnt bleiben, dass die ganze Angelegenheit auch komplett ohne einen Bibliotheksverweis auskäme. Folgender Ausdruck erzeugt nämlich ebenfalls das Shell-Objekt:

```
Dim objShell As Object
Set objShell = _
    CreateObject("Shell.Application")
```

Haben Sie nun das Hauptobjekt, so können Sie sogleich allerlei Methoden desselben ansprechen. So öffnet etwa die Methode **objShell.FileRun** den **Ausführen**-Dialog von Windows, oder **TrayProperties** den Dialog zu Einstellungen der Taskleiste. Warum Microsoft ausgerechnet solche Funktionen in die COM-Schnittstelle verbaute, ist nicht recht nachvollzieh-

bar, da man sie programmgesteuert wohl höchst selten aufrufen wird. Sie interessieren uns auch nicht weiter. Lediglich die Funktion **NameSpace** ist für unsere Belange wichtig: sie allein gibt ein **Folder**-Objekt, also ein Verzeichnis, zurück. Dazu übergibt man ihr als Parameter den Verzeichnisnamen:

```
Dim objFolder As Shell32.Folder
Set objFolder = objShell.NameSpace ( _
    "c:\windows")
```

Auf den Methodennamen **NameSpace** würde man wahrscheinlich nicht kommen und eher so etwas wie **GetFolder** erwarten. Grund für die Bezeichnung ist, dass sie eben nicht nur physische Verzeichnisse ermittelt, sondern auch virtuelle, wie dies beim Zip-Archiv vorliegt. Statt des Verzeichnispfads nimmt sie etwa auch Zahlen entgegen. Versuchen Sie dies:

```
? objShell.NameSpace(0).Title
```

Ergebnis wäre der String **Desktop**. (**Title** ist die Methode zum Auslesen der Bezeichnung eines Folder-Objekts.) Der komplette Pfadname ist etwas umständlicher zu ermitteln:

```
? objShell.NameSpace(0).Self.Path
```

Das ergibt etwa **C:\Users\André\Desktop**. Sie können nun mit den Zahlenparametern experimentieren. Übergeben Sie die 3, so erhalten Sie als Bezeichnung **Systemsteuerung**. Tatsächlich lässt sich die Systemsteuerung ja im Explorer-Baum anzeigen. Was aber ist deren Pfad? Sie erhalten den Ausdruck

```
::{21EC2020-3AEA-1069-A2DD-
08002B30309D}
```

Diese **GUID** symbolisiert einen virtuellen Ordner. Machen Sie die Probe aufs Exempel und setzen diese GUID als String für den Pfadnamen ein:

```
? objShell.NameSpace("::{21EC2020-
3AEA-1069-A2DD-08002B30309D}").Title
```

Auch das geht und liefert uns abermals die Bezeichnung **Systemsteuerung**.

Zusammengefasst also nimmt **NameSpace** als Parameter sowohl Verzeichnis-Strings entgegen, wie auch Zahlenwerte (die sogenannten **KnownFolder-IDs**) und schließlich GUIDs (**KnownFolder-CLSIDs**) als Strings.

Wenden wir uns nach diesem Ausflug in den Shell-Namespace wieder den Zip-Archiven zu. Sie können sicher erraten, wie eine Zip-Datei als **Folder**-Objekt zu erhalten ist: Sie übergeben einfach deren Dateinamen. Ein Beispiel:

```
Set objFolder = objShell.NameSpace ( _
    "d:\beispiele\test.zip")
```

Als nächstes interessiert uns natürlich der Inhalt eines Ordners – in diesem Fall die enthaltenen Dateien. Die **Folder**-Klasse sieht dafür die Auflistungs-Eigenschaft **Items** vor, welche sich über eine **For-Next**-Schleife durchlaufen lässt. Jedes Element der **Items**-Auflistung ist vom Typ **FolderItem**. Möchten Sie die Namen der Dateien im Ordner ausgeben, so käme etwa diese Routine infrage:

```
Dim objItm As FolderItem
For Each objItm in objFolder.Items
    Debug.Print objItm.Name
Next objItm
```

Auf diese Weise können Sie die in einem Zip-Archiv befindlichen Dateien inspizieren. Dabei sollte noch erwähnt werden, dass das Archiv wiederum Ordner enthalten kann. Nur dann weist die Eigenschaft **IsFolder** des entsprechenden FolderItems den Wert **True** auf, und die Eigenschaft **Type** gibt einen Text aus, der genau der Bezeichnung im Explorer entspricht. Je nach System kann diese Bezeichnung allerdings unterschiedlich sein. Ist etwa ein Packprogramm installiert, so könnte die Bezeichnung **WinRAR ZIP-Archiv** lauten.

FolderItems extrahieren

Wie jedoch lassen sich die Dateien oder Ordner des Archivs nun entpacken? Im Objektkatalog findet man weder zum **FolderItem**, noch zum **Folder**-Objekt Funktionsnamen, wie **Extract** oder **Uncompress**, die dafür dienlich sein könnten. Das aber ist einleuchtend, da die Shell das Archiv als normalen Ordner ansieht und somit lediglich jene Methoden anbietet, die auch für Dateien eines Verzeichnisses gälten. Im Explorer können Sie eine Datei aus dem Archiv extrahieren, indem Sie sie entweder per Drag&Drop verschieben, oder indem Sie sie kopieren und in einen anderen Ordner einfügen. Genau das geht auch hier über die **CopyHere**-Funktion eines **Folder**-Objekts. Diese Methode erwartet als Parameter ein **Folder**- oder **FolderItem**-Objekt. Der Vorgang hat folgendermaßen auszusehen:

Sie setzen zuerst ein Folder-Objekt auf das Zielverzeichnis. Dann benutzen Sie dessen **CopyHere**-Methode, um in dieses Verzeichnis Dateien oder Ordner zu kopieren, wobei diese in Form von zuvor erzeugten **FolderItems** daher kommen müssen.

Machen wir den Test und versuchen das Archiv **test.zip** in einen Ordner **d:\testordner** zu extrahieren:

```
Dim objZiel As Folder
Set objZiel = objShell.NameSpace( _
    "d:\testordner")
objZiel.CopyHere objZipFolder
```

Leider wird das Archiv damit nicht entpackt, sondern nur eine Kopie der Zip-Datei angelegt – eigentlich klar, da das Objekt **objZipFolder** ja das Archiv selbst kennzeichnet, nicht dessen Inhalt. Folglich muss hier die oben angeführte Routine zum Auslesen der **FolderItems** modifiziert werden:

```
Dim objItm As FolderItem
For Each objItm in objFolder.Items
    objZiel.CopyHere objItm
    DoEvents
Next objItm
```

Und tatsächlich: Im Zielordner laden damit die Dateien des Archivs! Sollte das Archiv Unterordner mit Dateien enthalten, so werden auch diese alle anstandslos entsprechend der Struktur angelegt. Es ist also nicht etwa nötig, diese Unterordner und Dateien in der Routine alle einzeln anzusprechen.

Wichtig ist auch die Zeile mit der **DoEvents**-Anweisung. Da der Shell-Prozess asynchron arbeitet, ist er unter Umständen noch mit dem Kopieren eines Items beschäftigt, während VBA bereits mit dem Code fortfährt – die **CopyHere**-Methode wartet nicht auf den Vollzug! Es hat sich gezeigt, dass ohne ein DoEvents manchmal Dateien unterschlagen werden.

Bei umfangreicheren Archiven werden Sie feststellen, dass sich beim Ausführen der Routine automatisch der Windows-Fortschrittsdialog öffnet. Das **Shell**-Objekt verhält sich also exakt gleich, wie beim manuellen Kopieren von Dateien im Explorer. In Maßen lässt sich das Verhalten jedoch über einen zweiten optionalen Parameter von **CopyHere** steuern, den wir bisher unterschlagen haben. Mit **Options** kann man eine Kombination von Steuerkonstanten übergeben, die Sie im Modul **mdlZipShell** der Beispieldatenbank kommentiert als Enumeration **eFileOp** finden. So bewirkt etwa die Angabe von **eOpNoProgress**, dass sich eben kein Fortschrittsdialog öffnet. Oder **eOpAllowUndo** weist an, dass die Operation wieder rückgängig gemacht werden kann. In unserem Fall hieße das, dass Sie im Explorer dann das Rückgängig-Symbol betätigen könnten, wodurch sich das Zielverzeichnis wieder leeren würde. Interessant ist auch die Konstante **eOpNewName**. Ohne deren Angabe würde eine existierende Datei **mein.txt** im Zielverzeichnis nicht überschrieben, sondern automatisch mit einem neuen Namen kopiert; also etwa in **Kopie von mein.txt** oder **mein(1).txt**.

Die komplette Routine zum Entpacken eines Zip-Archivs finden Sie in der Beispieldatenbank unter der Funktion **UnZipFile**. Sie übergeben ihr einfach den Pfadnamen der Zip-Datei und das Zielverzeichnis. Bei erfolgtem Entpacken gibt die Funktion ein **True** zurück. Wie Sie sehen, kommt man über die Shell-Bibliothek mit nur wenigen Code-Zeilen zum Ergebnis, und externe Zusatzkomponenten sind überflüssig.

Zip-Archive erzeugen

Zwar gibt uns Windows die Möglichkeit, Zip-Dateien zu entpacken, aber das Anlegen solcher scheint nicht vorgesehen zu sein. Ein Rechtsklick auf den freien Bereich eines Ordners im Explorer zeigt im Kontextmenü unter **Neu...** keinen Eintrag für das Erstellen eines Zip-komprimierten Ordners. Tatsächlich gibt es keinerlei Methoden in der Shell, um das zu bewerkstelligen.

Zum Glück ist eine Zip-Datei prinzipiell eine ziemlich einfache Sache, die sich auch über einen Workaround anlegen lässt. Nehmen Sie eine beliebige Zip-Datei und löschen aus ihr sämtliche Dateien. Das Archiv existiert dann immer noch, hat nun aber eine Größe von nur 23 Byte. Dieses Grundgerüst kann auch über VBA angelegt werden:

```
Dim arrZip(22) As Byte
arrZip(0) = 80 'P
arrZip(1) = 75 'K
arrZip(2) = 5
arrZip(3) = 6
```

Sie legen ein Byte-Array an und füllen die ersten vier Elemente mit den Vorgabewerten für eine Zip-Datei. Die restlichen Bytes enthalten Nullen.

Das Array muss nun nur noch in eine Datei gespeichert werden:

```
Open "d:\test\xy.zip" For Binary
As #1
Put F, , arrZip
Close #1
```

Im Testordner wird im Explorer jetzt für die Datei **xy.zip** die korrekte Bezeichnung **Zip-Archiv** angezeigt.

Das weitere Vorgehen können Sie sich denken. Die angelegte Zip-Datei wird über das Shell-Objekt als **Folder** behandelt und über dessen **CopyHere**-Methode Dateien hinzugefügt. Eine rudimentäre Routine könnte so aussehen:

```
Dim objZip As Shell32.Folder
Dim objSource As Folder
Set objZip = objShell.NameSpace( _
    "d:\test\xy.zip")
Set objSource = objShell.NameSpace( _
    "c:\windows\")
objZip.CopyHere objSource
```

Führen Sie diesen Code besser nicht aus! Er kopiert den gesamten Inhalt des Windows-Ordners in das Zip-Archiv **xy.zip**. Deutlich machen soll er nur, dass die Shell sich hier genauso verhält, wie beim Entpacken. Komplette Ordnerstrukturen können mit nur einer Zeile in ein Archiv verfrachtet werden.

Möchten Sie nur einzelne Dateien in das Archiv übernehmen, so benötigen Sie statt des **Forder**-Objekts ein **FolderItem**-Pendant. Eine Erweiterung der Routine käme etwa wie in Listing 1 daher (Ausschnitt). Dort

```
Dim objZip As Shell32.Folder
Dim objSource As Folder
Dim objItem As FolderItem
Set objZip = objShell.NameSpace("d:\test\xy.zip")
Set objSource = objShell.NameSpace("d:\test\")
Set objItem = objSource.Items.Item("mein.txt")
objZip.CopyHere objItem
```

Listing 1: Hinzufügen von Dateien zum Zip-Archiv

würde die Datei **mein.txt** des Quellverzeichnisses **d:\test** über die Variable **objItem** in das Archiv wandern.

Sollten Sie in einer Schleife mehrere Dateien in das Archiv übernehmen, so sollten Sie auch hier ein **DoEvents** einfügen. Sonst kommt es vor, dass im Archiv anschließend einige Dateien fehlen. Auch sonst gilt, etwa für die Optionen der **CopyHere**-Methode, dasselbe, wie für die Routine zum Entpacken des Archivs.

Ausführlichere Routinen zum Erzeugen von Zip-Archiven finden Sie in der Beispieldatenbank in Gestalt der beiden Prozeduren **CreateZip** und **CreateZipFromFolder**. Der ersten übergeben Sie als Parameter den Pfadnamen der zu erzeugenden Zip-Datei und alle hinzuzufügenden Dateien in einem String-Array, der zweiten stattdessen nur den Pfadnamen eines Ordners, den Sie im Archiv haben wollen. Beide Prozeduren werden vom Formular **frmZipper** verwendet, das eine Oberfläche für sie bereitstellt.

Zippen über Formular

Bild 2 zeigt Sie das Formular **frmZipper** der Beispieldatenbank nach dem Öffnen. Im obersten Textfeld geben Sie den Pfadnamen der Zip-Datei an, die Sie erzeugen wollen. Dafür können Sie auch die Schaltfläche rechts daneben verwenden. Sie öffnet einen **Datei speichern**-Dialog, in dem der Name **AB_Test.zip** auf das aktuelle Datenbankverzeichnis voreingestellt ist, wie in Bild 3. Diesen Dialog

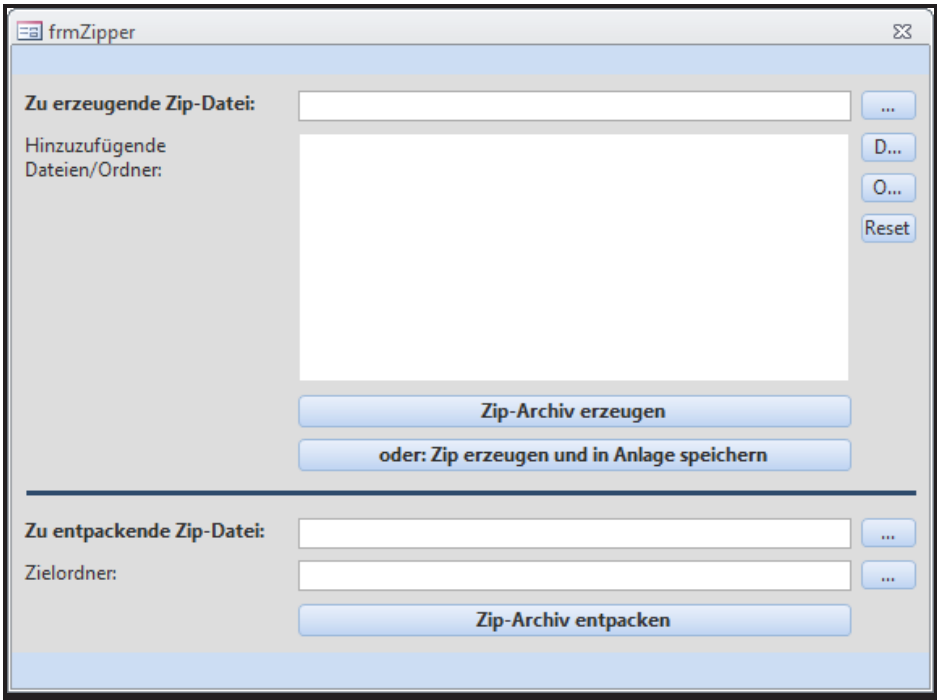


Bild 2: Das Formular frmZipper für die Verwaltung von Zip-Archiven

Navigationsbereiche gestalten

Ab Access 2007 wurde das Datenbankfenster zur Anzeige der Objekte einer Datenbank durch den Navigationsbereich ersetzt. War jenes eigentlich ausschließlich für den Entwickler bestimmt, sollte das Navigationsfenster auch für den Datenbankbenutzer von Nutzen sein. Wie dieser entsprechend angepasst werden kann, beleuchten wir in diesem Beitrag.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1502_Navigation.accdb**

Navigationsbereich

Ob der Navigationsbereich für den Entwickler wirklich von Gewinn ist, darüber gehen die Meinungen auseinander. Da hier eine nur eindimensionale Liste vorliegt, während die des Datenbankfensters von Access 2003 noch zweidimensional war, geht im Vergleich bei umfangreichen Datenbanken etwas die Übersicht verloren. Die Such-, Filter- und Gruppierungsfunktionen machen das andererseits teilweise wett. Was dem Entwickler vielleicht ein Dorn im Auge ist, kann jedoch für den Anwender der Datenbank eine nützliche Angelegenheit sein, denn das alte Datenbankfenster eignete sich nun wirklich nicht als Bedienoberfläche. Switchboards, Menüs oder andere Elemente wurden für die Navigation herangezogen.

In den neueren Access-Versionen lässt sich stattdessen das Ribbon verwenden, oder auch ein angepasster Navigationsbereich, beziehungsweise eine Kombination beider. Um die Anpassungen vornehmen zu können, sollte man sich zunächst mit den Grundelementen des Navigationsfensters vertraut machen. Bild 1 zeigt es für die Beispieldatenbank, die übrigens lediglich eine Kopie der für die Ausgabe 12/2014 zum Thema **Diagramme in Formularen und Berichten** verwandten ist.

Zuoberst finden Sie den Kategorietitel **Alle Access-Objekte** der Navigation. Über den Pfeilbutton können Sie eine andere Kategorie auswählen. Access definiert hier grundsätzlich einige Kategorien vor, die aber durch eigene erweitert werden können. Darunter lässt sich über das Textfeld Suche die Liste filtern, wobei dies über Teilausdrücke funktioniert – also quasi eine *LIKE*-Filterung. Daran schließt sich die eigentliche Liste der Objekte an, die wiederum in Gruppen gegliedert ist. Anzahl, Titel und Inhalt der Gruppen lassen sich ebenfalls erweitern und anpassen. Die Gruppen können aus- und zugeklappt, über die Optionen aber auch einzeln komplett ausgeblendet werden.

Alle Elemente des Navigationsbereichs weisen ein Kontextmenü auf. Ein Rechtsklick auf die Kategorieleiste bringt ein Menü hervor, über das der Anzeigemodus der Objektliste eingestellt wird. So lässt sich hier die Sortierung festlegen – etwa nach Name, Typ oder Erstell- und Änderungsdatum –, oder die Gestalt der Objekteinträge. Meist wird man es bei der Einstellung **Liste** belassen, denn die anderen Modi **Symbol** und **Details** vergeuden doch etwas sehr viel Raum.

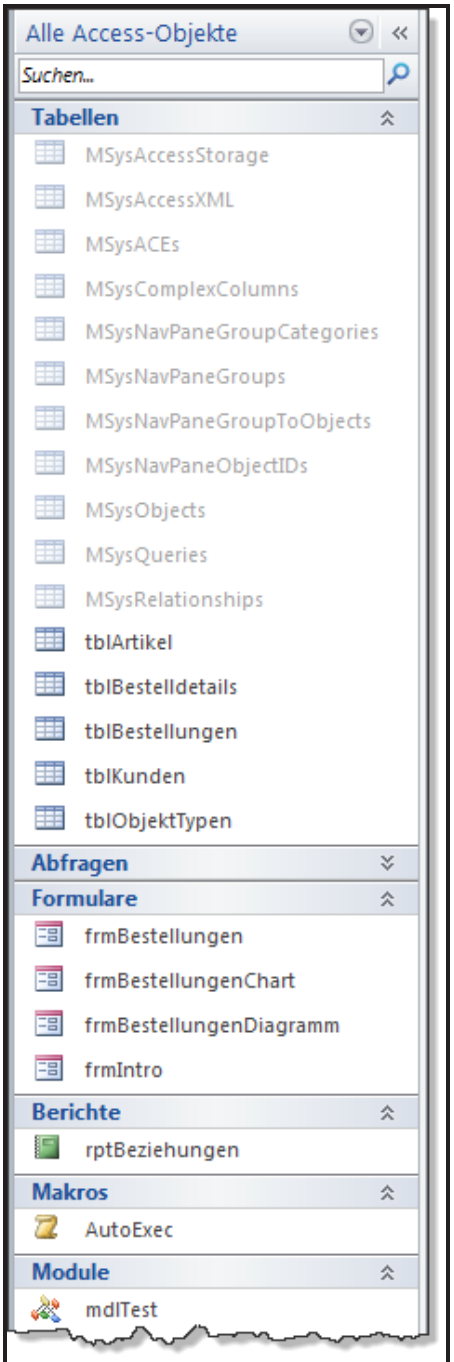


Bild 1: Datenbanknavigationsfenster

Anpassung vornehmen

Eigene Anpassungen erfolgen über den Aufruf der **Navigationsoptionen...** des Kontextmenüs (Bild 2). Das ruft ein Dialogfenster auf den Plan, der Steuerzentrale für das Navigationsfenster. In jungfräulichem Zustand präsentiert es sich wie in Bild 3. Im linken Listenfeld sind die Kategorien zu sehen, wobei die vordefinierte Kategorie **Objekttyp** sich weder modifizieren, noch löschen lässt. Über die Schaltfläche **Element hinzufügen** können Sie jedoch neue Kategorien anlegen. Sobald Sie eine Kategorie markieren, ändert sich der Inhalt des rechten Listenfelds, welche die zu dieser Kategorie gehörigen Gruppen darbietet. Für eigene Kategorien sind dann auch die Schaltflächen zum Erstellen oder Ändern neuer Gruppen aktiv.

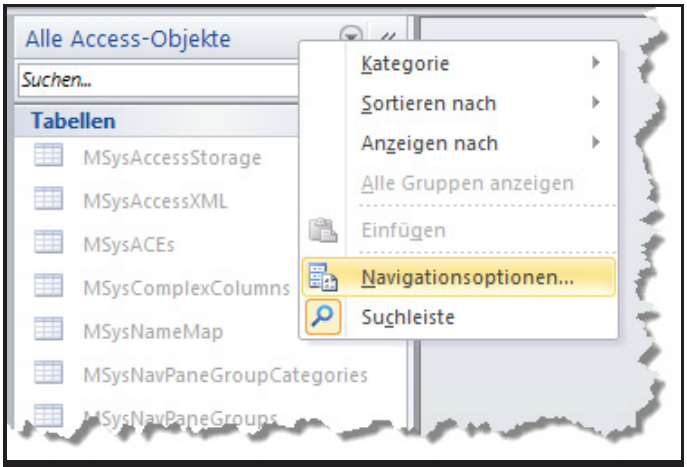


Bild 2: Kontextmenü des Navigationsbereichs (Titelleiste)

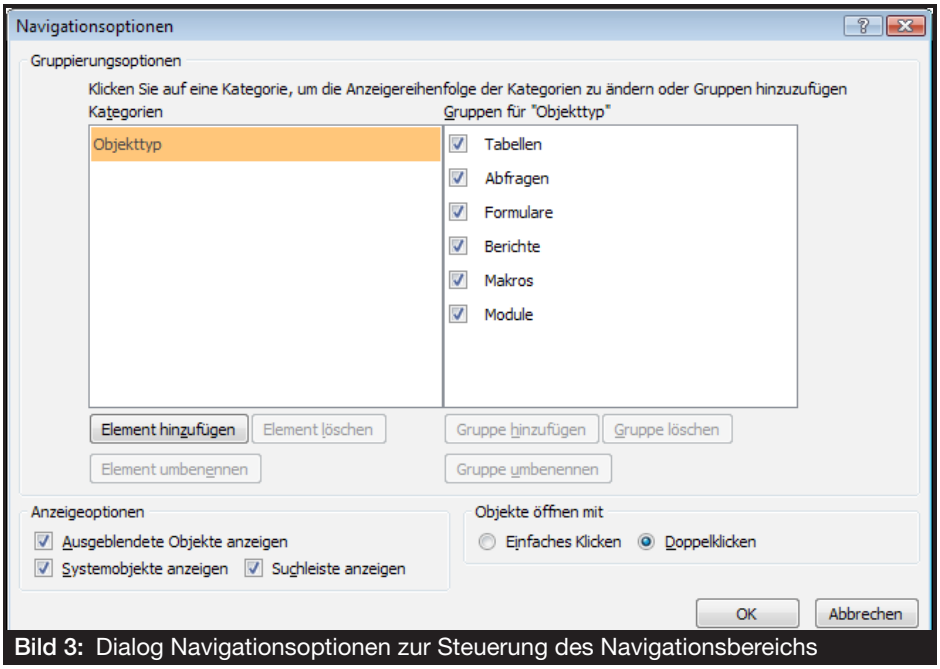


Bild 3: Dialog Navigationsoptionen zur Steuerung des Navigationsbereichs

Wie möchten wir nun den Navigationsbereich neu gestalten? Es hängt davon ab, welches Ziel Ihnen vor Augen schwebt. Als Entwickler werden Sie eine möglichst funktionelle Struktur anstreben. So könnten Sie etwa alle Elemente, die mit Bestellungen zu tun haben, in eine Kategorie **Bestellungen** nehmen und diese wiederum in Gruppen, wie **Bestellformulare** und **Bestellabfragen** untergliedern. So haben Sie alle zu Bestellungen gehörigen Elemente übersichtlich im Blick. Mit Kundenelementen könnten Sie analog verfahren.

Soll der Navigationsbereich jedoch für den Endanwender gestaltet sein, so möchten Sie wahrscheinlich Elemente, wie Abfragen, Unterformulare, Module, oder Systemobjekte ausblenden und nur jene anzeigen lassen, die direkt aufgerufen werden sollen. Wie auch immer, das Vorgehen ist in beiden Fällen gleich, so dass die folgenden Ausführungen grundsätzlich gelten. Anhand eines Beispiels möchten wir **Formulare und Berichte** als neue Kategorie definieren und für Systemtabellen ebenfalls eine eigene Kategorie einrichten.

Neue Kategorien und Gruppen

Klicken Sie auf Element hinzufügen, um eine neue Kategorie zu erstellen. Im Listenfeld entsteht eine neue Zeile, deren Bezeichnung Sie gleich editieren können. Verwenden Sie **Formulare und Berichte**. Im rechten Listenfeld findet sich nun immer der Eintrag **Nicht zugewiesene Objekte**. Wofür es diesen gibt und warum er zunächst immer markiert ist, sehen wir gleich noch bei der Zuordnung von Datenbankobjekten zu den Gruppen. Erstellen Sie jetzt aber erst die zwei neuen Gruppen **Formular** und **Bericht** über den Button **Gruppe hinzufügen**. Danach sollte der Dialog etwa so aussehen, wie in Bild 4. Nachdem Sie ihn schließen, finden Sie die neue Kategorie und deren

Access als MP3-Player

Nein, Access kennt keine eingebaute Möglichkeit, um MP3-Dateien abzuspielen. Dennoch lässt sich das über den Windows Media Player und dessen Fernsteuerung mit wenig Aufwand einrichten. Da es zudem bei der Verwaltung von MP3-Dateien um Aufgaben geht, für die sich eine Datenbank gut macht, kam die Idee auf, eine Player-Anwendung unter Access zu realisieren. Schauen Sie selbst ...

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1502_MP3.mdb**.

Access [basics]-MP3-PLAYER
Damit Sie gleich einen Eindruck davon bekommen, was Sie erwartet, sehen Sie in Bild 1 die stylische Oberfläche des Players – das muss in diesem Genre nun mal sein. Und schon vorneweg: Im Rahmen dieses Beitrags können nicht alle Finessen der Gestaltung und Programmierung der Datenbank besprochen werden. Dafür ist sie zu aufwändig, wenn auch nicht kompliziert. Wir beschränken uns an dieser Stelle auf die Darstellung der wesentlichen Kernfunktionen.

Auf der linken Seite befindet sich die Liste aller in die Datenbank aufgenommenen MP3-Dateien. Über das Suchfeld darunter lassen sich die Titel filtern. Bei Eingabe von Buchstaben in das Suchfeld filtert sich die Liste sofort und reduziert sich auf diejenigen Dateien, welche den eingegebenen Begriff enthalten. Da der Dateiname unter Umständen nicht so aussagekräftig ist, besteht zusätzlich über die Schaltfläche **Filter...** die Möglichkeit, die Titel nach weiteren Kriterien, wie etwa dem Artist zu filtern. Der Filter wird einfach

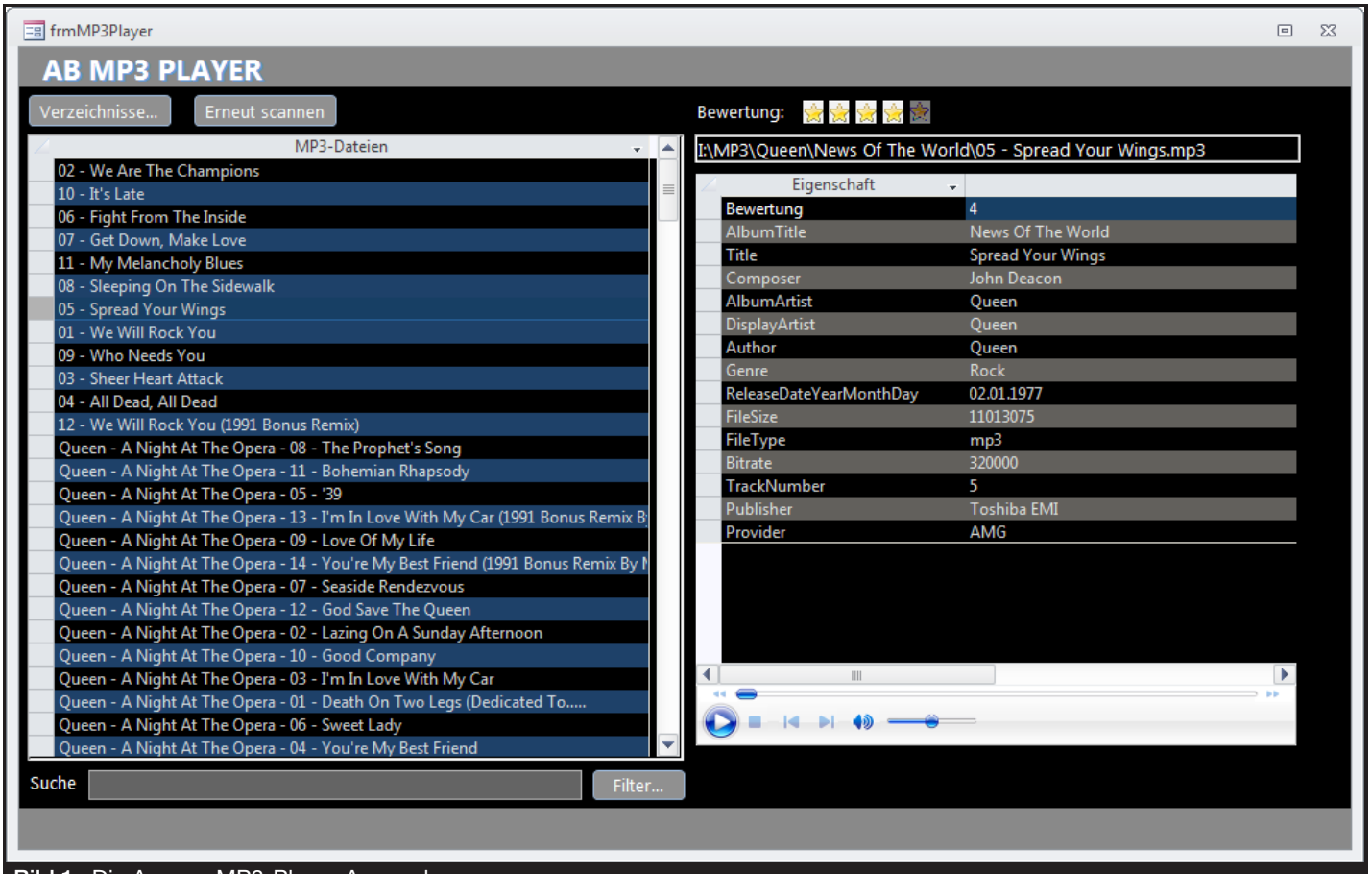


Bild 1: Die Access-MP3-Player-Anwendung

zurückgesetzt, wenn auf das Suchfeld doppelgeklickt wird.

Markieren Sie einen Eintrag der Titelliste, so zeigt das rechte Unterformular verschiedene Eigenschaften des Titels an, die im Wesentlichen aus den MP3-Tags der Datei bezogen wurden. Ein Doppelklick auf einen Eintrag der Liste spielt die Datei ab. Ebenso kann hierfür der Start-Button des Media Players verwendet werden.

Oben rechts können Sie den Titel bewerten, indem Sie bis zu fünf Sternchen vergeben. Das wird in der Datenbank gespeichert und kann später über die Filterfunktion zum Heraussuchen der Favoriten benutzt werden.

Im Auslieferungszustand ist die Datenbank noch leer, und Sie müssen zunächst die Verzeichnisse definieren, die in die Suche nach MP3-Dateien einbezogen werden sollen.

Dazu klicken Sie auf die Schaltfläche **Verzeichnisse...**, die das Formular in Bild 2 aufruft. Hier haben Sie die Möglichkeit, über die Pünktchen-Buttons neue Verzeichnisse in die Liste aufzunehmen, wobei jeweils angegeben werden kann, ob auch deren Unterverzeichnisse zu berücksichtigen sind.

Nach dem Aufnehmen der Verzeichnisse geht es an das Durchsuchen dieser nach MP3-Dateien. Klicken Sie dazu auf den Button **Erneut scannen** des Hauptformulars. Der Vorgang kann eine Weile dauern und zeigt den Vollzug dadurch an, dass die Titelliste sich mit Einträgen gefüllt hat. Dabei werden Sie außerdem über eine Messagebox gefragt, ob nun auch die MP3-Eigenschaften der Dateien ausgelesen werden sollen. Wenn Sie das bestätigen, was zu empfehlen ist, damit später die Filtermöglichkeiten größer sind, so liest die Datenbank jede einzelne Datei und deren MP3-Tags aus. Dass das noch länger braucht, als das bloße Scannen der Verzeichnisse, ist sicher selbstverständlich, und darum zeigt ein Fortschrittsbalken am unteren Ende des Formulars auch an, wie weit die Code-Ausführung gediehen ist.

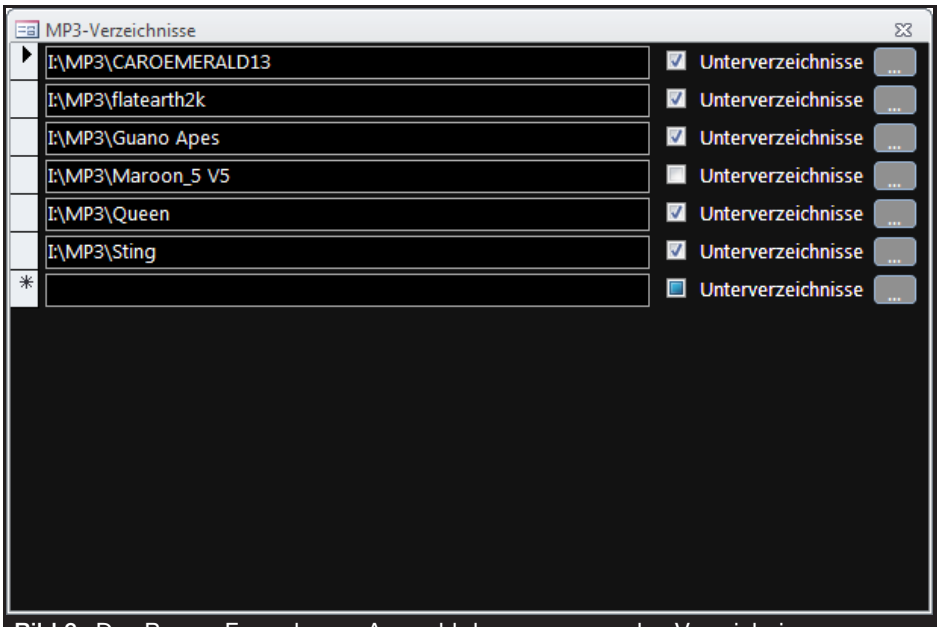


Bild 2: Das Popup-Formular zur Auswahl der zu scannenden Verzeichnisse

Soweit die Beschreibung der Anwendung. Sie können nun loslegen und sie einfach benutzen, oder sich in die Entwicklungshintergründe begeben...

Windows Media Player

Natürlich handelt sich bei der Player-Einrichtung der Datenbank nicht um eine Eigenentwicklung. Das wäre wirklich des Guten zuviel gewesen. Warum auch? Windows bringt mit dem **Media Player** nicht nur ein geeignetes Instrument mit, das sich per OLE-Automation und damit per VBA steuern lässt, sondern auch noch ein Steuerelement, das den Player beherbergt. Sie fügen das Control in ein Formular ein, indem Sie die Liste der **ActiveX-Steuerelemente (Ribbon)** im Entwurf nach dem Windows Media Player durchforsten. Das Steuerelement zeigt nun nicht nur die Leiste mit den Bedienelementen an, sondern auch den Bereich, der eigentlich zu Anzeige von Videos gedacht ist. Dort zeigt sich beim Abspielen von Sound-Dateien dann eine nette Animation, die wir aber nicht in der Anwendung haben möchten. Leider lässt sich dieser Bereich nicht ausblenden. Verkleinern Sie im Entwurf die Höhe des Steuerelements, so dass nur noch die Bedienleiste übrig bleibt, scheint dies das gewünschte Ergebnis zu zeitigen, doch in der Formularansicht verschwindet stattdessen die Leiste und der Animati-

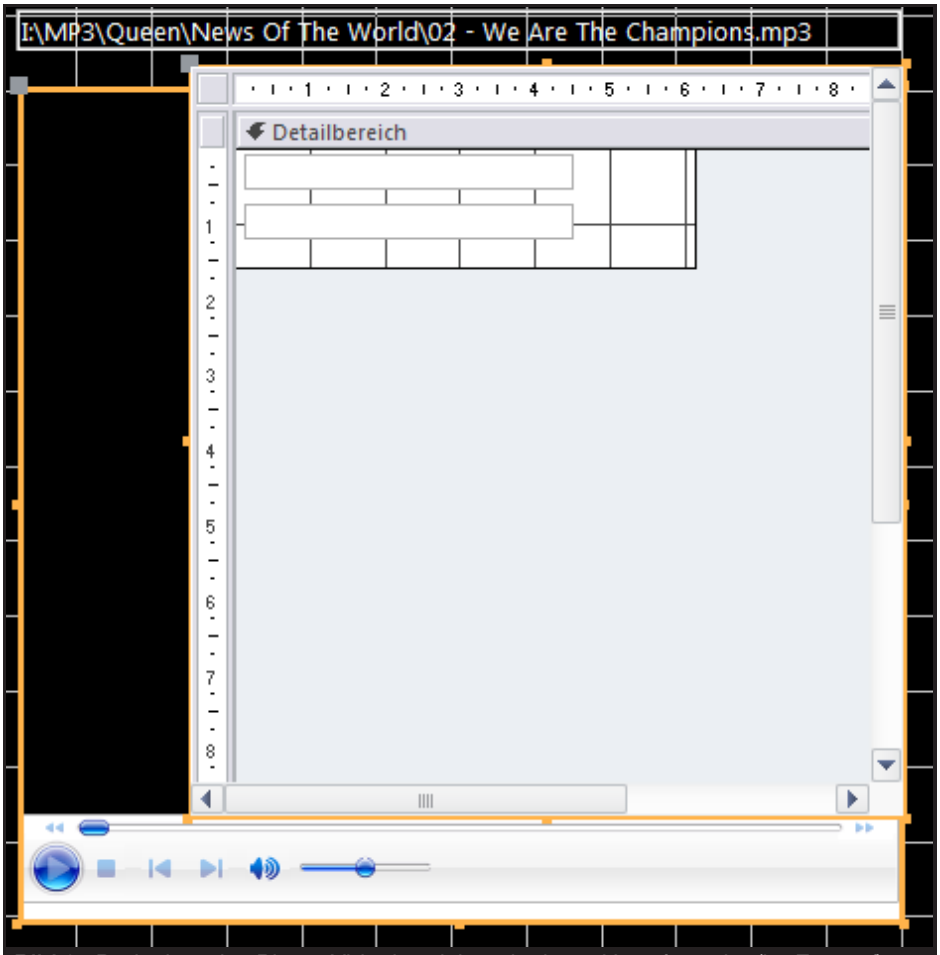


Bild 3: Bedecken des Player-Videobereichs mit einem Unterformular (im Entwurf)

onsbereich ist sichtbar. Das Control lässt sich unter Access nicht dazu überreden, diesen Bereich auszublenden. Darum wird er, wie in Bild 3, als Workaround einfach mit einem Unterformular überdeckt, was in der Anwendung jenes für die Anzeige der MP3-Eigenschaften ist (**sfrm-Eigenschaften**).

Markieren Sie nun im Entwurf das Steuerelement und bewegen sich im **Eigenschaftenblatt** auf den Reiter **Andere**. Markieren Sie hier den Eintrag **Benutzerdefiniert** und dort die Schaltfläche mit den Pünktchen. Es öffnet sich der Eigenschaftendialog des Players. Dort können Sie diverse Optionen wählen, die aber zumeist für die Darstellung von Videos von Belang sind. Wichtiger ist die Registerseite **Erweitert**: Dort sollte **Steuerelemente aktivieren** gesetzt sein, und besser **Kontextmenü aktivieren** nicht. Das bezieht sich auf die Bedienleiste des Players. Sind die Steuerelemente deaktiviert, so können Sie beim Abspielen etwa weder die Positionen einstellen, noch die Lautstärke ändern. Schließen Sie den Dialog mit **OK**.

Wie bringt man dem Player-Control nun bei eine Datei abzuspielen? Im Eigenschaftenblatt ist keine passende Eigenschaft, wie **File** oder ähnlich, sichtbar. Dafür ist nun doch etwas VBA-Programmierung nötig...

Und die lässt sich zwar mit nur wenigen Zeilen erledigen, ist aber auf den ersten Blick nicht sehr einleuchtend. Es gibt im Objektkatalog zur Bibliothek **WMPLib** – das ist die dem Media Player zugeordnete und automatisch von Access eingefügte Bibliothek – tatsächlich keine Eigenschaft **File** oder eine **Open**-Anweisung für das Control-Objekt **WindowsMediaPlayer**. Dieser spielt nämlich nur Playlisten ab. Eine Datei ohne Playlist gibt es in diesem System nicht. Folglich muss erst eine Playlist angesprochen werden, der man eine oder mehrere Dateien hinzufügt. Der verantwortliche Code dazu steht im Formular **frmMP3Player** in der Prozedur **SetMedia**, die in Listing 1 abgebildet ist. **oPlayer** ist die Objektvariable für das Player-Steuerelement. **sMP3** enthält den Pfad zur MP3-Datei. Zuerst werden die Bedienelemente des Players (**Controls**) angewiesen, eine eventuell gerade laufende Aktion zu stoppen. Dann wird der Player geschlossen (**Close**),

```
Private Sub SetMedia()
    Dim oMedia As IWMPMedia

    sMP3 = LblPath.Caption
    oPlayer.Controls.Stop
    oPlayer.Close
    oPlayer.currentPlaylist.Clear
    Set oMedia = oPlayer.newMedia(sMP3)
    oPlayer.currentPlaylist.appendItem oMedia
End Sub
```

Listing 1: Diese Routine bereitet das MediaPlayer-Control zum Abspielen einer MP3-Datei vor

was ihn quasi resettet. Die aktuelle Playlist – man muss sie nicht neu erzeugen, da es immer eine gibt – wird ebenfalls mit **currentPlaylist.Clear** geleert. Und nun kommen die zwei entscheidenden Zeilen: Mit der Methode **newMedia**, der als Parameter der Dateipfad mitzugeben ist, wird ein neues Medien-Objekt erzeugt. Und dieses Objekt **oMedia** wird über **currentPlaylist.appendMedia** der Playlist hinzugefügt. Damit ist das Player-Control bereit zur Wiedergabe der Datei. Die Wiedergabe wiederum wird grundsätzlich über die Eigenschaft **Controls** des Players gesteuert. Das Abspielen geschieht mit

```
oPlayer.Controls.play
```

Mehr braucht es dazu nicht. Wollen Sie die Wiedergabe per VBA anhalten, so genügt ein

```
oPlayer.Controls.pause
```

Für mehr Anweisungen schauen Sie sich die Funktionen der Klasse **IWMPControls** im Objektkatalog des VBA-Editors an.

MP3-Tags

Das Player-Control ist auch verantwortlich für das Auslesen der MP3-Eigenschaften. Bedenken Sie, dass hinter diesem Steuerelement der gesamte Windows Player und dessen System stehen, weshalb auch die Bibliothek recht umfangreich ist. Im Formular finden Sie in der Routine **GetMP3Properties** den Code für das Auslesen der MP3-Tags. In ihr sind zahlreiche Elemente enthalten, die sich mit dem Speichern der Eigenschaften in die Tabellen der Datenbank über Recordsets beschäftigen. Jene Zeilen, die das Player-Objekt betreffen, hier im Schnelldurchlauf:

Im Sprachgebrauch von Windows Media nennen sich die MP3-Tags **Attribute**. Zuerst ist über den Player, ähnlich, wie in Listing 1, ein Media-Objekt zu erzeugen. Statt es abzuspielen, wird nun die Anzahl der Tags dieses Objekts über die Eigenschaft **attributeCount** ermittelt und eine Schleife darüber gebaut:

```
Set oMedia = oPlayer.newMedia(sDatei)
For i = 0 To oMedia.attributeCount
    ...
Next i
```

Innerhalb der Schleife wird nun jeweils der Name und der Wert eines Tags ausgelesen:

```
sTagName = oMedia. _
    getAttributeName(i)
sValue = oMedia. _
    getItemInfo(sTagName)
```

Tatsächlich ist das auch schon alles, was das Player-Objekt angeht. Es wird zum Schluss mit der **Close**-Anweisung lediglich noch geschlossen. Dass man mit so wenig Code an die kompletten MP3-Tags gelangt, ist schon erstaunlich. Der Rest der Routine ist, wie gesagt, dem Speichern der Eigenschaften in die Tabellen vorbehalten.

Tabellen der Datenbank

Deshalb möchten wir an dieser Stelle kurz auf das Datenmodell der Anwendung eingehen. In Bild 4 wird das Beziehungs-Layout verdeutlicht. Die Tabelle **tblVerzeichnisse** wird über das Formular **frmVerzeichnisse** (siehe oben) gefüllt. Es basiert direkt auf dieser Tabelle und ist in Endlosformularansicht gestaltet. Die Tabelle **tblMP3Dateien** wird über den Scan-Vorgang bestückt, der im Hauptformular in der Prozedur **ScanMP3Files** vonstatten geht. Diese durchläuft alle Verzeichniseinträge von **tblVerzeichnisse** und ruft für jedes die Hilfsfunktion **FindFilesTree** im Modul **mdlFindFiles** auf. Die Rückgabeergebnisse speichert Sie als Dateiname neben der ID des Verzeichnisses ab.

Auf die Routine **GetMP3Properties** kamen wir schon zu sprechen. Sie speichert ihre Ermittlungen in die zwei Tabellen **tblEigenschaft** und

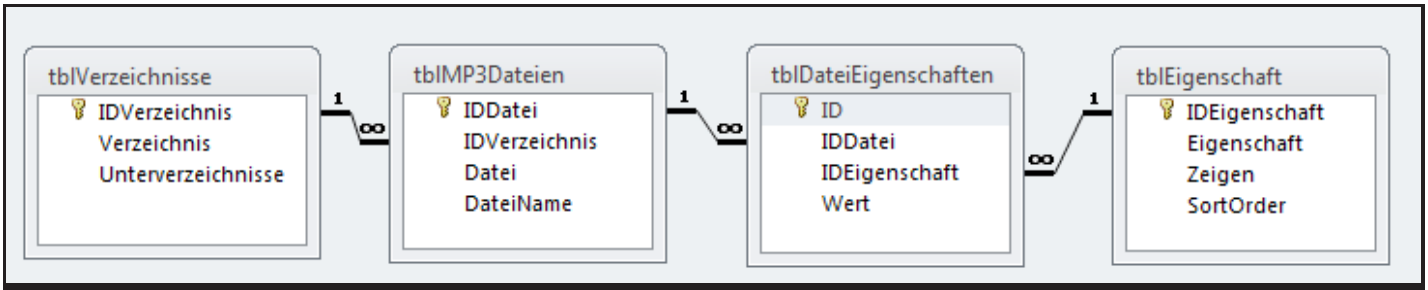


Bild 4: Das Datenmodell der Access-MP3-Player-Anwendung

Textdateien exportieren und einlesen

Wenn Sie Daten zwischen Access und anderen Anwendungen austauschen, dann ist der Weg über Textdateien oft der überschaubarste, wenn nicht gar der einzige. Der Import oder Export läuft als Zwischenschritt über ein Format, das einfach und kontrollierbar ist. Doch manchmal ringen Sie hier auch mit undurchsichtigen Problemen, denn Text ist nicht gleich Text ...

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1503_Textumwandlung.mdb**.

Import und Export in Access

Möchten Sie eine Tabelle aus Access in eine andere Anwendung exportieren, die nur mit Textdateien umgehen kann, so verwenden einfach Sie den Export-Assistenten, welchen Sie über den Ribbon und dessen Register **Externe Daten** erreichen. Klicken Sie hier auf **Exportieren|Textdatei**.

Zuvor haben Sie die Tabelle oder Abfrage mit den gefragten Daten im Navigationsfenster markiert – etwa die **tblAddressen** der Beispieldatenbank.

Im nun erscheinenden Dialog geben Sie den Pfad für die zu erstellende Textdatei an (siehe Bild 1). Sollte das Kontrollkästchen mit der Beschriftung **Exportieren mit Formatierung und Layout** aktiviert werden? Davon ist abzuraten, wenn die Textdatei später als Datenquelle herhalten soll, denn die Formatierung macht unter Umständen die Struktur zunichte und ist nur dann sinnvoll, wenn Sie die Textdatei anschließend lediglich in einem Texteditor betrachten wollen.

Mit Klick auf **OK** gelangen Sie zu weiteren Seiten des Assistenten-Dialogs, über die Sie das Feldtrennzeichen und andere Einstellungen für das Format angeben können. Access belegt alles mit Voreinstellungen, die mehr oder weniger nützlich sind (Bild 2), aber meist der Modifikation bedürfen. So sollte ein Häkchen auf **Feldnamen in erste Zeile einbeziehen** gesetzt werden, denn sonst ist beim späteren Import unbekannt, wie die Spalten der Tabelle lauten.

Möchten Sie etwa die Textdatei mit Adressen in Outlook importieren, dann fragt dieses für jede Spalte die Zuordnung zu seinen internen Adressfeldern ab. Da ist es optimaler, wenn gleich die Feldnamen der Tabelle ohne Begutachtung deren Inhalts ersichtlich werden. Dasselbe gilt für den Reimport der Textdatei in Access.

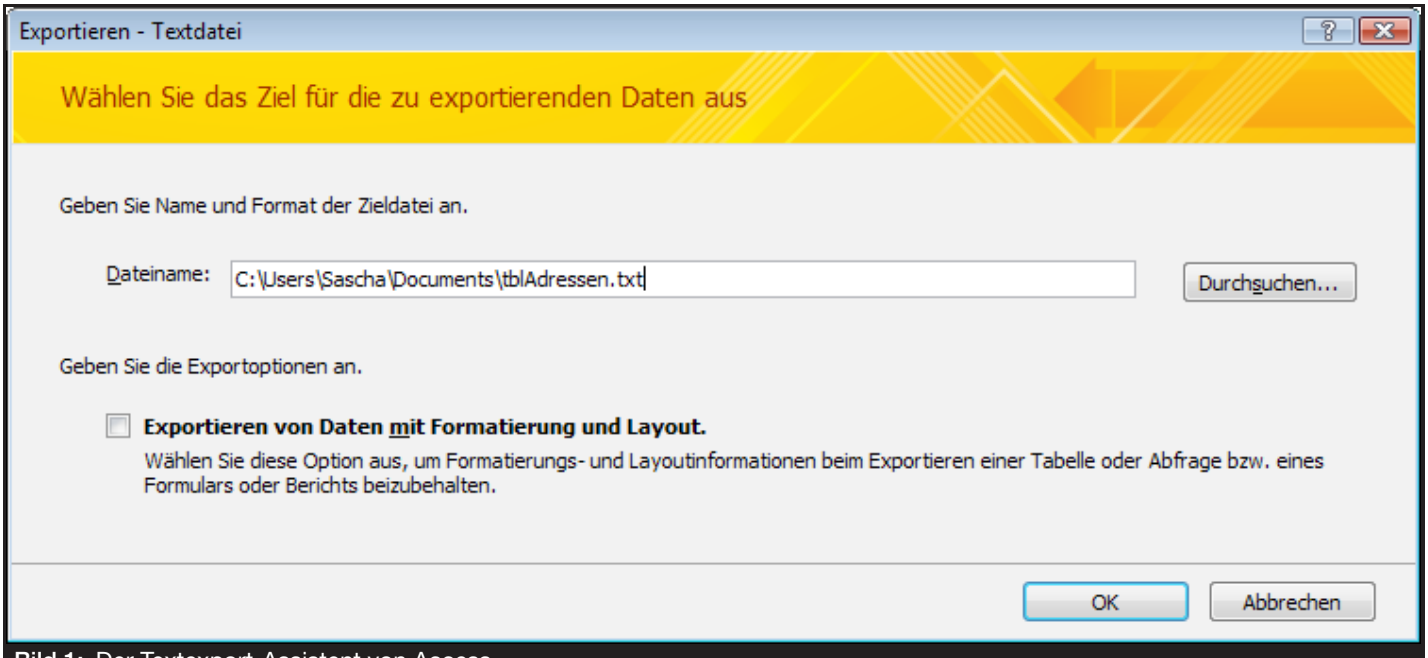


Bild 1: Der Textexport-Assistent von Access

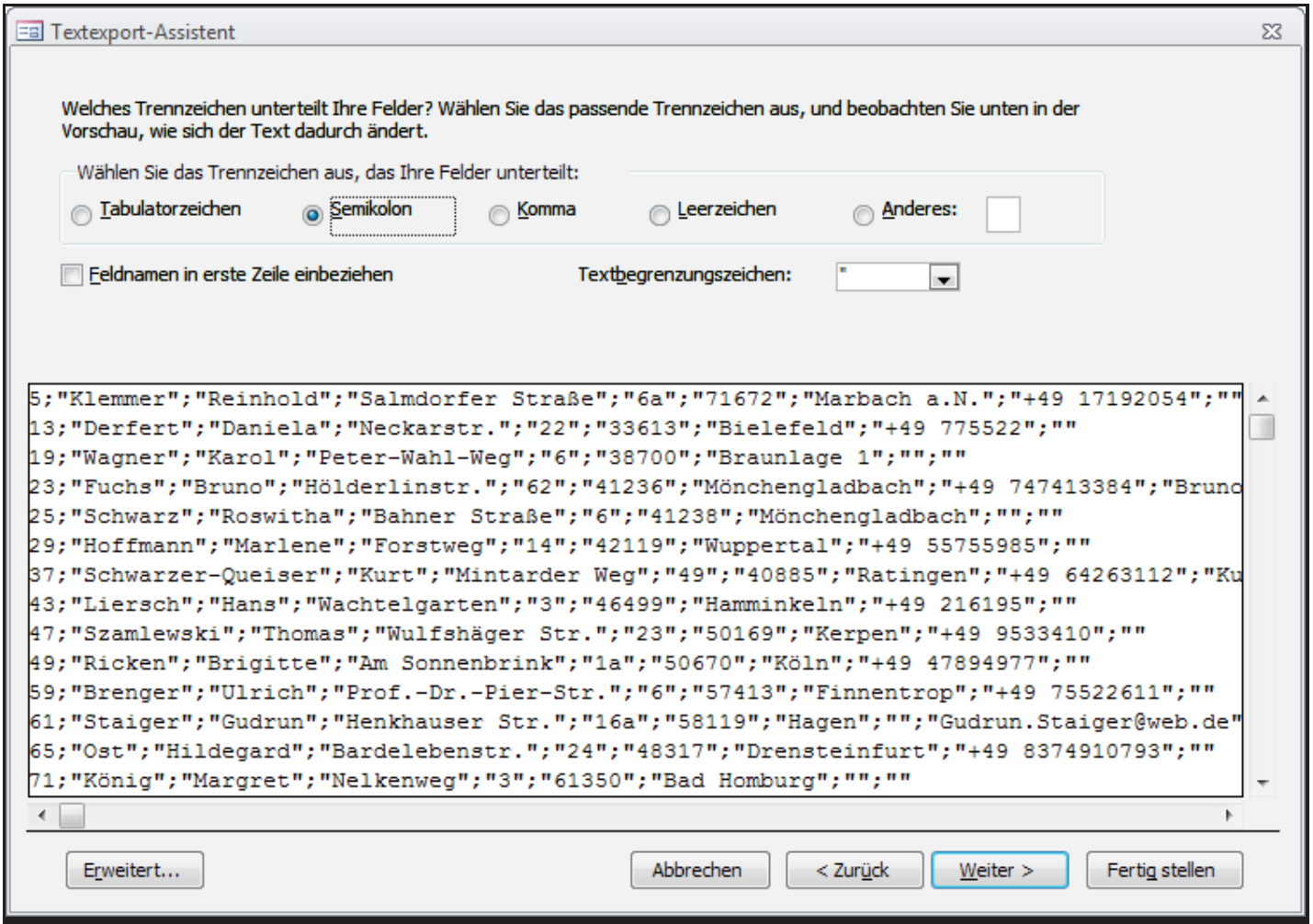


Bild 2: Einstellungenseite des Assistenten für den Textexport

Beim Feldtrennzeichen sind andere Überlegungen anzustellen. Das vorgeschlagene Semikolon ist oft ungeeignet. Nämlich immer dann, wenn Textinhalte der Tabelle selbst Semikola enthalten könnten.

Diese würden später ebenfalls als Feldtrenner verstanden werden, was den Import fehlschlagen lässt. Günstiger ist da das **Tabulatorzeichen**, welches sicher selten in Textfeldern auftauchen wird – schließlich bewegen Sie sich mit der **Tab**-Taste in einem Datenblatt zur nächsten Spalte, anstatt ein Tab-Zeichen einzugeben.

Allerdings kommt es auch auf die Zielanwendung an. Oft wird eben gerade das Semikolon als Trennzeichen erwartet, und dann müssen Sie dieses auch aktivieren, aber zuvor sicherstellen, dass die Tabelle selbst keine Semikola enthält.

Beim **Textbegrenzungszeichen** sieht es ähnlich aus. Die Anführungszeichen sind zwar Standard, erfordern aber ebenfalls, dass solche nicht im Text selbst vorkommen dürfen. Sollte das jedoch nicht gegeben sein, so wählen Sie aus dem Kombinationsfeld den Eintrag **{kein}** aus. Eine Fremdanwendung kann dann beim Import zwar nicht so leicht erkennen, dass es sich bei einer Spalte um ein Textfeld handelt, aber dieser Umstand ist weniger problematisch, als falsch positionierte Textbegrenzungszeichen.

Nicht verschwiegen werden sollte, dass Sie sich mit fehlendem Textbegrenzungszeichen neue Probleme einhandeln können. Da diese neben dem Trennzeichen den Inhalt einer Zelle eindeutig abgrenzen, kann ein enthaltener String beliebige Zeichen aufweisen. Fehlt das Begrenzungszeichen, so werden eventuell bestimmte Zeichen anders interpretiert.

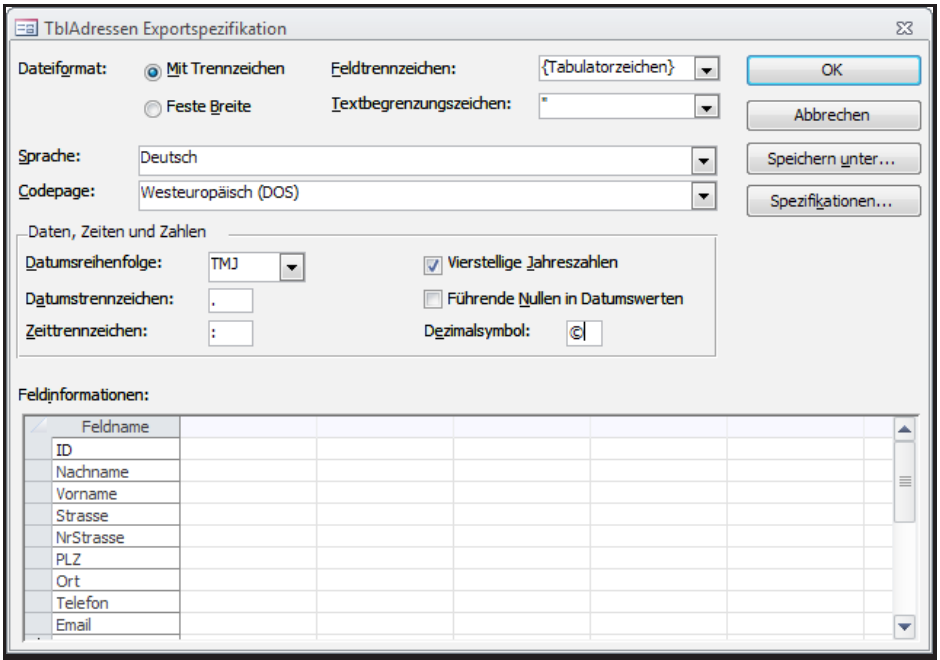


Bild 3: Exportspezifikationen

Häufige Kandidaten für das Schlamassel sind das Punktzeichen und das Komma, denn diese werden oft als Dezimalpunkt oder -komma für Zahlen verstanden. Schon beim Export aus Access kann es dabei zu Fehlern kommen, sofern Sie nicht über den Button **Erweitert...** ein anderes **Dezimalsymbol** bestimmten (Bild 3).

Soll die Textdatei später wieder in Access importiert werden, so können Sie hier ein Zeichen nehmen, dessen Vorkommen in der Tabelle ausscheidet. Solche Zeichen lassen sich mit die Tastatur über den Ziffernblock ausgeben. Nehmen Sie etwa für ein © die Kombination ALT-0169. Noch sicherer wäre die Kombination ALT-0253.

Mit einem Klick auf **Weiter >** im Export-Assistenten erstellen Sie schließlich die Textdatei.

Das Einlesen dieser Datei als Tabelle in Access übernimmt der **Textimport-Assistent**, der im weitgehend gleichen Gewand daherkommt, wie der Export-Assistent. Sie rufen ihn über den Ribbon und **Externe Daten|Importieren und Verknüpfen|Textdatei** auf. Auf den Seiten des Dialogs machen Sie dann exakt dieselben Angaben, wie beim Export. Es müssen Feldtrennzeichen, Begrenzungszeichen, Dezimalsymbol und Zeilenüberschriften übereinstimmen. Die so importierte Adressentabelle sollte dann die genau gleichen Daten zeigen, wie die Ursprungstabelle. Andere Eigenschaften der Tabelle, wie die Einstellungen zu Indizes, Default-Werten oder Beschränkungen, müssen Sie selbstverständlich manuell nachholen.

Nachname	Vorname	Strasse	N
Bl„sing	Hans Peter	Lommatzscher Platz	2
Wollschl„ger	Hermann	Lettestr.	19
Sch„der	Therese	H"kerstr.	16
Str„hler	Karin	Paul-Lincke-Str.	25
Langenk„mper	Lena	Finkenhofstr.	33
Lichtenb„umer	Brigitte	Erlenmeyerstr.	21
Sch„fer	Petra	Grner Wall	44
W„ldele		Vor den Weiden	1
M„ttig	Christine	Seitenstr.	14
R„ck		Kapellenstr.	5
Sch„fer	Claudia	Heidenstock	42
Gr„tz	Juliane	Mnsterdeich	27
B„uml	Josefa	Sch"llbronner Str.	59
Sch„fers	Klaus	Garrelts Kamp	

Bild 4: Fehlende Umlaute nach dem Import

Import fremder Textdaten

Beim Export und Import von Tabellen über Textdateien aus Access heraus gibt es selten Probleme, wenn Sie die vorherigen Ausführungen beachten. Anders sieht das aus, wenn die Quelle der Textdatei eine fremde Anwendung ist. Vor allem dann, wenn diese etwa auf einem Mac oder unter Linux erstellt wurde, in einem anderen Land oder gar zu Zeiten, in denen MS-DOS noch das Maß der Dinge war. Zwar kann über den Assistenten und dessen Vorschau kontrolliert werden, welches die korrekten Trennzeichen oder Dezimalsymbole sind, aber das Ergebnis einer auf die Schnelle importierten Textdatei sieht dann unter Umständen aus, wie in Bild 4. Zwar fand der Import korrekt statt, weil der Import-Assistent erfolgreichen Vollzug meldete und sich keine Tabelle Einfügefehler bildete – was meist der Fall ist, wenn der Inhalt eines Textfelds mehr als 255 Zeichen enthielt, wofür ein Memofeld angelegt werden müsste –, aber die Texte zeigen seltsame Zeichen, die sich bei näherer Betrachtung als falsch konvertierte Umlaute herausstellen.

Grund dafür ist, dass sich Sprache und sogenannte

Kodierte Texte mit VBA umwandeln

Auch wenn die Assistenten von Access zum Import und Export von Textdateien über ihre Einstellungsseiten genügend Möglichkeiten bieten, um mit allen erdenklichen Formaten zurechtzukommen, gibt es häufig Situationen, in denen die Verarbeitung über VBA-Programmierung benötigt wird. Hier geht es in erster Linie um den Umgang mit Texten in unterschiedlicher Sprachkodierung.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1503_Textumwandlung.mdb**.

Sprachen, Codepages, CharacterSets

Beleuchten wird zunächst, was es mit den Sprachcharakteristiken von Textdateien auf sich hat. Wenn Sie sich eingehender mit Textformaten beschäftigen – Stichwort Internationalisierung –, so begegnen Sie immer wieder Begriffen, wie **Language Id**, **Locale Identifier** (**LCID**), **Codepage**, **Unicode** und **Character Set**. Jeder hat seine eigene Bedeutung und doch hängen alle irgendwie miteinander zusammen. Grundsätzlich geht es dabei nicht nur um Textdateien, sondern generell um den Umgang mit Texten im System. Betroffen sind damit etwa die Tastatur, die Textausgabe auf dem Bildschirm, Schriftarten, oder auch die Interaktion mit dem Internet.

An oberster Stelle steht die Sprache eines Textes, die wiederum einer Gruppe von Sprachen angehören kann. Deutsch etwa gehört zur Gruppe **Westeuropa und USA**. Windows unterstützt von Haus aus mehrere Sprachgruppen und Sprachen. Dabei gibt es einen Unterschied zwischen nur unterstützten und den installierten Sprachen. Während es genau 17 Gruppen gibt, kann die Zahl der zugehörigen Sprachen selbst variieren. Für den Umgang mit Texten ist die Gruppierung weniger interessant, denn sie betrifft vor allem die Tastaturbelegung, die in der Regel pro Sprachgruppe eindeutig ist.

Welche Sprachen Windows unterstützt können Sie mit einem Doppelklick auf die Tabelle **tblLCIDs** der Beispieldatenbank herausfinden (siehe Bild 1). In der zweiten Spalte finden Sie die Sprache, in der dritten das zugehörige Land. Auf die Bedeutung der vierten (**CP**) kommen wir noch zu sprechen. Das Feld **LCID** legt die eindeutige Id der Sprache fest. Diese Zahl werden Sie häufiger unter Windows finden. Schauen Sie etwa in den Order von **Microsoft Office** im Explorer zu Ihrer Office-Version. Dort werden Sie als Unterverzeichnisse die von Ihrem Office unterstützten Sprachen als **LCID** finden. Die **1033** für **Englisch (USA)** ist immer mit dabei, die **1031** für Deutsch wahrscheinlich auch. In diesen Verzeichnissen befinden sprachspezifische Ressourcen für Office. Installieren Sie etwa zusätzlich die Rechtschreibprüfung für Französisch, so entsteht hier ein weiterer Ordner mit dem Namen **1036**.

Ist die Sprache mit ihrer **LCID** noch eindeutig, so ist es das Land nicht. Filtern Sie die Tabelle nach dem Land **Deutschland**, so ergeben sich vier Datensätze, unter denen sich etwa auch das sehr spezielle **Niedersorbisch** befindet.

Der Inhalt dieser Tabelle ist keineswegs in Stein gemeißelt. Je nach

tblLCIDs

LCID	Language	Country	CP
0	(Unbekannt)	(Unbekanntes Land)	0
1025	Arabisch (Saudi-Arabien)	Saudi-Arabien	1256
1026	Bulgarisch (Bulgarien)	Bulgarien	1251
1027	Katalanisch (Spanien)	Spanien	1252
1028	Chinesisch (traditionell, Taiwan)	Taiwan	950
1029	Tschechisch (Tschechische Republik)	Tschechische Republik	1250
1030	Dänisch (Dänemark)	Dänemark	1252
1031	Deutsch (Deutschland)	Deutschland	1252
1032	Griechisch (Griechenland)	Griechenland	1253
1033	Englisch (USA)	Vereinigte Staaten von Amerika	1252
1034	Spanisch (Spanien, traditionelle Sortierung)	Spanien	1252
1035	Finnisch (Finnland)	Finnland	1252
1036	Französisch (Frankreich)	Frankreich	1252
1037	Hebräisch (Israel)	Israel	1255
1038	Ungarisch (Ungarn)	Ungarn	1250
1039	Isländisch (Island)	Island	1252
1040	Italienisch (Italien)	Italien	1252

Bild 1: Inhalt der Tabelle tblLCIDs mit den von Windows unterstützten Sprachen

Windows-Version, -Installationsoptionen und -Sprache kann sie andere Datensätze aufweisen. Sie können die Tabelle auf Ihrem System selbst erstellen. Dazu gibt es im VBA-Modul **mdlTextconverter** der Beispieldatenbank eine Prozedur **EnumLanguageGroupsLocales**. die Sie aus dem Direktfenster des VBA-Editors heraus aufrufen können. Sie löscht zunächst den Inhalt der Tabelle und erstellt ihn dann über verschiedene API-Funktionen neu. Es gibt allerdings keine API-Funktion, die die Sprachen direkt aufzählen könnte. Erst müssen die einzelnen Sprachgruppen ermittelt (**EnumLanguageGroups**), in einer Collection zwischengespeichert und anschließend der Sprach-Enumeration übergeben werden. In Listing 1 sehen Sie den Code der Prozedur, der hier nur exemplarisch angeführt wird.

```
Sub EnumLanguageGroupsLocales()
    Dim vID As Variant

    Set colInfo = New VBA.Collection
    CurrentDb.Execute "DELETE FROM tblLCIDs"
    CurrentDb.Execute "INSERT INTO tblLCIDs(LCID,Language,Country,CP) " & _
        "VALUES (0,'(Unbekannt)','(Unbekanntes Land)',0)"

    EnumLanguageGroups
    For Each vID In colInfo
        EnumLanguageGroupLocales AddrOf(AddressOf LanguageGroupLocalesProc), _
            ByVal CLng(Val("&H" & vID)), ByVal 0&, 0&
    Next vID
End Sub
```

Listing 1: Prozedur zum Füllen der Tabelle tblLCIDs mit Sprachdatensätzen

Denn im Rahmen dieses Beitrags würde es eindeutig zu weit führen, die nicht unkomplizierten und mit API-Aufrufen gespickten Funktionen näher zu erläutern. Und ohne API-Funktionen kommt man allein mit den Bordmitteln der VBA-, Access- und Office-Bibliotheken bei diesem Thema nicht weiter.

Kommen wir zu der vierten Spalte der Tabelle mit dem Namen **CP**, was ein Kürzel für **Codepage** darstellt. Dort ist zu jeder Sprache die Vorgabe-**Codepage** zu finden. Was aber ist eine **Codepage**? Dazu ein Ausflug in die binäre Speicherung von Texten: Für jedes Zeichen wird bei Single-Byte-Formaten (**ANSI, OEM**) genau ein Byte festgelegt. Schreiben Sie ein **A** in einen Text, so finden Sie unter einem Hex-Editor an dieser Stelle die Zahl 65. Genauso gut können Sie auch VBA bemühen:

```
? Asc ("A")   -> 65
? Chr (65) -> "A"
```

Die **65** kennzeichnet, welches Zeichen innerhalb der deutschen Sprach-Codepage zu finden ist. Eine **Codepage** ist damit imgrunde eine Tabelle mit 255 Datensätzen, die jeder Position ein bestimmtes Zeichen zuordnet. Und eben diese Position ist, je nach Sprache, unterschiedlich. So ermitteln Sie etwa für ein **Ü** den Code **220**. In Russland aber kennt man kein **Ü**. Die dort verwendete **Codepage** ordnet der 220 ein anderes, ein kyrillisches Zeichen zu. Ein Text in deutscher Sprache mit Umlauten wird dort falsch dargestellt, wenn er mit dem russischen **Notepad** geöffnet wird. Ein russisches **Word** hingegen wird beim Öffnen der Textdatei nachfragen, welche **Codepage** zu verwenden ist, und erst nach Einstellung von **Westeuropa (Windows)** das korrekte Ergebnis zeitigen.

Die **Codepage** ist also die zentrale Schaltstelle, wenn es um Textverarbeitung geht. Erst sie ermöglicht es, einem Byte-Code ein bestimmtes Zeichen zuzuordnen.

Nun braucht es nicht für jede einzelne Sprache eine gesondert Codepage. Die **ANSI-Codepage** für Westeuropa (**1252**) etwa kennt fast alle benötigten Zeichen für die Sprachen Europas, also die deutschen Umlaute und die französischen oder spanischen Akzente.

Es ist demnach ökonomischer, mehreren Sprachen eine Codepage zuzuweisen. So steht in der **CP**-Spalte der Sprachtabelle für Deutsch, Italienisch, Französisch, Spanisch, Englisch überall die **1252**. Filtern Sie die Tabelle nach diesem Wert: Erstaunlicherweise gibt es über Hundert Sprachen, die sich diese **Codepage** teilen.

Nun kann man sich eine Zahl als Bezeichner für eine **Codepage** nur schwer merken. Deshalb haben Codepages auch Namen. Öffnen Sie die Tabelle **tblCodepages** der Beispieldatenbank (siehe Bild 2). Sie finden dort unter dem Feld **Description** die Bezeichnungen der **Code-**

pages. Die **1251** nennt sich exakt **ANSI - Lateinisch I**. Das ist gut zu wissen, denn eine Bezeichnung mit dem Teilinhalt **Westeuropa** werden Sie nämlich vergeblich suchen.

In der Tabelle sind allerlei kryptische **Codepages** aufgeführt, die zum großen Teil veraltet und nicht mehr in Gebrauch sind.

Sie werden sie unter Umständen dennoch benötigt, wenn Sie es mit Textdateien zu tun haben, die aus IT-Urzeiten stammen, wie die **IBM**-Codes. Eher noch treffen Sie auf Texte von MS-DOS-Systemen, die in der Tabelle durch das Präfix **OEM** gekennzeichnet sind.

Es gibt überdies ein Problem mit diesen Bezeichnungen, die leider nicht so eindeutig sind. Die Bezeichnungen in der Tabelle sind das Resultat einer Windows-API-Funktion, die in der Prozedur **GetCPName** der Beispieldatenbank zum Einsatz kommt:

```
? GetCPName (1252)
- > "(ANSI - Lateinisch I)"
```

Mithilfe dieser Prozedur und derjenigen zum Auflisten der **Codepages** (**EnumCodepages**), aufgerufen aus dem VBA-Direktfenster, können Sie den Inhalt der Tabelle neu erstellen. Wie wir noch sehen werden, gibt es aber auch noch die Möglichkeit, sich über **DAO** und die **Access Database Engine** alle **Codepages** ausgeben zu lassen. Dieses Ergebnis gibt die Tabelle **tblCodepagesDAO** wieder. Und dort steht für die **Codepage 1252** die Bezeichnung **Westeuropäisch (Windows)**. Damit Sie alle Sprachen, die zugehörigen **Default-Codepages** und deren Bezeichnungen im Blick haben, fasst die Abfrage **qry_LCID_CP** die drei besprochenen Tabellen zusammen und gibt sie wie in Bild 3 wieder. Sie sollte genügen, um aus einer Bezeichnung die zugehörige Codepage zu identifizieren. Denn diese Bezeichnungen listen etwa die Kombinationsfelder von Access oder Word auf, wenn es um die Auswahl einer **Codepage** geht.

tblCodePages

CodePage	Description
0	(Unbekannt)
37	IBM EBCDIC - USA/Kanada
437	OEM - Vereinigte Staaten
500	IBM EBCDIC - International
708	Arabisch - ASMO
720	Arabisch- Transparent ASMO
737	OEM - Griechisch 437G
775	OEM - Baltisch
850	OEM - Multilingual Lateinisch I
852	OEM - Lateinisch II
855	OEM - Kyrillisch
857	OEM - Türkisch
858	OEM - Multilingual Lateinisch I + Euro
860	OEM - Portugiesisch
861	OEM - Isländisch
862	OEM - Hebräisch
863	OEM - Französch (Kanada)
864	OEM - Arabisch
865	OEM - Nordisch
866	OEM - Russisch
869	OEM - Modernes Griechisch
870	IBM EBCDIC - Multilingual/ROECE (Lateinisch-2)

Bild 2: Inhalt der Tabelle tblCodepages

qry_LCID_CP

Language	LCID	CP	CodepageNameAPI	CodepageNameDAO
Bulgarisch (Bulgarien)	1026	1251	ANSI - Kyrillisch	Kyrillisch (Windows)
Katalanisch (Spanien)	1027	1252	ANSI - Lateinisch I	Westeuropäisch (Windows)
Chinesisch (traditionell, Taiwan)	1028	950	ANSI/OEM - Chinesisch, traditi	Chinesisch traditionell (Big5)
Tschechisch (Tschechische Republik)	1029	1250	ANSI - Osteuropa	Mitteleuropäisch (Windows)
Dänisch (Dänemark)	1030	1252	ANSI - Lateinisch I	Westeuropäisch (Windows)
Deutsch (Deutschland)	1031	1252	ANSI - Lateinisch I	Westeuropäisch (Windows)
Griechisch (Griechenland)	1032	1253	ANSI - Griechisch	Griechisch (Windows)
Englisch (USA)	1033	1252	ANSI - Lateinisch I	Westeuropäisch (Windows)
Spanisch (Spanien, traditionelle Sortierung)	1034	1252	ANSI - Lateinisch I	Westeuropäisch (Windows)
Finnisch (Finnland)	1035	1252	ANSI - Lateinisch I	Westeuropäisch (Windows)
Französisch (Frankreich)	1036	1252	ANSI - Lateinisch I	Westeuropäisch (Windows)
Hebräisch (Israel)	1037	1255	ANSI - Hebräisch	Hebräisch (Windows)
Ungarisch (Ungarn)	1038	1250	ANSI - Osteuropa	Mitteleuropäisch (Windows)

Bild 3: Alle Sprachen und deren Codepages gibt die Abfrage qry_LCID_CP aus

Zuletzt fehlt noch die Erläuterung des Begriffs **Character Set**, zu Deutsch Zeichensatz. Die Zuordnung der Bytes eines Textes zu bestimmten Zeichen über die Codepage ist das Eine, die visuelle Darstellung dieses Zeichens über Schriften das Andere. Eine Schriftart kann mit der Codepage allein noch nichts anfangen. Der Satz von darstellbaren Zeichen ist durch den Zeichensatz definiert. Die wiederum ist ebenfalls über eine bestimmte Nummer gekennzeichnet. Die Funktion **CharSetFromCP** der Beispieldatenbank nimmt als Parameter eine Codepage entgegen und gibt die ID des zugehörigen Zeichensatzes zurück:

```
? CharSetFromCP(1252)
 -> 0 (Deutsch, System-Zeichensatz)
? CharSetFromCP(1251)
 -> 204 (Russisch, Kyrillisch)
```

Das Ergebnis 0 bedeutet, dass es sich um den Vorgabezeichensatz des installierten Systems handelt. Um kyrillische Zeichen darzustellen, benötigen Sie das CharacterSet 204.

Die Standardschriftarten von Windows, wie **Arial**, **Times New Roman**, **Courier New**, et cetera, enthalten alle diese Zeichen. Die Nummer des Zeichensatzes identifiziert dann, welcher Block von Zeichen zu verwenden ist.

Unicode

Bisher ging es nur im Texte im **Single-Byte-Format**, wobei **ANSI** der Standard ist. Die Beschränkung auf 256 mögliche Zeichen durch ein Byte führt hier zum beschriebenen System mit den **LCIDs** und **Codepages**, welches einige Kopfschmerzen bereiten kann. Um einiges angenehmer ist da das **Unicode**-Format, das jedem Zeichen mehrere Bytes spendieren kann.

Bei **UTF-16** sind das pro Zeichen in der Regel zwei Bytes. Dadurch steigt die Zahl zuordenbarer Zeichen auf 2^16, was für alle Zeichen dieser Welt ausreicht. Um **Codepages** müsste man sich im Prinzip keine Gedanken mehr machen, wenn alle Texte in diesem Format gespeichert wären. Allerdings bedeutet diese Erweiterung noch nicht, dass damit auch die Sprache des Textes bekannt ist und welcher Zeichensatz zur Darstellung kommen soll. Insofern behält das System um **LCIDs**, **Codepages** und **Charactersets** weiterhin seine Bedeutung.

Bei **UTF-8** wird es komplizierter, weil ein Zeichen meist ein Byte belegt, aber bei spezielleren Zeichen bis zu vier. Der Beginn eines Mehr-Byte-Zeichens wird durch bestimmte Flag-Bytes eingeleitet.

Dasselbe gilt für das wenig verbreitete **UTF-7**, welches eine noch komplexere Kodierung vorsieht, denn ein Zeichen wird hier durch nur 7 Bits eines Bytes identifiziert. Infolgedessen sind speziellere Zeichen durch eine ganze Kette von Byte-Anweisungen repräsentiert.

Eigentlich ist das **UTF-8**-Format das optimalste, weil es alle Zeichen darstellen kann, aber wenig Platz benötigt. Weil es jedoch durch variable Byte-Anzahl pro Zeichen etwas schwieriger zu handhaben ist, verwendet VBA ausschließlich das **UTF-16**-Format. Tatsächlich sind alle Strings unter VBA solche Unicode-Strings. Das lässt sich leicht nachweisen. Während die Funktion **Len** die Zahl der Zeichen eines Strings zurückgibt, ermittelt **LenB** die Anzahl der Bytes, die dafür benötigt werden. Sie gibt grundsätzlich das Doppelte aus:

```
? Len ("ACCESS")  -> 6
? LenB ("ACCESS") ->12
```

Intern steht dieser Text mit diesen Bytes im Speicher:

```
65 - 0 - 67 - 67 - 0 - 69 - 0 - 83 -0
- 83 - 0
```

Das macht auch schon deutlich, dass **UTF-16** zur Speichervergeudung neigt. Der häufigste Wert in diesen Strings ist 0. Da Strings in Textfeldern seit Access 2000 ebenfalls grundsätzlich in UTF-16 gespeichert sind, sieht Access die Möglichkeit vor, für diese Felder als Eigenschaft die Unicode-Kompression einzuschalten, um Speicherplatz zu sparen. Die Kompression elimiert die ganzen Nullen und führt intern zu einem Format, das UTF-8 ähnelt.

Es gibt also einige Varianten von **Unicode**, und bei Unicode-Textdateien kommt noch ein weiterer Stolperstein hinzu. Es gibt zum einen solche Dateien, die die Bytes vertauschen. Der String oben sieht dann so aus:

```
0 - 65 - 0 - 67 - 67 - 0 - 69 - 0 - 83
-0 - 83
```

Hier handelt es sich um die LE-Variante (**Low Endian**), während VBA die

BE-Variante (**Big Endian**) nutzt. Zum anderen kann eine Unicode-Datei noch sogenannte **Lead-Bytes** ganz zu Anfang beherbergen, die sie eindeutig als im Unicode-Format kodiert kennzeichnen soll. Das ist im Allgemeinen die Byte-Folge 255, 254 oder 254, 255, je nachdem, ob es sich um **Big Endian** oder **Low Endian** handelt. Bei **UTF-8** stehen hier sogar 3 **Lead-Bytes**.

All dies macht klar, dass auch Textdateien im Unicode-Format Probleme bescheren können. Damit VBA mit Strings korrekt umgehen kann, müssen diese immer in UTF-16 vorliegen. Auch alle Steuerelemente, etwa die Textbox, erwarten dieses Format, sonst kommt es zu seltsamen Zeichenfolgen.

Texte konvertieren

Die wichtigste VBA-Funktion zur Umwandlung von **ANSI** in **Unicode** und zurück ist **StrConv**:

```
? StrConv ("ACCESS", vbUnicode)
-> "A C C E S S"
? StrConv ("ACCESS", vbFromUnicode)
-> "???"
```

Da der übergebene String bereits im Unicode-Format vorliegt, verdoppelt VBA nicht nur das Zeichen-Byte, sondern auch die interne Null, was zu drei Nullen führt, die als Leerzeichen ausgegeben werden. Umgekehrt bildet sich bei der Konvertierung aus Unicode nach ANSI zwar eine gültige Byte-Folge, die aber von VBA nicht angezeigt werden kann, was durch die Fragezeichen deutlich wird. Kombinieren Sie beide Funktionen, so erhalten Sie ein sinnvolles Ergebnis und den Nachweis, dass bei der Konvertierung nach ANSI nichts verloren ging:

```
? StrConv(StrConv("ACCESS", _
        vbFromUnicode),vbUnicode)
- > "ACCESS"
```

Eine Besonderheit von **StrConv** ist, dass es nicht nur Strings als Parameter entgegennimmt, sondern auch Byte-Arrays. Oder aus Byte-Arrays wieder Strings erzeugen kann:

```
Dim bin() As Byte
bin = StrConv("ACCESS", vbFromUnicode)
? UBound(bin)  '-> 5 = 6 Bytes
? StrConv(bin, vbUnicode)
-> "ACCESS"
```

Über Byte-Arrays können Strings so am einfachsten manipuliert werden.

Ein weiterer Parameter von **StrConv**, der optional angegeben werden kann, ist **LCID**. Damit kann eine Sprache festgelegt werden, in die der String konvertiert werden soll und umgekehrt. Das macht es möglich einen String von einer Sprach-Id in die andere umzuwandeln. Die Funktion **ConvertUnicodeToUnicode** (Listing 2) der Beispieldatenbank macht sich das zunutze. Hier wird aus dem übergebenen Text zunächst ein Byte-Array erzeugt. Dabei wird der Parameter **LCID** weggelassen, was bedeutet, dass **StrConv** die Sprach-Id des Systems verwendet und damit einen westeuropäischen Text. Beim Zurückverwandeln in einen String als Rückgabe der Funktion jedoch wird die **LCID** eingesetzt, die als optionaler Parameter übergeben wurde. Rufen Sie die Funktion etwa so auf:

```
? ConvertUnicodeToUnicode( _
"André",1031)
```

Das übersetzt **André** nach Englisch. Heraus kommt genau dasselbe, da Englisch die gleiche Codepage verwendet, wie Deutsch (1252). Übergeben Sie stattdessen 1042, was für **Koreanisch** steht, so gibt die Funktion dies aus:

```
? ConvertUnicodeToUnicode( _
"André",1031)
- > Andr?
```

Offenbar hat sich der String also geändert. Verwechseln Sie den **LCID**-Parameter von **StrConv** nicht mit dem Wert für die **Codepage**!

```
Public Function ConvertUnicodeToUnicode(ByVal sText As String, _
    Optional lLCID As Long) As String
    Dim bin() As Byte

    On Error GoTo Fehler
    If lLCID = 0 Then lLCID = GetSystemDefaultLangID
    bin = StrConv(sText, vbFromUnicode)
    ConvertUnicodeToUnicode = StrConv(bin, vbUnicode, lLCID)
    Exit Function

Fehler:
    MsgBox "Ungültige Sprach-ID (LCID)", vbCritical
End Function
```

Listing 2: Funktion zum Konvertieren eines Textes mithilfe einer Sprach-ID (LCID)

Diese wird aus der **LCID** intern von der Funktion selbst berechnet. Das in Bild 3 gezeigte Abfrageergebnis gibt darüber Aufschluss.

Die Anwendung Textconverter

Damit Sie sich nicht mit zu vielen VBA-Prozeduren und "API-Geraffel" beschäftigen müssen, ist in die Beispieldatenbank das Formular **frmTextconverter** eingebaut, das Ihnen manuelle Programmierarbeit abnehmen kann. Beim Start des Formulars werden Sie gefragt, ob die Beispieltexte zum Beitrag importiert werden sollen. Sie müssten sich im Unterverzeichnis **Textdateibeispiele** befinden. Es handelt sich um einige Dateien, die in den unterschiedlichsten Formaten abgespeichert wurden, um die Funktionsfähigkeit des Formulars zu testen. Bejahen Sie die Nachfrage, so liest eine Prozedur alle Dateipfade in die dem Formular zugrundeliegende Tabelle ein. Mehr geschieht noch nicht. Ein Klick auf den Button **Einlesen** nimmt sich der angezeigten Datei an und liest ihren Inhalt ein, um ihn im Textfeld unten auf den Bildschirm zu bringen (Bild 4). Über den Pünktchen-Button können Sie aber ebenso eine andere Textdatei Ihrer Wahl laden.

Bei Textfeld handelt es sich übrigens nicht um eine Access-Textbox, sondern um ein **MS-Forms-Textfeld**, also ein ActiveX-Steuerelement. MS-Forms ist Bestandteil jeder Office-Installation und damit garantiert verfügbar. Die **MS-Forms-Textbox** hat gegenüber dem Pendant von Access einige Vorteile.

Sie lässt sich ebenso an ein Datenfeld des Formulars binden, aber der Text lässt sich besser scrollen, und vor allem können für die angegebene Schrift nicht nur die üblichen Verdächtigen, wie **Font.Name** und **Font.Size** angegeben werden, sondern auch **Font.Characterset**, was zu unserem Thema besonders passt.

In der Abbildung wird der Text nach dem Einlesen korrekt dargestellt. Kein Wunder, da es sich um einen **ANSI**-Text mit der verwendeten **Codepage** 1252 handelt. Um das zu verifizieren, können Sie nun auf die Schaltfläche **Analysieren** klicken. Dabei wird die Routine **DetectCPFileDAO2** des Moduls **mdlTextconverter** angesprochen, die über einen später zu erläuternden Code die **Codepage** des Textes zu ermitteln versucht – die **Access Database Engine** hält dazu eine verborgene Anweisung bereit. Nach dieser Analyse wird die ermittelte **Codepage** namentlich im Kombinationsfeld rechts angezeigt. Die Datenherkunft dieser Combobox ist über einen SQL-String auf die Tabelle **tblCodepages** eingestellt. Zusätzlich zeigt ein Meldungsfenster an, wie lange die Routine für ihre Analyse benötigt hat. Grund: Es gibt im Modul noch zwei weitere gleichartige Routinen, die für diese Analyse herangezogen werden können. Die eine verwendet eine temporäre Ta-

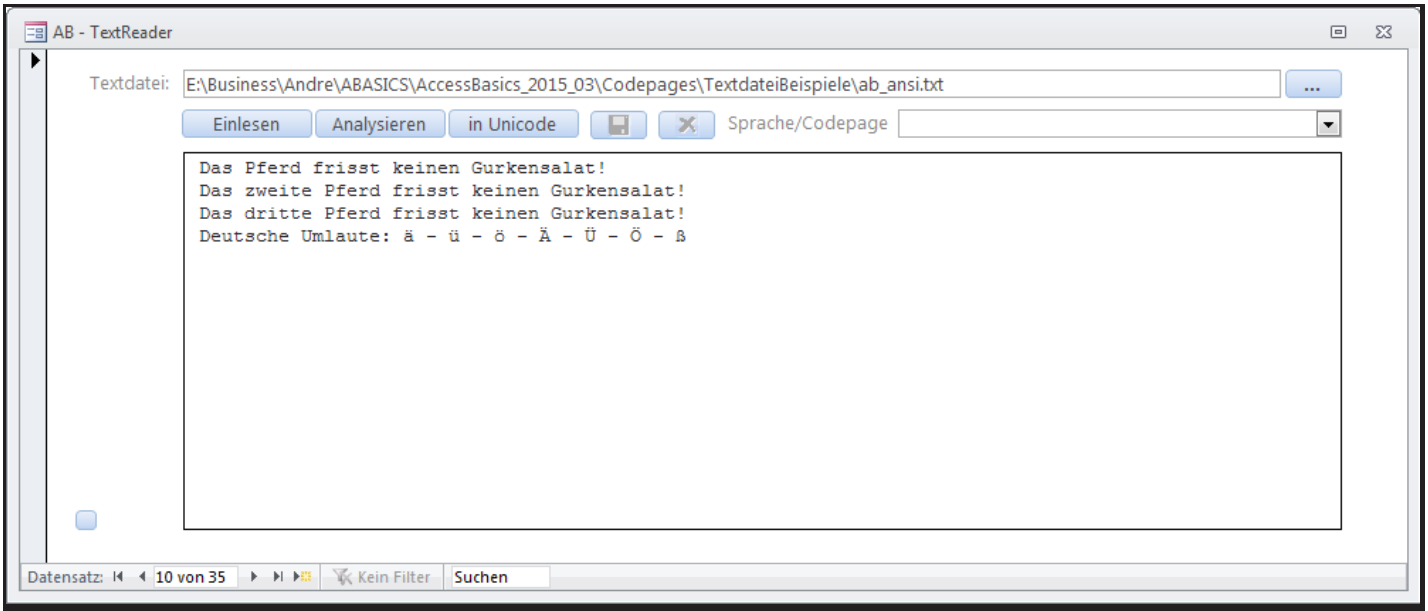


Bild 4: Das Formular frmTextconverter der Beispieldatenbank nach dem Einlesen einer Textdatei

belle über DAO (**DetectCPFileDAO**), die andere ADO (**DetectCPFileADO**) für den gleichen Zweck. Die voreingestellte Routine hingegen arbeitet mit einer temporären Abfrage. Sie können nun in der **Click**-Ereignisprozedur des **Analysieren**-Buttons abwechselnd einer der drei Routinen setzen und schauen, wie es sich mit der Performance verhält. Es zeigt sich, dass die Abfragen-Routine **...DAO2** eindeutig die schnellste ist. Zwar geht es hier nur um Sekundenbruchteile, doch immerhin dauert die Berechnung etwa eine Sekunde, was heutzutage nicht gar so wenig ist. Die Analyse-Methoden der **Access Database Engine** müssen hier ja aufwändigere statistische Algorithmen ausführen.

Nach dem Analysieren der **Codepage**, die in unserem dargestellten Fall korrekt die **1252** ermittelte, wird der Button **in Unicode** aktiv. Ein Klick auf ihn konvertiert den Originaltext mithilfe der **Codepage** in das Unicode-Format und bringt diesen im Textfeld zur Ansicht. Wenn die Darstellung nun immer noch korrekt ist, können Sie auf den Speichern-Button gehen, wodurch der Unicode-Text in der Tabelle **tblText** als konvertierter Text gespeichert wird (Bild 5). Wenn Sie später wieder auf diesen Datensatz gehen, wird automatisch dieser konvertierte Text im Formular angezeigt. Sie können ihn aber auch wieder über den Button mit dem Löschen-Symbol entfernen. Der Originaltext bleibt auf jeden Fall erhalten, wie auch die ermittelte **Codepage**. Gehen wir an dieser Stelle auf den Aufbau der Tabelle **tblText** ein, deren Entwurf in Bild 6 zu sehen ist.

File ist das Feld, welches den Pfad zur Textdatei aufnimmt. **TextOriginal** wird beim Einlesen der Datei gefüllt und erhält damit deren Inhalt. Es macht Sie stutzig, dass hier der Datentyp **OLE-Objekt** gewählt wurde? Wie bereits ausgeführt, speichert Access Texte immer im **UTF-16**-Format. Wir möchten jedoch die Textdatei binär Byte für Byte wie im Original abspeichern, und dazu eignet sich als Binär-Container ein OLE-Feld am besten. In der Datenblattansicht erscheinen die Texte dann lediglich mit dem Vermerk **Long Binary-Daten**. (Der Begriff **OLE-Objekt** ist veraltet und verweist auf Zeiten, in denen dieser Feldtyp für die tatsächliche Speicherung von OLE-Objekten, also Automationsobjekten, vorgesehen war. Tatsächlich handelt es sich dabei nur um einen Container für Byte-Arrays, weshalb der Begriff **LongBinary** eigentlich angemessener wäre, obwohl Access durchaus zwischen Binärdaten und OLE-Daten in diesem Feld unterscheidet.)

TextConverted wird beim Konvertieren des Textes nach Unicode im Formular gefüllt. Hier wird ein Memo-Feld verwendet, das dann auch in der Datenblattansicht der Tabelle den lesbaren Text anzeigt. Und **Codepage** enthält nach Auslösen des **Analysieren**-Buttons oder der Auswahl über das Kombinationsfeld die ermittelte oder gesetzte **Codepage** als Wert.

Über dieses Formular der Anwendung können Sie bequem alle Textdateien testen und in für Access akzeptables **Unicode** bringen. Die Analysefunktion ist dabei nur eine

tblText

Feldname	Felddatentyp
ID	AutoWert
File	Text
TextOriginal	OLE-Objekt
TextConverted	Memo
Codepage	Zahl

Bild 6: Entwurfsansicht der Formulartabelle tblText

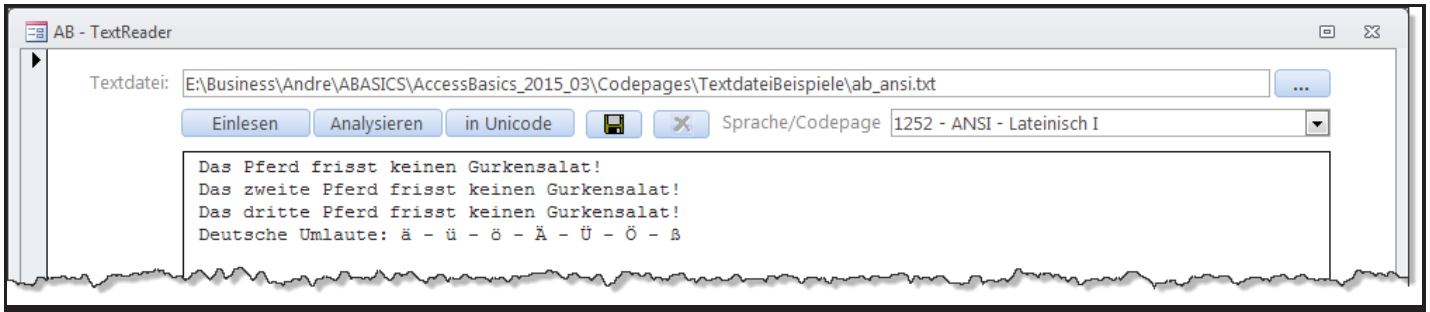


Bild 5: Das Formular frmTextconverter der Beispieldatenbank nach dem Konvertieren in Unicode per ermittelter Codepage

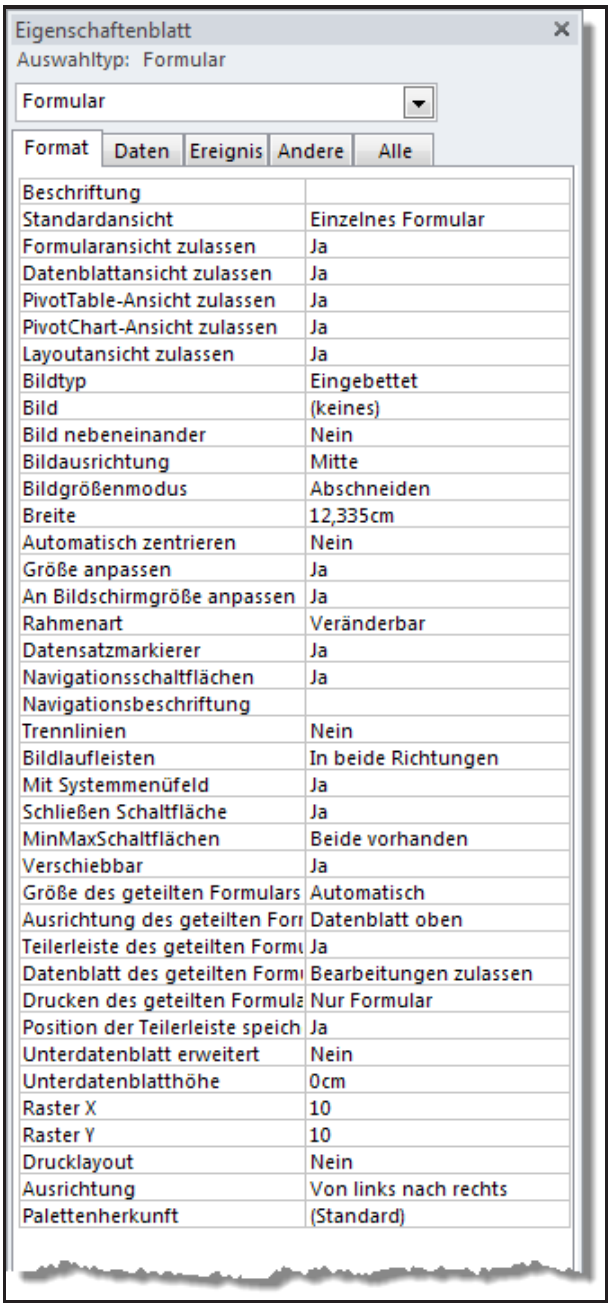
Navigieren über Formulartricks

Die Voreinstellungen, die Access für ein neues Formular im Entwurf anbietet, sind für die üblichen Standardmasken in Anwendungen sicher sinnvoll. Durch Modifikation dieser Einstellungen bieten Formulare aber noch ganz andere Möglichkeiten – so etwa zu Navigationszwecken, wie in der hier vorgestellten Lösung.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1504_Formulartricks.accdb**

Formulareinstellungen
Erzeugen Sie ein neues leeres Formular in Ihrer Datenbank, so versieht Access es mit Standardeinstellungen für das Layout und Format. Unter Access 2010 sind diese im **Eigenschaftenblatt** des Entwurfs zu finden (Bild 1). Da es einige Mühe macht, diese Einstellungen jeweils neu anzupassen, finden Sie sie so auch in vielen Datenbanken wieder. Nehmen wir aber einmal einige der wichtigsten Elemente genauer unter die Lupe.

Grundsätzlich ist die **Standardansicht** auf **Einzelnes Formular** gestellt. Das dürfte auch der am häufigsten gebrauchte Typ sein. Fraglich ist, ob die anderen Ansichtstypen wirklich zugelassen sein sollten. Wer braucht schon in einer Adressenverwaltung die **Pivot-Ansichten**? Erst recht gilt dies für die ab Access 2007 eingeführte **Layoutansicht** mit ihrer Mischung aus Entwurf und Formularansicht, die bis auf seltene Fälle eher der Entwicklungsphase vorbehalten sein dürfte. Die **Endlosansicht** jedoch wird immer wieder gebraucht.



Beschriftung	
Standardansicht	Einzelnes Formular
Formularansicht zulassen	Ja
Datenblattansicht zulassen	Ja
PivotTable-Ansicht zulassen	Ja
PivotChart-Ansicht zulassen	Ja
Layoutansicht zulassen	Ja
Bildtyp	Eingebettet
Bild	(keines)
Bild nebeneinander	Nein
Bildausrichtung	Mitte
Bildgrößenmodus	Abschneiden
Breite	12,335cm
Automatisch zentrieren	Nein
Größe anpassen	Ja
An Bildschirmgröße anpassen	Ja
Rahmenart	Veränderbar
Datensatzmarkierer	Ja
Navigationsschaltflächen	Ja
Navigationsbeschriftung	
Trennlinien	Nein
Bildlaufleisten	In beide Richtungen
Mit Systemmenüfeld	Ja
Schließen Schaltfläche	Ja
MinMaxSchaltflächen	Beide vorhanden
Verschiebbar	Ja
Größe des geteilten Formulars	Automatisch
Ausrichtung des geteilten Forr	Datenblatt oben
Teilerleiste des geteilten Formu	Ja
Datenblatt des geteilten Form	Bearbeitungen zulassen
Drucken des geteilten Formula	Nur Formular
Position der Teilerleiste speich	Ja
Unterdatenblatt erweitert	Nein
Unterdatenblatthöhe	0cm
Raster X	10
Raster Y	10
Drucklayout	Nein
Ausrichtung	Von links nach rechts
Palettenherkunft	(Standard)

Bild 1: Diese Einstellungen für das Layout spendiert Access 2010 einem neuen Formular

Dass **Automatisch zentrieren** auf Nein steht, ist dem Standardverhalten von Fenstern in MDI-Anwendungen unter Windows geschuldet: Öffnen Sie mehrere Formulare gleichzeitig, so werden diese versetzt auf dem Arbeitsbereich positioniert, so dass auf alle zugegriffen werden kann. Selten sind Access-Formulare jedoch so klein, dass mehrere davon auf dem Bildschirm Platz hätten. Für größere Formulare stellen Sie die Zentrierung auf dem Bildschirm also auf **Ja**.

Datensatzmarkierer und **Navigationsschaltflächen** sind ebenfalls immer aktiviert. Während die **Navigationsschaltflächen** sicher nützlich sind, wenn das Formular nicht nur zur Eingabe oder Anzeige eines einzigen Datensatzes ausgelegt ist, so ist der **Datensatzmarkierer** in produktiven Anwendungen wieder fraglich. In der Formularansicht erkennen viele Anwender nicht, wofür er da ist, und eine Fehlbedienung durch versehentliches Betätigen der Entfernen-Taste bei markiertem Datensatz kommt immer wieder vor. In

Formularen, wie dem **Switchboard** der Beispieldatenbank zum Beitrag, sind alle beide überflüssig.

Dass ein Formular eine **Schließen-Schaltfläche** aufweist, ist immer dann erforderlich, wenn Sie es nicht per VBA entladen möchten. **Min-MaxSchaltflächen** hingegen braucht man in der Regel nicht. Formulare sind üblicherweise auf eine bestimmte Größe designet, so dass ein Maximieren keinen Vorteil bringt. Damit wären wir auch schon bei der Einstellung für die **Rahmenart**, die im Standard auf **Veränderbar** steht. Sie können es dann zur Laufzeit mit der Maus in der Größe ändern. In der **Formularansicht** ist auch das nicht notwendig. Eher verwenden Sie hier die Einstellungen **Dialog** oder **Dünn**, die zum gleichen Ergebnis führen: Die Titelleiste des Fensters ist sichtbar – und damit auch die Schließen-Schaltfläche –, die Abmessungen lassen sich aber nicht per Maus ändern. Schalten Sie den Modus **Keine** für die Rahmenart ein, so verschwindet die Titelleiste komplett. Das Formular kann dann nur per VBA geschlossen werden. Das Formular **frmIntro** der Beispieldatenbank, ein **Splash-Screen,** etwa macht davon Gebrauch. Erst ein Klick auf den Detailbereich des Formulars löst eine Ereignisprozedur aus, die das Formular über **DoCmd.Close** entlädt.

Interessant, aber wenig beachtet, ist die Eigenschaft **Verschiebbar** eines Formulars. Normalerweise können Sie es zur Laufzeit mit der Maus über die Titelleiste an die gewünschte Position ziehen. Ist **Verschiebbar** deaktiviert, so ist dies jedoch in keiner Ansichts- oder Rahmenart mehr möglich. Das Formular bleibt fest im Arbeitsbereich angepinnt. Dennoch können Sie es über VBA und die Methode **Move** bewegen, doch dazu später mehr.

Die **Bildlaufleisten** sind im Standard zwar eingeschaltet, aber in der Formularansicht bis auf Ausnahmefälle unnötig. Das wären Formulare, deren Inhalt so umfangreich ist, dass er nicht auf den Bildschirm passt und nur über Scrollen erreicht werden kann. Im Interesse von guter Ergonomie ist ein solches Verhalten weniger.

Ein Teil der Formulareinstellungen schließlich ist gar nicht im jeweiligen **Eigenschaftenblatt** zu finden, sondern in den **Optionen** der Datenbank. Klicken Sie im Ribbon auf **Datei | Optionen** und wählen die Abteilung **Aktuelle Datenbank** aus. Ein Ausschnitt der Einstellungen ist hier in Bild 2 zu sehen. **Überlappende Fenster** ist der übliche MDI-Modus, wie Sie ihn aus anderen Mehrfensteranwendungen kennen. Das **Registerkartenformat** jedoch führt zu einer Anordnung der Fenster mit **Tabs**. Die Einstellung **Automatisch zentrieren** eines Formulars etwa wird damit ignoriert, denn es bleibt immer oben am Arbeitsbereich angeheftet. Ebenso lässt sich hier auch die **Layoutansicht** deaktivieren, was der entsprechenden Eigenschaft **Layoutansicht zulassen** eines Formulars übergeordnet ist.

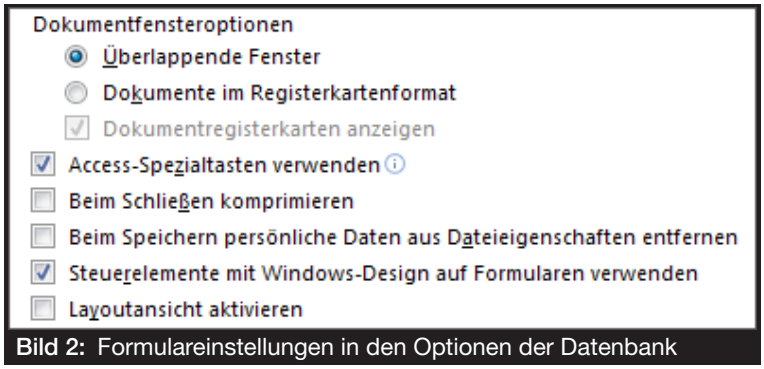


Bild 2: Formulareinstellungen in den Optionen der Datenbank

Nicht verschiebbare Fenster

Ist die Eigenschaft Verschiebbar eines Formulars auf Nein gesetzt, so bleibt es zur Laufzeit, wie erwähnt, an einer festen Position angenagelt. Selbst mit einer Titelleiste ausgestattet reagiert es nicht auf ein Ziehen mit der Maus. Das eröffnet die Möglichkeit, mehrere Formulare nach Belieben auf dem Arbeitsbereich zu positionieren, ohne dass sie sich überlappen oder der Anwender die Anordnung beeinflussen kann.

Wie aber bringt man ein Formular an die gewünschte Stelle? Dazu sieht VBA die recht einfache Methode **Move** vor, die übrigens nicht nur die Formularklassen, sondern auch alle Steuerelemente aufweisen:

```
Form1.Move Links, Oben, Breite, Höhe
```

Der Clou der Anweisung besteht darin, dass sämtliche Parameter optional sind. Um ein Fenster etwa 100 Pixel vom linken Rand des Arbeitsbereichs unterzubringen, setzten Sie einfach dies ab:

```
Form1.Move  100*15
```

Der Faktor 15 übersetzt hier die Pixelmaße in die

Twips-Einheiten, welche Access immer erwartet. Soll das Formular oben bündig abschließen, so genügt ein

```
Form1.Move 100*15, 0
```

Und sollen Breite und Höhe zusätzlich vorgegeben werden, käme diese Anweisung infrage:

```
Form1.Move 1500, 0, 400*15, 300*15
```

Diese Methode können Sie etwa im **Beim Laden**-Ereignis (**Form_Load**) absetzen, aber auch an jeder anderen Stelle des Formular-Codes.

Versuchen wir mit diesem Wissen ein **Switchboard** zu realisieren, das dem Navigationsbereich von Access ähnelt.

Navigationsmenü per Formular

Das Formular **frmNavi** der Beispieldatenbank sieht zur Laufzeit aus, wie in Bild 3. Für das Format ist im Eigenschaftenblatt die Rahmenart **Keine** angegeben, **Datensatzmarkierer** und **Navigationsschaltflächen** wurden auf **Nein** gesetzt, und der Modus **Verschiebbar** ist deaktiviert. Damit das Formular links oben angeheftet bleibt, wird im Ereignis **Beim Laden** diese Code-Zeile ausgeführt:

```
Me.Move 0, 0, 1500
```

Das bewirkt, dass die Positionen für Links und Oben auf 0 eingestellt und die Breite auf 100 Pixel festgelegt werden.

ID	Caption	Function	Sort	Type	Ribbon	BackColor
1	Kunden	frmKunden	1	1	☐	13820902
2	Bestellungen	frmBestellungen	2	1	☐	11193849
3	Rechnungen	frmRechnungen	3	1	☑	11525325
4	Einstellungen	frmEinstellungen	4	1	☐	14013909
6	Kundenbericht	rptKunden	6	2	☐	8446207
9	Kundenbericht 2	rptKunden2	7	2	☑	8446207
12			19	0	☐	15849926
5	Testfunktion	fuTest	20	3	☐	8446207
8			98	0	☐	15849926
7	Schließen	fuQuit	99	3	☐	6455551
(Neu)					☐	

Bild 4: Die Tabelle tblNavi mit den Werten zur Steuerung des Menüformulars

Die Höhe bleibt von der Anweisung unberührt, weil der vierte Parameter fehlt.

Die beschrifteten Schaltflächen sollen nicht fest im Formular angelegt sein, sondern sich aus den Daten einer Tabelle **tblNavi** (siehe Bild 4) speisen, damit sich das Menü über diese steuern lässt. Leider lässt Access das Anlegen von Steuerelementen zur Laufzeit nicht zu. Deshalb ist für die Standardansicht des Formulars nicht die **Formularansicht** ausgewählt, sondern die **Endlosansicht**.

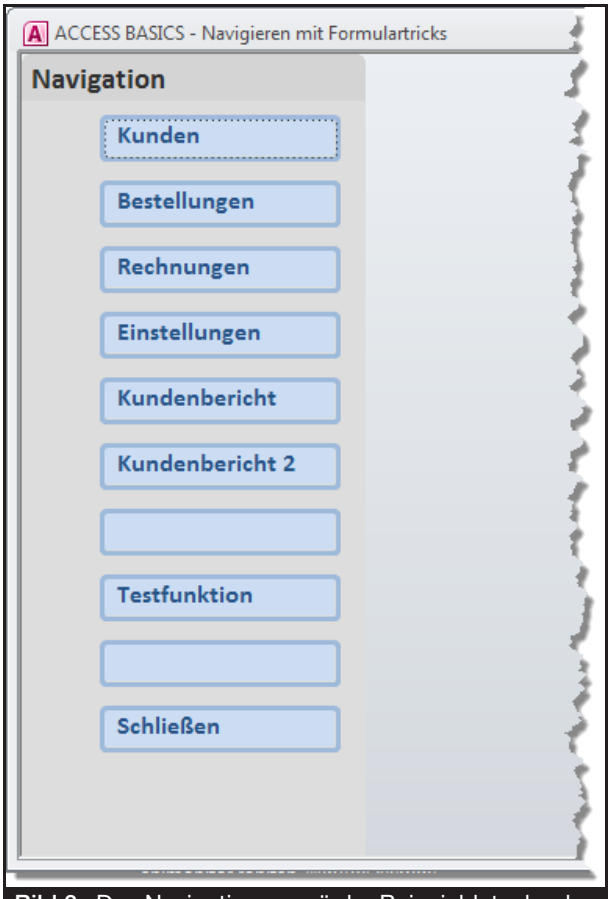


Bild 3: Das Navigationsmenü der Beispieldatenbank

Kommen wir zur Bedeutung der einzelnen Felder der Tabelle. Unter **Caption** wird angegeben, welche Beschriftung eine Schaltfläche des Menüs haben soll. **Function** ist eine Parameterangabe, die dem VBA-Code später mitteilt, welches Objekt beim Klick auf eine Schaltfläche geöffnet wird. Die vertikale Reihenfolge der Schaltflächen wird über den Zahlenwert unter **Sort** festgelegt, nach der über eine Abfrage die Datensätze sortiert werden. **Type** gibt an, welche Aufgabe ein Klick auf den Button ausführt. Dabei steht die 1 für **Formular öffnen**, die 2 für **Bericht öffnen** und 3 für das Auslösen einer selbst angelegten VBA-Prozedur. In **Function** findet sich dabei der Name des

Formulars, Berichts oder der Funktion. Im **Bool**-Feld **Ribbon** aktiviert ein Häkchen, dass nach dem Klick der Ribbon von Access eingeblendet wird. Beim Starten der Anwendung wird dieser nämlich zunächst ausgeblendet. Schließlich können Sie dem Feld **BackColor** jeweils noch einen RGB-Wert spendieren, der die Hintergrundfarbe des Buttons nach dem Klick bestimmt. Den hier benötigten Long-Wert erhalten Sie etwa über das VBA-Direktfenster und die Anweisung

```
? RGB( Rotanteil, Grün, Blau)
```

Diese Tabelle dient als Datenherkunft des Endlosformulars, wobei sie allerdings über eine Abfrage sortiert wird:

```
SELECT * FROM tblNavi ORDER BY Sort;
```

Der Formularentwurf

Nun kann eine Schaltfläche in Access nicht direkt an ein Datenfeld gebunden werden. Zwar könnten Sie per VBA im Ereignis **Beim Anzeigen**, die in einem Endlosformular für jeden Datensatz automatisch ausgelöst wird, die Caption-Eigenschaft der Schaltfläche aus der Datenherkunft setzen, doch dann würden alle Buttons hintereinander dieselbe Beschriftung erhalten, nämlich die des letzten Datensatzes – denn die einzelnen Instanzen des Buttons leiten sich alle von einem Objekt ab. Also scheidet dieses Verfahren aus. Greifen wir deshalb zu einem Trick, der vielleicht in der Entwurfsansicht in Bild 5 deutlich wird. Ganz oben ist der Formularkopf eingeblendet, der ein Bezeichnungsfeld enthält, das schlicht mit **Navigation** beschriftet

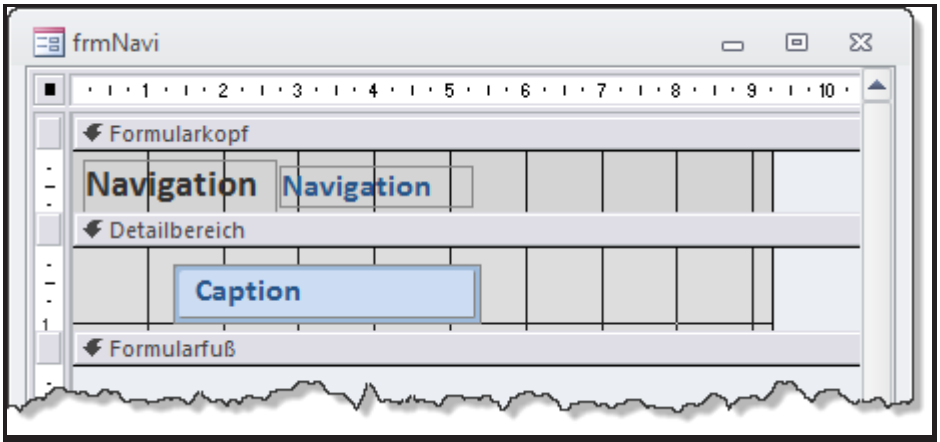


Bild 5: Das Formular frmNavi in der Entwurfsansicht

ist. Rechts daneben befindet sich ebenfalls ein Bezeichnungsfeld **LblTitle** mit der gleichen Beschriftung, das aber zur Laufzeit unsichtbar ist. Auf dessen Bedeutung kommen wir noch zu sprechen.

Der Detailbereich, welcher sich in einem Endlosformular für jeden Datensatz wiederholt, ist mit einer Schaltfläche bestückt, deren Hintergrund scheinbar hellblau daherkommt. Tatsächlich aber – und das ist der Trick! – sieht man hier den Hintergrund eines Textfeldes, welches direkt hinter der Schaltfläche **cmdFunction** liegt und die gleichen Abmessungen aufweist. Die Hintergrundart der Schaltfläche selbst hingegen ist auf Transparent gestellt. Und dennoch reagiert sie auf einen Klick in den Hintergrund des Textfeldes.

Über das Textfeld ist die Möglichkeit der Datenbindung an das Feld **Caption** gegeben, so dass das Ganze zur Laufzeit so wirkt, als hätte man lediglich eine blaue Schaltfläche vor sich. Damit das Textfeld dann nicht etwa den Mauszeiger als Textcursor entgegennimmt, ist seine Eigenschaft **Aktiviert** auf **Nein** eingestellt.

Bedingte Formatierung

Bevor die Programmierung des Formulars erläutert wird, gehen wir noch auf ein Spezialverhalten der Schaltfläche ein. Beim Klick auf sie nimmt sie die Farbe an, die in der Tabelle für BackColor hinterlegt ist. Dabei kommt diese Funktionalität fast ohne VBA-Code aus, indem das Textfeld hinter der Schaltfläche mit einer **Bedingten Formatierung** ausgestattet wird. Markieren Sie dazu das Textfeld **txtFunction** und wechseln im Ribbon auf den Bereich **Format | Steuerelementformatierung**. Dort ist der Button **Bedingte Formatierung** aktiv. Klicken Sie ihn an, um den Dialog für die Festlegung der Bedingungen zu öffnen. In Bild 6 ist die entsprechende Bedingungszeile bereits hinterlegt. Entstanden ist Sie über das Hinzufügen einer Bedingung über die Schaltfläche **Neue Regel**, welche den Dialog in Bild 7 lädt. Die Berechnungsregel lautet hier im Prinzip so:

```
Feldwert = [LblTitle].Beschriftung
```

Das bedeutet im Klartext, dass die unter der Zeile eingestellte Formatierung, nämlich gelber Hintergrund, immer dann zur Geltung kommt,

wenn der Wert des Datenfeldes **Caption**, das dem Textfeld ja zugrunde liegt, genau dem Wert für die Eigenschaft **Beschriftung** des **Label**-Steuerelements **LblTitle** entspricht. Und dieses befindet sich unsichtbar im Formularkopf.

Das mag ihnen spanisch vorkommen, denn das **Label** hat ja die feste Beschriftung **Navigation**. Und da es in der Tabelle keinen einzigen Eintrag gibt, der unter **Caption** diese Bezeichnung führt, dürfte die Bedingung ja nie zutreffen, was die gelbe Formatierung nie auslösen würde. Tatsächlich braucht es hier doch noch etwas VBA-Code, um der Sache auf die Sprünge zu helfen.

In Listing 1 steht die Prozedur für das Ereignis **Beim Anzeigen** des Formulars. Sie wird dann aufgerufen, wenn ein Datensatz des Formulars aktiviert wird, was etwa beim Klicken auf eine Schaltfläche des Detailbereichs passiert. In der ersten Zeile wird dann der Wert des Datenfeldes Caption dem versteckten Label **LblTitle** im Formularkopf zugewiesen. Und schon trifft die Regel für die **Bedingte Formatierung** zu! Und zwar nur für diese eine Schaltfläche beziehungsweise deren unterlegtes Textfeld. Denn alle anderen Textfelder des Endlosformulars haben ja einen anderen Wert für **Caption** eingetragen. Die **Bedingte Formatierung** tritt indessen nicht sofort in Kraft, sondern wird in der letzten Zeile über eine **Requery** des Textfelds aktiv.

Nun erschiene die Schaltfläche in leuchtendem Gelb. Da wir ihr jedoch die in der Tabelle durch **BackColor** angegebene Hintergrundfarbe verabreichen wollten, muss zuvor noch in der zweiten Zeile die **Bedingte Formatierung** per VBA geändert werden. Über **FormatConditions(0)** gelangen wir an die einzige Regel des Textfelds **txtFunction**. Ihre Eigenschaft **BackColor** wird mit dem gleichnamigen Wert aus der Tabelle versehen. Und damit wird das Gelb durch diese neue Farbe ersetzt, ohne dass die Regel selbst sich ändert.

Das Zusammenwirken von **Bedingter Formatierung** mit VBA ist hier sicher nicht so leicht nachzuvollziehen, zeigt aber anschaulich, was sich mit den Bordmitteln von Access ohne

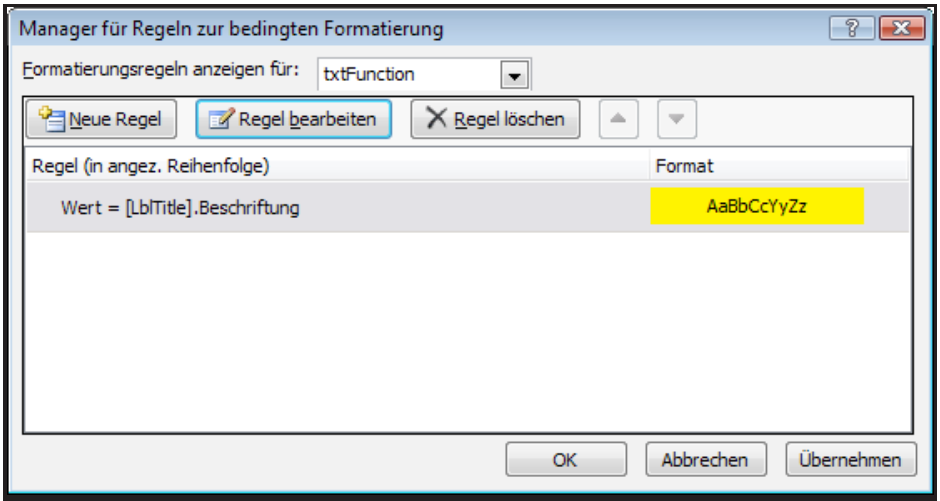


Bild 6: Dialog zur Bedingten Formatierung des Textfelds txtFunction

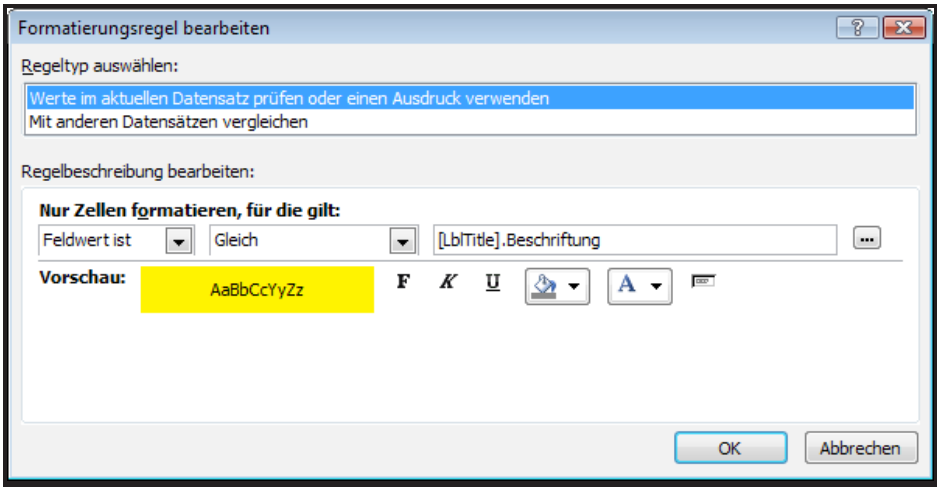


Bild 7: Eingabe der Berechnungsregel für die Bedingte Formatierung

```
Private Sub Form_Current()
    Me!LblTitle.Caption = Me!Caption.Value
    Me!txtFunction.FormatConditions(0).BackColor = Me!BackColor.Value
    Me!txtFunction.Requery
End Sub
```

Listing 1: Ereignisprozedur Beim Anzeigen des Formulars

Nützliche Helferlein per VBScript

Das Entwickeln von Access-Anwendungen ist das eine, der Umgang mit der Peripherie derselben das andere. So will Ihre Anwendung erst verteilt werden, oder Sie benötigen Systemaktionen, die sich unter Access nicht so ohne weiteres ausführen lassen. Dann könnten Ihnen möglicherweise extern gestartete Skripte weiterhelfen. Solche Skripte schauen wir uns in diesem Beitrag an.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1504_ScriptingDemo.zip** und den enthaltenen Dateien.

Script-Sprachen

Wenn Sie sich nach Möglichkeiten umsehen, um abseits von Access- und VBA-Programmierung unter Windows Makros auszuführen, so bieten sich etwa **Batch-Dateien** mit der Endung **bat** oder **cmd** an, die Kommandozeilenaktionen starten. Viel Freude werden Sie dabei wahrscheinlich nicht haben, da der Sprachumfang reichlich beschränkt ist und zudem die Kenntnis zahlreicher Kommandozeilenparameter erfordert. Erheblich ausgefeilter kommt da die **Windows-Powershell** daher, über die sich fast alles realisieren lässt. Sie verlangt Ihnen jedoch noch mehr Einarbeitung ab, die zudem wegen des objektorientierten Ansatzes nicht gerade leicht fällt.

So bleibt als leistungsfähiger Ausweg das standardmäßig unter Windows installierte **Visual Basic Scripting**, kurz **VBScript**. Das Gute an der Sache: Wenn Sie sich halbwegs mit VBA-Programmierung auskennen, so wird ihnen das Erstellen von VBScripten leicht von der Hand gehen, denn Syntax und Sprachumfang sind nahezu identisch. Auf einige Unterschiede gehen wir noch ein.

VBScripte

VBScript-Dateien sind einfache Textdateien mit der Endung **vbs**, die VBA-Code enthalten und von der **Windows-Scripting-Engine** ausgeführt werden. Diese Dateiendung ist automatisch mit einem der beiden Script-Prozessoren **wscript.exe** oder **cscript.exe** verknüpft.

Doppelklicken Sie also auf eine **vbs**-Datei, so startet in der Regel die **wscript.exe**, welche die Anweisungen dann abarbeitet. Dabei nimmt sich die Engine den Code zunächst genau unter die Lupe und testet ihn auf Syntaxfehler. Im Fehlerfall wird eine entsprechende Meldung geöffnet. Danach kompiliert sie ihn und startet die Prozedur. Aber auch dann, wenn die Syntax stimmt, kann es zu Fehlern etwa infolge falscher Parameter oder inkorrekter Objektansprache kommen. Dann ereignet sich ebenfalls eine Fehlermeldung und das Script wird an dieser Stelle abgebrochen.

An dieser Stelle ein Hinweis: VBScript ist an sich noch kein Sicherheitsrisiko, arbeitet jedoch nicht in abgeschotteter Umgebung. Zwar können Windows-API-Funktionen nicht aufgerufen – hier ein Unterschied zu VBA –, dafür aber beliebige ActiveX-Komponenten erzeugt werden, die ihrerseits allerlei Unbill hervorzubringen imstande wären. Aus diesem Grund werden die Script-Engines von Administratoren zuweilen gesperrt und die Ausführung von **vbs**-Dateien verhindert. Inzwischen kommt das immer seltener vor, denn über VBA-Programmierung, die ja Bestandteil von beliebigen Access-, Word- oder Excel-Dateien sein kann, sind noch viel mehr unheilvolle Aktionen möglich. Und diese Dateien werden ja auch nicht gesperrt.

VBScripte erzeugen

Nicht einfacher, als das! Legen Sie eine neue Textdatei an und speichern sie unter der Endung **vbs** ab. Schreiben Sie nur diese eine Zeile in sie:

```
Msgbox "Hallo! Bin ein VBScript!"
```

Doppelklicken Sie auf die dergestalt modifizierte Datei, und schon werden Sie mit dem Meldungsfenster beglückt, wenn Scripting nicht auf Ihrem System deaktiviert ist.

Die Tatsache, dass es sich nur um Textdateien handelt, eröffnet die Möglichkeit, Scripte auch aus Access heraus zu erstellen. Dazu wird der Code in eine String-Variable geschrieben und diese in das Dateisystem abgespeichert, wie etwa in der Prozedur in Listing 1. Der Variablen **sScript** wird die Codezeile von oben zugewiesen und über die **Open**-Anweisung eine Datei mit dem Namen **test.vbs** angelegt, wobei als Pfad

das Verzeichnis der Datenbank (**CurrentProject.Path**) verwendet wird. Anschließend wird die **vbs**-Datei über die Script-Engine (**wscript.exe**) und die **Shell**-Anweisung aufgerufen.

Rufen Sie einfach die Prozedur **CreateVBScript** aus dem VBA-Direktfenster heraus auf und überzeugen Sie sich vom Erfolg der Routine.

```
Sub CreateVBScript()
    Dim sScript As String
    Dim sVBSFile as string

    sVBSFile = CurrentProject.Path & "\test.vbs"
    sScript = "Msgbox ""Ich bin ein VBScript!"""
    Open sVBSFile For Binary As #1
    Put #1, , sScript
    Close #1
    DoEvents
    Shell "wscript.exe " & sVBSFile
End Sub
```

Listing 1: Aus VBA heraus erzeugte VBS-Datei

Anwendungsfälle

Warum sollte man aus Access heraus ein **VBScript** aufrufen? Schließlich lässt sich ja über VBA dasselbe erreichen, wie über ein externes Skript. Das stimmt jedoch nicht immer. Manche Aufgaben können nur über administrative Rechte erledigt werden – etwa das Setzen von bestimmten **Registry**-Schlüsseln. Und da Access nun Mal nur unter normalen Benutzerrechten ausgeführt wird, kann derlei nicht vollzogen werden. Von der Lösung, Access dennoch als Administrator auszuführen, sollten Sie besser die Finger lassen, weil dann die Add-Ins von Access und VBA durcheinander geraten können.

Hier hilft ein **VBScript** aus, das Aufgaben unter erhöhten Benutzerrechten ausführt. Den dafür benötigten Code schauen wir uns später an.

Oft gebraucht wird ein Skript, das Ihre Datenbank überhaupt erst installiert. Nehmen wir an, Sie hätten alle Teile Ihrer Anwendung in einer Zip-Datei untergebracht. Da wäre es schön, wenn ein Script auf einem Zielrechner etwa folgende Aufgaben erledigen würde:

Entpacken des Archivs; Anlegen von Zielverzeichnissen; Kopieren von Dateien in die Zielverzeichnisse; Setzen des Vertrauenswürdigen Orts für die Datenbank; Anlegen einer Verknüpfung auf dem Desktop; und eventuell noch das Registrieren benötigter ActiveX-Komponenten. Auch hierfür gibt es ein umfangreiches Beispiel-Skript, welches auf den folgenden Seiten erläutert wird.

VBScript vs. VBA

Unter **VBScript** brauchen Sie, wie wir weiter oben bereits gesehen haben, keine Prozedur zu deklarieren. Das Script selbst stellt schon eine Prozedur dar. Sie können gleich zu Beginn im Code mit der ersten Anweisung beginnen. Es ist Ihnen jedoch unbenommen, trotzdem Prozeduren auf die gleiche Weise anzulegen, wie unter VBA. Solche **Sub**- oder **Function**-Prozeduren sind dann sinnvoll, wenn sie als Unterroutinen im Code öfters aufgerufen werden.

Variablen müssen unter **VBScript** nicht deklariert werden. Dies kommt dem Verhalten von VBA gleich, wenn kein **Option Explicit** im Modulkopf steht. **VBScript**-Variablen sind immer vom Typ **Variant**, was aber nicht bedeutet, dass ihr Sub-Typ nicht von Belang wäre. Funktionen etwa, die einen String als Parameter erwarten, kann kein Long-Wert übergeben werden. So eine Konversion nimmt VBS nicht automatisch vor.

Wenn Sie es gewohnt sind, unter VBA in For..Next-Schleifen die Zählervariable namentlich einzusetzen, so müssen Sie unter VBScript davon Abstand nehmen.

```
For i = 0 To 10
...
Next i
```

führt zu einem VBScript-Syntaxfehler. Korrekt wäre

```
For i = 0 To 10
...
Next
```

Darüber hinaus unterstützt **VBScript** nicht die Deklaration von **API**-Funktionen. Windows- oder andere DLLs können damit nicht angesprochen werden.

Außerdem fehlt ausgerechnet eine ganze Reihe von Datei-Anweisungen, wie **Dir**, **FileCopy**, **FileLen** oder **FileDateTime**. Auch der Befehl **Shell** zum Ausführen anderer Anwendungen existiert nicht. Solche Operationen müssen durch COM-Objekte ersetzt werden, die sich per **CreateObject** erzeugen lassen. Die am häufigsten verwendeten Objekte sind das **FileSystemObject**, das **Shell**-Objekt aus **shell32** und andere speziell für **VBScript** entwickelte Komponenten, wie der **Windows**

Scripting Host (**wshom.ocx**). All diesen Objekten begegnen Sie noch bei der Besprechung des Installations-Skripts.

Zusammenfassend kann man feststellen, dass sich VBA-Code zu großen Teilen direkt in ein **VBScript** übernehmen lässt. Entwickeln und debuggen Sie ihren Code unter VBA und ändern Sie ihn nach Übernahme in ein **VBScript** überall da, wo es notwendig ist.

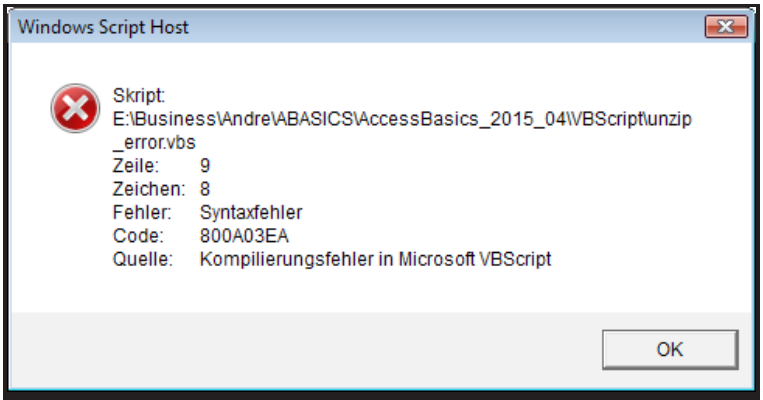


Bild 1: Fehlermeldung der Script-Engine bei einem Syntaxfehler

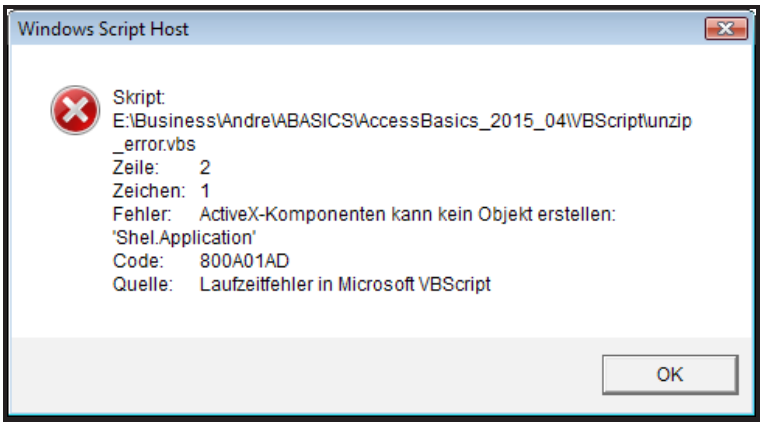


Bild 2: Fehlermeldung der Script-Engine bei einem Objektfehler

VBScripte debuggen

Leider ist der Komfort, wie Sie ihn von VBA her kennen, beim Debuggen von VBScripten nicht vorhanden. Der Debugger zeigt lediglich eine Meldung an, wenn sich ein Fehler ereignet. Bild 1 ist ein Beispiel für einen Syntaxfehler, Bild 2 eines für einen Fehler bei der Ansprache eines Objekts. Der Debugger zeigt immerhin die Code-Zeile und die Spalte an, in der sich der Fehler zutrug. Was genau dort schiefging, bleibt aber weitgehend verborgen. Eine Inspektion von Variablen oder Haltepunkte bleibt außen vor. Meist streut man dann diverse Message-Boxen in den Code ein, die die Werte von Variablen anzeigen, um dem Fehler auf die Schliche zu kommen. Wer viel mit **vbs**-Dateien hantiert kann als **Power User** zu **Visual Studio** greifen, oder zu einem Tool, wie dem kostenpflichtigen **VBSEdit** (http://www.vbsedit.com). Eine freie Alternative stellt der freie **Admin Script Editor** (http://www.itninja.com/community/admin-script-editor) dar. Diese können alle den Script-Code auf ähnliche Weise debuggen, wie VBA.

Hilfsbibliotheken für VBScript

Wo Funktionen für bestimmte Aufgaben im Sprachumfang von **VBScript** fehlen, greift man zu zusätzlichen Bibliotheken. Unter VBA tun Sie ja Ähnliches, indem Sie zusätzliche **Verweise** in das VBA-Projekt einklinken. **VBScript** allerdings sieht das so nicht vor, kann aber fast alle Objekte erzeugen, die in der Registry unter dem Zweig **HKEY_CLASSES_ROOT** zu finden sind. Also jene, die eine **ProgId** aufweisen und durch einen mit Punkt getrennten Begriff gekennzeichnet sind, wie zum Beispiel **Access.Application**.

Die wichtigsten Aufgaben lassen sich mit zwei Bibliotheken erledigen, der **Shell32**-Bibliothek und dem **Windows Scripting Host**. Um sich über deren Inhalt zu informieren, laden Sie unter VBA Verweise auf die Bibliotheken **Microsoft Shell Controls And Automation** (**Shell32**) und **Windows Script Host Object Model** (**IWshRuntimeLibrary**) und inspizieren sie im VBA-Objektkatalog. Die Syntax für das Anlegen der wichtigsten Objekte unter VBA und **VBScript** finden Sie hier im Vergleich:

```
VBA:
Set oShell = New Shell32.Shell
Set oShell2 = New WshShell
Set fso = New FileSystemObject

VBScript:
Set oShell = CreateObject _
  ("Shell.Application)
Set oShell2 = CreateObject _
  ("WScript.Shell")
Set fso = CreateObject _
  ("Scripting.FileSystemObject")
```

Über die so instanzierten Objekte lassen sich alle Aufgaben durchführen, die auch VBA ermöglicht. Das wären vor allem die Datei- und

Verzeichnisoperationen über das FileSystemObject und das WshShell-Objekt. Aber auch die Registry lässt sich lesend und schreibend mit sehr einfachen Einzeilern ansprechen.

Öffnen Sie die Beispieldatenbank, in der die erwähnten Verweise bereits gesetzt sind. Im Modul **mdlScripting** finden Sie verschiedene Prozeduren, die die Ansprache der Objekte verdeutlichen.

```
Set oShell = CreateObject("Shell.Application")
If (WScript.Arguments.Count = 1) Then ElevateUAC
Sub ElevateUAC
     sCommand = WScript.Arguments(0)
    'oShell.ShellExecute "c:\windows\syswow64\wscript.exe", _
         '          WScript.ScriptFullName & " | " & sCommand, , "runas", 1
         '-> unter Winx64
         oShell.ShellExecute "c:\windows\system32\wscript.exe", _
                   WScript.ScriptFullName & " | " & sCommand, , "runas", 1
         WScript.Quit
End Sub

sFolder = WScript.Arguments(1)
Msgbox "Berechtigungen werden gesetzt auf " & sFolder
oShell.ShellExecute "Icacls.exe ", sFolder & " /grant Jeder:F /T", , "open", 1
```

Listing 2: Das Skript startet als Adminstrator und setzt dann Berechtigungen

Script als Administrator starten

Kommen wir nach den theoretischen Ausführungen zu einem praktischen Beispiel. Für viele Dateioperationen ist es notwendig, dass das ablaufende Skript Administratorrechte besitzt. Sei es, um Dateien in geschützte Verzeichnisse zu kopieren oder darin zu löschen, oder, um Rechte auf Ordner zu setzen. Doppelklicken Sie jedoch auf eine **vbs**-Datei, so wird das Skript unter normalem Benutzerkontext gestartet. Im Prinzip müssten Sie die **wscript.exe Als Administrator ausführen** und ihr als Argument den Pfad der **vbs**-Datei mitteilen. Das ist etwas umständlich. Mit einem Trick lässt sich das aber auch innerhalb der **vbs**-Datei selbst bewerkstelligen. Nehmen Sie den Code des Beispielskripts **setpermission.vbs** in Listing 2.

Zunächst wird ein **Shell**-Objekt **oShell** erzeugt, welches für das Ausführen anderer Dateien über seine Methode **ShellExecute** benötigt wird. Das ist in etwa ein Ersatz für die **Shell**-Anweisung von VBA. **WScript** ist eine Objektvariable, die in einem VBScript immer automatisch zur Verfügung steht und die **Scripting-Engine** symbolisiert. Ihre Auflistungsmethode **Arguments** enthält die Kommandozeilenparameter, welche der Skript-Datei übergeben wurden. Das Skript erwartet nämlich als Parameter die Angabe eines Verzeichnispfades, auf den Vollberechtigungen für den Benutzer Jeder gesetzt werden sollen. Diesen Vorgang kann in der Regel das verwendete Kommandozeilen-Tool **Icacls** von Windows nur unter Admin-Rechten durchführen.

Das Skript ermitteln nun per **Count**, wie viele Parameter ihm übergeben wurden. Ist es genau einer, so wird die Unterroutine **ElevateUAC** angesprungen. Sie sehen hier, dass eine **Sub**- oder **Function**-Prozedur mitten im Code an beliebiger Stelle stehen kann. Dennoch setzt das Skript die Ausführung nach diesem Prozedureinschub unbeirrt weiter fort. Die Subroutine speichert jetzt den übergebenen Parameter, also den Ordnerpfad, in der String-Variablen **sCommand**. Und nun wird die **wscript.exe** über **ShellExecute** erneut aufgerufen, wobei ihr aber nun zwei Parameter übergeben werden: ein **Pipe**-Zeichen und der Inhalt der Variablen **sCommand**. Die **Scripting-Engine** sieht Leerzeichen als Trenner für Parameter an. Zu achten ist außerdem darauf, dass unter 64-bit-Systemen der Ort der **wscript.exe** ein anderer ist, als unter Windows **x32**.

Fullname ist eine Eigenschaft des **WScript**-Objekts, die den Pfad der eben ausgeführten **vbs**-Datei wiederspiegelt. Diese wird nun gestartet, wobei **ShellExecute** trickreich als Aktionsparameter der Ausdruck **runas** übergeben wird. Das entspricht dem Starten einer ausführbaren Datei als Administrator. Danach wird das laufende Skript selbst über die **Quit**-Methode beendet. Das Skript wurde nun also erneut gestartet, diesmal jedoch administrativ und mit zwei Parametern. **Arguments.Count** ist deshalb 2 und nicht 1, weshalb das Anspringen der Subroutine **ElevateUAC** übergangen wird und die Hauptroutine mit den Anweisungen nach ihr fortfährt. In **sFolder** wird der zweite Parameter eingelesen, der ja den gewünschten Ordner enthält - der erste Parameter ist das **Pipe**-Zeichen als **Dummy**.

Schließlich wird **Icacls** per **ShellExecute** aufgerufen, wobei der Ordner angegeben und die Berechtigung für **Jeder** gesetzt wird.

Da das Skript universell verwendbar sein und einen Ordner-Parameter annehmen soll, muss es indirekt gestartet werden, in einer Batch-Datei etwa einfach über einen solchen Ausdruck:

```
setpermission.vbs c:\data
```

Verwenden des WMI

Das **Windows Management Instrumentarium** (**WMI**) ist das leistungsfähigste Tool, um jegliche Informationen über das System zu erhalten. Es kann in **VBScript** als Objekt angesprochen und über seine eingebaute Abfragesprache die gewünschte Info ermittelt werden. Listing 3 zeigt ein Beispiel.

```
Set oWMI = GetObject("winmgmts:{impersonationLevel=impersonate}!\\.\root\cimv2")
Set oResult = oWMI.ExecQuery ("Select Version from Win32_OperatingSystem")
For Each itm In oResult
        lWinVers = itm.Version
        lWinVersInt = Int(Left(lWinVers,1))
Next

Set oResult = oWMI.ExecQuery ("Select AddressWidth from Win32_Processor")
For Each itm In oResult
        Is64 = (Int(itm.AddressWidth)=64)
Next

If Is64 = 0 Then
        Msgbox "Windows-Version: " & lWinVers & " 32-bit"
Else
        Msgbox "Windows-Version: " & lWinVers & " 64-bit"
End If
```

Listing 3: Verwenden des WMI zum Auslesen der Windows-Version

Das **WMI**-Objekt **oWMI** erhält man über einen etwas kryptischen Ausdruck in **GetObject**, der mit **winmgmts:** eingeleitet wird. Das Objekt befragt man über **ExecQuery**, wobei zunächst die Sparte **Win32_OperatingSystem** gewählt wird und darin nur das Feld **Version**. Die Ergebnismenge **oResult** muss in einer Schleife abgearbeitet werden, obwohl hier nur ein Ergebnissatz enthalten ist. Die Variable **lWinVers** nimmt den String für die Versionsnummer von Windows auf, etwa **7.0.526**, **lWinVersInt** als Integer-Variable nur die Hauptnummer 7.

Leider sagt die Versionsnummer von Windows nichts darüber aus, ob es sich um ein 64- oder 32-bit-System handelt. Darum wird eine neue Abfrage abgesetzt, die diesmal die Rubrik **Win32_Prozessor** befragt. Steht deren Feld **AddressWidth** auf 64, so haben wir ein 64-bit-System vor uns. Das wird in der **Bool**-Variablen **Is64** festgehalten. Am Schluss gibt eine Messagebox in Abhängigkeit dieser Variablen das Ergebnis wieder.

Wichtig ist diese Routine deshalb, weil die UAC erst ab Version 6 von Windows eingeführt wurde. Das Elevieren des Skript-Prozesses, wie im vorigen Beispiel geschehen, ist so nur dann nötig, wenn die Windows-Version größer als 5 ist. Da nun auch bekannt ist, ob ein 64-bit-OS läuft, kann der Ort bestimmter Verzeichnisse bestimmt werden. Das Systemverzeichnis kann sich unterscheiden und einmal den Pfad **c:\windows\system32** haben, oder **c:\windows\SysWOW64**.

Standardpfade ermitteln

Es gibt aber auch noch alternative Wege, um den Systempfad und andere Standardverzeichnisse zu erfahren. Das **Shell**-Objekt ermittelt über die Methode **NameSpace** alle möglichen Spezialordner, indem ihr als Parameter bestimmte ID-Zahlenwerte übergeben werden:

```
Set oShell = CreateObject _
   ("Shell.Application")
Msgbox oShell.NameSpace(0)
```

Die 0 ist die **ID** für den Desktop. Die Messagebox gibt darum den Ausdruck **Desktop** aus. Das ist allerdings erst die Bezeichnung des Spezialordners. **NameSpace** gibt nämlich nicht etwa einen String zurück, sondern ein **Folder**-Objekt, dessen Standardeigenschaft **Title** ist. Möchten Sie aber den Verzeichnispfad bekommen, so muss eine andere Eigenschaft angesprochen werden:

```
Set oFolder = oShell.NameSpace (0)
sFolder = oFolder.Self.Path
Msgbox sFolder
```

Welche **IDs** mit welchen Windows-Spezialordnern korrelieren, erfahren Sie, indem Sie die Prozedur **GetSpecialFolders** der Beispieldatenbank aufrufen. Sie gibt im VBA-Direktfens-

Daten ins Web exportieren

Nicht selten tritt der Fall ein, dass die Inhalte Ihrer Datenbank auch anderen Personen zur Verfügung gestellt werden sollen, die eben kein **Access** installiert haben. Abseits anderer Austauschformate ist der Browser die wahrscheinlich einfachste Methode, um Daten zur Anzeige zu bringen – und das auch auf mobilen Geräten! Sehen wir nach, welche Mittel **Access** mitbringt, um Daten tabellarisch in HTML-Dateien für das Internet zu speichern.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1505_Webexport.accdb**

Access-Daten im Internet

Um es vorneweg klarzustellen: Hier geht es nicht um die Veröffentlichung von Datenbanken über die seit Access 2007 mögliche Interaktion mit den **Sharepoint**-Servern von **Microsoft Office 365**. Zwar ist dieses eingebaute Feature die sicher komfortabelste Lösung, um Daten interaktiv ins Web zu bringen, erfordert aber einen Server-Platz und einen Account bei Microsoft mit zahlreichen Beschränkungen.

Die performanteste Lösung ist es überdies auch nicht. Und warum sollten Sie, so Sie bereits im Besitz eines Providers und **Web-Space** sind, noch zusätzliche Kosten auf sich nehmen, die mit aktuell mindestens 69 Euro pro Jahr gar nicht so gering ausfallen?

Für den nur lesenden Zugriff auf Tabellendaten ist **Office 365** wie mit Kanonen auf Spatzen geschossen. Es geht auch einfacher, wie wir sehen werden. Die einzige Voraussetzung, die erfüllt sein muss: Sie haben einen **Web-Space** mit FTP-Zugang, über den Sie HTML-Dateien veröffentlichen können.

HTML-Export mit dem Export-Assistenten

Access verfügt von Haus aus über die Exportmöglichkeit in das HTML-Format. Dazu markieren Sie eine Tabelle oder Abfrage im Navigationsbereich und öffnen das Kontextmenü über Rechtsklick. Wählen Sie dann **Exportieren | HTML-Dokument**. Alternativ kann dies auch über den Ribbon und **Externe Daten | Exportieren | Weitere | HTML-Dokument** erfolgen. Das ruft den **Exportieren**-Dialog von Access auf den Plan, welcher Sie nach dem Speicherort für die anzulegende HTML-Datei fragt. Hier ist selbstverständlich die Speicherung im Dateisystem gemeint und nicht bereits etwa eine Schnittstelle über FTP ins Internet.

Neben der Zieldatei lassen sich im Dialog noch drei weitere Optionen aktivieren. **Exportieren von Daten mit Layout und Formatierung** führt dazu, dass Access die Darstellung der Tabelle in der Datenblattansicht in das HTML-Dokument zu übertragen versucht. **Öffnen der Zieldatei nach Abschluss des Exportvorgangs** bewirkt das Öffnen des Standardbrowsers mit der angelegten HTML-Datei. **Exportieren der ausgewählten Datensätze** ist nur dann aktivierbar, wenn Sie bereits eine Tabelle oder Abfrage geöffnet und in dieser Datensätze markiert haben. Der Assistent berücksichtigt solche offenen Datenblätter automatisch, jedoch nur dann, wenn sie bei seinem Start schon geöffnet sind. Nachträglich lässt sich das nicht machen, da der Dialog modal ist und keine weiteren Aktionen der Access-Oberfläche zulässt.

Formulare lassen sich übrigens auf die gleiche Weise exportieren. Erwarten Sie aber nicht, dass Access auch hier das Layout nachzubilden versucht. Es nimmt schlicht die Datenherkunft des Formulars, so, als wäre sie in eine Datenblattansicht geladen, und exportiert diese.

In Bild 1 ist die Abfrage **Wettkampfergebnisse** der Beispieldatenbank zum Export in das HTML-Format ausgewählt, ein Block von Datensätzen markiert und schließlich der Assistent aufgerufen worden. Alle drei Optionen des Dialogs sind aktiviert. Nach Klick auf **OK** startet der Export sofort, aber auch nur dann, wenn die Option zu **Layout und Formatierung nicht** eingeschaltet war. Andernfalls nämlich fragt der Assistent in einem Zwischenschritt noch nach einer etwaigen Vorlage und nach der Textkodierung, wie in Bild 2. Während die Angabe der Codierung, welche standardmäßig ANSI vorsieht, im Zweifelfall bei internationalen Inhalten oder Sonderzeichen in der Datenherkunft

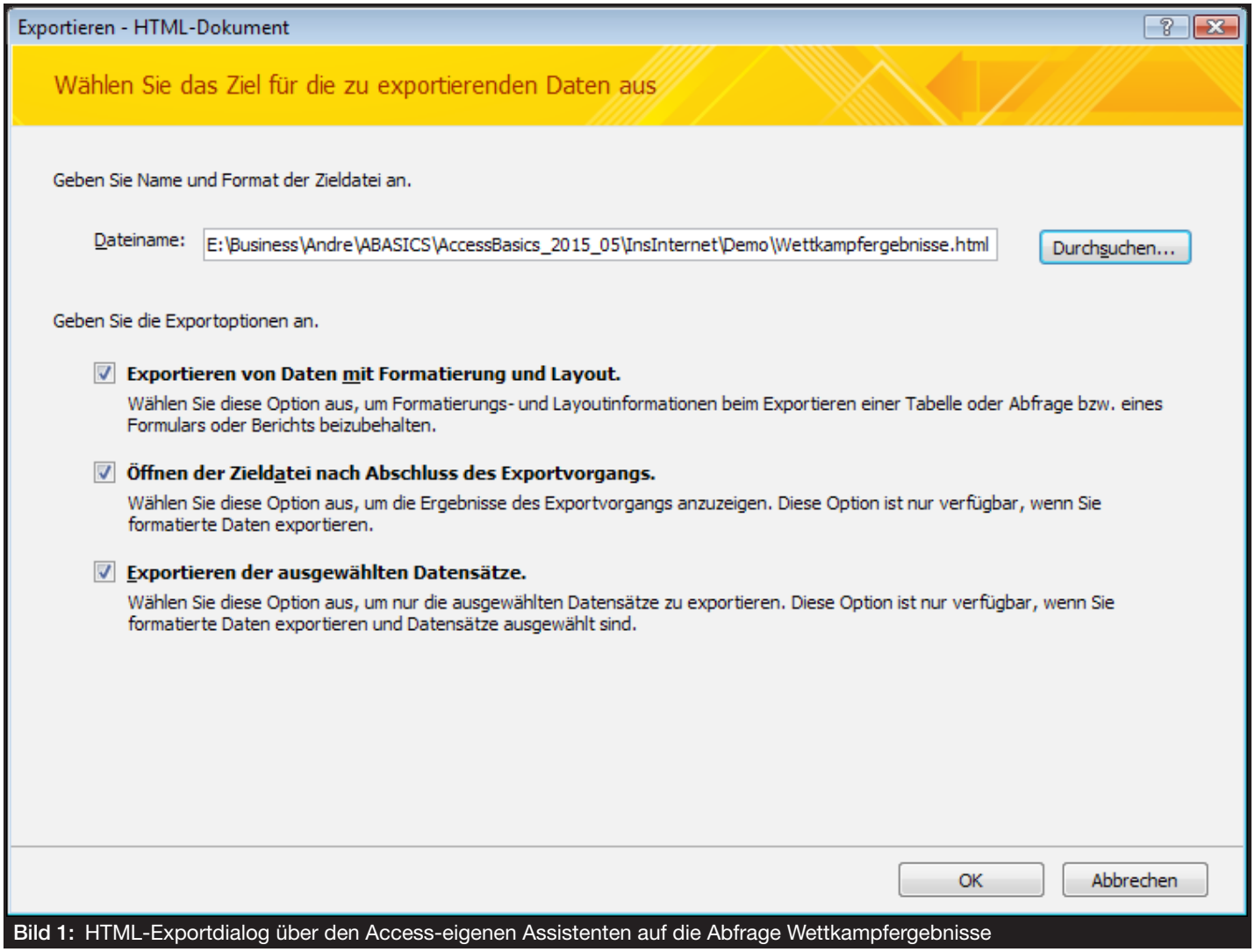


Bild 1: HTML-Exportdialog über den Access-eigenen Assistenten auf die Abfrage Wettkampfergebnisse

Sinn macht und ohne Weiteres auf **Unicode** gesetzt werden darf, wobei **UTF-8** wegen der kleineren resultierenden Dateigröße zu bevorzugen wäre, stellt einen die Option zur **HTML-Vorlage** vor Rätsel. Denn in den aktuellen Hilfedateien von Access werden Sie vergeblich nach dem Aufbau einer solchen Vorlage forschen. Auch im Internet sind nur rudimentäre Angaben zu finden. Machen wir es kurz:

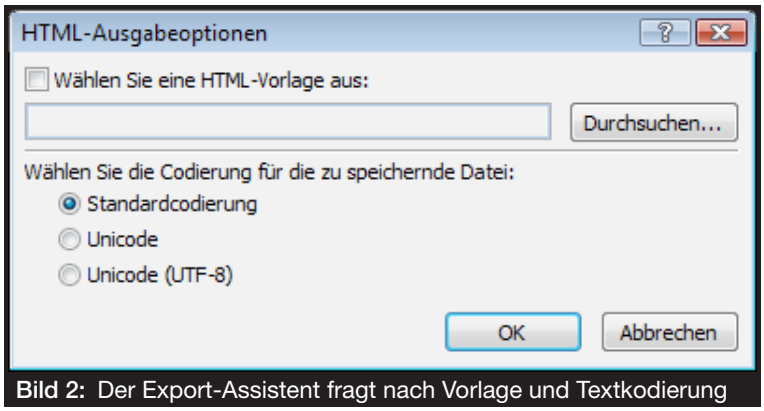


Bild 2: Der Export-Assistent fragt nach Vorlage und Textkodierung

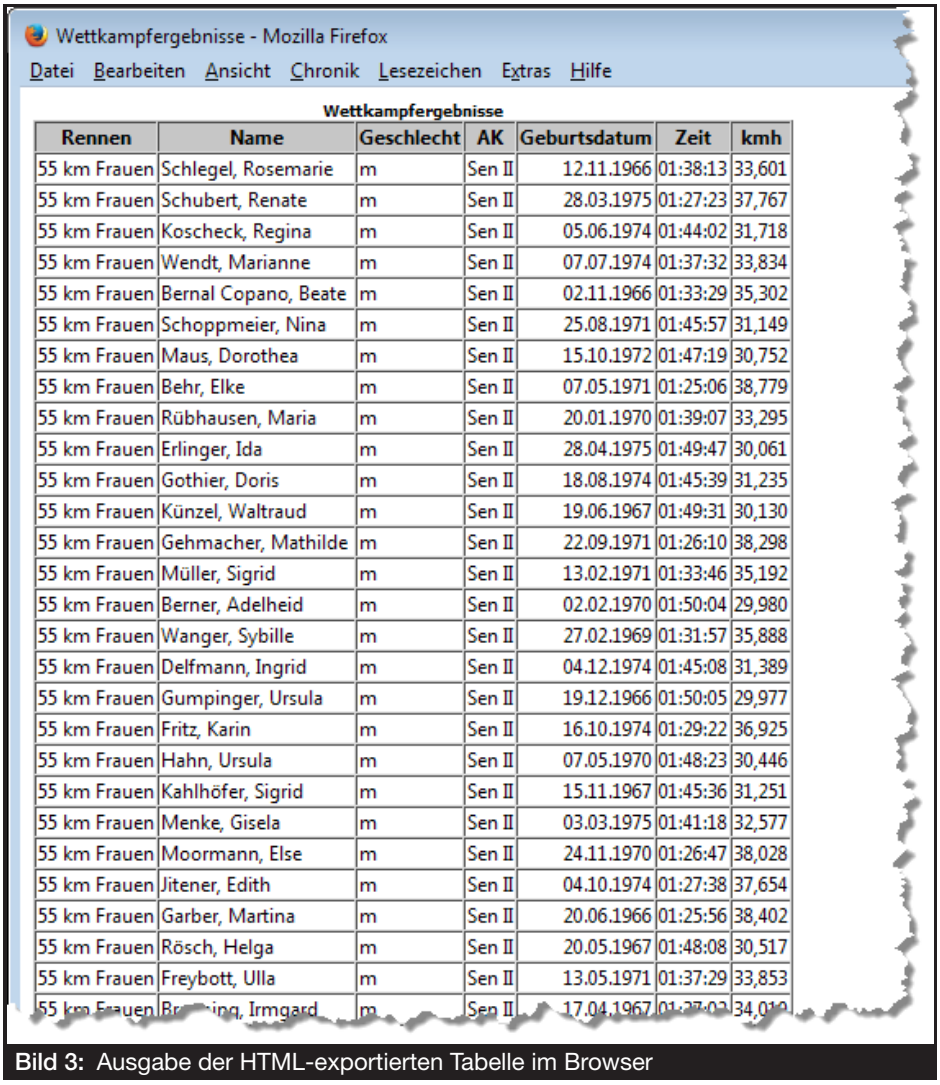
Diese Option ist ein Überbleibsel aus **Office 2000**-Zeiten und die Möglichkeiten zur Gestaltung des HTML-Dokuments über die Vorlage sind stark beschränkt und nicht mehr zeitgemäß. Ignorieren Sie diese Option besser. Sie werden mit der Vorlage mehr Arbeit haben, als mit einem Formatieren der HTML-Datei über VBA.

Sehen wir uns nun das Ergebnis des Exports im Browser an. Die Anzeige entspricht weitgehend der Datenblattansicht der Abfrage, kommt jedoch etwas altbacken daher (siehe Bild 3). Das Layout der

HTML-Tabelle lässt sich leider in keiner Weise beeinflussen. Zwar können Schriftarten, -größen und -farben über die Datenblattansicht der Abfrage vordefiniert werden, was sich dann auch im Export niederschlägt, aber über den Tabellenkopf oder die Zellbegrenzungen und -ränder lässt Access nicht mit sich reden.

Werfen wir noch einen Blick in den Quellcode der HTML-Datei. Dieser wird jedem Web-Designer die Haare zu Berge stehen lassen. Hier ist jede einzelne Zelle (**TD**) mit **Style**-Informationen versehen, die ihren Ursprung in der Datenblattformatierung hat, wie in Listing 1. Von **CSS-Formatierungen** oder Ähnlichem keine Spur. Das treibt die Dateigröße bei umfangreicheren Tabellen gewaltig in die Höhe und macht dem Browser zudem einige Arbeit. Die Wettkampf-Abfrage der Beispieldatenbank mit ihren 3.300 Datensätzen etwa kommt so auf eine HTML-Datei von 3 MB, obwohl der reine Textinhalt nur mit 200 KB zu veranschlagen ist.

Damit eignet sich diese Methode eher für kleinere Datenmengen, bei denen auf die Gestalt der Browserdarstellung kein Wert gelegt wird.



Wettkampfergebnisse

Rennen	Name	Geschlecht	AK	Geburtsdatum	Zeit	kmh
55 km Frauen	Schlegel, Rosemarie	m	Sen II	12.11.1966	01:38:13	33,601
55 km Frauen	Schubert, Renate	m	Sen II	28.03.1975	01:27:23	37,767
55 km Frauen	Koscheck, Regina	m	Sen II	05.06.1974	01:44:02	31,718
55 km Frauen	Wendt, Marianne	m	Sen II	07.07.1974	01:37:32	33,834
55 km Frauen	Bernal Copano, Beate	m	Sen II	02.11.1966	01:33:29	35,302
55 km Frauen	Schoppmeier, Nina	m	Sen II	25.08.1971	01:45:57	31,149
55 km Frauen	Maus, Dorothea	m	Sen II	15.10.1972	01:47:19	30,752
55 km Frauen	Behr, Elke	m	Sen II	07.05.1971	01:25:06	38,779
55 km Frauen	Rübhausen, Maria	m	Sen II	20.01.1970	01:39:07	33,295
55 km Frauen	Erlinger, Ida	m	Sen II	28.04.1975	01:49:47	30,061
55 km Frauen	Gothier, Doris	m	Sen II	18.08.1974	01:45:39	31,235
55 km Frauen	Künzel, Waltraud	m	Sen II	19.06.1967	01:49:31	30,130
55 km Frauen	Gehmacher, Mathilde	m	Sen II	22.09.1971	01:26:10	38,298
55 km Frauen	Müller, Sigrid	m	Sen II	13.02.1971	01:33:46	35,192
55 km Frauen	Berner, Adelheid	m	Sen II	02.02.1970	01:50:04	29,980
55 km Frauen	Wanger, Sybille	m	Sen II	27.02.1969	01:31:57	35,888
55 km Frauen	Delfmann, Ingrid	m	Sen II	04.12.1974	01:45:08	31,389
55 km Frauen	Gumpinger, Ursula	m	Sen II	19.12.1966	01:50:05	29,977
55 km Frauen	Fritz, Karin	m	Sen II	16.10.1974	01:29:22	36,925
55 km Frauen	Hahn, Ursula	m	Sen II	07.05.1970	01:48:23	30,446
55 km Frauen	Kahlhöfer, Sigrid	m	Sen II	15.11.1967	01:45:36	31,251
55 km Frauen	Menke, Gisela	m	Sen II	03.03.1975	01:41:18	32,577
55 km Frauen	Moormann, Else	m	Sen II	24.11.1970	01:26:47	38,028
55 km Frauen	Jitener, Edith	m	Sen II	04.10.1974	01:27:38	37,654
55 km Frauen	Garber, Martina	m	Sen II	20.06.1966	01:25:56	38,402
55 km Frauen	Rösch, Helga	m	Sen II	20.05.1967	01:48:08	30,517
55 km Frauen	Freybott, Ulla	m	Sen II	13.05.1971	01:37:29	33,853
55 km Frauen	Br[illegible]ing, Irmgard	m	Sen II	17.04.1967	01:[illegible]	34,0[illegible]

Bild 3: Ausgabe der HTML-exportierten Tabelle im Browser

Export über VBA

Zum gleichen Ergebnis kommen Sie, wenn Sie eine Tabelle oder Abfrage nicht manuell, sondern über VBA in eine HTML-Datei exportieren. Dafür reicht bereits eine einzige Codezeile aus, welche die generelle Export-Anweisung DoCmd.**OutputTo** enthält:

```
<TR VALIGN=TOP>

<TD BORDERCOLOR=#d0d7e5  ALIGN=RIGHT><FONT style=FONT-SIZE:9pt FACE="Segoe UI" COLOR=#000000>1</FONT></TD>
<TD BORDERCOLOR=#d0d7e5  ALIGN=RIGHT><FONT style=FONT-SIZE:9pt FACE="Segoe UI" COLOR=#000000>34663</FONT></TD>
<TD BORDERCOLOR=#d0d7e5 ><FONT style=FONT-SIZE:9pt FACE="Segoe UI" COLOR=#000000>Standard</FONT></TD>
<TD BORDERCOLOR=#d0d7e5 ><FONT style=FONT-SIZE:9pt FACE="Segoe UI" COLOR=#000000>m</FONT></TD>
<TD BORDERCOLOR=#d0d7e5 ><FONT style=FONT-SIZE:9pt FACE="Segoe UI" COLOR=#000000>Schulz, Katharina</FONT></TD>
<TD BORDERCOLOR=#d0d7e5  ALIGN=RIGHT><FONT style=FONT-SIZE:9pt FACE="Segoe UI" COLOR=#000000>04.12.1990</FONT></TD>
<TD BORDERCOLOR=#d0d7e5  ALIGN=RIGHT><FONT style=FONT-SIZE:9pt FACE="Segoe UI" COLOR=#000000>01:22:32</FONT></TD>
<TD BORDERCOLOR=#d0d7e5  ALIGN=RIGHT><FONT style=FONT-SIZE:9pt FACE="Segoe UI" COLOR=#000000>39,981</FONT></TD>

</TR>
```

Listing 1: HTML-Tags einer Tabellenzeile der aus Access exportierten Abfrage

```
DoCmd.OutputTo acOutputQuery, _
    "Wettkampfergebnisse", _
    acFormatHTML, _
    "d:\XYZ\export.html", True, , _
    "UTF-8"
```

Ihre Parameter im Einzelnen: **acOutputQuery** sagt Access, dass eine Abfrage zum Export ansteht. (**acOutputTable** wäre das Pendant für eine Tabelle.) Der Name der Abfrage oder Tabelle folgt als nächster Parameter – hier **Wettkampfergebnisse**. Dass es beim Export um das HTML-Format gehen soll, wird über die Konstante **acFormatHTML** angegeben. Darauf folgt der volle Pfad zur Zieldatei. Das anschließende **True** weist Access an, nach dem Export den Browser mit der Datei zu öffnen. Den Parameter für eine mögliche Vorlage überspringen wir. Die Codierung des HTML-Codes wird als letzter Parameter hier mit dem String **UTF-8** definiert. Die letzten Parameter sind optional, obligatorisch nur die ersten vier.

Das Ergebnis dieses VBA-Exports stimmt mit dem des Assistenten exakt überein, wobei hier automatisch die Option **Mit Formatierung und Layout** angenommen wird, denn das lässt sich per VBA leider nicht einstellen.

Export über Programmierung

Nach dem bisher Dargestellten wird deutlich, dass der HTML-Export über Access-Bordmittel keinen höheren Ansprüchen genügt. Zum Glück handelt es sich aber bei HTML um ein reines Textformat, dessen Code somit auch mit VBA-String-Funktionen zusammengesetzt werden kann.

Natürlich kommt man in diesem Fall nicht ohne grundlegende HTML-Kenntnisse aus, aber die hat man sich schnell angeeignet.

Eine ganz rudimentäre HTML-Datei ergibt sich etwa über eine Prozedur, wie die folgende:

```
Sub ExportHTML()
  Dim sHTML As String
  sHTML = "<html><body>" & _
   "Hallo!</body></html>"
  Open "d:\xyz.html" For Binary As #1
  Put #1, , sHTML
  Close #1
End Sub
```

In der String-Variablen **sHTML** wird der HTML-Code der Datei abgelegt. Der zugewiesene Code besteht lediglich aus den grundlegendsten **Tags** mit einem **Hallo!** im **Body**. Die String-Variable wird binär in eine per **Open**-Anweisung erzeugte Datei gespeichert, wobei die **Put**-Methode zum Einsatz kommt. Das ist die Grundstruktur, die sich dann erweitern lässt. In **sHTML** lassen sich durch String-Verkettung beliebige Inhalte unterbringen, die etwa auch aus den Daten einer über ein **Recordset** geöffneten Tabelle oder Abfrage stammen können.

Neben den reinen Daten können außer Style-Auszeichnungen ebenso JavaScript-Funktionen eingebaut werden. Wie die Lösung aussehen könnte, wird am Beispiel der Demo-Datenbank erläutert.

Wettkampfergebnisse

Als Beispiel haben wir eine Datenbank erstellt, die fiktive Radrennen verwaltet. Die Zeitnahmen der Teilnehmer kämen etwa über ein Interface mit einem Zeitnehmersystem in die Tabelle **tblResultate**, wo sie den Teilnehmern der Tabelle **tblTeilnehmer** zugeordnet sind. Als Hilfstabellen finden sie die **tblAltersklassen**, welche zu den jeweiligen Altersgruppen die Bezeichnungen enthält, sowie die **tblWettkampfe**, die alle ausgetragenen Rennen verzeichnet. Das Datenmodell stellt Bild 4 dar. Es ist an einigen Stellen redundant. So ist die Spalte **kmh** der Resultate-Tabelle etwa ein berechnetes Feld,

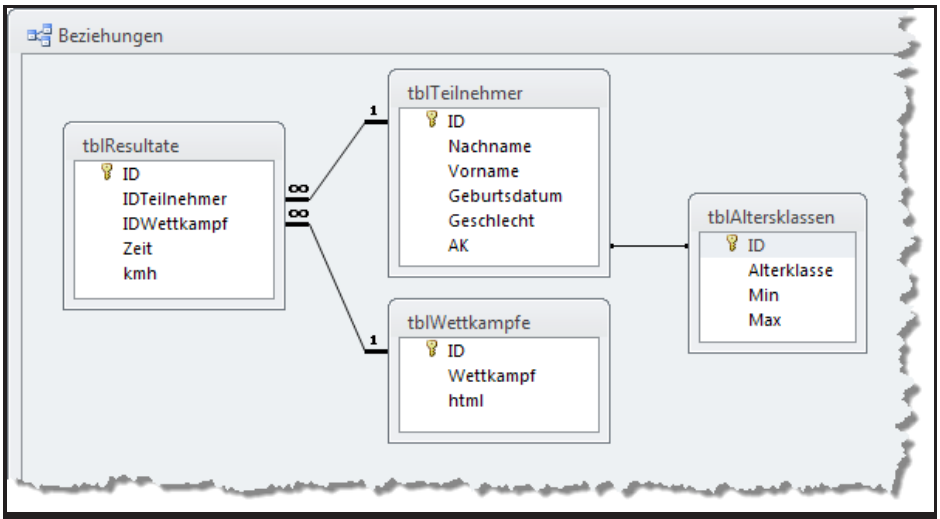


Bild 4: Datenmodell der Beispieldatenbank zur Verwaltung von Radrennen

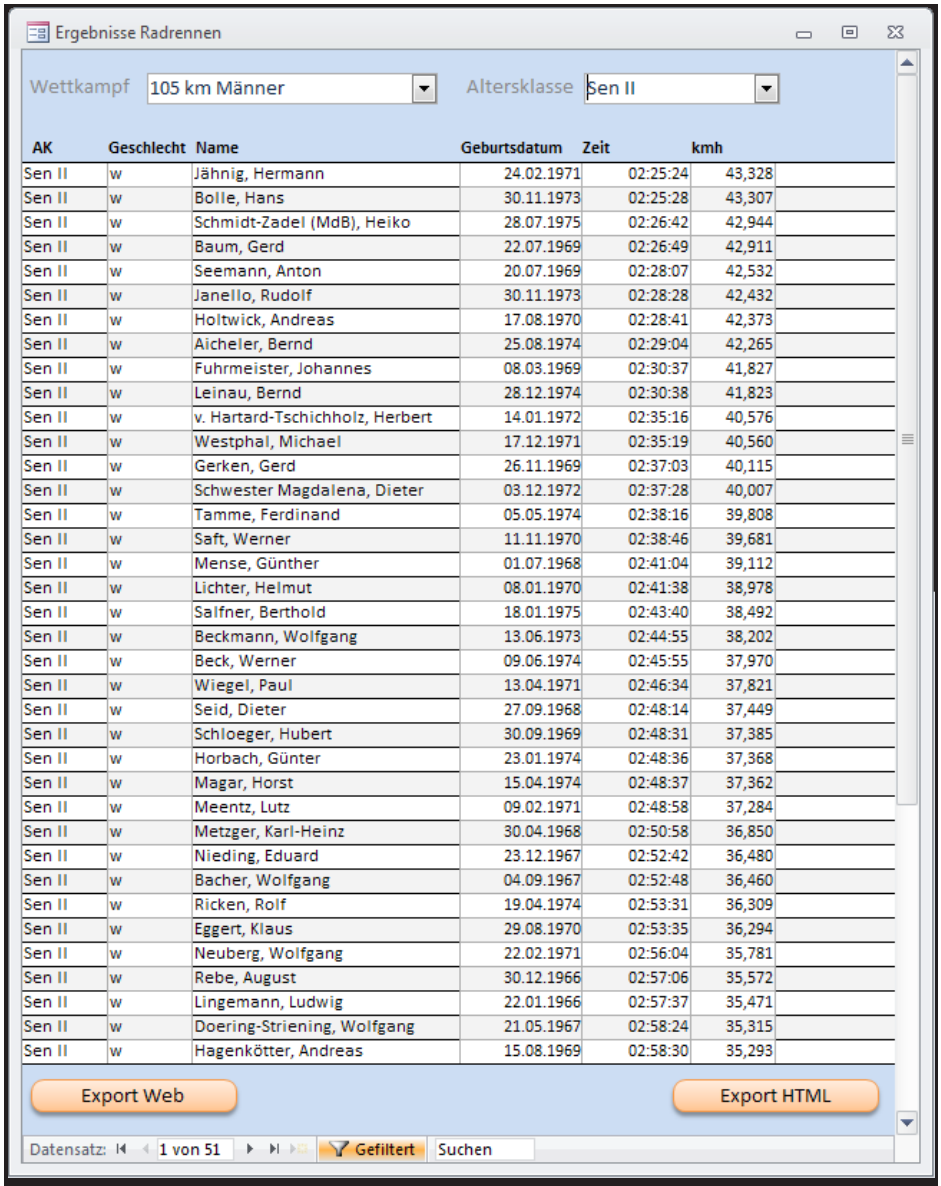
dessen Inhalt sich aus der Strecke und der genommenen Zeit ergibt, also über eine Abfrage ausgegeben werden könnte. Für den leichteren Export wurde hier aber der Wert diskret eingerechnet. Ähnlich verhält es sich mit der Altersklasse **AK** der Teilnehmertabelle. Da sich diese über das Alter der Personen ergibt, welches durch das angegebene **Geburtsdatum** definiert ist, wäre die Angabe an sich überflüssig. Auch hier dient das berechnete Feld dem vereinfachten Export.

Das ganze Vorhaben ist durchaus realistisch. Bei solchen Rad- oder Laufwettbewerben stehen die Ergebnisse meist unmittelbar nach Zieleinlauf im Internet. Wir simulieren hier die Veranstaltung.

Dazu dient ein einziges Formular, welches in Bild 5 gezeigt ist. Die Datenquelle basiert auf der Tabelle **tblResultate**, dies jedoch über eine Abfrage ähnlich zu Bild 6. Es handelt sich um ein **Endlosformular**, welches im Formularkopf zwei Kombinationsfelder integriert hat, die zur Auswahl und Filterung der Daten dienen.

Im Formularfuß sind zwei Schaltflächen untergebracht, wovon die rechte nur zur Ausgabe der Ergebnisse in eine HTML-Datei über die besprochene einfache Methode **DoCmd.OutputTo** dient, die linke aber für das Anlegen einer Datei über String-Verkettung aus den Abfragedaten.

Klicken Sie auf eine der Schaltflächen, so werden Sie zunächst mit einem Bestätigungsdialog für den Export konfrontiert. Ein weiterer



AK	Geschlecht	Name	Geburtsdatum	Zeit	kmh
Sen II	w	Jähnig, Hermann	24.02.1971	02:25:24	43,328
Sen II	w	Bolle, Hans	30.11.1973	02:25:28	43,307
Sen II	w	Schmidt-Zadel (MdB), Heiko	28.07.1975	02:26:42	42,944
Sen II	w	Baum, Gerd	22.07.1969	02:26:49	42,911
Sen II	w	Seemann, Anton	20.07.1969	02:28:07	42,532
Sen II	w	Janello, Rudolf	30.11.1973	02:28:28	42,432
Sen II	w	Holtwick, Andreas	17.08.1970	02:28:41	42,373
Sen II	w	Aicheler, Bernd	25.08.1974	02:29:04	42,265
Sen II	w	Fuhrmeister, Johannes	08.03.1969	02:30:37	41,827
Sen II	w	Leinau, Bernd	28.12.1974	02:30:38	41,823
Sen II	w	v. Hartard-Tschichholz, Herbert	14.01.1972	02:35:16	40,576
Sen II	w	Westphal, Michael	17.12.1971	02:35:19	40,560
Sen II	w	Gerken, Gerd	26.11.1969	02:37:03	40,115
Sen II	w	Schwester Magdalena, Dieter	03.12.1972	02:37:28	40,007
Sen II	w	Tamme, Ferdinand	05.05.1974	02:38:16	39,808
Sen II	w	Saft, Werner	11.11.1970	02:38:46	39,681
Sen II	w	Mense, Günther	01.07.1968	02:41:04	39,112
Sen II	w	Lichter, Helmut	08.01.1970	02:41:38	38,978
Sen II	w	Salfner, Berthold	18.01.1975	02:43:40	38,492
Sen II	w	Beckmann, Wolfgang	13.06.1973	02:44:55	38,202
Sen II	w	Beck, Werner	09.06.1974	02:45:55	37,970
Sen II	w	Wiegel, Paul	13.04.1971	02:46:34	37,821
Sen II	w	Seid, Dieter	27.09.1968	02:48:14	37,449
Sen II	w	Schloeger, Hubert	30.09.1969	02:48:31	37,385
Sen II	w	Horbach, Günter	23.01.1974	02:48:36	37,368
Sen II	w	Magar, Horst	15.04.1974	02:48:37	37,362
Sen II	w	Meentz, Lutz	09.02.1971	02:48:58	37,284
Sen II	w	Metzger, Karl-Heinz	30.04.1968	02:50:58	36,850
Sen II	w	Nieding, Eduard	23.12.1967	02:52:42	36,480
Sen II	w	Bacher, Wolfgang	04.09.1967	02:52:48	36,460
Sen II	w	Ricken, Rolf	19.04.1974	02:53:31	36,309
Sen II	w	Eggert, Klaus	29.08.1970	02:53:35	36,294
Sen II	w	Neuberg, Wolfgang	22.02.1971	02:56:04	35,781
Sen II	w	Rebe, August	30.12.1966	02:57:06	35,572
Sen II	w	Lingemann, Ludwig	22.01.1966	02:57:37	35,471
Sen II	w	Doering-Striening, Wolfgang	21.05.1967	02:58:24	35,315
Sen II	w	Hagenkötter, Andreas	15.08.1969	02:58:30	35,293

Bild 5: Das Formular frmErgebnisse der Beispieldatenbank in Aktion

Dialog fragt danach, ob die Ergebnisse sogleich im Browser begutachtet werden wollen.

Und schließlich können Sie beim Webexport-Button noch der Frage zustimmen, ob die Ergebnisdatei per FTP an den Webserver übertragen werden soll. Betrachten wir anhand einiger Listings in den folgenden Abschnitten, wie die Sache umgesetzt ist.

Filtern der statischen Abfrage

Für den Export über die Methode **DoCmd.OutputTo** kann auf die Ergebnismenge nicht direkt Einfluss genommen werden. Der Anweisung muss ja ein fester Abfragename als Parameter mitgegeben werden, doch der Export soll genauso gefiltert sein, wie es die Auswahl im Formular anzeigt. Hier ist ein Trick in die Abfrage Wettkampfergebnisse in Bild 6 eingebaut. Die Art des Rennens ist über

Reimen leicht gemacht

Kennen Sie das? Zur Feier des 70. Geburtstags Ihrer Mutter soll mit einigen Darbietungen aufgewartet werden, und Ihnen fällt das Los zu, ein würdigendes heiteres Gedicht zu verfassen. Ohne Reime darf dieses natürlich nicht auskommen, und so ringen Sie eines Tages mit Ihrer Kreativität und dem deutschen Wortbestand, um etwas Sinnvolles zu Papier zu bringen. Aber lassen Sie sich dabei doch einfach von Access und einer Reim-Hilfe-Datenbank unterstützen...

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1505_Reimer.accdb**

Computer-Reimen

Nicht jeder ist ein Sprachgenie, und das Auffinden von gleichklingenden Ausdrücken in den neuronalen Untiefen des Sprachzentrums macht wohl Allen einige Mühe. Der Vorgang ist aber nicht so kompliziert, dass man ihn nicht auch an eine Software-Anwendung delegieren könnte.

Der Kern dieser wäre eine Liste aller deutschen Worte, deren Endungen dann nur noch verglichen werden müssten. Dafür eignet sich natürlich eine Datenbankanwendung am besten. Tatsächlich ist die Beispieldatenbank des Autors zu diesem Beitrag aus ähnlichen Motiven entstanden, wie im Intro erwähnt.

Wortliste

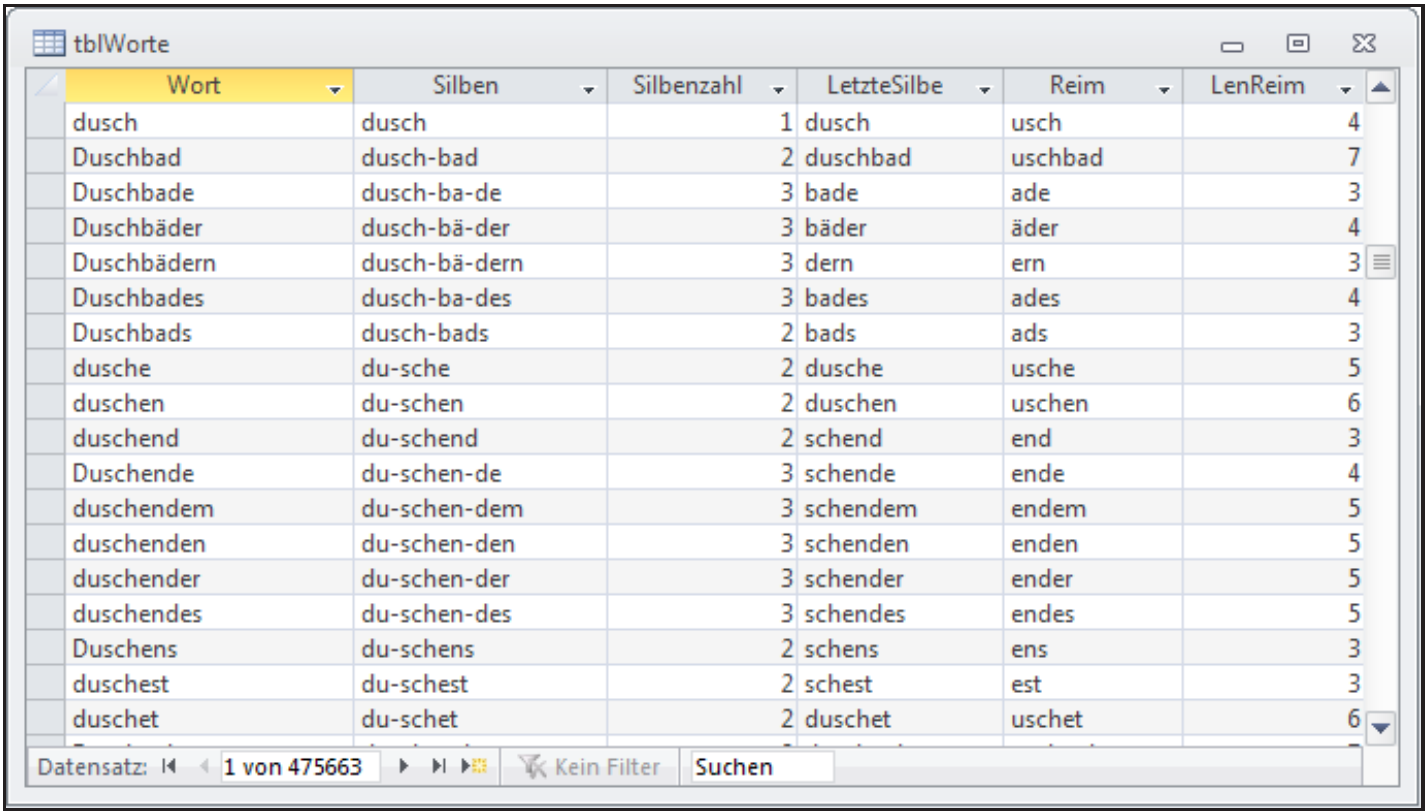
Auf der Suche nach Dateien, die alle deutschen Begriffe in allen Formen, wie Flexionen, et cetera, enthalten, stößt man im Netz auf verschiedene Quellen. Diese sind leider allesamt weder komplett, noch fehlerfrei.

Die einzige Tabelle **tblWorte** der Demo-Datenbank ist daher aus verschiedenen Dateien erstellt und nachbearbeitet worden.

Meist liegen diese Wortdateien in Textform vor und haben einen **CSV**-ähnlichen Aufbau. So war eine Quelle etwa der **OpenThesaurus**, welcher auch von **Open-Office** eingesetzt wird, oder die Ergebnisse des Sprachwunders **Morphy**, das auch der Syntaxanalyse der Begriffe mächtig ist. Diese Textdateien wurden in Tabellen importiert und über Anfügeabfragen zusammengeführt. Dann erfolgte noch eine Nachbearbeitung über VBA-Routinen, die die Begriffe eindeutig filterten und die Groß- und Kleinschreibung anpassten. Schließlich wurde die Tabelle auch noch für das Reimen vorbereitet und erweitert, indem eine automatische Silbentrennung aller Begriffe erfolgte.

Ergebnis ist eine Tabelle mit etwa 470.000 Begriffen, die aber dennoch keinen Anspruch auf Vollständigkeit erhebt. Viele umgangssprachliche Ausdrücke oder Modeworte fehlen, und die Groß- und Kleinschreibung lässt stellenweise zu wünschen übrig. Vor allem aber ist die Silbentrennung ganz und gar nicht perfekt, denn diese entstand aus Performancegründen über einen Algorithmus, der phonetische Gesichtspunkte außer Acht ließ und Konsonanten- und Vokalgruppen nur rudimentär untersuchte. Dennoch ist sie für den angepeilten Einsatz brauchbar. Damit die Tabelle schnell durchsucht werden kann, sind einige Felder indiziert, was dazu führt, dass die Datenbank mit 48 MB recht groß geraten ist.

Einen Ausschnitt der Wortliste der **tblWorte** finden Sie in Bild 1. Die Spalte **Wort** verzeichnet die Begriffe, welche in der gebräuchlichen Schreibweise gespeichert sind. Natürlich kann das Verb **duschen** auch im Nominativ daherkommen und schriebe sich dann korrekt **Duschen**. Im Feld **Silben** finden Sie dann jeweils den Begriff mit dem Trennzeichen **-** für die Silbentrennung. Für Abfragen, die etwa den Rhythmus eines Ausdrucks berücksichtigen wollen, ist zur schnelleren Bearbeitung noch das berechnete Feld **Silbenzahl** eingefügt, dessen Werte sich aus der Zahl der Trennstriche plus eins ergeben. Mit **Letzte-Silbe** und **Reim** folgen nun die für uns besonders interessanten Werte. Sie wurden über eine VBA-Routine aus dem **Wort** gebildet, wobei die Endsilbe und voranstehende Vokale untersucht wurden. Dass dies nicht fehlerfrei funktionierte, sehen Sie etwa am Begriff **Duschbad**, der aus unerfindlichem Grund ohne Silben angegeben ist. Das letzte Feld lautet **LenReim** und gibt die String-Länge des Ausdrucks in **Reim** wieder. Auch dieses berechnete Feld soll zur be-



tblWorte

Wort	Silben	Silbenzahl	LetzteSilbe	Reim	LenReim
dusch	dusch	1	dusch	usch	4
Duschbad	dusch-bad	2	duschbad	uschbad	7
Duschbade	dusch-ba-de	3	bade	ade	3
Duschbäder	dusch-bä-der	3	bäder	äder	4
Duschbädern	dusch-bä-dern	3	dern	ern	3
Duschbades	dusch-ba-des	3	bades	ades	4
Duschbads	dusch-bads	2	bads	ads	3
dusche	du-sche	2	dusche	usche	5
duschen	du-schen	2	duschen	uschen	6
duschend	du-schend	2	schend	end	3
Duschende	du-schen-de	3	schende	ende	4
duschendem	du-schen-dem	3	schendem	endem	5
duschenden	du-schen-den	3	schenden	enden	5
duschender	du-schen-der	3	schender	ender	5
duschendes	du-schen-des	3	schendes	endes	5
Duschens	du-schens	2	schens	ens	3
duschest	du-schest	2	schest	est	3
duschet	du-schet	2	duschet	uschet	6

Datensatz: 1 von 475663 | Kein Filter | Suchen

Bild 1: Der Screenshot verdeutlicht den Feldaufbau der Tabelle tblWorte, die (fast) alle deutschen Begriffe enthält

schleunigten Weiterverarbeitung bei der Reimsuche helfen.

Die VBA-Routinen, die zu dieser Tabelle führten, sind in der Beispieldatenbank nicht enthalten. Ihre Erläuterung hätte angesichts der Komplexität sicher den Rahmen des Beitrags gesprengt.

Formular zur Reimhilfe

Das einzige Formular **frmReimer** der Datenbank sieht zur Laufzeit aus, wie in Bild 2. Betrachten wir zunächst, welche Elemente es aufweist.

Im Textsteuerelement links kann der Text des Gedichts eingetragen werden. Sobald Sie eine Zeile mit der **Return**-Taste beendet haben, tritt der Reim-Algorithmus auf den Plan und sucht aus dem Wortschatz alle Begriffe, die sich auf das letzte Wort der Zeile reimen könnten. Sie werden dann in der **Listbox** rechts aufgeführt, und Sie können sich einen auswählen, der für die nächste Reimzeile infrage kommt. Der Suchvorgang dauert angesichts der fast halben Million von Begriffen je nach Rechnerausstattung einen Moment, was durch die dann laufende Sanduhr angezeigt wird.

Nun werden Sie nicht unbedingt nur **Paarreime** erfinden wollen, also solche, in denen sich aufeinanderfolgende Zeilen reimen (**A-A-B-B**). Vielleicht sollen es **Kreuzreime** sein (**A-B-A-B**), oder **Blockreime** (**A-B-B-A**). Deshalb aktualisiert die **Listbox** ihren Inhalt nicht nur nach Beenden einer Zeile, sondern auch dann, wenn Sie ein beliebiges Wort mit der Maus im Text doppelklicken. Folglich lässt sich zu jedem Zeitpunkt ein Reimwort auffinden.

Was hat es mit der **Combobox** rechts oben im Formular mit dem **Label Anzahl Buchstaben** auf sich? Per **Default** steht sie auf dem Kürzel **A**, kann aber auf **B** oder 2 bis 8 umgestellt werden. Hier handelt es sich um den Typ des Algorithmus, der zum Einsatz kommt. Mit **A** durchsucht die Datenbank alle Begriffe, die im Feld **Reim** der Tabelle **tblWorte** die gleiche Endung aufweisen, wie das zum Reimen markierte Wort. Mit **B** wird stattdessen das Feld **Letzte-Silbe** herangezogen. Wie die Praxis zeigte, sind beide Methoden jedoch unzureichend, wenn es etwa um kurze Begriffe geht. Deshalb wurde noch eine Methode hinzugefügt, die

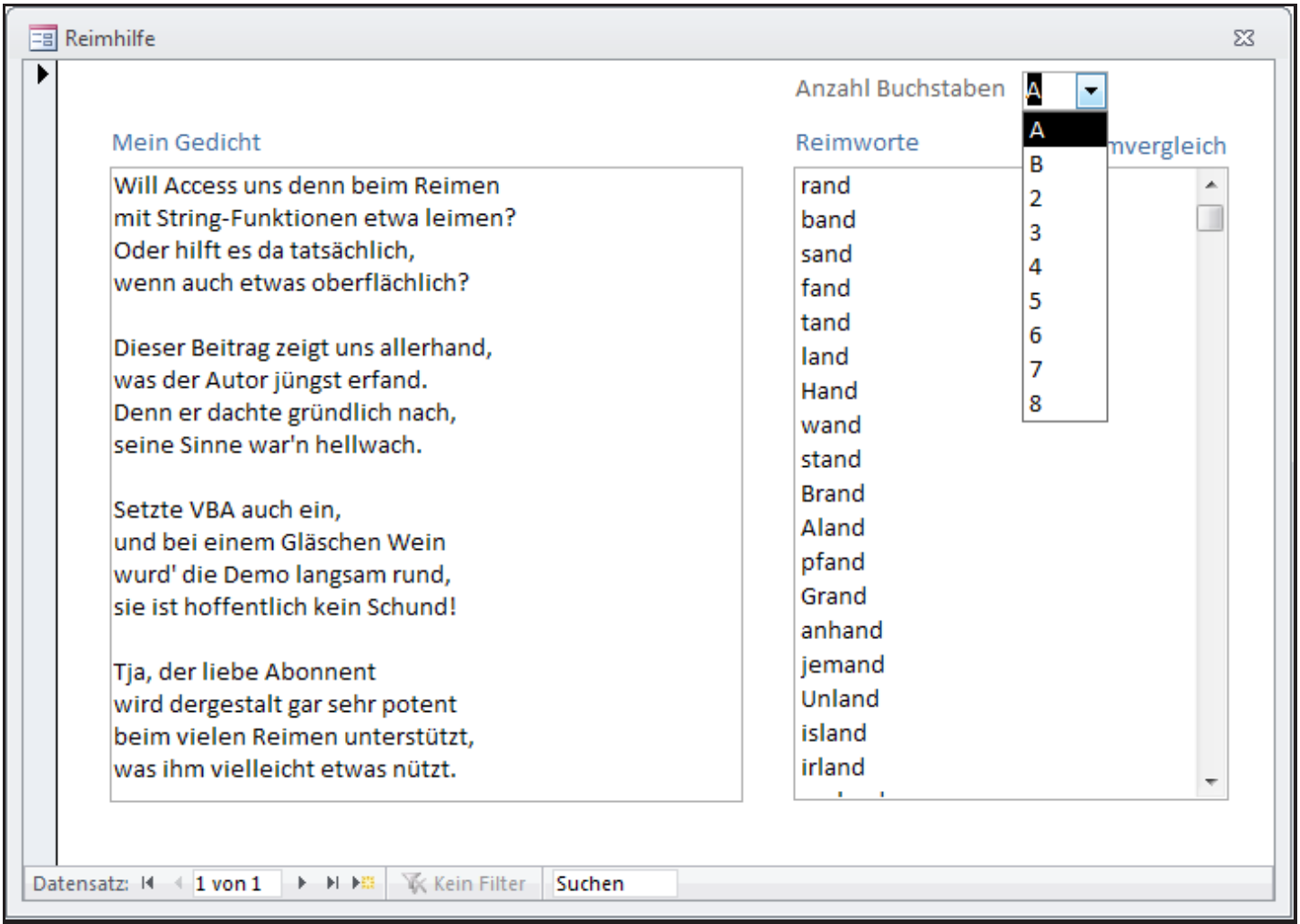


Bild 2: Dieses Formular frmReimer soll Sie beim Reimen unterstützen und verwendet dazu die Tabelle tblWorte

direkt die letzten Buchstaben der Begriffe (Feld **Wort**) mit der gleichen Anzahl im Suchbegriff vergleicht. Und hierbei ist wählbar, wie viele Buchstaben berücksichtigt werden. Sie können die **Combobox** jederzeit umstellen, um eine andere Liste von Reimbegriffen zu erhalten.

Die Reihenfolge der angebotenen Begriffe ist übrigens nicht alphabetisch, sondern aufsteigend nach deren Länge. Damit stehen oben die einfacheren Begriffe, von denen sich längere eventuell ableiten. In Bild 2 wurde der Begriffe **allerhand** im Text doppelgeklickt. An fünfter Stelle der vorgeschlagenen Reimworte findet sich **land** – zwar in Kleinschreibung, aber man weiß, was gemeint ist. Scrollte man die Liste weiter nach unten, fände man etwa auch **Aufenthaltsland**. Bei alphabetischer Sortierung stünde dieser Begriff weiter oben. Es spart jedoch Hirnschmalz und ist ergonomischer, wenn der Grundbegriff weiter oben steht: Das Wort **Urlaubsland** fällt Ihnen sicher schneller ein, wenn es von **Land** abgeleitet ist, als bei **Aufenthaltsland**.

Verankerung

Wenn Ihnen das Formular zu klein ist, so ziehen Sie es mit der Maus einfach größer. Die Steuerelemente sind alle mit dem **Verankern**-Feature von Access ausgestattet, wodurch sie sich in Größe und Position dem Formular anpassen.

Ein Blick in die Entwurfsansicht des Formulars (Bild 3) verdeutlicht dies, wenn Sie im Ribbon gleichzeitig das **Tab Anordnen** ausgewählt haben. Aktivieren Sie nun ein Steuerelement und inspizieren die Verankerung über Klick auf den **Anker-Button** (Bild 4). Der Typ der Verankerung wird durch die gelbe Markierung bestimmt - im Beispiel wurde die **Listbox lstWorte** zur Anzeige der Reimworte akti-

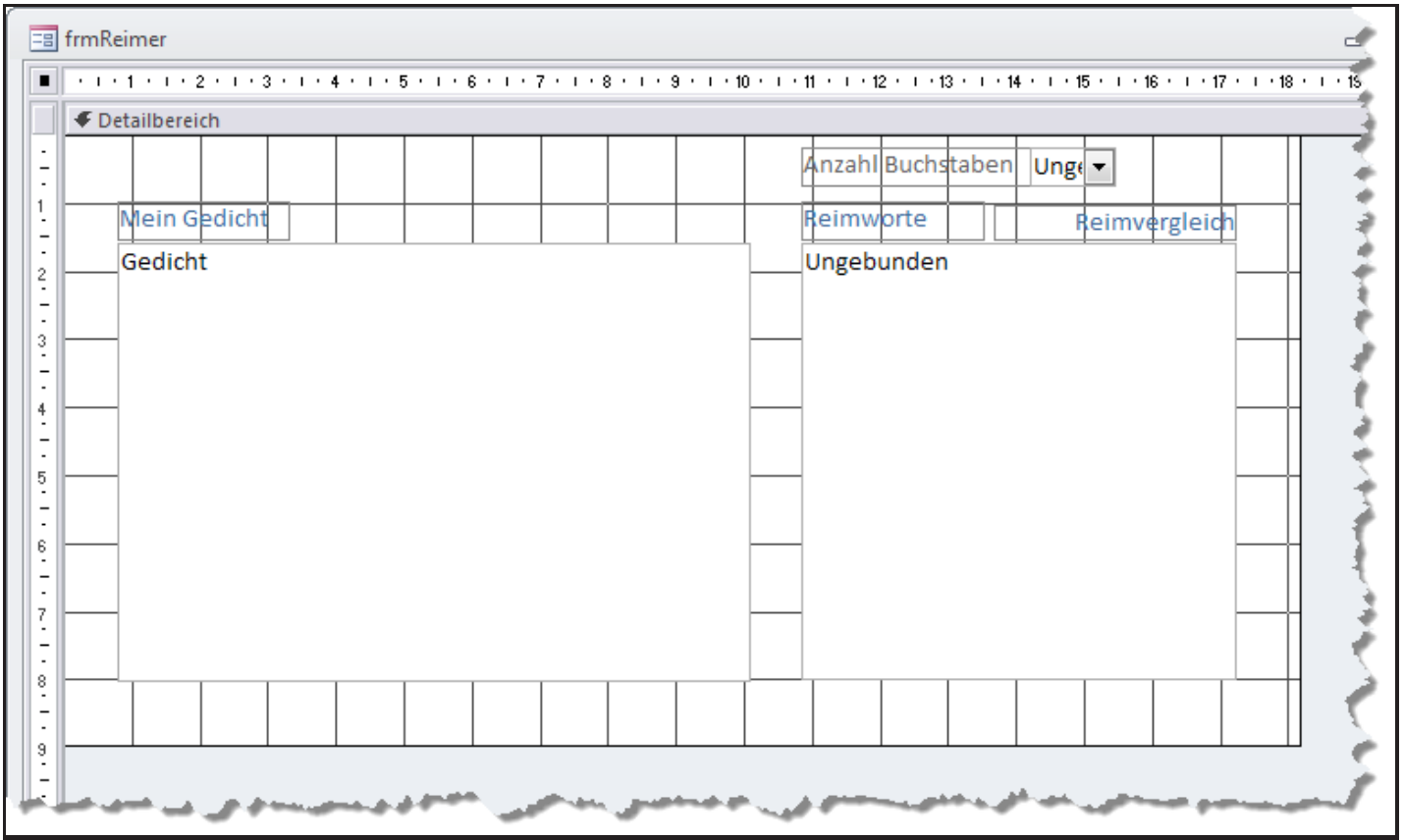


Bild 3: Das Formular frmReimer und seine Steuerelemente in der Entwurfsansicht

viert, die auf **Nach unten und rechts dehnen** eingestellt ist. Damit bleibt ihre Position beim Vergrößern des Formulars relativ zur Texteingabe links erhalten, ihre Breite bleibt gleich, verschiebt sich also absolut, die Höhe passt sich jedoch entsprechend an.

Das Textfeld selbst hingegen ist auf **Nach unten und quer dehnen** gesetzt, wodurch die Position erhalten wird, sich bei Größenänderung aber sowohl Breite, wie Höhe, anpassen. Sämtliche anderen Steuerelemente des Formulars – **Labels** und die **Combobox** – stehen auf der **Verankerung Oben rechts**. Damit verschiebt sich lediglich deren Position, während die Größe unbeeinflusst bleibt.

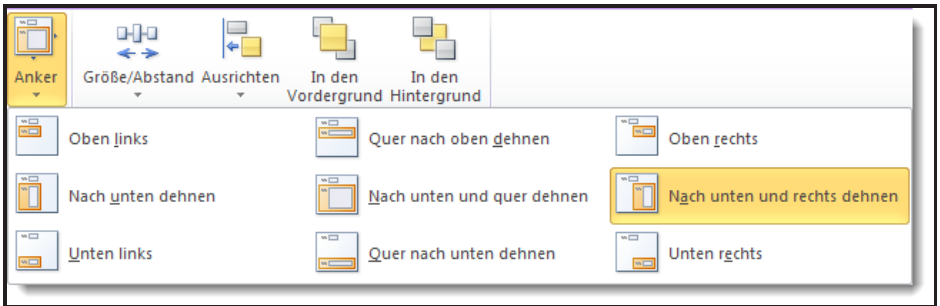


Bild 4: Die Verankerung des Listbox-Steuerelements lstWorte des Formulars

Die Ausdrücke, welche Microsoft den einzelnen **Verankern**-Modi verpasst hat, sind nicht unbedingt einleuchtend. Experimentieren Sie bei Ihren eigenen Formularen einfach etwas herum, um dem Verhalten bei Laufzeit auf die Schliche zu kommen.

Wie auch immer, in vergrößertem Zustand präsentiert sich durch die Verankerung das Formular dann wie in Bild 5. Der Raum für die Texteingabe hat sich ausgedehnt, die **Listbox** vertikal erweitert, und die restlichen Elemente haben sich verschoben.

Nur ein Verkleinern machen die Steuerelemente nicht mit: Die minimalen Ausmaße werden durch jene bestimmt, die Sie ihnen in der Entwurfsansicht verpassten. Dies gilt dann

Datenbankvorlagen und Designs

Sie sparen sich Arbeit, wenn Sie Ihre Datenbanken nicht immer von Grund auf neu entwickeln, sondern auf bereits vollbrachtem Werk aufbauen. Access bietet hier in den neueren Versionen einige Hilfsmittel an, die eine schnellere Gestaltung fördern. Wir zeigen, wie Sie Ihren Workflow mit Datenbankvorlagen, Anwendungsparts und Designs beschleunigen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1506_VorlagenDB.accdb**

Vorlagen

Seit der Version 2007 von Access bietet Microsoft bei Anlegen einer neuen Datenbank Vorlagen an, die vom Hersteller selbst stammen. Bild 1 zeigt im unteren Teil etwa die Vorlagen **Posten** und **Kontakte** an, welche, so aktiviert, neue Datenbanken auf der Grundlage der von Microsoft zusammengestrickten Vorgaben erstellen. Ob diese Beispielvorlagen Ihren Ansprüchen genügen entscheiden Sie selbst. Da es sich in den meisten Fällen um komplexere Datenbanken handelt, die nicht nur Formulare enthalten, sondern auch zugehörige Tabellen und Makros, dürfte das Einsatzfeld ziemlich beschränkt sein und selten

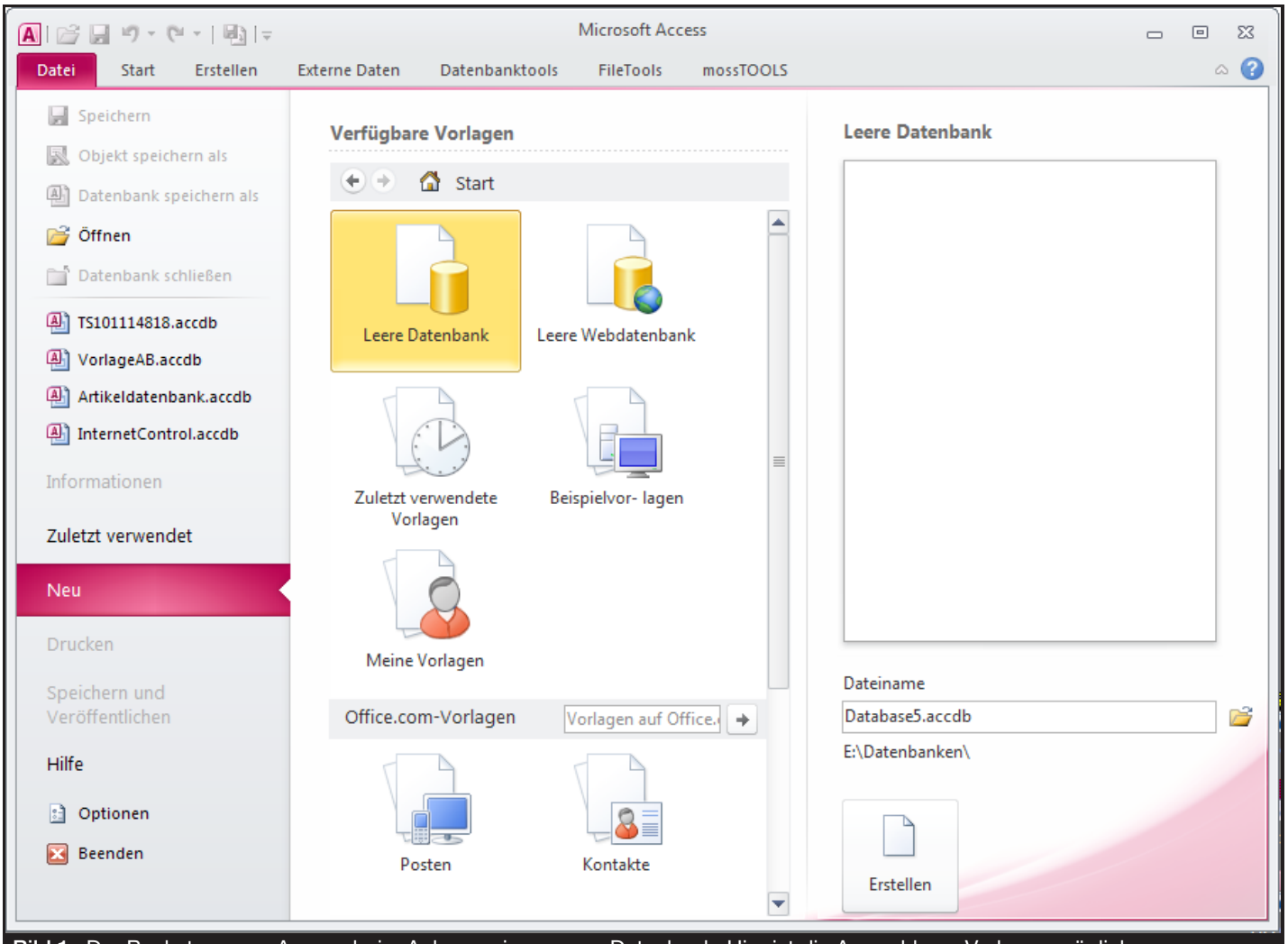


Bild 1: Der Backstage von Access beim Anlegen einer neuen Datenbank. Hier ist die Auswahl von Vorlagen möglich.

auf Ihre Anforderungen passen. Die vorgefertigten Vorlagen sollen auch gar nicht Thema dieses Beitrags sein. Wir versuchen hier vielmehr, solche Vorlagen selbst zu bauen, und im **Backstage**-Bereich neben den Microsoft'schen zur Ansicht zu bringen.

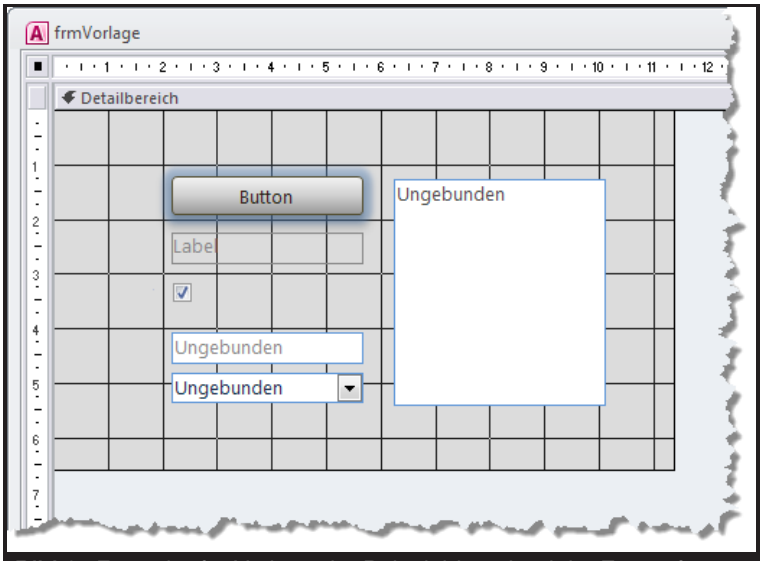


Bild 2: Formular frmVorlage der Beispieldatenbank im Entwurf

Datenbankvorlagen können alles enthalten, was es in einer Access-Datenbank gibt: Tabellen, Abfragen, Formulare, Berichte, Makros und Module, zuzüglich der eingestellten Optionen. Um eine Vorlage neu zu erstellen, legen Sie also einfach eine neue Datenbank an und entwerfen in ihr jene Elemente, die Sie später auch in anderen Datenbanken gebrauchen könnten.

Das könnten etwa Hilfsmodule mit allerlei immer wieder benötigten Grundfunktionen sein, Dialoge, Formulare mit einem bestimmten Layout, oder auch Tabellen, welche häufig genutzt werden. Adresstabellen etwa wären solche Kandidaten. Sollten Sie später in einer auf dieser Vorlage basierenden Datenbank die entsprechenden Elemente nicht benötigen, so löschen Sie sie einfach. Bauen Sie also eher mehr, denn weniger, in die Vorlage ein.

Die Beispieldatenbank enthält ein Formular **frmVorlage**, im Entwurf wiedergegeben in Bild 2. Ziel war es hier weniger, ein funktionsfähiges Formular zu erstellen, das direkt in anderen Datenbanken zum Einsatz kommen könnte. Das Augenmerk lag eher auf dem Design der Steuerelemente, die abweichend vom Standard formatiert wurden. Also wurden gängige Steuerelemente eingefügt und anschließend der **Corporate Identity** entsprechend formatiert. Die Schaltfläche etwa bekam ein Hintergrundleuchten, alle **Controls** einen blauen Rahmen, und die Schriftarten sowie -größen wurden auf benutzerdefinierte Werte gesetzt. Sie gestalten dies über den Ribbon und die Schaltflächen unter dem Reiter **Format | Steuerelementformatierung**. Damit alle neuen Steuerelemente später immer in genau diesem Gewand erscheinen, muss das jeweilige Format für dieses Formular allerdings erst zum Standard gemacht werden.

Dazu markieren Sie ein Steuerelement und klappen im **Ribbon**, Reiter **Entwurf**, die Galerie der Steuerelemente über die **DropDown-Box** rechts unten auf (Bild 3).

Wenn Sie nun auf den Eintrag **Steuerelementvorgaben festlegen** klicken, so speichert Access die Formatvorgaben für die markierte Steuerelementart in diesem Formular. Fügen Sie also später einen neuen Button in das Formular ein, so erscheint dieser ebenfalls mit dem Hintergrundleuchten. Ein neues Listenfeld hätte den blauen Rahmen. Der Vorgang zum Festlegen der Formatierung ist für jedes Steuerelement gesondert zu wiederholen. Diese Formatvorgaben bleiben auch dann erhalten, wenn

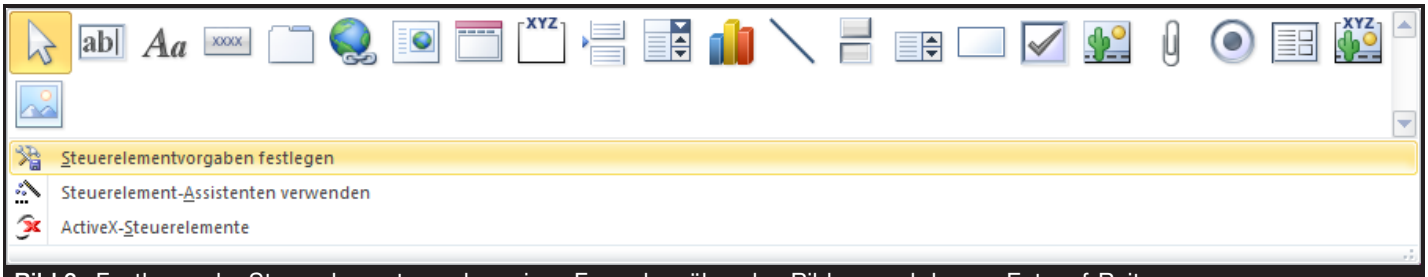


Bild 3: Festlegen der Steuerelementvorgaben eines Formulars über den Ribbon und dessen Entwurf-Reiters

Sie alle Steuerelemente aus dem Formular löschen!

Das Formular **frmVorlage** dient somit lediglich als Container für Formatstandards. Verwenden Sie es später in einer neuen Datenbank, so löschen Sie zuerst alle Steuerelemente in ihm und legen anschließend die wirklich benötigten an, wobei sie dann im neuen Design erstrahlen.

Ähnlich verfahren Sie mit Berichten oder zusätzlichen Formularen. Neben den Formatierungen macht es Sinn, zusätzlich Eigenschaften voreinzustellen. Das könnte etwa die Sichtbarkeit des Datensatzmarkierers, der Navigationsschaltflächen oder der Bildlaufleisten sein. Oder Sie geben vor, dass immer Kopf- und Fußbereich in einer bestimmten Größe vorhanden wären. Legen Sie in der Vorlage am besten einen ganzen Satz an grundlegenden Formularen und Berichten an, aus denen Sie später das jeweils für einen Anwendungszweck geeignetste auswählen.

Verfrachten Sie immer benötigte VBA-Funktionen in ein Modul und benennen es eindeutig; zum Beispiel mit **modHilfsfunktionen**. Auch ein **AutoExec**-Makro macht sich gut. Sein Inhalt könnte ja zunächst aus lediglich einem Kommentar bestehen.

Sinnvoll ist es zudem, die **Datenbankoptionen** der Vorlage auf brauchbare Einstellungen zu bringen. Über **Datei | Optionen | Aktuelle Datenbank** gelangen Sie zum für die Vorlage relevanten Teil, wie in Bild 4. Deaktivieren Sie hier etwa die von Access zunächst immer eingeschaltete **Objektnamen-Autokorrektur**, welche die Datenbanken aufbläht und verlangsamt. Auch **Entwurfsänderungen an Tabellen in der Datenblattansicht** haben in einer produktiven Datenbank nichts zu suchen. Nachdem Sie diese Optionen eingestellt haben, werden Sie automatisch auch in der auf der Vorlage basierenden Datenbank aktiv.

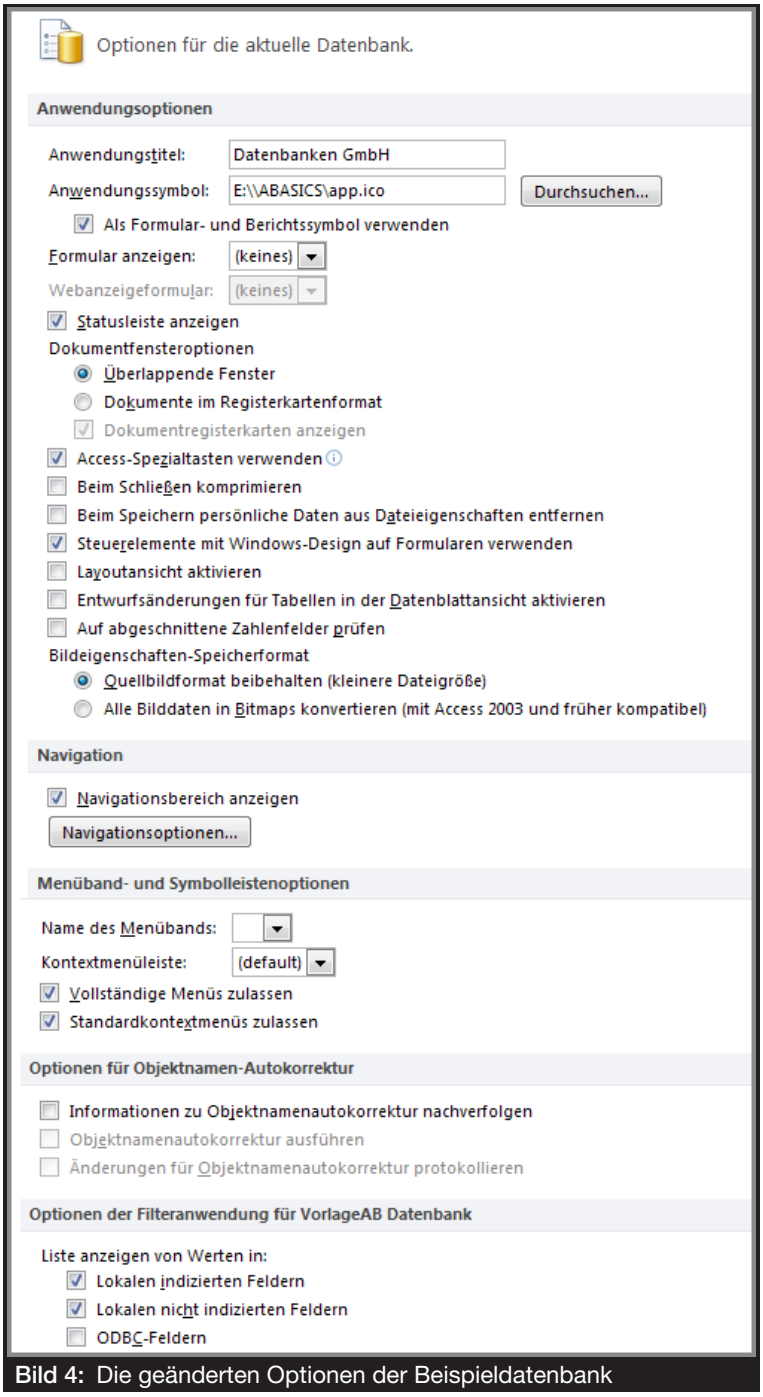


Bild 4: Die geänderten Optionen der Beispieldatenbank

Schließlich speichern Sie die Vorlage über den Backstage (**Datei|Speichern und veröffentlichen|Datenbank speichern als...**), indem Sie als Zielformat den Eintrag **Vorlage (*.accdt)** auswählen. Abweichend vom Verhalten beim Speichern einer normalen Datenbank (**accdb**) werden Sie nun mit einem Dialog konfrontiert, der zusätzliche Angaben zur Vorlage abfragt (Bild 5). Außer informativen Einträgen lässt sich auch das Verhalten bei Anwendung der Vorlage bestimmen. Möchten Sie auch die Tabellen der Vorlage transferieren, so

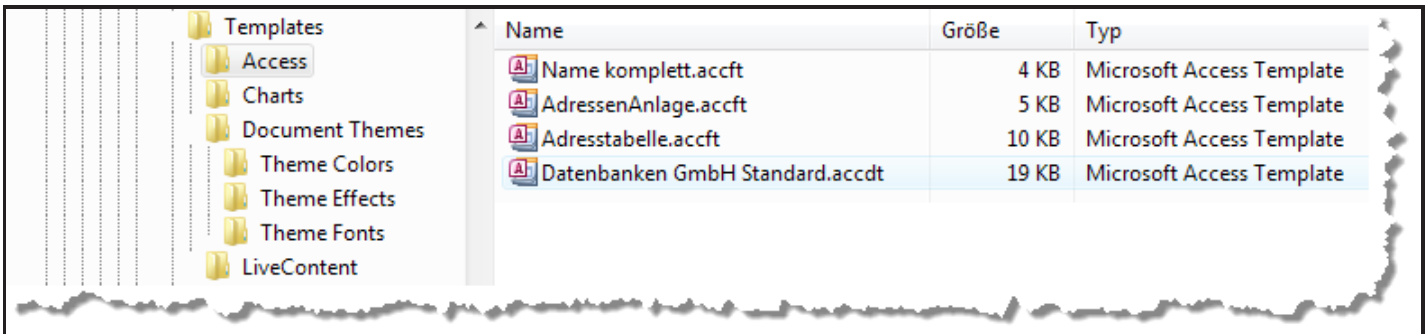


Bild 6: Unter diesem Zweig speichert Access die Vorlagendateien automatisch pro Benutzer ab

aktivieren Sie das Kontrollkästchen **Daten in Vorlage einbeziehen.** Zur Bedeutung der Einstellung **Anwendungspart** kommen wir noch später. Außerdem bestimmt der etwas skurrile Begriff **Instanziierungsformular** dasjenige Formular, welches später als **Startformular** der Datenbank in Erscheinung treten soll. Das können Sie sich allerdings sparen, wenn diese Angabe bereits in den **Datenbankoptionen** hinterlegt wurde.

Als **Kategorie** können Sie lediglich den Eintrag **Benutzervorlagen** auswählen, und als **Symbol** eine Grafikdatei vorzugsweise im **PNG-Format** angeben, die später als Bild neben dem Titel der Vorlage erscheint. Alle Angaben in diesem Dialog, außer dem Namen, sind optional.

Nach Klick auf **OK** speichert Access die Vorlage ab, wobei eine Fortschrittsanzeige den Vorgang nachvollziehbar macht. Nach dem Ort der Speicherung, also dem Speicherverzeichnis, werden Sie indessen nicht gefragt. Der wird nämlich immer das Benutzerverzeichnis Ihres Windows sein. Bild 6 zeigt den entsprechenden Zweig im Explorer unter

```
C:\Users\<Benutzer>\AppData\Roaming\
    Microsoft\Templates\
```

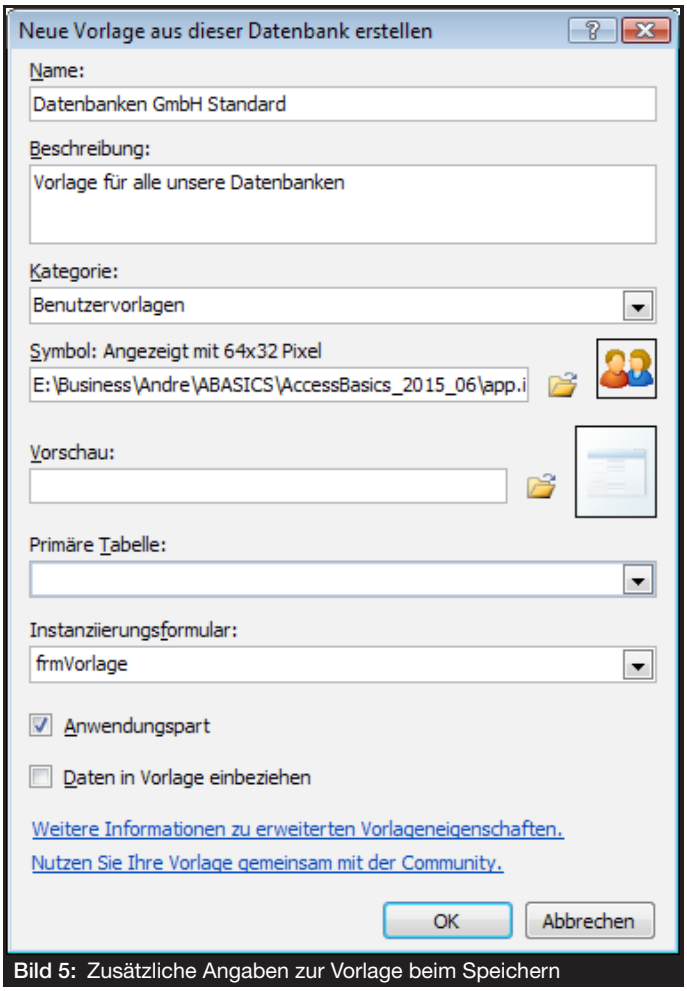


Bild 5: Zusätzliche Angaben zur Vorlage beim Speichern

an. Die Vorlagendatei hat dabei denselben Dateinamen, der für die Vorlage als Name vergeben wurde, gefolgt von der Endung **accdt**.

Sie sind neugierig und möchten erfahren, was Access eigentlich in der Vorlagendatei abgespeichert hat? Sie könnten auf die Idee kommen, die Datei in ***.accdb** umzubenennen und sie dann zu öffnen. Das aber wird fehlschlagen, weil es sich bei diesen Dateien nicht um Access-Datenbanken handelt, sondern um ein **Open-Document-Format**, ähnlich zum Format **docx** unter Word. Die Dateien sind ZIP-komprimiert und enthalten eine Menge **XML-Dateien**, die die Eigenschaften der Elemente der Vorlage bestimmen. Außerdem finden sich Textdateien in ihnen, die den gleichen Inhalt haben, wie Dateien, die über die versteckte VBA-Anweisung **SaveAsText** entstehen. Access entnimmt also diesen Dateien den Aufbau einer Datenbank – es handelt sich somit um Steuerdateien. Im Prinzip braucht

Datentypen konvertieren unter VBA

Ohne Variablen kommt fast keine Programmiersprache aus. Variablen wiederum sind an bestimmte Datentypen gebunden, die die Programmiersprache vorgibt. Die Anforderung, den Wert einer Variablen in den einer anderen Variablen unterschiedlichen Typs zu überführen, stellt sich über die Parameter von Funktionen häufig. Was es dabei zu beachten gibt, beleuchten wir in diesem Beitrag.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1506_Konvertierungen.accdb**

Datentypen unter VBA
Imgrunde müssten Sie sich unter VBA keine Gedanken über Dateitypen machen. Denn der Standarddatentyp ist hier der **Variant**. Wenn in den Modulen Ihrer Datenbank kein **Option Explicit** im **Modulkopf** steht, so nimmt VBA für jede Variable automatisch an, dass es sich um einen **Variant** handelt. Sie brauchen die Variablen dann weder über **Dim**-Anweisungen zu deklarieren, noch ihren Datentyp vorzubestimmen. VBA versucht dann intern selbst, den Datentyp zu erraten. Denn ein Variant ist natürlich kein wirklicher Datentyp, sondern eine Kapselung von allen erdenklichen Typen. Ob Text, Zahl, Datum oder Objekt, alle können im Variant unterkommen, ohne dass dem Programmierer bewusst wird, was hier im Hintergrund geschieht.

VBA leitet aus der Wertzuweisung ab, um welchen Typ es sich handeln dürfte. Bei

```
sText = "ACCESS BASICS"
```

ist durch die Anführungszeichen deutlich gemacht, dass es sich um einen **String** handelt. VBA speichert den Text nun als Variant, der intern ein Kennzeichen enthält, welches mit dem Marker für einen String versehen wird. Die eigentlichen Daten werden je nach Typ in einer bestimmten Datenstruktur abgelegt, über die man ebenfalls nichts erfährt. Ein Zeiger auf die Struktur macht den Variant vollständig.

Nicht immer ist die Sache jedoch so eindeutig. Nehmen Sie etwa diese Wertzuweisung:

```
Dim A As Variant
A = 123
```

Was nimmt VBA hier als Datentyp an? Dass es sich um eine Zahl handelt, ist nicht schwer zu erraten. Doch liegt hier ein **Long**-Wert vor, ein **Integer**, oder ein **Double**-Wert?

Sie sind nicht auf Spekulationen angewiesen, denn die Funktion **TypeName** von VBA kann Auskunft darüber geben, wie VBA die Variable sieht:

```
Debug.Print TypeName ( A)
> "Integer"
```

Hoffentlich liegt es hier richtig! Denn folgende Formel könnte zu einem Fehler wegen Überlaufs führen:

```
A = A * A * A
Debug.Print A, TypeName(A)
```

Eine Fehlermeldung tritt jedoch nicht auf. Das Ergebnis wird korrekt angezeigt, und der Datentyp ist auf **Long** gewechselt. Ähnlich verhält es sich mit

```
Debug.Print TypeName (A / 3)
> "Double"
```

Aber auch dies ist möglich:

```
A = "123"
Debug.Print TypeName (A)
A = A / 3
Debug.Print TypeName (A)

> "String"
  "Double"
```

Obwohl hier VBA zunächst einen **String** annimmt, stolpert es im nächsten Schritt über den Teilungsoperator und verwandelt deshalb schnell den Datentyp in **Double**, um die Berechnung durchführen zu können.

Anderes Beispiel:

```
A = 123
Debug.Print TypeName(A)
A = A & "ACCESS"
Debug.Print A, TypeName(A)
> "Integer"
  "String"
```

Durch den Verknüpfungsoperator **&** für Strings errät VBA, dass das Ergebnis ebenfalls ein String sein muss, weshalb der Wert der **Integer**-Variablen **A** intern zunächst in einen **String** umgewandelt und schließlich der hartkodierte Text angefügt wird.

VBA scheint also ziemlich intelligent zu sein, wenn es um das Erraten der benötigten Datentypen geht. Das geht auch solange gut, wie Sie lediglich die VBA-Bibliothek selbst benutzen. Sobald Sie aber auch andere Bibliotheken verwenden, etwa die Datenzugriffsbibliothek **DAO**, ist VBA häufig überfordert.

Denn bei der Zuweisung eines Variablenwerts etwa zum Feldinhalt einer Tabelle weiß VBA unter Umständen nichts über den Datentyp, den Sie beim Entwurf der Tabelle für das Feld vorsahen. Hier kann es zu Fehlern wegen nicht übereinstimmender Datentypen kommen, oder zu Fehlern durch Überläufe.

Option Explicit

Um solche undurchsichtigen Fehler zu vermeiden, setzen Sie besser in den Modulkopf jedes Ihrer Module ein **Option Explicit**. Das weist VBA an, jede im Programmcode auftauchende Variable zu überprüfen und deren Deklaration zu suchen. Findet es keine, so tritt beim Kompilieren eine entsprechende Fehlermeldung auf. Das manuelle Einsetzen der **Option-Explicit-Anweisung** können Sie sich sparen, wenn Sie in den **Optionen** von VBA (Menü **Extras**) den Eintrag Variablendeklaration erforderlich aktivieren (siehe Bild 1). Dann nämlich setzt VBA beim Neuanlegen eines Moduls die Anweisung automatisch in den Modulkopf.

Die Variablendeklarationen können Sie, je nach deren Gültigkeitsbereich, in den Modulkopf setzen, oder in die einzelnen Funktionsroutinen. Die Stelle spielt dabei keine Rolle. Eine **Dim**-Anweisung kann auch irgendwo mitten im Funktionstext stehen.

Die Schlüsselwörter für die Deklarationen lauten **Dim**, **Private**, **Public**, **Friend** und **Static**. **Static** ist nur innerhalb von Routinen erlaubt, **Public**, **Friend** und **Private** nur im Modulkopf.

Neben dem Namen der Variablen ist zusätzlich deren Datentyp anzugeben:

```
Dim A As Long
A = 123
```

Ganz stimmt das allerdings nicht, denn auch ein schnödes

```
Dim A
```

ist möglich. In diesem Fall nimmt VBA für A aber den Typ Variant an. Die Typbezeichnung lässt sich auch durch ein Kürzel ersetzen:

```
Dim A$
```

etwa deklariert die Variable als **String**. Die Kürzel im Überblick:

```
&  String (Text)
%  Integer (16bit-Integer)
&  Long (32bit-Integer)
^  LongLong (64bit-Integer:
       nur unter Office 64 verfügbar)
#  Double (Fließkomma, genau)
!  Single (Fließkomma, einfach)
@  Currency (Währung)
```

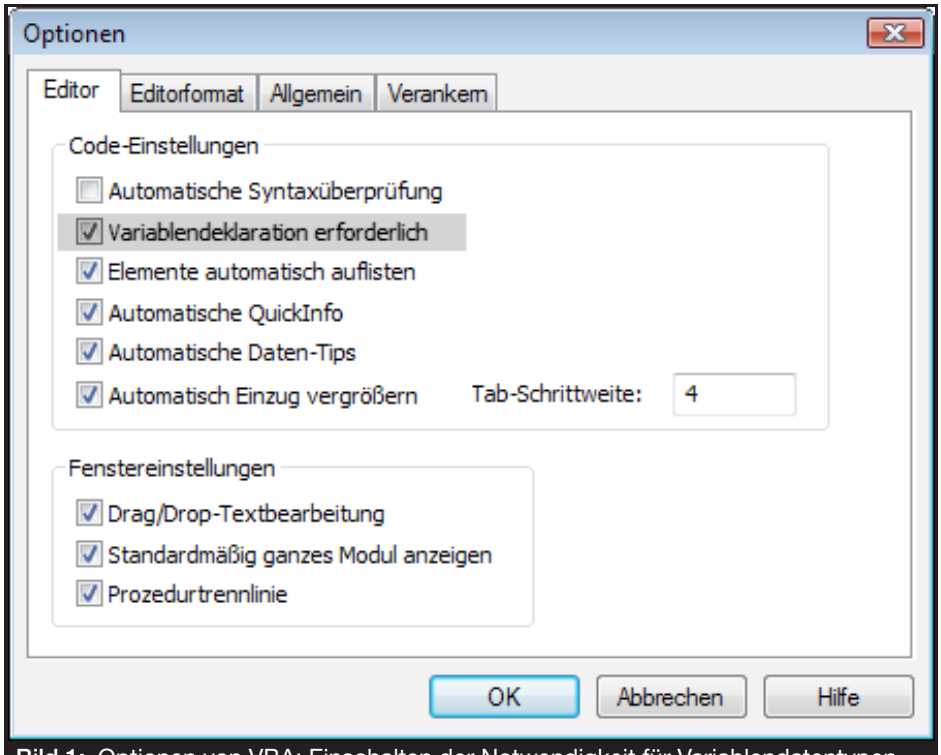


Bild 1: Optionen von VBA: Einschalten der Notwendigkeit für Variablendatentypen

Diese Kürzel machen den Programmcode allerdings schlecht lesbar.

Das strikte Erfordernis zur Variablendeklaration führt nun natürlich dazu, dass die Beispiele von oben nicht mehr alle funktionieren. Ist die Variable **A** als **Long** deklariert, so kann ihr natürlich kein durch Division zustande gekommener Double-Wert mehr zugewiesen werden. Die interne beim Variant von VBA vorgenommene Konversion schränkt sich damit ein, ist aber nicht außer Kraft gesetzt. Denn dies etwa geht nach wie vor fehlerfrei über die Bühne:

```
Dim A As Integer
Dim B As Long
B = 123
A = B
```

VBA weiß hier über beide Datentypen Bescheid, überprüft die Werte der Variablen und kann dann den einen Wert erfolgreich in den anderen überführen, solange der Bereich nicht überläuft, was beim **Long**-Wert 123 der nicht der Fall ist.

Wenn man sich auf die eingebaute Intelligenz von VBA aber nicht verlassen will, dann gibt einen ganzen Satz von Konvertierungsfunktionen, die die Bibliothek VBA bereithält.

VBA-Conversion

Rufen Sie den **Objektkatalog** im VBA-Editor auf, setzen die Combobox links oben auf die Bibliothek **VBA** und markieren die Klasse **Conversion**.

Rechts werden Ihnen nun sämtliche Konvertierungsmethoden aufgelistet (Bild 2). Die reinen Konvertierfunktionen beginnen alle mit dem Präfix C. Angehängt daran ist ein Kürzel für den Ergebnisdatentyp der Funktion. **CBool** etwa hat als Ergebnis einen **Boolean-Wert**, **CLng** einen **Long** und **CDate** ein Datum.

Als Parameter kann diesen Funktionen grundsätzlich ein beliebiger Datentyp verabreicht werden. Der übergebene Parameter hat also den Typ Variant. Was VBA ohne **Option Explicit** intern macht, führen diese Funktionen dabei dezidiert aus.

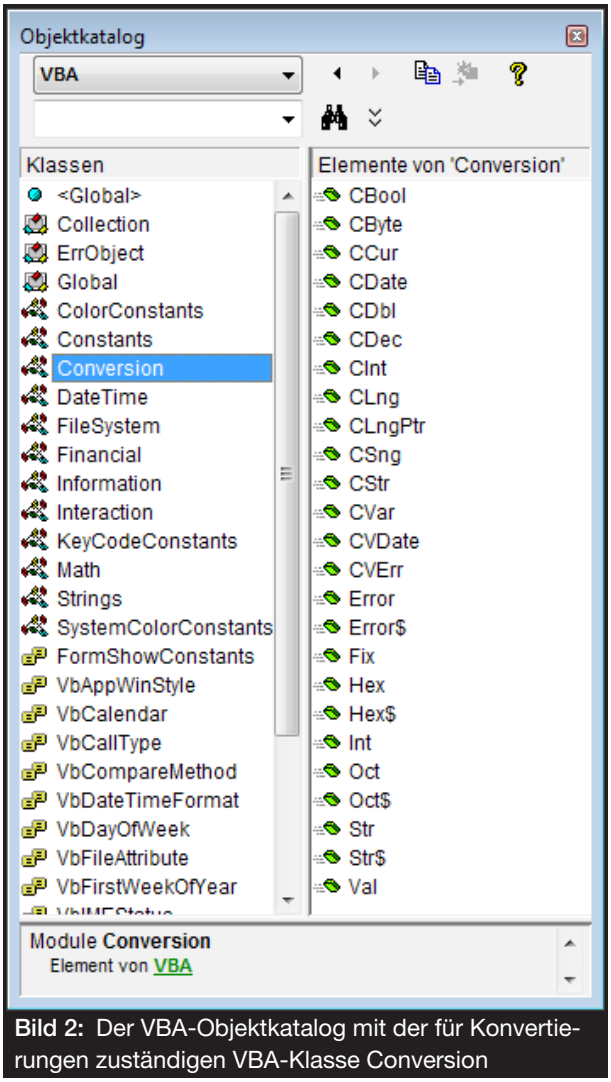


Bild 2: Der VBA-Objektkatalog mit der für Konvertierungen zuständigen VBA-Klasse Conversion

Das Beispiel zum Verketten des Werts einer Long-Variablen mit einem Text müsste nun so geschrieben werden:

```
Dim A As Long
Dim B As String
A = 123
B = "ACCESS"
B = B & CStr(A)
```

CStr macht aus der Zahl **123** in **A** einen String **"123"**, und dieser kann dann an die String-Variable **B** angehängt werden.

Beleuchten wir im Folgenden, was es mit den einzelnen Funktionen der Bibliothek auf sich hat.

Konvertierungsfunktionen in VBA-Routinen

Im Modul **mdlConverter** der Beispieldatenbank finden Sie etliche Funktionen, die wiederum die angesprochenen **C**-Funktionen von VBA verwenden. Listing 1 zeigt diese im Ausschnitt für die Konvertierung von Werten in **Boolean**- oder **Byte**-Werte. Alle weiteren Funktionen des Moduls sind ähnlich aufgebaut.

Der Rückgabetyp der **fBool**-Funktion ist **Boolean**. Damit dieser Rückgabewert gesetzt werden kann, muss der als **Variant** übergebene Parameter **Value** über die **CBool**-Methode dezidiert umgewandelt

Mathematische VBA-Funktionen

Abseits der Grundrechenarten kommen mathematische Berechnungen sowohl immer wieder in VBA-Routinen zum Einsatz, wie auch in Abfragen auf Tabellenfelder. Einen Überblick über das, was VBA uns zu diesem Thema anzubieten hat, zeigen wir im Folgenden – und ebenso, wie sich der Funktionsumfang durch die Excel-Bibliothek erweitern lässt.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1506_Mathe.accdb**

Die Bibliothek Math von VBA

Öffnen Sie den Objektkatalog des VBA-Editors und wählen links oben aus der Combobox den Eintrag **VBA** aus. Markieren Sie dann die Zeile **Math** in der Klassenauflistung links. Rechts werden Ihnen nun die Funktionen der Klasse angezeigt – siehe Bild 1.

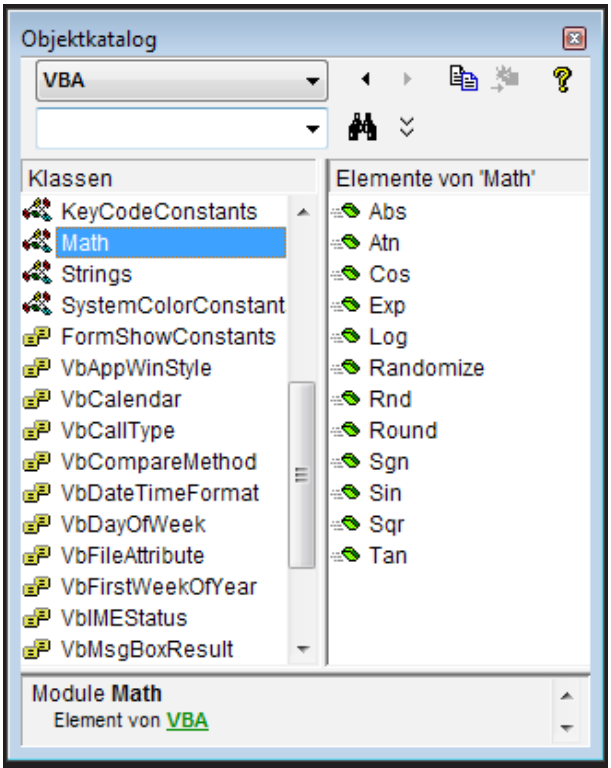


Bild 1: Die VBA-Klasse Math im Objektkatalog

Ziemlich überschaubar, was uns da geboten wird. Gerade mit zwölf Methoden werden wir beglückt. Wenn es um naturwissenschaftliche oder finanzielle Berechnungen geht, dann bleibt Ihnen nicht erspart, die komplexeren Funktionen selbst in Modulen nachzurüsten und zu programmieren.

Aber durchlaufen wir einmal das Vorhandene. Zunächst für die einfacheren Methoden...

Abs gibt den **Absolutwert** einer Zahl zurück. Das bedeutet nichts anderes, als dass eine negative Zahl in eine positive verwandelt wird. **+13** und **-13** haben beide den gleichen Absolutwert **13**.

Sgn ist ähnlich, gibt aber lediglich das Vorzeichen der Zahl zurück. Allerdings nicht als **String**, sondern als Zahl. Jede Zahl größer als 0 ergibt eine 1, jede kleiner als 0 eine -1. Nur bei 0 ist auch der Rückgabewert eine 0. **Round** rundet eine Fließkommazahl auf eine wählbare Anzahl von Nachkommastellen:

```
? Round (2.4674)
> 2,47
```

Lassen Sie Vorsicht walten, wenn es um das Runden von Zahlen geht, besonders wenn es um Währungsberechnungen handelt! Denn dort ist Runden nicht gleich Runden. Mathematisches Runden und kaufmännisches Runden etwa unterscheiden sich. Unterziehen Sie einmal die Zahl **2,465** einem VBA-**Round**. Als Ergebnis erhalten Sie **2,46**! Die **5** am Ende wird also nicht aufgerundet. Für korrektes Runden benötigen Sie daher eine eigene Funktion, die etwa so aussähe:

```
Function Round2( _
    No As Variant, _
    Dgts As Long) _
    As Variant
    Dim N As Double
    Dim F As Double
    F = 10 ^ Dgts
    N = 0.5 + No * F
  Round2 = CLng(N) / F
End Function
```

Hier wird **F** die Zehnerpotenz der Nachkommastellen zugewiesen. Für 2 stünde in der Variablen dann die 100. Das wird mit der übergebenen Zahl **No** multipliziert und 0.5 addiert.

Schließlich wird per **CLng** der Nachkommateil abgeschnitten und das Ergebnis wieder durch den Faktor **F** dividiert. Sie können das im Modul **mdlMathe** der Beispieldatenbank nachvollziehen, indem Sie die Routine **TestRound** aufrufen.

Rnd im Verein mit **Randomize** sind keine mathematischen Funktionen im eigentlichen Sinne. **Rnd** erzeugt eine Zufallszahl (**Single**) im Bereich von 0 bis 1. Um andere Wertebereiche für die Zufallszahlen zu generieren, multiplizieren Sie einfach den Rückgabewert mit einem Faktor und/oder addieren Werte dazu. Beispiel für eine Funktion, die Zufallszahlen im Bereich 10 bis 100 generiert:

```
Function Rnd100() As Single
    Randomize
    Rnd100 = 10 + 90 * Rnd()
End Function
```

Das **Randomize** können Sie auch weglassen. Es initialisiert lediglich den internen Generator von VBA neu, der unter anderem auf der Systemzeit beruht und darauf eine Sequenz ablaufen lässt. Die Gefahr von Wiederholungen ist aber nahezu ausgeschlossen. Mit **Sqr** ermitteln Sie die **Quadratwurzel** einer Zahl. Die Funktion ist weitgehend überflüssig, da das Ergebnis auch über Potenzierung zu erhalten ist:

```
? Sqr(10)
? 10 ^ 0.5
```

Log berechnet den natürlichen Logarithmus der übergebenen Zahl. **Natürlich** bedeutet, dass die Basis **e** herangezogen wird. Wenn Sie aber den Zehnerlogarithmus benötigen, so ermitteln Sie diesen über einen Einzeiler:

```
Function Log10 (Value As Double) _
                As Double
    Log10 = Log(Value ) / Log(10)
End Function
```

Die Umkehrfunktion ist **Exp**. Sie potenziert die Basis **e** mit der übergebenen Zahl. Das lässt sich allerdings auch einfach anders berechnen. Folgende Zeilen kommen zum gleichen Resultat:

```
? Exp(3)
? Exp(1) ^ 3
? 2,71828182845905 ^ 3
```

Das war es dann auch schon mit den naturwissenschaftlichen Funktionen. Trigonometrische gibt es auch nur vier:

Sin berechnet den **Sinus**, **Cos** den **Cosinus** des übergebenen Double-Arguments, welches nicht etwa als Winkel, sondern in der Einheit Bogenmaß vorliegen muss.

Eine Funktion, die den Winkel-Sinus berechnet sähe etwa so aus:

```
Function SinWinkel(Value As Double) _
                    As Double
    Dim pi As Double
    pi = 4 * Atn(1)
    SinWinkel = Sin(Value / 360# * pi)
End Function
```

Für das Umrechnen von Winkel in Bogenmaß benötigen wir die Zahl **PI**. VBA kennt diese jedoch nicht. Abhilfe schafft die Funktion **Atn** (**ArcTangens**), die mit dem Parameter 1 ein Viertel des **PI** erzeugt. Der Rest ist selbsterklärend.

Tan ermittelt den **Tangens** eines Winkels. Auch hier natürlich im Bogenmaß. Die Angabe einer eigenen Funktion für den Winkel-Tangens braucht es hier nicht – sie sähe analog aus, wie der Ersatz zuvor für den Sinus.

Eine Funktion für den **CoTangens** fehlt. Die wird aber auch nicht wirklich vermisst, errechnet sich dieser doch einfach über den Kehrwert des **Tangens**:

```
Function CoTan(Value As Double)
    CoTan = 1 / Tan(Value)
End Function
```

Atn berechnet den **ArkusTangens** eines Werts. Auch hier kommt ein Bogenmaß heraus. Soll es ein Winkel sein, so dividieren Sie das Ergebnis eben einfach durch **PI**, wie in der Routine **AtnWinkel** des Moduls.

Weitergehende Mathe-Funktionen

Wenn Sie naturwissenschaftliche Berechnungen anstellen müssen, die über den bescheidenen Umfang von VBA hinausgehen, so programmieren Sie dafür eigene Routinen. Anregungen dafür gab es bereits im Vorhergehenden. Aber auch im Web gibt es Unmengen von Beispielen und Sammlungen, die eher auf **Excel**-Seiten beheimatet sind, als auf Access-Seiten. Warum das Rad neu erfinden, wenn das Glück auf der Straße liegt? An dieser Stelle jedenfalls ist eine Darstellung solcher Funktionen in Listings nicht erforderlich – zu unterschiedlich und umfangreich sind die Bedürfnisse.

Excel als Mathe-Helfer

Da **Excel** in der Regel auch da installiert ist, wo sich Access befindet, wäre es töricht, ausgerechnet den Spezialisten in Sachen Mathematik-

Arbeiten mit externen Bibliotheken

Wahrscheinlich sind Sie schon einmal auf Begriffe gestoßen, wie Verweise, VBA-Bibliotheken oder Type Libraries. Alle bezeichnen dabei dieselbe Sache. Klären wir auf, was es damit auf sich hat, und wie Sie als VBA-Programmierer mit VBA-Bibliotheken umgehen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1507_Libraries.accdb**

Das VBA-Projekt

Sobald Sie in eine Datenbank neu anlegen, erzeugt Access automatisch auch ein VBA-Projekt in ihr, das zunächst den gleichen Namen trägt, wie die Datei. Öffnen Sie den VBA-Editor, so steht dieser Name ganz oben im **Projekt-Explorer**. Dieser Name bezeichnet bereits die grundlegende Bibliothek des VBA-Projekts dieser Datenbank. Der Inhalt dieser Bibliothek wäre die Sammlung von Modulen und Klassen mit ihren Methoden, die Sie gegebenenfalls entwickeln.

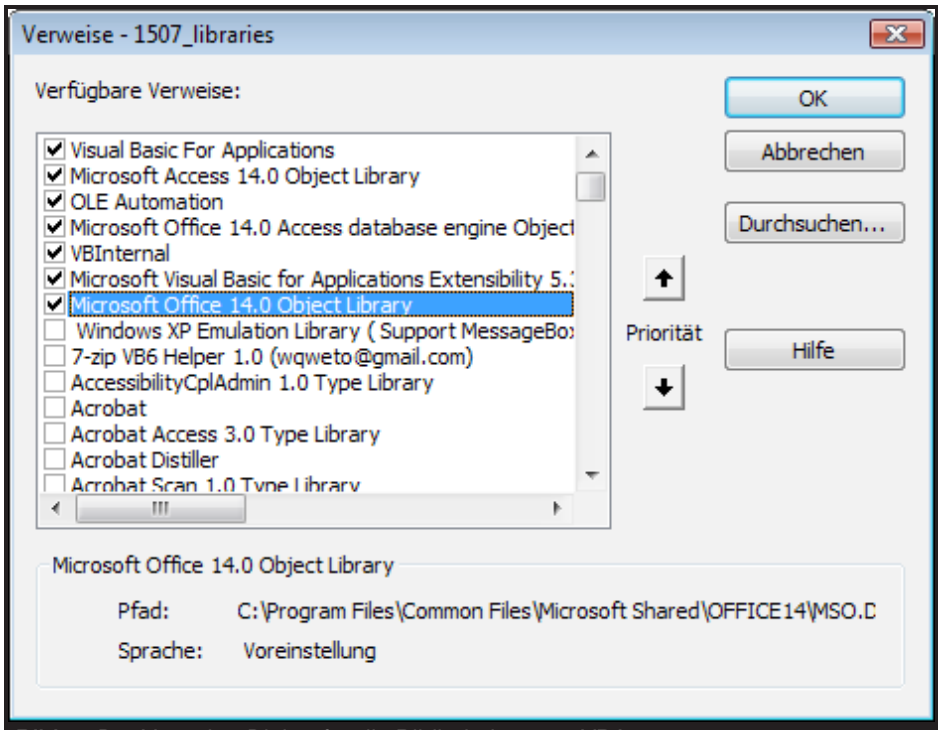


Bild 1: Der Verweise-Dialog für die Bibliotheken von VBA

Verweise

VBA funktioniert grundsätzlich nicht ohne solche Bibliotheken. Um bestimmte Funktionen und Anweisungen ausführen zu können, benötigt es eine Referenz auf diese. Von Haus aus weiß es nichts über Namen, wie **Left**, **Right**, **Abs**, **CurrentDb**, und so weiter. All diese Namen müssen aus Bibliotheken kommen, die erstens die Bezeichnungen kennen, und zweitens den Ort, wo sich die Funktionen befinden, auf die sie sich beziehen.

Auf den ersten Blick ist dieser Zusammenhang nicht unbedingt gleich ersichtlich. Denn beim Anlegen des VBA-Projekts einer Datenbank legt Access sogleich Referenzen auf zwei Bibliotheken fest, die Teil der Office-Installation sind: **VBA** und **Access**. Das sind die beiden grundlegenden Bibliotheken, mit denen unter VBA einer Datenbank gearbeitet wird.

Left und **Right** etwa stammen aus der Bibliothek **VBA**, während **CurrentDb** Teil der **Access-Library** ist. **Type Library**, kurz **Library**, und **Bibliothek** sind synonyme Bezeichnungen. Während diese Begriffe auch unter anderen Programmiersprachen gängig sind, spricht man unter VBA eher von **Verweisen**, oder englisch **References**. Korrekt wäre eigentlich die ausführliche Bezeichnung **Verweis auf eine Bibliothek**.

Welche Bibliotheken in das VBA-Projekt der aktuellen Datenbank integriert sind – ohne Datenbank auch kein VBA-Projekt und somit keine Bibliotheken –, erfahren Sie über den **Verweise**-Dialog des VBA-Editors. Öffnen Sie ihn über sein Menü und die Abfolge **Extras|Verweise...** . Dort (siehe Bild 1) finden Sie links eine Liste mit allen Bibliotheken, die auf Ihrem Rechner zur Verfügung stehen. Über die Kontrollkästchen können Sie darin beliebige Einträge aktivieren oder deaktivieren. Nach dem Schließen des Dialogs über **OK** sind

die jeweiligen Bibliotheken dann in das VBA-Projekt integriert oder aus ihm entfernt. Um zu erläutern, woher die Einträge dieser Liste überhaupt kommen und wie sie funktionieren, ist etwas weiter auszuholen.

VBA holt sie sich aus der **Registry** von **Windows** und da aus dem Zweig **HKEY_CLASSES_ROOT\TypeLib**. Dort stehen alle Bibliotheken, die im System installiert wurden. Genauer gesagt handelt es sich um sogenannte **COM-Bibliotheken**, oft auch als **ActiveX-Komponenten** bezeichnet. Zu beschreiben, wie diese Komponenten genau arbeiten und wie die Registry-Schlüssel mit ihnen zusammenhängen, würde hier zu weit führen. Wesentlich ist nur, dass diese **COM-Bibliotheken** durch Installation von Komponenten zustande kamen und eine Art Inhaltsverzeichnis darstellen. Über die **GUID**-Schlüssel wird eine Verbindung von Name der Bibliothek und der ausführbaren Datei hergestellt, die in der Regel als DLL daher kommt, oder als EXE-Datei. Ein VBA-Verweis ist also nichts anderes, als eine Referenz auf eine ausführbare Datei samt ihrer Funktionsliste.

Diese speziellen **COM-DLLs** unterscheiden sich von anderen dadurch, dass sie außer den enthaltenen Exportfunktionen eben auch noch eine integrierte **Type Library** aufweisen. Nur diese kann VBA lesen. Der **Objektkatalog** von VBA ist fähig, ihren Inhalt darzustellen. Es gibt aber zusätzlich auch solche Verweise, die **nur** aus einer **Type Library** bestehen und gar keine binären Funktionen enthalten. Das sind etwa die Dateien mit der Endung **tlb** oder **olb**. Ein Beispiel wäre die Access-Bibliothek selbst, die in der Datei **msacc.olb** steckt, welche sich im Office-Installationsverzeichnis befindet. Bei erfolgreicher Installation von Access hat sie sich selbst als Type Library in die Registry eingetragen.

Dort findet VBA sie und lädt ihren Inhalt, also das Verzeichnis aller Funktionen von **Access**, falls ein Verweis auf sie im VBA-Projekt der Datenbank eingetragen ist. Das allerdings ist immer der Fall. Sie können ihn über den Verweise-Dialog auch gar nicht entfernen. Access verhindert die Manipulation der **Access**-Bibliothek, wie auch die von **VBA**. Mit alle anderen Verweisen können Sie verfahren, wie Sie möchten.

Sehen wir uns die Funktionsweise folgender VBA-Zeile an:

```
S = Mid$("ACCESS BASICS", 8, 2)
> "BA"
```

Der **Parser** von VBA analysiert sie und stößt auf den ihm unbekannten Ausdruck **Mid$**. Daraufhin klappert er die Namen der Funktionen aller geladenen Bibliotheken ab und wird unter **VBA** fündig. Deren Klasse Strings enthält diese Funktion **Mid$**. Weil es eine globale Funktion ist, deren Klasse immer zur Verfügung steht und nicht erst instanziiert werden muss, ist dieser Funktionsname gültig. Im weiteren Verlauf schaut der **Parser** nach der ausführbaren Datei, welche diese Funktion enthält. Er erfährt dies aus der **Registry**, wo festgehalten wurde, dass die Datei **vbe6.dll** für den Verweis verantwortlich sei. Im Windows-Suchpfad findet er sie schließlich unter **Common Files\Microsoft Shared\VBA** und lädt sie in den Prozess von Access. Da die Funktion Teil einer Objektklasse **Strings** ist, wird weiter automatisch eine Instanz dieser Klasse erzeugt. Nun kann in ihrem **Interface** die Funktion **Mid$** angesprungen werden, deren Speicheradresse ebenfalls über die **Type Library** ermittelt wird. Die Parameter für die Funktion können jetzt übergeben werden.

Wäre ein Verweis auf **VBA** nicht im Projekt geladen, so fände der **Parser** die Funktion **Mid$** auch nicht und quittierte dies beim Kompilieren mit der Fehlermeldung **Unbekannte Funktion in Ausdruck**.

Verweiseigenschaften

Was Sie in der Liste der Verweise zu sehen bekommen, sind nicht die Namen der Bibliotheken, sondern deren Beschreibungstexte. Für den Namen **VBA** lautet die Beschreibung etwa **Visual Basic For Applications**, für **Access** ist es **Microsoft Access 14.0 Object Library**. Sie laden den Verweis also über den Beschreibungstext, finden im Objektkatalog die Bibliothek dann aber über deren Namen.

VBA interessieren intern allerdings beide nicht. Der einzig wichtige Identifikator für die Bibliothek ist nämlich ihre **GUID**, die für Access den Wert

```
{4AFFC9A0-5F99-101B-AF4E-00AA003F0F07}
```

aufweist. Unter diesem Schlüssel ist sie in der **Registry** zu finden. Sowohl der Verweise-Dialog, wie auch der

Das Web im Formular

Links zu Web-Seiten findet man häufig in Anwendungen. Ein Klick auf sie öffnet in der Regel den Webbrowser des Systems, der zur gewünschten Seite navigiert und sie anzeigt. Wenn Sie derlei unter Access benötigen, so sind Sie nicht auf den externen Browser angewiesen, sondern können diesen direkt in Ihre Datenbankanwendung einbauen. Denn Access wartet mit einem eingebauten Steuerelement für diesen Zweck auf. Nehmen wir dieses hier genauer unter die Lupe.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1507_Web-Control.accdb**

Webdaten

Es gibt immer wieder Gründe, um Webdaten in Datenbanken zur Ansicht zu bringen. Ob das nun Informationen zu Produkten sind, Hilfeseiten, oder gar ein Web-Support, Anwendungsfälle gibt es viele. Den Trend, Hilfesysteme etwa nicht statisch im System zu speichern, gibt Microsoft selbst vor: Die Hilfe zu Office kommt mindestens zu einem Teil direkt aus dem Web, was die Möglichkeit offenlässt, den Inhalt kontinuierlich dynamisch zu verbessern. Auch Sie könnten Ihrer Anwendung solche Hilfeseiten spendieren und schlügen damit zwei Fliegen mit einer Klappe, denn einerseits kann die Hilfe dann innerhalb der Anwendung und andererseits abseits der Anwendung im Browser begutachtet werden. Die Quelle bliebe jedoch dieselbe.

Aber auch für die Anzeige von HTML-Inhalten, die lokal gespeichert sind, gibt es gute Gründe, denn die Gestaltungsmöglichkeiten für Webseiten sind nun mal weitaus vielseitiger, als das, was uns etwa **Textboxen** anbieten. Sobald Sie den Benutzer zum Lesen umfangreicheren Texts nötigen, ist die Strukturierung über HTML eine Überlegung wert, auch wenn Access mit dem **Richtext-Format** für **Textboxen** hier etwas nachgelegt hat.

Webbrowsersteuerelement

Seit **Access 2007** bietet **Microsoft** dieses Steuerelement an, welches HTML-Daten rendern kann. Dabei handelt es sich lediglich um eine Kapselung des **Internet Explorers** in einem Steuerelement. In ein Formular eingebaut zeigt es eine schlichte weiße Fläche, wie in Bild 1 für das Formular **frmWeb** der Beispieldatenbank. Wie die meisten anderen Steuerelemente auch weist es eine Datenbindung auf, die hier in der Bezeichnung **URL** zum Ausdruck kommt, denn das ist das gleichnamige Feld der Tabelle **tblHTML**, die als Datenquelle für das Formular herangezogen wurde. Im Eigen-

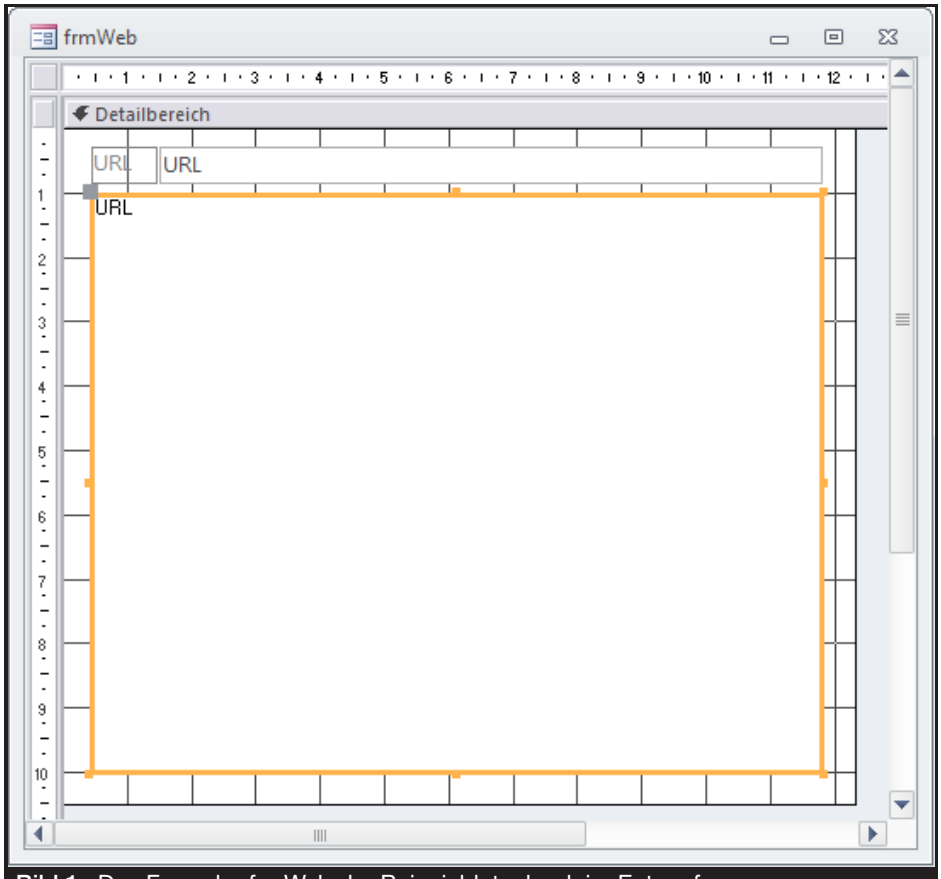


Bild 1: Das Formular frmWeb der Beispieldatenbank im Entwurf

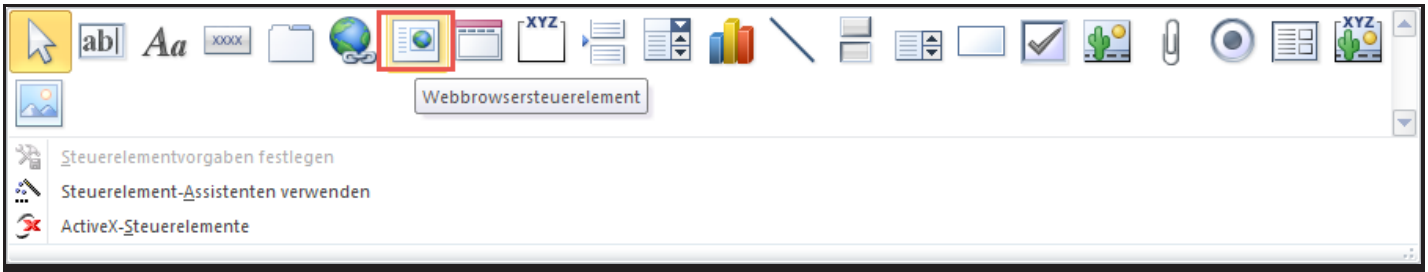


Bild 2: Das Webbrowsersteuerelement fügen Sie über den Ribbon-Reiter Entwurf in ein Formular ein

schaftenblatt zum **Webbrowsersteuerelement** geben Sie also unter **Steuerelementinhalt** jenes Feld einer Tabelle an, die den Link zur HTML-Seite enthält. Das Feld muss vom Datentyp Text oder Memo sein. Es ist wichtig festzuhalten, dass hier wirklich nur die URL zur Seite gespeichert sein darf, und nicht etwa der HTML-Inhalt selbst.

Das **Webbrowsersteuerelement** fügen Sie über den **Ribbon** in ein Formular oder einen Bericht ein, indem Sie im **Tab Entwurf** das entsprechende Symbol anklicken (siehe Bild 2). Gegebenenfalls versehen Sie es dann noch mit der **Verankern**-Einstellung **Quer und nach unten dehnen**, damit es sich der Größe des Formular anpasst.

Rufen Sie das Formular zur Laufzeit auf, so zeigt sich zunächst eine leere Fläche. Denn als **URL** steht im ersten Datensatz der Tabelle **tblHTML** der Eintrag **about:blank**. Was passiert, wenn Sie das Feld stattdessen einfach leer lassen, können Sie mit dem Sprung zum letzten Datensatz über die Navigationsschaltflächen des Formulars ermitteln. Dann nämlich zeigt der Browser die Meldung **Adresse ungültig** an, wie Sie das auch von Links zu nicht existierenden Webseiten her kennen. Aus diesem Grund ist für das Tabellenfeld **URL** auch der Vorgabewert **about:blank** eingetragen.

Klicken Sie sich im Formular zum zweiten Datensatz, so rendert das Steuerelement augenblicklich die Home-Seite von **Access im Unternehmen (siehe** Bild 3), der dritte führt zu **Access [basics]** und der fünfte zu einem **Youtube**-Video. Dies alles geschieht völlig ohne jegliche Programmierung! Sie brauchen sich lediglich darum zu kümmern, dass die zugrundeliegende Tabelle die korrekten URLs enthält, wobei es sich um Webadressen handeln kann, aber auch um URLs zu lokalen Dateien, wobei Sie dann den kompletten Pfad zur HTML-Datei angeben. Auf die Datenbindung an ein Feld können Sie sogar komplett verzichten, wenn Sie die URL als Steuerelementinhalt eintragen, wie in Bild 4. Hier ist die URL als **String** hinterlegt und der Webbrowser rendert diese Datei dann unabhängig von der Tabelle des Formulars, egal, auf welchem Datensatz Sie sich gerade befinden.

Innerhalb der Seite können Sie selbstverständlich auf Links klicken, um zu weiteren Seiten zu gelangen.

Bild 3: Der zweite Datensatz navigiert zur Webseite von Access im Unternehmen

Dass das nicht immer klappt, wie gewünscht, werden wir noch sehen.

Webbrowser-ActiveX-Steuerelement

Bisher wurde unterschlagen, dass es auch vor **Access 2007** bereits die Möglichkeit gab, einen Webbrowser in einem Formular unterzubringen. Der **Internet Explorer** bringt von Haus aus grundsätzlich dieses **ActiveX-Control** mit, das Sie über den **Ribbon Entwurf** und den Button **ActiveX-Steuerelemente...** einfügen (siehe Bild 2 unten). Wählen Sie dazu aus der Liste der ActiveX-Steuerelemente den **Microsoft Web Browser** aus.

Das Formular **frmWebAX** ist eine Kopie des vorigen **frmWeb**, nur dass statt des Access-Steuerelements das ActiveX-Steuerelement zum Einsatz kommt und zwei zusätzliche Schaltflächen zum Vor- und Zurücknavigieren eingefügt wurden. Bild 5 zeigt das Formular in der Entwurfsansicht. Im Unterschied zum **Access-Control** zeigt das **ActiveX-Control** das Windows-Logo. Ansonsten sind Funktionalität und Ergebnis zur Laufzeit gleich. Allerdings gibt es eine Unannehmlichkeit: Das ActiveX-Control weist keine Datenbindung auf. Die Eigenschaft **Steuerelementinhalt** fehlt ihm also. Dieses Manko kann aber mit nur einer Zeile Programm-Code ausgeglichen werden. In die Ereignisprozedur **Form_Current** (**Beim Anzeigen**) des Formulars schreiben Sie einfach diese Zeile:

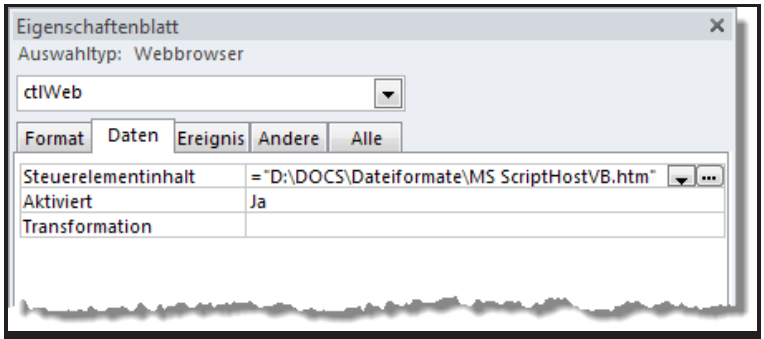


Bild 4: Die URL für das Webbrowsersteuerelement kann auch direkt ins Eigenschaftenblatt eingetragen werden

```
Me!ctlWebAX.Navigate2 Me!URL.Value
```

Das ActiveX-Control hat im Formular den Namen **ctlWebAX**. Seiner Methode **Navigate2** übergeben Sie als Parameter die URL, welche gerendert werden soll. Und die entnehmen Sie dem Tabellenfeld **URL** der zugrundeliegenden Tabelle **tblHTML**.

Das Access-eigene **Webbrowsersteuerelement** hat gegenüber dem ActiveX-Control des **Internet Explorer** also nur den einen Vorteil, dass diese kleine Programmierung entfallen kann. Dass man um eine Programmierung dann doch nicht herumkommt, zeigen die weiteren Ausführungen.

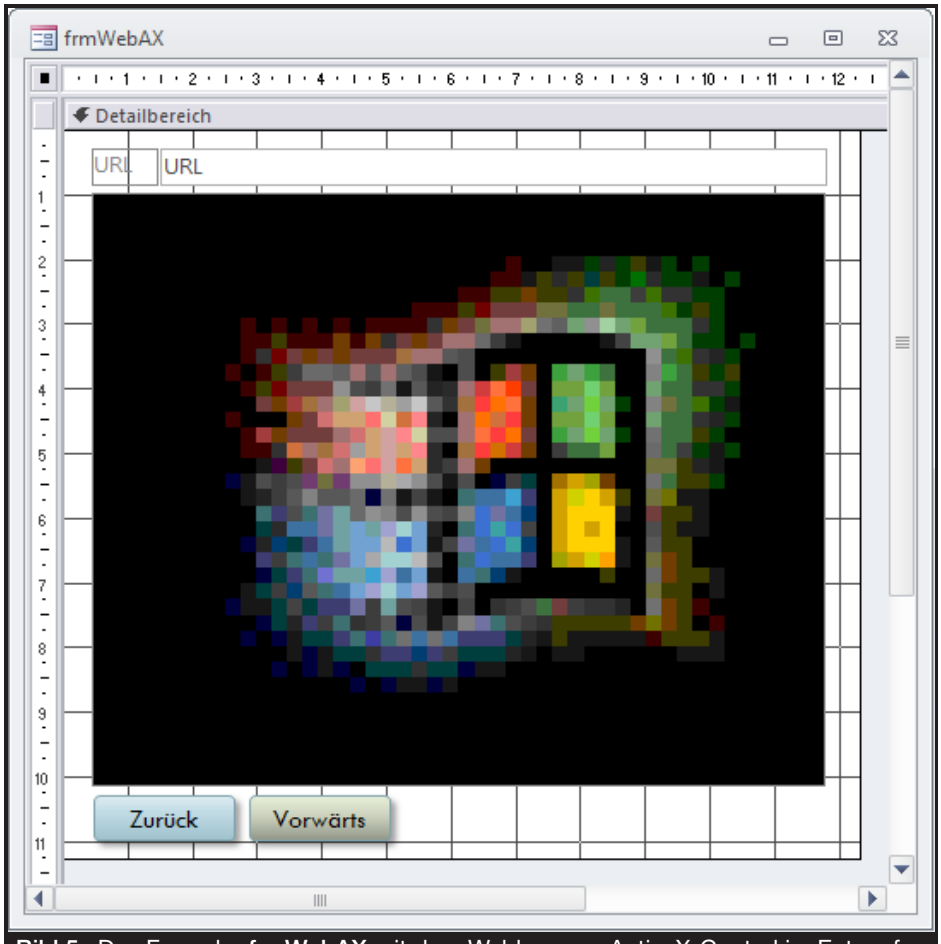


Bild 5: Das Formular **frmWebAX** mit dem Webbrowser ActiveX-Control im Entwurf

Scripting-Fehler

Sobald eine Webseite fehlerhafte **Skripte** enthält, wird dies in beiden **Webbrowser-Controls** über einen Dialog gemeldet (siehe Bild 6). In Ihrem externen Webbrowser wird die Anzeige dieser Fehler glücklicherweise unterbunden. Denn fehlerhafte **JavaScripts** sind auf den heutigen komplexen Websites eher schon die Regel, denn die Ausnahme, aber meist unbedeutend. Und da nervt die Fehleranzeige der **Webbrowser-Controls** ungemein.

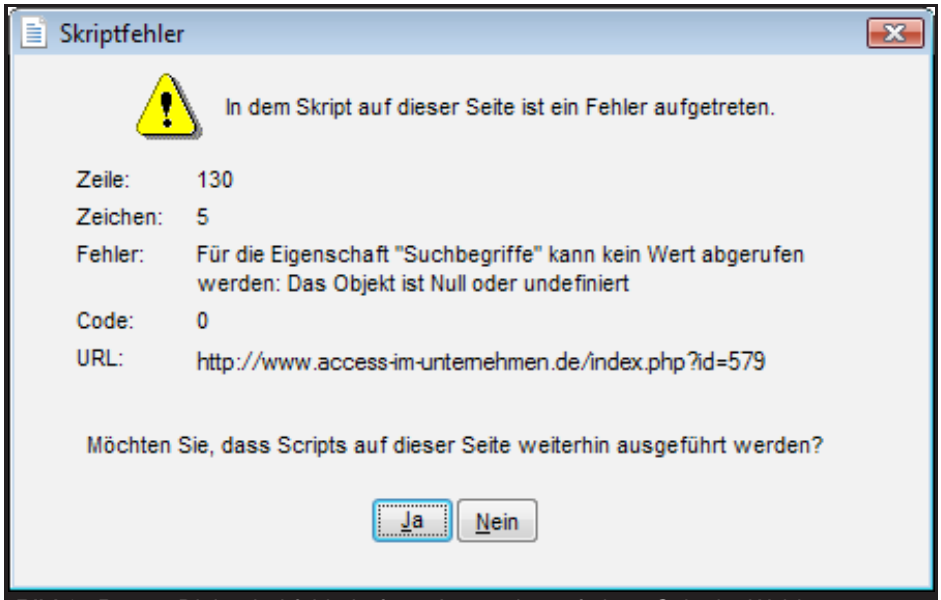


Bild 6: Popup-Dialog bei fehlerhaftem Javascript auf einer Seite im Webbrowser

Dieses Verhalten lässt sich jedoch unterbinden, indem die Eigenschaft **Silent** der Controls auf **True** gestellt wird. Markieren Sie das **Webbrowser-ActiveX-Control** im Formular **frmWebAX** und schauen Sie sich das Eigenschaftenblatt unter **Andere** an. Dort gibt es diese Eigenschaft **Silent**, die Sie nun auf **Ja** einstellen (Bild 7). Leider merkt sich das Steuerelement unter Access diese Einstellung nicht dauerhaft, weshalb es eine gute Idee ist, dies über Programmierung zu lösen. In die Ereignisprozedur **Beim Laden** (**Form_Load**) des Formular schreiben Sie dazu diese Zeile:

```
Me!ctlWebAX.Silent = 1
```

Und schon werden Sie, wie im stationären Browser, nicht mehr mit den Fehlermeldungen behelligt.

Inspizieren Sie hingegen das **Webbrowsersteuerelement** im Formular **frmWeb**, so suchen Sie die Eigenschaft **Silent** vergeblich. **Microsoft** hat im Steuerelement nämlich sämtliche Eigenschaften des **Internet Explorers** ausgeblendet. Sie kommen deshalb nicht umhin, auch hier programmatisch tätig zu werden:

```
Me!ctlWebAX.Object.Silent = 1
```

Der Unterschied zur vorigen Zeile besteht nur darin, dass hier der Ausdruck **Object** eingeschoben ist. Denn **Object** ist jene Eigenschaft, welche direkt das **Webbrowser-Objekt** zurückgibt, und dieses ist absolut identisch mit dem **ActiveX-Control**.

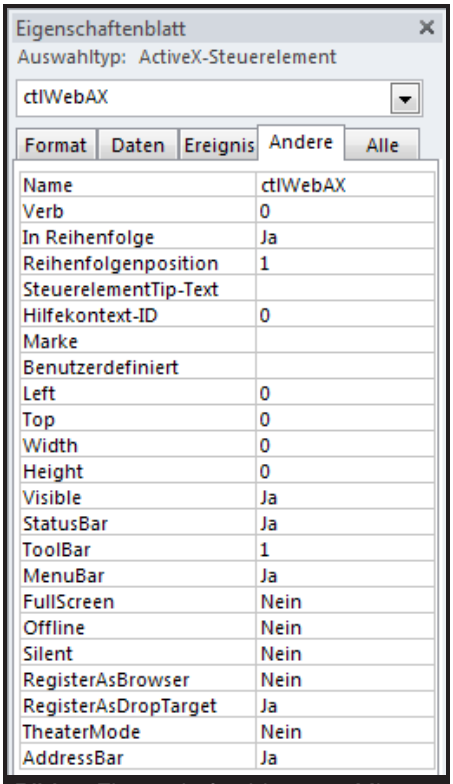


Bild 7: Eigenschaftenblatt zum Microsoft Web Browser-ActiveX-Control

Navigation über Seiten-Links

JavaScript-Fehler sind leider nicht das einzige Erfordernis, um das **Webbrowsersteuerelement** per Code zu manipulieren. Das nächste Problem tritt auf, wenn Sie auf einer Seite einen Link anklicken, der normalerweise ein neues Fenster oder einen neuen **Tab** im Browser öffnen würde. Das die **Webbrowser-Controls** solche **Tabs** nicht kennen, wird dann nämlich einfach ein echtes **Internet Explorer-Fenster** geöffnet und mit der Anzeige des Links beauftragt. Sie haben nun die Wahl, wie Sie es per Code lösen: Entweder Sie unterbinden die Navigation zu diesen Links ganz, oder Sie leiten die URL auf das aufrufende Fenster weiter, in

Besseres Code-Layout

Die Lesbarkeit von Programmcode beschleunigt sowohl die Entwicklung, wie auch die Wartung Ihrer Anwendungen. Wenn Sie ein Freund der VBA-Programmierung sind, so überprüfen Sie Ihre Routinen doch einmal auf Struktur und Gestalt. Gibt es hier Verbesserungspotential?

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1507_Code-Layout.accdb**

Strukturierung

Umso logischer Ihr Code aufgebaut ist, desto mehr Freude werden Sie später an ihm haben. **Später** kann dabei bedeuten, dass Sie schon am nächsten Tag möglichst schnell wieder in die Routinen hineinfinden und nicht unnötig Zeit damit verbringen müssen, die Funktionsabfolgen mühsam nachzuvollziehen. Erst recht gilt dies, wenn Sie sich zur Wartung oder Weiterentwicklung nach einem Jahr wieder an eine Datenbank machen müssen. Früher galt einmal das Credo, den Code ausführlich zu kommentieren, um später so einen Leitfaden vorzufinden, doch was nützt eine Kommentierung oder Dokumentation, wenn die Struktur unzureichend ist?

Modulstrukturen

Sie finden Funktionen leichter auf, wenn Sie sie auf mehrere Module aufteilen, statt alles in ein Modul zu packen. Sie haben eine Sammlung von immer wieder gebrauchten Routinen, etwa Dateifunktionen, Datumsberechnungen, String-Operationen und solche für den Datenzugriff? Leicht können dies Hunderte werden, auf die der Zugriff, falls in nur in einem Modul gespeichert, umständlich wird. Die Navigation ist behindert und ausdauerndes Scrollen wird notwendig. Der Überblick geht verloren.

Da ist es günstiger, wenn Sie Routinen mit ähnlicher Funktionalität in thematisch passenden Modulen vereinen. Legen Sie ein Modul für die Dateioperationen an, eines für die String-Bearbeitung und eines für für den Datenzugriff. Damit Sie dann wissen, um was es sich handelt, sind sprechende Namen für die Module angeraten. Eine Bezeichnung, wie **mdlFunktionen**, sagt wenig über den Inhalt aus. Besser fahren Sie da etwa mit **mdlFileAccess**, **mdlDataAccess-DAO**, **mdlStringHandling**.

Ähnlich, wie bei Variablen- oder Steuerelementnamen hat sich die Verwendung von Präfixen eingebürgert. Für normale Module kommt meist das Präfix **mdl** oder **mod** zum Einsatz, für Klassenmodule ein **cls** oder schlicht ein großes **C**. Englische Bezeichnungen oder deutsche? Das bleibt Ihnen überlassen. Da jedoch davon auszugehen ist, dass die meisten Programmierer mit englischen Fachbegriffen vertraut sind, liegt man mit englischen Bezeichnungen auf der sicheren Seite. So ist sichergestellt, dass es nicht zu Konfusionen kommt, wenn die Datenbankanwendung oder einzelne Module einmal die Ländergrenzen überschreiten sollten.

Die Frage, wie kleinteilig man Routinen auf Module verstreut, ist schwer zu beantworten. Natürlich könnten Sie Ihre Module so gestalten, dass der Code immer komplett auf eine Bildschirmseite passt. Das aber würde zu überhäufigem Umschalten zwischen den Modulen führen – schließlich werden die Prozeduren ja wahrscheinlich von anderen aufgerufen. Je umfangreicher andererseits ein Modul ausfällt, desto mehr Scrollen ist angesagt. Finden Sie hier also einen Mittelweg.

Prozedurstrukturen

Innerhalb eines Moduls macht eine logische Anordnung der Prozeduren ebenfalls Sinn. Was hier logisch ist, oder nicht, hängt von der Bedeutung der Routinen ab. Nehmen Sie etwa ein Formularmodul. Sinnvoll wäre hier eine Anordnung in der chronologischen Abfolge von Ereignissen. **Form_Open** (**Beim Öffnen**) ist das erste Ereignis, das ein Formular auslöst. Gefolgt wird es von **Form_Load** (**Beim Laden**) und schließlich **Form_Current** (**Beim Anzeigen**). Das Schließen des Formulars löst **Form_Close** (**Beim Schließen**) und **Form_Unload** (**Beim Entladen**) aus. Also würde eine Anordnung solcher Ereignisprozeduren von oben nach unten später nützlich sein. Ähnliches gilt für die Steuerelementereignisse, die sich an die Formularereignisse anschließen könnten. Hilfsfunktionen,

die von diesen Ereignisprozeduren aufgerufen werden, dürfen getrost ganz unten im Modul ihren Raum finden.

Auch bei normalen Modulen ist eine Anordnung der Routinen von bedeutsam bis Hilfsfunktion in vertikaler Richtung praktisch. Denn beim Öffnen eines Moduls werden Sie in der Regel mit den ersten Zeilen konfrontiert. Da macht es sich gut, wenn hier gleich die relevantesten Code-Teile zu finden sind.

Allerdings gibt es hier in punkto Lesbarkeit auch Einschränkungen. Nehmen Sie etwa folgenden Pseudocode:

```
Funktion A()
    [Code]
    Call Funktion B
    [Code]
Ende Funktion A

Funktion B()
    [Code]
Ende Funktion B
```

Funktion **B** wird also von Funktion **A** benötigt und aufgerufen. Als Hilfsprozedur fände Sie weiter unten im Modul ihren Platz. Stießen Sie in Funktion **A** bei Durchsicht später auf den Aufruf von **B**, so müssten Sie weit nach unten scrollen, oder über das Kontextmenü den Eintrag **Definition** auswählen, um zur Hilfsfunktion **B** zu gelangen. Besser ist es da, wenn die Hilfsfunktion direkt nach der aufrufenden steht. Versuchen Sie, die Prozedurreihenfolge möglichst kompakt zu halten. Würde Funktion **C** von mehreren Prozeduren benötigt, sähe das zum Beispiel so aus:

```
Funktion A()
    [Code]
    Call Funktion C
    [Code]
Ende Funktion A

Funktion B()
    [Code]
    Call Funktion C
Ende Funktion B

Funktion C()
    [Code]
Ende Funktion C

Funktion D()
    [Code]
Ende Funktion D
```

Überhaupt ist die Frage, in wie viele Prozeduren eine Aufgabe aufgeteilt werden kann und soll. Im Beispiel hätte man auf die Hilfsfunktion **C** ja auch verzichten und deren Code wiederholt in Prozedur **A** und **B** einsetzen können. Hier gilt aber der Grundsatz, dass wiederverwendbarer Code sich nicht wiederholen, sondern in eigene Funktionen ausgelagert werden sollte. Erstens macht es die Routinen abermals besser wartbar, denn Änderungen oder Weiterentwicklungen müssen so nur an einer Stelle (Funktion **C**) vollzogen werden, statt an mehreren. Und zweitens sind kürzere Prozeduren leichter überschaubar, als Spaghetti-Code, der sich länglich hinzieht.

Dennoch sollte man diese Strukturierung auch nicht zu weit treiben. Wenn Funktion **A** die Funktion **B** aufruft, diese wiederum Funktion **C**, und **C** verlangt nach **D**, dann ist zum Verstehen von A die ganze verteilte Kette **B** bis **D** zu überblicken. Ist das Ganze dann auch noch über mehrere Module verteilt, dann kann von Überblick eben keine Rede mehr sein. Von sehr komplexen Aufgaben abgesehen, sollte daher in der Regel die Verteilung der Funktionalität auf ein bis zwei Unterebenen ausreichen.

Prozedurstrukturierung

Innerhalb einer Prozedur gibt es eine Menge Gestaltungspotential. Neben dem reinen Layout, auf das wir später noch zu sprechen kommen, fragt sich etwa, wo sich die Dim-Anweisungen zur lokalen Variablendeklaration befinden sollten.

Meist stehen diese ganz am Anfang, müssen dies aber nicht. VBA ist es egal, ob die Variablen alle versammelt zu Anfang deklariert werden, oder irgendwo mitten im Code stehen. Zwar ist es weniger fehleranfällig, wenn die Dim-Anweisungen in einem Block stehen, bei Routinen mit 20 Variablen aber ist dieser Block dann schon ziemlich umfangreich. Stoßen Sie in der Prozedur auf eine Variablenzuweisung, haben den Datentyp der Variablen aber nicht mehr im Kopf, so ist abermals Scrollen zum Prozedurkopf angesagt. Aus diesem Grund kann es sinnvoll sein, die Variablendeklarationen in langen Prozeduren in mehrere Blöcke aufzuteilen. Variablen, die erst weiter unten verwendet werden, könnten dazu eben auch erst später in einem gesonderten Block deklariert werden. Das erhöht wieder die Übersichtlichkeit.

Andere Deklarationen

Im Kopf eines Moduls stehen die Deklarationen zu modulweit oder global gültigen Variablen, sowie gegebenenfalls die API-Deklarationen. Unter Umständen sammelt sich hier viel an, wie etwa Listing 1 zeigt.

```
Option Compare Database
Option Explicit

'Öffentliche Konstanten
Public Const GWL_STYLE As Long = -16
Public Const GWL_WNDPROC As Long = -4

'Private Konstanten
Private Const GW_CHILD As Long = 5
Private Const GW_HWNDFIRST As Long = 0
Private Const GW_HWNDLAST As Long = 1
Private Const GW_HWNDPREV As Long = 3

'Öffentliche Typen
Public Type Guid
    Data1 As Long
    Data2 As Integer
    Data3 As Integer
    Data4(0 To 7) As Byte
End Type

'Private Typen
Private Type LUID
    LowPart As Long
    HighPart As Long
End Type

'Öffentliche API-Funktionen
Public Declare Function GetWindow Lib "user32.dll" ( _
    ByVal hwnd As Long, _
    ByVal wCmd As Long) As Long

'Private API-Funktionen
Private Declare Function SetWindowPos Lib "user32.dll" ( _
    ByVal hwnd As Long, _
    ByVal hWndInsertAfter As Long, _
    ByVal x As Long, _
    ByVal y As Long, _
    ByVal cx As Long, _
    ByVal cy As Long, _
    ByVal wFlags As Long) _
    As Long

'Öffentliche Variablen
Public dbs As DAO.Database

'Private Variablen
Private i As Long, j As Long

'Prozeduren ----------
```

Listing 1: Beispiel für Deklarationen in einem Modulkopf

Es ist nützlich, wenn Sie wissen, wo Sie hier was finden können. Stoßen Sie im Code einer Prozedur auf eine Konstantenbezeichnung und möchten deren Wert erfahren, so ist es einfacher, einen Block von Konstanten zu begutachten, als den gesamten Modulkopf nach ihr abzusuchen.

Packen Sie also besser alle Konstanten in einen Block und verfahren Sie mit Typdefinitionen, API-Deklarationen und Variablen genauso. Wenn Sie immer den gleichen Aufbau erzeugen, finden Sie später schneller die gewünschte Stelle im Modulkopf.

Als geeignet hat sich dabei dieser Aufbau erwiesen: Stellen Sie Konstanten ganz oben ins Modul. Das ist nicht unwichtig, denn unter Umständen kommen diese Konstanten im weiteren Verlauf in Typen oder API-Funktionen zum Einsatz. VBA findet diese Konstanten aber nur dann, wenn Sie vor diesen Definitionen deklariert wurden. Dasselbe gilt für die Typendefinitionen im folgenden Block. API-Funktionen oder Variablen könnten ebenfalls nach benutzerdefinierten Typen verlangen, die dann bereits zuvor im Modul definiert sein müssen. Für Typdefinitionen gibt es noch eine weitere Besonderheit, wie folgendes Beispiel zeigt:

```
Private Type XYZ
    X As Long
    Y As Long
    N As ABC
End Type

Private Type ABC
    A As Integer
    B As Integer
    C As Long
End Type
```

Das wird VBA nicht zulassen, weil der Typ **XYZ** im Element **N** den Typ **ABC** benötigt, jener aber erst später folgt. Die umgekehrte Reihenfolge ist nötig!

An den Block der Typdefinitionen schließen Sie die API-Deklarationen an. Zum Schluss folgen die Variablendeklarationen, denn diese stehen den Prozeduren quasi am nächsten.

Da alle Deklarationen sowohl private, wie auch öffentliche Gültigkeit aufweisen können, ist zudem ein Aufteilen in diese Gültigkeitsbereiche sinnvoll. Stellen Sie etwa erst alle öffentlichen Elemente zusammen, dann die privaten, so, wie auch im Listing geschehen.

Prozedur-Layout

Hier geht es um die optische Gestaltung des Code. Führen Sie sich dazu das Negativbeispiel in Listing 2 zu Gemüte.

Hier gibt es keine Strukturierung durch Leerzeilen oder Einrückungen. Alles ist seriell linksbündig heruntergeschrieben. Das tut dem Auge weh und macht bei der Analyse der Funktion reichlich Mühe. Besser kommt man wohl mit einem Layout zurecht, wie in Listing 3, welches weitgehend die gleiche Prozedur enthält. Was ist hier anders?

Zunächst ist der gesamte Code um vier Leerzeichen eingerückt. Damit setzt er sich klar von der Prozedurdeklarationszeile ab. Innerhalb des Moduls werden so die Prozeduren insgesamt deutlicher. Keiner Tipp am Rande: Die Einrückung erfolgt über die **Tab**-Taste zu Beginn einer Zeile. Wie viele Leerzeichen dies hervorruft, können Sie in den VBA-Optionen festlegen **(Extras|Optionen|Editor|Tab-Schrittweite)**. Der Standard ist 4. Übrigens rückt **die** Tab-Taste auch gleich mehrere Zeilen ein, wenn Sie diese zuvor markiert haben.

Weiter ist allen Variablendeklarationen oben eine eigene Zeile spendiert worden, statt sie kommagetrennt aneinanderzufügen. Das macht es dem Auge leichter, den Datentyp jeder Variablen zu erkennen, auch wenn das auf Kosten der vertikalen Ausdehnung geht.

Auf die Variablendeklarationen folgt eine Leerzeile, um den Beginn des

```
Public Function fuCodeLayout(Optional ByVal x As Long, Optional _
ByVal y As Long) As Boolean
Dim i As Long, j As Long, dbs As DAO.Database, rs As DAO.Recordset
On Error GoTo Fehler
Set dbs = CurrentDb
Set rs = dbs.OpenRecordset("SELECT [Name],[Id],DateCreate,DateUpdate " & _
"FROM MSysObjects WHERE [Type]=1 AND Flags>0", dbOpenDynaset)
With rs
Do While Not .EOF
For j = 0 To 3
Debug.Print .Fields(j).Value,
Next j
i = i + 1
.MoveNext
Loop
.Close
End With
fuCodeLayout = True
Debug.Print i & " Datensätze"
Ende:
On Error Resume Next
rs.Close
Set dbs = Nothing
Exit Function
Fehler:
MsgBox "Fehler in Zeile " & Erl & vbCrLf & Err.description, vbCritical
Resume Ende
End Function
```

Listing 2: Eine Code-Wüste ohne jegliche optische Strukturierung

```
Public Function fuCodeLayoutBad(x As Long, y As Long) As Boolean
    Dim i As Long
    Dim dbs As DAO.Database
    Dim sText As String
    Dim j As Long
    Dim lRet As Long
    Dim sTmp As String
    Dim rs As DAO.Recordset

    Set dbs = CurrentDb
    Set rs = dbs.OpenRecordset("SELECT * FROM MSysObjects WHERE [Type]=1 AND Flags>0")

    Do While Not rs.EOF
        Debug.Print rs.Fields("Name"), rs.Fields("Id"), _
             rs.Fields("DateCreate"), rs.Fields("DateUpdate")
        i = i + 1
        rs.MoveNext
    Loop
    rs.Close

    fuCodeLayoutBad = True

    Debug.Print i & " Datensätze"
End Function
```

Listing 3: Code-Strukturierung durch Leerzeilen und Einrückungen

ADODB als Alternative zu DAO

Der Umgang mit Daten unter VBA findet in der Regel über die Bibliothek DAO statt, welche Access über den gleichnamigen Verweis beim Erzeugen einer neuen Datenbank automatisch in das VBA-Projekt einbindet. Microsoft hat sie damit zum Standard erkoren. Tatsächlich gibt es aber noch ein anderes Schwergewicht für den Datenzugriff, das unter dem Kürzel ADODB daherkommt. Führen wir uns diese Bibliothek einmal als Alternative zu DAO zu Gemüte.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1508_ADO.zip**.

Datenzugriffsbibliotheken

Die Zeiten ändern sich. War **DAO** (**Data Access Objects**) als Schnittstelle zu den Datenobjekten unter VBA mit dem Erscheinen von Access noch die einzige Bibliothek, so führte Microsoft mit **Office 2000** eine neue ein, die die alte zunehmend ersetzen und ihr angeblich in vielerlei Hinsicht überlegen sein sollte. Die Bezeichnung lautete **ADODB** als Abkürzung für das etwas schräge **Microsoft ActiveX Data Objects Data Base**.

Ein Hintergrund dafür war die Einführung der **Access-Projekte**, die den unmittelbaren Zugriff auf Objekte eines verbundenen SQL-Servers erlaubten und sogar deren Verwaltung ermöglichen. Derlei war mit **DAO** unmöglich, und so nahm man sich der bereits für Visual Basic vorliegenden Implementation von **ADO** an und pfropfte sie Access auf.

Statt **DAO** wurde beim Neuanlegen einer Datenbank automatisch der Verweis auf die neue Bibliothek gesetzt. Das änderte indessen nichts daran, dass die Access-Objekte, wie Formulare und Berichte, unter der Haube weiterhin mit **DAO** arbeiteten.

Diese Inkonsistenz ist neben der Tatsache, dass die Programmierung über **ADO** deutlich aufwändiger und komplizierter ist, mit ein Grund dafür, dass diese Technik von Entwicklern nicht wirklich angenommen wurde.

So elegant sie bei der Einbindung von Daten auf SQL-Servern ist, so überflüssig beim Zugriff auf die Tabellen der lokalen Access-Datenbank oder eines verknüpften Backends.

Microsoft wurde sich dessen offenbar bewusst, weshalb ab **Access 2007** die Umkehr stattfand. Nun sollte **DAO** nicht nur für **JET-Datenbanken**, sondern sogar für die Ansprache von SQL-Servern über **ODBC** eingesetzt werden. Dafür dürfte allerdings nicht nur Einsicht Vater des Gedankens gewesen sein, sondern auch der Fakt, dass die Technik der **Access-Projekte** nicht mehr weiterentwickelt wurde und nach und nach dem Rotstift anheimfiel.

Auch, wenn **ADO** mittlerweile ein Stiefkind zu sein scheint, kann es nicht schaden, sich die Datenzugriffsbibliothek einmal genauer anzusehen. Denn sie verfügt über einige interessante Features, die **DAO** fehlen, weshalb ihr Einsatz in manchen Fällen doch noch sinnvoll sein kann.

ADODB im VBA-Projekt

Den Verweis auf die ADO-Bibliothek legt Access, wie erwähnt, nicht mehr selbst an, wenn Sie eine neue Datenbank erzeugen. Sie müssen Sie selbst einbinden. Dazu öffnen Sie im VBA-Editor über **Extras|Verweise...** den **Verweise-Dialog** und klappern dessen Einträge nach der **Microsoft ActiveX Data Objects 2.x Library** ab. Sie werden gleich mehrere davon finden, die die Versionsnummern **2.1** bis **2.8** tragen. Tatsächlich handelt es sich dabei aber lediglich um verschiedene Typbibliotheken für den gleichen Satz an DLLs, also das gleiche System, die aus Kompatibilitätsgründen angelegt sind. Aktivieren Sie einfach die höchste Version **2.8**. Nach dem Schließen des **Verweise-Dialogs** finden Sie die Bibliothek im VBA-Objektkatalog, wenn Sie im Kombinationsfeld links oben den Eintrag **ADODB** auswählen.

Die Struktur von ADO

Schon am Umfang der Klassen der Bibliothek ist ersichtlich, dass deren Programmierung etwas komplexer ist, als unter DAO. Um die Bedeutung der einzelnen Objekte verständlich zu machen, ist es nützlich, sie in einen Vergleich zu jenen der DAO-Bibliothek zu setzen. In Bild 1 sind die wichtigsten Objekte gegenübergestellt.

Während Sie unter DAO auf die aktuelle Datenbank (**Database-Objekt**) unmittelbaren Zugriff haben, indem Sie die Funktion **CurrentDb** des **Access-Application**-Objekts aufrufen, oder die Methode **OpenDatabase** des ebenfalls bereits vorliegenden **DbEngine-Objekts**, muss in ADO erst eine Verbindung zur Datenbank aufgebaut werden.

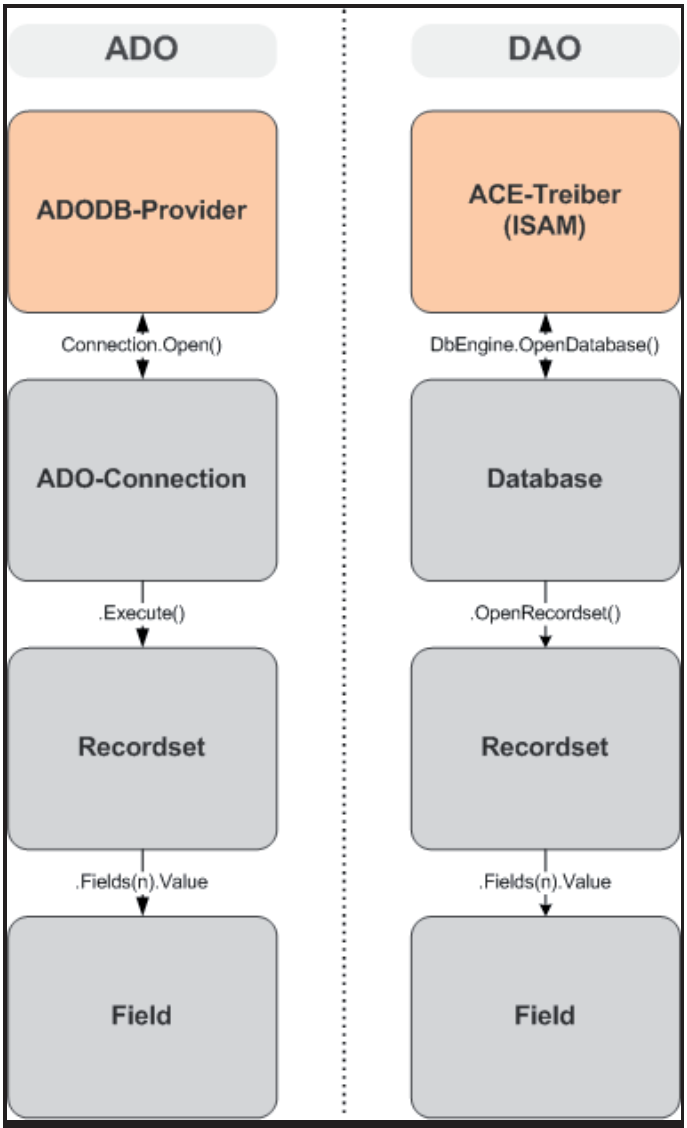


Bild 1: Die Datenzugriffsbibliotheken ADO und DAO in der logischen Gegenüberstellung

Das ist auch schon der Hauptunterschied: Access ist beim Öffnen einer Datenbank ganz unbemerkt bereits mit der **JET-Engine** verbandelt. Die darüber liegende Schicht, die Treiber der **Access Database Engine** (**ACE**) samt ihrer dateispezifischen **ISAM-Treiber**, bleibt Ihnen verborgen. Verwendet werden dabei die Treiber für **MDB**- und **ACCDB**-Dateien. Neben diesen gibt es ja etwa auch noch jene zum Öffnen von **Excel**-, **Text**- oder **DBase**-Dateien. Um diese Schicht brauchen Sie sich nicht zu kümmern – die Integration vollzieht Access selbst. Anders unter **ADO**: Hier muss zunächst dezidiert der sogenannte **Provider** angegeben werden.

Dabei handelt es sich Imgrunde um einen im System registrierten Treiber, der zum Öffnen einer Datei herangezogen werden soll. Diese **Provider** werden von **Office** installiert, teilweise aber auch von **Windows** über **ODBC** ins Spiel, wenn der **MSDASQL.1**-Provider verwendet wird. Wie die jeweiligen Provider lauten, lässt sich aus Access heraus nicht ermitteln. Greifen Sie im Zweifelsfall zu einer Dokumentation im Internet, wie etwa der Domain **connectionsstrings.com**. Dort klicken Sie auf das gewünschte das Datenbanksystem, um die Verbindungszeichenfolgen zu erfahren. Die Seite unterscheidet dabei zwischen Zeichenfolgen für **ODBC**, **OLEDB** und **.NET**. Uns interessieren an dieser Stelle nur die **OLEDB-Angaben**, ein Synonym für **ADODB**.

Neben dem Namen des Providers finden Sie dort auch noch weitere Angaben, die Sie zum **Provider** machen können, nämlich die Optionen-Strings, welche die Treiber steuern. Doch dazu später mehr.

Haben Sie die Angaben zum **Provider** beisammen, so können Sie sich an das Anlegen eines **Connection-Objekts** machen. **ADO** kennt kein **Database-Objekt**, sondern nur das Verbindungsobjekt **Connection**, welches diesem aber in der Funktionalität ähnelt. Die Verbindung wird über die Methode **Open** hergestellt. Steht diese, so kann analog zur **Database** ein **Recordset** auf die Datenbankobjekte geöffnet werden. Nur lautet die hier verwendete Methode nicht **OpenRecordset**,

oder anderen Anwendungen. Alle **Provider** sind namentlich gekennzeichnet. So lautet der Name des **Providers** zum Öffnen einer Access-Datenbank aktuell etwa **Microsoft.ACE.OLEDB.12.0**. Früher nahm man zum Ansprechen einer **MDB-Datei** den **Jet-Provider Microsoft.Jet.OLEDB.4.0**. Soll ein **MS-SQL-Server** verbunden werden, dann heißt der Provider **SQLOLEDB.1**. Und schließlich kommen weitere **DBMS-Server**

sondern **Execute**. In beiden Fällen wird ein **Recordset** zurückgegeben.

Allerdings ist ein **ADO-Recordset** nicht dasselbe, wie ein **DAO-Recordset**! Zwar gleichen sich dessen Methoden weitgehend, doch es handelt sich dennoch um zwei unterschiedliche Klassen. Wenn Sie Objektvariablen zu diesen Recordsets deklarieren, so geben Sie deshalb ausdrücklich die Bibliothek an, aus dem das **Recordset** stammen soll:

```
Dim rs As DAO.Recordset
Dim rs As ADODB.Recordset
```

Das beugt Verwirrung vor, denn im VBA-Projekt können Sie auf beide Bibliotheken zugleich verweisen!

Die Unterschiede zwischen dem einen und dem anderen **Recordset** sind marginal. Beide weisen die üblichen Methoden zur Navigation (**MoveNext**, **MoveLast**, etc.) auf und haben jeweils eine **Fields-Auflistung**, über die auf die einzelnen Felder des Ergebnisses zugegriffen werden kann.

Connection anlegen

Gehen wir nun Schritt für Schritt vor, um an die Daten einer Datenbank über **ADO** zu gelangen.

Dass Access kein ADO-Pendant zu **CurrentDb** enthält, stimmt nicht ganz. Seit Einführung von **ADODB** gibt es das **CurrentProject-Objekt**. Dessen Eigenschaft **Connection** stellt die **ADO-Connection** zur aktuellen Datenbank dar. Sie können es direkt verwenden, um ein **ADO-Recordset** zu öffnen:

```
Dim con As ADODB.Connection
Dim rs As ADODB.Recordset
Set con = _
   CurrentProject.Connection
Set rs = _
  con.Execute( _
  "SELECT * FROM tblKunden")
Debug.Print rs(3).Value
```

Hier wird zunächst mit der **Connection-Variablen con** auf das **Connection-Object** von **CurrentProject** verwiesen. Das **Recordset** wird über die **Execute**-Anweisung der **Connection** gefüllt, welcher der gewünschte SQL-Ausdruck übergeben wird. Im Beispiel ist das der komplette Inhalt der Tabelle **tblKunden**. Schließlich wird das Feld Nummer 3 und dessen Wert abgefragt. (Die 3 in Klammern ist die abgekürzte Syntax für **rs.Fields(3).Value**, da die **Fields**-Auslistung die Standardeigenschaft eines **ADO-Recordsets** ist.)

Zum Vergleich die analoge Routine in der **DAO**-Version:

```
Dim dbs As Database
Dim rs As DAO.Recordset
Set dbs = CurrentDb
Set rs = dbs.OpenRecordset( _
    "SELECT * FROM tblKunden")
Debug.Print rs(3).Value
```

Wenn Sie nicht die aktuelle Datenbank ansprechen möchten, sondern eine andere Access-Datei, so muss ein Connection-Objekt von Grund auf neu erzeugt werden:

```
Dim con As ADODB.Connection
Set con = New ADODB.Connection
con.Provider = _
    "Microsoft.ACE.OLEDB.12.0"
```

Über die **New**-Anweisung wird ein neues **Connection-Objekt** angelegt. Damit es weiß, welche Treiber zur Anwendung kommen sollen, wird der Provider für **ACE-Datenbanken** angegeben. Damit ist die oberste Schicht klargestellt.

Doch woher weiß die Connection, welches das Objekt der Begierde, die Datenbankdatei, ist? Die Angabe des Pfads muss in der Eigenschaft ConnectionString untergebracht werden:

```
con.ConnectionString = _
"Data Source=c:\xyz\test.accdb;"
```

Der **ConnectionString** ist der Ort, wo alle Optionen zur Verbindung eingestellt werden. Im Falle einer **ACE-Datenbank**, also einer **MDB** oder **ACCDB**, lautet dabei die Option für den Zugriffspfad **Data Source**. Bei anderen Providern könnte dieser Ausdruck auch einen anderen Namen haben.

Zusätzlich lassen sich weitere Optionen durch Semikola getrennt im String einbauen:

```
"Data Source=c:\datenbanken\test.
accdb;Mode=Share Deny None;"
```

Mit **Mode** wird bestimmt, dass der Datenzugriff ohne Mehrbenutzersperren erfolgen soll. Und selbst der **Provider** kann hier untergebracht werden, was das Setzen in der Eigenschaft **.Provider** überflüssig macht:

```
"Provider=Microsoft.ACE.
OLEDB.12.0;Data Source=c:\datenbanken\
test.accdb;Mode=Share Deny None;"
```

Um eine Vorstellung zu bekommen, welche weiteren Optionen gesetzt sein können, setzen Sie diesen Ausdruck im VBA-Direktfenster ab:

```
? CurrentProject.Connection. _
    ConnectionString
```

Auf eine Abbildung des Ergebnisses wird hier verzichtet – Access setzt Unmengen ACE-spezifischer Verbindungsoptionen.

Sind alle nötigen Angaben im **ConnectionString** gemacht, so kann es an das physische Herstellen der Verbindung gehen, welches die **Open-Methode** bewirkt:

```
con.Open
```

Tritt kein Fehler bei dieser Anweisung auf, so ist die Verbindung geöffnet und bereit zum Abfragen oder Schreiben von Daten über die **Execute-Methode**, wie im Recordset-Beispiel oben.

ADO-Routinen

Da der Schreibaufwand bei ADO-Programmierung etwas höher ist, als bei DAO, empfiehlt es sich, für grundlegende Aufgaben ein separates VBA-Modul anzulegen. In der Beispieldatenbank nennt dieses sich **mdlADODB**. Im Folgenden sind einige Abschnitte des Moduls dargestellt.

MyConnectString (Listing 1) ist eine Hilfsfunktion, die eine gültige Verbindungszeichenfolge zurückgibt. Sie beschränkt sich auf das Zusammensetzen des **ConnectionStrings** über String-Verkettung. Neben dem

```
Function MyConnectString() As String
    MyConnectString = "Provider=Microsoft.ACE.OLEDB.12.0;" & _
        "Data Source=" & CurrentProject.Path & "\1508_Backend.accdb;" & _
        "Mode=Share Deny None;"
End Function
```

Listing 1: Zusammensetzen eines ConnectionStrings für ADO

```
Function DBConnection(Optional sConnect As String) As ADODB.Connection
    If Len(sConnect) = 0 Then
        Set DBConnection = CurrentProject.Connection
    Else
        Set DBConnection = New ADODB.Connection
        DBConnection.ConnectionString = sConnect
        DBConnection.CursorLocation = adUseClient
        DBConnection.Open
    End If
End Function
```

Listing 2: Funktion zum Zurückgeben eines ADO-Connection-Objekts

Provider wird der Pfad zur anzusprechenden Access-Datenbank (**1508_Backend.accdb**) im Verzeichnis der aktuellen Datenbank (**CurrentProject.Path**) hier hartkodiert eingebaut. So eine Hilfsfunktion ist unter Umständen sinnvoll, damit der String nicht an mehreren Stellen im Code neu aufgebaut werden muss.

Auch das **Connection**-Objekt wird in einer eigenen Prozedur angelegt, die den Namen **DBConnection** trägt. In Listing 2 wird dabei zunächst ermittelt, ob der Parameter **sConnect** leer ist. In diesem Fall wird das **Connection-Objekt** der aktuellen Datenbank zurückgegeben. Andernfalls wird eine neue **Connection** angelegt, deren **ConnectionString** gesetzt und schließlich die Verbindung über **Open** geöffnet.

Wichtig ist noch die Zeile davor, wo der **CursorLocation** der Wert **adUseClient** zugewiesen wird. Mit dieser Eigenschaft wird bestimmt, wie die Kommunikation mit dem **OLEDB-Provider** vonstattengehen soll.

asUseClient sagt ihm dabei, dass die Client-Anwendung (=**Access**) die Steuerung übernimmt, während **adUseServer** hieße, dass diverse Zugriffsmethoden vom Server übernommen würden. Da eine Access-Datenbankdatei kein SQL-Server ist, macht diese Einstellung keinen Sinn. Leider ist sie aber für ein neues **Connection**-Objekt voreingestellt, so dass die Angabe der **CursorLocation** und **adUseClient** immer obligatorisch ist. Ohne sie treten beim anschließenden Ansprechen der Daten Fehlermeldungen auf.

Fehlt noch eine Funktion, die auf allgemeine Weise ein ADO-Recordset zur Verfügung stellt. ADORecordset in Listing 3 bewerkstelligt dies.

Als Parameter wird ihr in **sConnect** der **ConnectionString** übergeben und mit **bReadOnly** optional geregelt, ob das resultierende **Recordset** schreibgeschützt sein soll (**bReadOnly = True**). Die abzufragenden Daten sollen im Parameter sSQL stehen. Die Routine legt zuerst ein

```
Function ADORecordset(sConnect As String, sSQL As String, _
        Optional bReadOnly As Boolean) As ADODB.Recordset
    Set ADORecordset = New ADODB.Recordset
    With ADORecordset
        Set .ActiveConnection = DBConnection(sConnect)
        .CursorLocation = adUseClient
        .CursorType = adOpenDynamic '= adOpenKeyset
        If Not bReadOnly Then .LockType = adLockOptimistic
        .Open sSQL
    End With
End Function
```

Listing 3: Zurückgeben eines ADO-Recordset-Objekts über steuernde Parameter

leeres Recordset-Objekt an. Unter **ADODB** ist das möglich, unter **DAO** nicht. Dort kann ein Recordset-Objekt immer nur über die **OpenRecordset**-Methoden von **Database**, **TableDef** oder **QueryDef** erhalten werden. Zu welcher Datenquelle eine Verbindung aufgebaut werden soll, wird über die Eigenschaft **ActiveConnection** angegeben

Hier muss ein zuvor bereits angelegtes **Connection**-Objekt vorliegen. In unserem Fall kommt dieses aus der besprochenen Funktion **DBConnection**. Auch das **ADO-Recordset** kennt die Eigenschaft **CursorLocation**, die auf gleiche Weise gesetzt werden kann, wie beim **Connection-Objekt**. Allerdings ist das nicht zwingend erforderlich, weil eine auf **adUseClient** gesetzte Verbindung das **Recordset** implizit auf diesen Modus einstellt.

Mit **CursorType** kann gesteuert werden, wie sich die Datenabfrage verhalten soll. Sie kennen das schon von **DAO**: über den optionalen Parameter von **OpenRecordset** können Sie etwa den Wert **dbOpenDynaset** setzen, der bestimmt, dass das Recordset erst beim Bewegen durch die Datensätze auf die Quelle zugreift. **dbOpenStatic** wäre die andere Möglichkeit, die zur sofortigen Abfrage sämtlicher Datensätze führte. Beim **ADO-Recordset** entspricht **adOpenDynamic** dem **dbOpenDynaset** von **DAO**. Beim Zuweisen dieses Werts tritt kein Fehler auf, tatsächlich aber verwendet die **ACE-Provider-Engine** intern den Wert **adOpenKeyset**. Sie kann nur diesen dynamischen Typ. Es ist also egal, welche der beiden Angaben Sie machen. Sie unterscheiden das Verhalten des Recordsets in der Folge auch nicht nennenswert.

Nun folgt das Bestücken des **LockType** des Recordsets mit dem Wert **adLockOptimistic**. Dabei handelt es sich um den Sperrmodus der Datensätze. Ohne diese Zuweisung befänden sie sich nämlich automatisch im Modus schreibgeschützt (**adLockReadOnly**). Daten könnten zwar gelesen, jedoch nicht bearbeitet werden. Das Ganze geschieht in Abhängigkeit vom Parameter **bReadOnly**. Ist ihm False übergeben, so trifft die **If-Then-Bedingung** zu und **LockType** wird entsprechend gesetzt.

Zu guter Letzt öffnet die **Open**-Anweisung das **Recordset** ähnlich, wie dies beim **Connection**-Objekt erfolgte. Nur ist hier zusätzlich ein SQL-String anzugeben, der die abzufragenden Daten bestimmt.

Schauen wir uns an, wie sich dieses Routinen-Set praktisch einsetzen lässt. Listing 4 zeigt ein Beispiel. Erst wird die **Recordset-Variable rs** über die Funktion **ADORecordset** gefüllt, der als Parameter der **ConnectionString** aus der Hilfsfunktion **MyConnectString** und die gewünschten Daten – hier alle Datensätze der Tabelle **tblKunden** – als SQL-Ausdruck übergeben wird.

Es reicht hier also bereits eine Code-Zeile aus, um ein geöffnetes Recordset mit den gefragten Datensätzen zu erhalten. Der Rest der Routine wertet dann nur noch die Daten aus. **Source** ist eine Eigenschaft des Recordsets, die die gesetzte Datenabfrage wie-

```
Sub TestADORecordset()
    Dim rs As ADODB.Recordset
    Set rs = ADORecordset(MyConnectString, _
        "SELECT * FROM tblKunden")
    Debug.Print rs.Source
    Debug.Print rs.Fields.Count & " Felder"
    Debug.Print rs.Fields(1).Name & ": " & rs.Fields(1).Value
    rs.MoveNext
    Debug.Print rs.Fields("Kundencode").Name & ": " & rs.Fields(1).Value
    rs.Fields(1).Value = 1579518
    rs.Update
End Sub
```

Listing 4: Testprozedur für die ADO-Routinen im Modul mdlADODB

Frontends aktualisieren

Die Nutzung von Access-Datenbanken in einer Mehrbenutzerumgebung erfordert etwas mehr Aufmerksamkeit von Ihnen als Entwickler, als wenn nur Sie selbst mit einer Datenbank umgehen. Was Sie können, ist den Anwendern wahrscheinlich nicht so geläufig. Fassen wir in diesem Beitrag zusammen, was da auf Sie zukommen kann.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1508_FrontBackend.zip**.

Mehrbenutzerumgebung

Wenn eine Access-Datenbank von mehreren Nutzern gleichzeitig in Betrieb ist, so sieht das übliche System folgendermaßen aus: Auf den Arbeitsstationen liegen die **Frontend-Datenbanken**, die für die Anwenderoberfläche verantwortlich sind. Sie werden durch Doppelklick auf die **accdb-Datei** oder über eine Desktop-Verknüpfung auf sie in Access gestartet.

In diesem **Frontend** befinden sich jedoch nicht die gemeinsam zu bearbeitenden Daten, denn dafür ist genau eine **Backend-Datei** zuständig, die die entsprechenden Tabellen enthält. Sie liegt in der Regel auf einem **File-Server** in einem Verzeichnis, auf das alle Benutzer Lese- und Schreibrechte haben. Allerdings reicht auch das meist noch nicht aus, denn beim Komprimieren der Backenddatei wird eine temporäre Datei im Verzeichnis angelegt, die alte gelöscht und sie anschließend umbenannt. Damit sind Verwaltungsrechte auf das Verzeichnis nötig, also die Erlaubnis für das Löschen und Erzeugen von Dateien.

Die Tabellen dieses **Backends** werden nun in die **Frontends** verknüpft. Auf diese verknüpften Tabellen bauen die Formulare und Berichte der Anwendung auf.

Einmal eingerichtet könnte mit diesem System in aller Ruhe gearbeitet werden, wären da nicht ein paar Fallstricke. Sowohl **Backend**, wie **Frontend** blähen sich im Betrieb mit der Zeit auf. Deshalb ist das regelmäßige Komprimieren beider zu empfehlen, wenn nicht die Performance irgendwann einbrechen und die Anwender über Kaffeepausen beim Bearbeiten ihrer Daten klagen sollen. Auch die Gefahr von Datenkorruption nimmt zu, wenn das Komprimieren nie geschieht.

Und dann bleibt es selten bei der Urfassung einer Datenbank. Weiterentwicklung ist an der Tagesordnung. Mit jeder neuen Version stehen Sie vor der Aufgabe, das neue **Frontend** auf die Arbeitsstationen zu verteilen.

Das kann man manuell erledigen, falls man Vorort weilt, aber komfortabler wäre doch eine automatisierte Lösung. Dann können Sie das neue Frontend einfach auf einen **File-Server** legen, auf den Sie unter Umständen sogar per **VPN** Fernzugriff haben.

Blöd nur, dass dann die Pfade der verknüpften Tabellen im **Frontend** nun nicht mehr stimmen. Auf Ihrem Entwicklerrechner haben Sie wahrscheinlich ganz andere Verzeichnisstrukturen, als die Anwender in der Firma. Also müssen in den **Frontends** die Tabellen neu mit dem **Backend** verknüpft werden, was wiederum eine Aufgabe ist, die Sie besser nicht dem Anwender überlassen. Auch das sollte automatisch im Hintergrund vor sich gehen.

Damit stellen sich diese vier Grundanforderung:

- Das **Frontend** soll automatisch auf die Arbeitsrechner verteilt werden
- Das **Backend** soll automatisch regelmäßig komprimiert werden
- Das **Frontend** ebenfalls
- Die **Backend-Tabellen** sollen automatisch im **Frontend** verknüpft und aktualisiert werden

Schauen wir uns dazu einige einfach gehaltene Lösung an.

Frontend aktualisieren

Im Prinzip geht es dabei um nichts anderes, als Ihr **Entwickler-Frontend**

als Datei auf die einzelnen Rechner zu kopieren. Mancher sieht hier eine Versionierung vor. Das Kopieren geschähe demnach nur, wenn eine neuere Version des **Frontends** vorläge. Dafür gibt es aber wenig plausible Gründe. Am Einfachsten ist es, das Kopieren schlicht von einem Skript durchführen zu lassen, welches entweder beim Starten der Anwendung angestoßen wird, oder im **Autostart-Ordner** der Rechner liegt, so dass es jeweils beim Anmelden am Rechner ausgeführt wird.

Wir entscheiden uns hier für die erste Lösung, weil das **Frontend** so auf Zuruf etwa auch mittags aktualisiert werden kann. Eigentlich reicht für diesen Zweck eine kurze **Windows-Batch-Datei** aus. Da wir uns jedoch in der Ausgabe 04/2015 von **Access-Basics** bereits mit **VB-Script** auseinandersetzten, verwenden wir diese Sprache auch an dieser Stelle. Mit **VB-Script** hat man etwa mehr Möglichkeiten zur Kontrolle der Vorgänge und kann eine Fehlerbehandlung vorsehen.

Das Kopier-Skript finden Sie in Listing 1. Speichern Sie den Text als Datei unter dem Namen **copy_frontend.vbs** im Verzeichnis der Datenbank. Leider gibt es unter **VB-Script** keine direkte Funktion zum Ermitteln des aktuellen Verzeichnisses. Mit einem Trick gelingt jedoch auch dies. **ScriptFullName** ist eine Eigenschaft, die den Dateinamen des aktuell ausgeführten Skripts selbst wiederspiegelt. Damit ist der Pfad ebenfalls enthalten. Über die **InstrRev**-Funktion und **Left** wird das Verzeichnis aus dem Dateipfad herausgeschnitten.

```
sScriptFile = WScript.ScriptFullName
n = InstrRev(sScriptFile,"\")
sPathTarget = Left(sScriptFile,n)
sPathSource = sPathTarget & "FileShare"
sFileSource = sPathSource & "\1508_Frontend.accdb"

Set fso = CreateObject("Scripting.FileSystemObject")

If fso.FileExists(sFileSource) Then
        fso.CopyFile sFileSource, sPathTarget, True
        Msgbox "Info: Datenbank-Frontend aktualisiert."
End if
```

Listing 1: VBS-Script zum Aktualisieren des Frontends aus einem File Share

Damit steht das Zielverzeichnis für den Kopiervorgang in der Variablen **sPathTarget** bereit. In der Demo zum Beitrag ist das Verzeichnis, in dem sich das Entwickler-Frontend befindet, durch den Unterordner **FileShare** simuliert. **sPathSource** wird entsprechend gebildet. In natura setzten Sie hier eher hartkodiert den Pfad zum Server-Verzeichnis ein.

Zum Kopieren von Dateien müssen Sie unter **VB-Script** ein **Shell-Objekt** bemühen, wie es etwa im **FileSystemObject** vorliegt. Die Objektvariable **fso** nimmt es auf. Mit **FileExists** wird zunächst überprüft, ab das Entwickler-Frontend überhaupt vorhanden ist. Im Erfolgsfall übernimmt **CopyFile** dann letztendlich das Kopieren. Der Parameter **Overwrite** steht auf **True**, was bedeutet, dass eine eventuell vorhandene Datei gleichen Namens sang- und klanglos überschrieben wird. Der Erfolg der Aktion wird am Schluss über eine **Messagebox** quittiert. Diese Zeile können Sie natürlich löschen.

Frontend komprimieren und starten

Auch für diese Aufgabe steht ein **VB-Script** bereit. Das Problem ist ja, dass eine Datenbank per VBA nicht komprimiert werden kann. Das Komprimieren schließt zunächst die Datenbank, wodurch eine laufende Prozedur sofort abgebrochen würde. Aus diesem Grund hat Microsoft allen Methoden zum Komprimieren der aktuellen Datenbank einen Riegel vorgeschoben. Ergo muss das **Frontend e**xtern komprimiert werden.

Das Skript in Listing 2 speichern Sie unter dem Namen **startdb_compressed.vbs** ebenfalls im Datenbankordner. Ähnlich, wie im vorigen Skript, ermittelt die Routine zunächst den Pfad des Datenbankordners über **ScriptFullName**, um darüber den Namen der **Frontend-Datei** in **sFile** zu generieren.

Für das Komprimieren benötigen Sie nicht zwingend Access als Anwendung. Es reicht dafür die **ACE-Engine** aus. Sie können eine Instanz dieser über **CreateObject** und die Klassenbezeichnung **DAO.DbEngine.120** produzieren. Die Methode **CompactDatabase** des **Engine-Objekts** komprimiert und repariert eine Access-Datenbankdatei. Als Parameter erwartet sie dabei den Pfadnamen der **ACCDB-Datei** und die Angabe einer Zieldatei, die das komprimierte Resultat aufnimmt.

Arrays als Funktionsergebnis

Eine Sub-Prozedur führt unter VBA eine Aktion durch, eine Funktion möglicherweise auch, doch wichtiger ist in der Regel das berechnete Ergebnis, welches sie dem Aufruf zurückgibt. Fast immer handelt es sich dabei um einen bestimmten gesuchten Wert. Gelegentlich erwartet die aufrufende Prozedur aber auch eine Menge von Rückgabedaten. Hier kommen Arrays ins Spiel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1508_Array.accdb**.

Multiple Ergebnisse

Gerade in der Datenbankprogrammierung ist die Rückgabe von Ergebnissen, die mehr, als einen Wert enthalten, trivial. Nur kommen hier zur Auswertung gewöhnlich Datenbankobjekte zum Einsatz, deren Hauptprotagonist das **Recordset** ist.

So gibt die Methode **OpenRecordset** von **DAO** nicht nur eine Serie von Daten zurück, sondern sogar eine zweidimensionale: in der einen Dimension die Spalten, also Felder, eines Datensatzes, in der anderen die Datensätze selbst.

An die Zellen dieser zweidimensionalen Wertemenge gelangt man dann über spezielle Methoden des Objekts, statt über Indizes, wie das bei Arrays der Fall ist. Auf der einen Seite ist dies praktisch, auf der anderen Seite verlangt es mehr Programmierung.

Hierzu ein Beispiel. Sie möchten den Wert der Zelle einer Adressentabelle ermitteln, sei es etwa die dritte Spalte des vierzehnten Datensatzes. Unter **Excel** könnte das Arbeitsblatt nun tatsächlich über Koordinaten abgefragt werden:

```
Debug.Print WorkSheet.Cells(13,2)
```

Bei einem Recordset ist das nicht direkt möglich. Hier sähe die Routine so aus:

```
Set rs = CurrentDb. _
    OpenRecordset("tblAdressen")
rs.Move 13
Debug.Print rs(2).Value
```

Hier haben wir eine Zeile mehr Code und die Übersicht ist geringer, weil die Koordinaten auf unterschiedliche Zeilen aufgeteilt sind.

Befänden sich alle Adressdaten in einem Array, so könnte über zwei Indizes auf die Zelle zugegriffen werden. Immerhin hält **DAO** dafür eine eigene Methode bereit:

```
Set rs = CurrentDb. _
    OpenRecordset("tblAdressen")
V = rs.GetRows
Debug.Print V(2,13)
```

GetRows verwandelt nämlich die Datensätze eines Recordsets in ein zweidimensionales Array, wobei dieses als Variant deklariert sein muss und die einzelnen Elemente darin ebenfalls diesen Datentyp aufweisen.

Wollten Sie dieses Array als Ergebnis einer Funktion zurückgeben, dann käme diese Routine infrage:

```
Function TabellenArray() As Variant
    Dim rs As DAO.Recordset
    Set rs = CurrentDb. _
        OpenRecordset("tblAdressen")
    TabellenArray = rs.GetRows
End Function
```

Aber schauen wir uns den Umgang mit Arrays in Funktionen einmal etwas genereller an.

Array mit Elementen füllen

Es muss nicht eine Funktion sein, wenn ein Array in einer Sub-Prozedur mit Werten gefüllt werden soll. Sie können ein Array der Prozedur auch als Argument übergeben. Die folgende Routine vollzieht das:

```
Sub FillArray(cnt As Long, _
              arr As Variant)
    Dim i As Long
    ReDim arr(cnt)
    For i = 0 To cnt
        arr(i) = "Element " & i
    Next i
End Sub
```

Für den Aufruf der Prozedur könnte man dies schreiben:

```
Dim arr As Variant
```

```
FillArray 4, arr
```

Die Prozedur nimmt als Parameter die Zahl an Elementen in der Variablen **cnt** entgegen und dimensioniert das Array **arr** dann entsprechend, um es schließlich in einer Schleife mit Werten zu besetzen. Die Auswertung des so erhaltenen Arrays erfolgte ebenfalls in einer Schleife:

```
For i = 0 To UBound(arr)
    Debug.Print arr(i)
Next i
```

Das erstaunliche an der Sache ist, dass in der aufrufenden Prozedur das Array gar nicht als solches deklariert wurde, sondern als **Variant**. Der Datentyp der Variablen **arr** steht zu diesem Zeitpunkt also noch gar nicht fest.

Dennoch lässt sich die **Redim-Anweisung** auf den Variant anwenden, wodurch VBA ihn automatisch zu einem Array macht, dessen Elemente ebenfalls vom Typ Variant sind. Deshalb kann in der Folge die Funktion **UBound** zum Ermitteln der Grenzen des Arrays angewandt werden.

In dieser Version ist die vorherige Deklaration einer Variablen, die der Prozedur übergeben wird, zwingend. Anders sieht der Fall aus, wenn, wie in Listing 1, eine Funktion **fuArray** zum Einsatz kommt, die als Rückgabe direkt ein Array herausrückt.

Als Parameter wird ihr die gewünschte Zahl von Elementen in **cnt** übergeben. Innerhalb der Funktion wird ein eigenes Array **arr** angelegt, dimensioniert und in der Schleife gefüllt.

```
Sub Test()
    Dim arr As Variant
    arr = fuArray(4)
    For i = 0 To UBound(arr)
        Debug.Print arr(i)
    Next i
End Sub

Function fuArray(cnt As Long) As String()
    Dim i As Long
    Dim arr() As String

    ReDim arr(cnt)
    For i = 0 To cnt
        arr(i) = "Element " & i
    Next i
    fuArray = arr
End Function
```

Listing 1: String-Array als Ergebnis einer Funktion fuArray

```
Sub Test()
    Dim itm As Variant
    For Each itm In fuCollection(4)
        Debug.Print itm
    Next itm
    Debug.Print fuCollection(4)("i2")
End Sub

Function fuCollection(cnt As Long) _
                    As VBA.Collection
    Dim i As Long

    Set fuCollection = New VBA.Collection
    For i = 0 To cnt
        fuCollection.Add "Element " & i, _
                         "i" & i
    Next i

End Function
```

Listing 2: Enumerieren einer Collection-Funktion anstatt eines Arrays

Dieses Array wird am Schluss der Funktion als Ergebnis überreicht. Dabei ist das Resultat von vorn herein nicht als Variant deklariert, sondern als **String-Array,** was durch die Klammern deutlich wird: **String()**.

Der Aufwand ist in der Test-Routine nicht geringer, als bei der besprochenen Sub-Prozedur. Auch hier muss das Ergebnis der Funktion in einer Variablen **arr** zwischengespeichert werden, um es in einer Schleife zu enumerieren.

Aber möchte man nur einen Wert der Ergebnismenge abfragen, so reicht in diesem Fall bereits eine Zeile:

```
Debug.Print fuArray(4)(3)
-> "Element 3"
```

Eine Variable zum Zwischenspeichern entfällt. Daraus leitet sich ab: Die Deklaration einer Funktion führt Imgrunde zur Deklaration einer Variablen selben Namens und desselben Typs.

Der Vollständigkeit halber hier noch eine Routine, die statt des **Arrays** eine **Collection** heranzieht. In Listing 2 wird die Funktion **fuCollection** per **Set-Anweisung** direkt als neue **Collection** deklariert und angelegt. Die Elemente, die sie aufnimmt, werden zusätzlich und optional mit einem **Key** versehen, der sich aus dem Schleifenzähler **i** und dem String **"i"** zusammensetzt.

Das ist leider nötig, weil **Keys** von **Collections** alphanumerischer Natur sein müssen. In der **Test**-Routine kann das Ergebnis dann in einer **For-Each-Schleife** enumeriert werden. Wozu die **Keys** gut sind, zeigt die letzte Zeile, wo auf das Element **"i2"** direkter Bezug genommen wird.

Arrays mit benutzerdefinierten Typen

In den bisherigen Beispielen hatten die Elemente der Arrays einfache Datentypen, wie **String** oder **Variant**. Ein

Rekursive Tabellen

Ihr Kumpel, der Gemeinderat, kommt auf die geniale Idee, für alle Bewohner des Dorfes einen Stammbaum anzulegen, und betreut Sie als Programmierer mit der Aufgabe, diese in einer Datenbank anzulegen, um das anschließend zu Papier bringen zu können. Das kann kein so kompliziertes Unterfangen sein, denken Sie, und machen sich frohgemut an die Arbeit. Das Datenmodell ist der erste Schritt, der zu bewältigen ist...

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1509_RekursiveTabellen.accdb**

Eltern, Kinder, Enkel im Datenmodell

Es scheint gar nicht schwer zu sein. Es muss einfach eine Tabelle für die Eltern her, die einen Primärschlüssel aufweist. Dann eine für die Kinder, in der die Datensätze über einen Fremdschlüssel **ElterID** auf die Datensätze der Eltern verweisen. Und schließlich gibt es noch Enkel, deren Tabelle ähnlich aufgebaut ist, wie die der Kinder. Im Ergebnis erhalten Sie das Beziehungslayout aus Bild 1. Die Tabellen sind über **Referenzielle Integrität** eindeutig miteinander verknüpft, jeweils Indizes auf das Feld **ElterID** gelegt, und die eigentlichen Einträge der Daten bestehen aus dem Feld **Person** für den Namen der Bürgers und dem Feld **mw**, welches sein Geschlecht wiedergeben soll. Flux die Daten aus der Excel-Grundlage in die Tabellen eingepflegt, und schon lässt sich dank der automatischen Unterdatenblätter von Access – ein Feature,

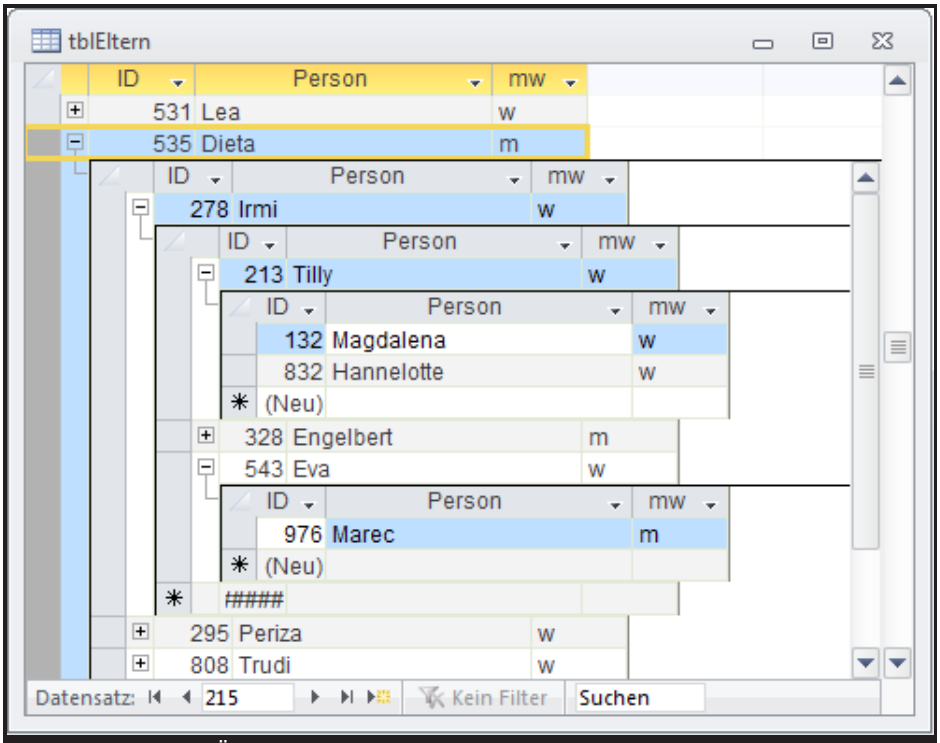


Bild 2: Nach dem Öffnen der Tabelle tblEltern lassen sich die automatisch generierten Unterdatenblätter mit den Kindern und Enkeln ausklappen

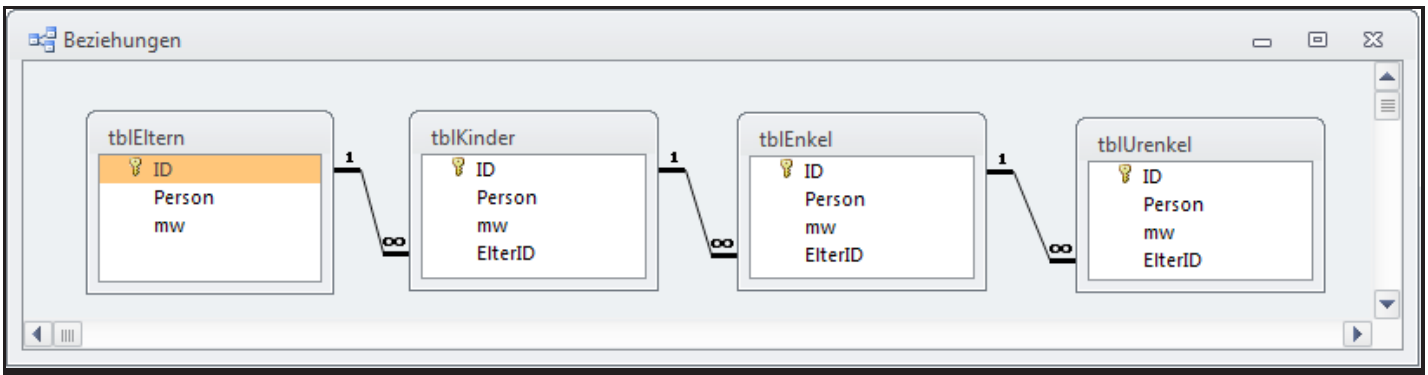
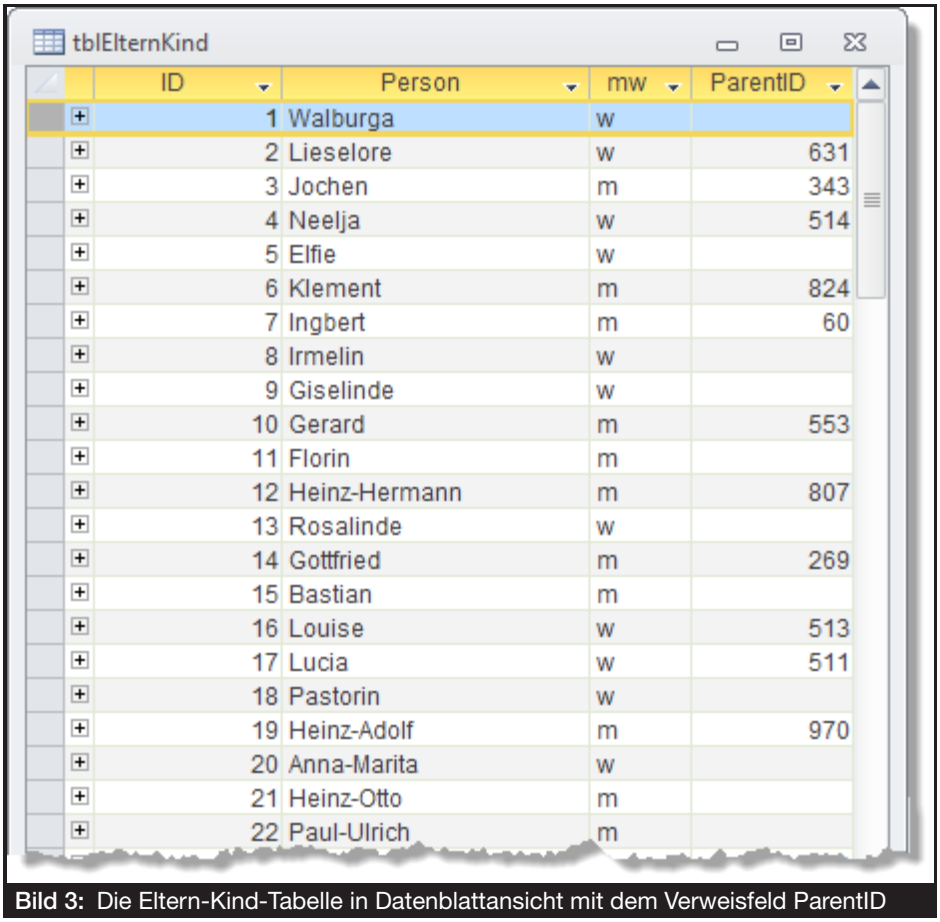


Bild 1: Der erste Entwurf des Datenmodells für die Stammbaumdatenbank mit allen Tabellen und deren Beziehungen

welches hier wirklich einmal sinnvoll zum Einsatz kommt – ein Baum ausklappen, wie in Bild 2. Spätestens hier kommen Ihnen Bedenken wegen der etwaigen Urenkel oder gar noch weiterer Abkömmlinge. Sollten noch weitere Tabellen für diese angelegt werden? Und wie sieht es später mit der Auswertung dieser Struktur aus? Werden die dafür benötigten Abfragen da nicht recht umfangreich und unübersichtlich?

Es geht natürlich auch anders! Sie kommen tatsächlich mit nur einer einzigen Tabelle aus, in der rekursiv eine Beziehung zwischen den Datensätzen für Eltern und Kinder hergestellt wird



tblElternKind

ID	Person	mw	ParentID
1	Walburga	w	
2	Lieselore	w	631
3	Jochen	m	343
4	Neelja	w	514
5	Elfie	w	
6	Klement	m	824
7	Ingbert	m	60
8	Irmelin	w	
9	Giselinde	w	
10	Gerard	m	553
11	Florin	m	
12	Heinz-Hermann	m	807
13	Rosalinde	w	
14	Gottfried	m	269
15	Bastian	m	
16	Louise	w	513
17	Lucia	w	511
18	Pastorin	w	
19	Heinz-Adolf	m	970
20	Anna-Marita	w	
21	Heinz-Otto	m	
22	Paul-Ulrich	m	

Bild 3: Die Eltern-Kind-Tabelle in Datenblattansicht mit dem Verweisfeld ParentID

Rekursive Tabellen

Das Grundprinzip ist ganz einfach. Eine rekursive Tabelle muss einen Primärschlüssel aufweisen, der jeden Datensatz eindeutig identifiziert. Und zusätzlich ein weiteres indiziertes Feld, welches als Fremdschlüsselverweis für den Primärschlüssel dient. Dazu kommen dann die eigentlichen Inhalte der Tabelle. In Bild 3 ist das für die Tabelle **tblElternKind** realisiert. Die **ID** ist als Primärschlüssel vom Typ **Autowert**. **ParentID** vom Typ **Long** enthält dann jeweils einen Wert, der auf einen anderen Datensatz mit dieser **ID** verweist. **Person** und **mw** sind die eigentlichen Inhalte der Tabelle. **Ingbert** in Datensatz 7 hat die **ParentID** 60, was besagt, dass sein Elter weiter unten im Datensatz mit eben dieser **ID** 60 zu finden ist. Das wäre dann, außerhalb des in Bild 3 sichtbaren Bereichs, **Günter**. Scrollen Sie zu **Günter**, so kann es sein, dass dieser ebenfalls eine **ParentID** enthält, mithin also etwa ein Kind von **Klara** wäre, und somit **Ingbert** ein Enkel von **Klara**. Tatsächlich ist aber das Feld **ParentID** von Günter leer, was besagt, dass in der Datenbank kein Elter von ihm verzeichnet ist. **Günter** wäre ein **Root-Datensatz** der rekursiven Tabelle.

Mehr gibt es dazu eigentlich nicht zu sagen. Immer dann, wenn Datensätze hierarchisch voneinander abhängen und die Zahl der Kindelemente, also der Unterebenen, nicht bekannt ist, kommt so eine rekursive Struktur infrage.

Sie finden in jeder Access-Datenbank dafür ein Beispiel. Die Systemtabelle **MSysObjects** enthält rekursive Einträge. Auch sie hat den Primärschlüssel **ID** und das Verweisfeld **ParentID**. Öffnen Sie sie einmal in der Beispieldatenbank, was voraussetzt, dass in den Optionen des Navigationsbereichs der Eintrag **Systemobjekte anzeigen** aktiviert ist. Unter der Name Spalte **Name** suchen Sie die Tabelle **tblElternKind**. Im Datensatz finden Sie unter **ParentID** den Wert **251658241**.

Nun navigieren Sie zum Datensatz mit dieser **ID**. Dort steht unter **Name** **Tables**. Das ist der Container für alle Tabellen der Datenbank. Er selbst hat die **ParentID 251658240**, was allerdings zu keinem gültigen Datensatz in der Tabelle führt...

Ein anderes Beispiel wäre die Systemtabelle **MSysAccessStorage**, welche alle Objekte der Datenbank enthält, außer den Tabellen und Abfragen. Sie ist noch stärker verschachtelt und hat als **Root-Datensatz** die **ID** 1 für den bezeichnenden Eintrag **MSysAccessStorage_ROOT**.

Sie sparen sich mit dieser rekursiven Tabelle also eine Menge Arbeit für die Stammbaumdatenbank ein. Doch wie sieht es nun mit der Darstellung des Baums aus? Gab es im ersten Entwurf noch voneinander abhängige Tabellen, die als Unterdatenblätter der Elterntabelle eingeblendet werden konnten, so scheint diese Möglichkeit bei nur einer Tabelle nunmehr auszuscheiden.

Anzeige rekursiver Tabellen

Tatsächlich kommt aber nur Access von sich aus nicht auf die Idee, in einer Tabelle als Unterdatenblatt eben die gleiche Tabelle einzutragen. Doch genau dies ist machbar. Nehmen Sie den Entwurf der Tabelle **tblElternKind** in Bild 4. Dort ist ein Ausschnitt des Eigenschaftenblatts der Tabelle eingeblendet. Als Unterdatenblatt ist die **tblElternKind** selbst eingetragen und die Verknüpfung über **ParentID** und **ID** gewährleistet. Nachdem Sie diese Änderung abgespeichert haben, zeigt das Datenblatt die bekannten **Plus-Symbole** und klappt zu Eltern tatsächlich die Kinder aus, diese wiederum die Enkel, diese die Urenkel, und so fort. Access erzeugt hier also für jede Unterebene eine neue Instanz der Tabelle und verknüpft diese im Hintergrund mit dem Datensatz der übergeordneten Instanz.

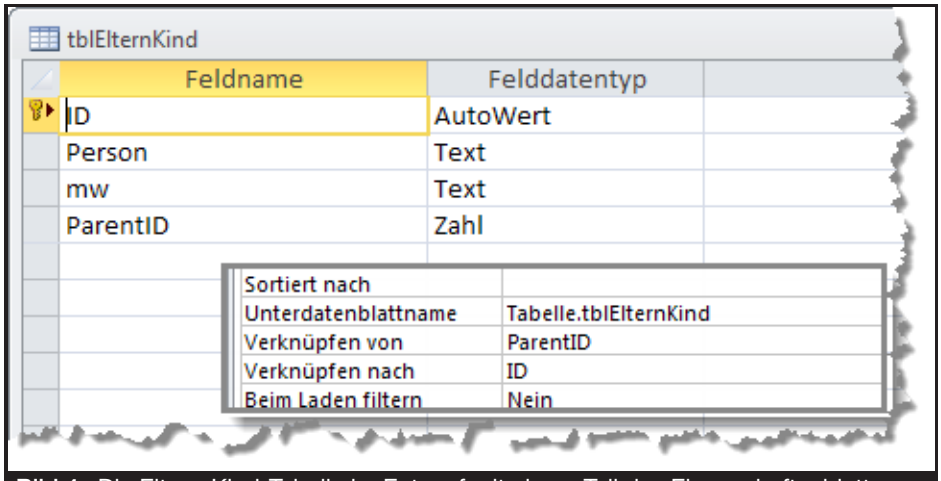


Bild 4: Die Eltern-Kind-Tabelle im Entwurf mit einem Teil des Eigenschaftenblatts

Für einen ersten Überblick mag das praktisch sein. Doch bei genauerem Hinsehen stört dann doch, dass bei allen Kindebenen etwa überflüssigerweise eine Zeile für einen neuen Datensatz angezeigt wird, und überhaupt sind die Spalten **ID** und **ParentID** wenig informativ. Hier greifen wir besser zu Abfragen, die die Ansicht aufräumen.

Die Tabelle selbst zeigt in der Datenblattansicht auf erster Ebene alle Datensätze an. Also nicht nur die Eltern, sondern genauso die Enkel und Urenkel. Wir möchten dies so einschränken, dass nur **echte** Eltern dargestellt werden. Dazu müssen die Daten gefiltert werden, wie in der Abfrage **qryElternKind** aus Bild 5. Das Kriterium dafür, dass eine Person kein Elter besitzt, ergibt sich aus **ParentID**. Ist hier nichts eingetragen, so handelt es sich um einen **Root-Datensatz**. Das Auswahlkriterium lautet folglich **ParentID Ist Null**. Da das Feld ansonsten für die Ansicht nichts hergibt, wird es zudem ausgeblendet. Weil sie nun aber eben dieses Verknüpfungsfeld nicht mehr im Ergebnis ausgibt, kann die Abfrage als Unterdatenblatt nicht

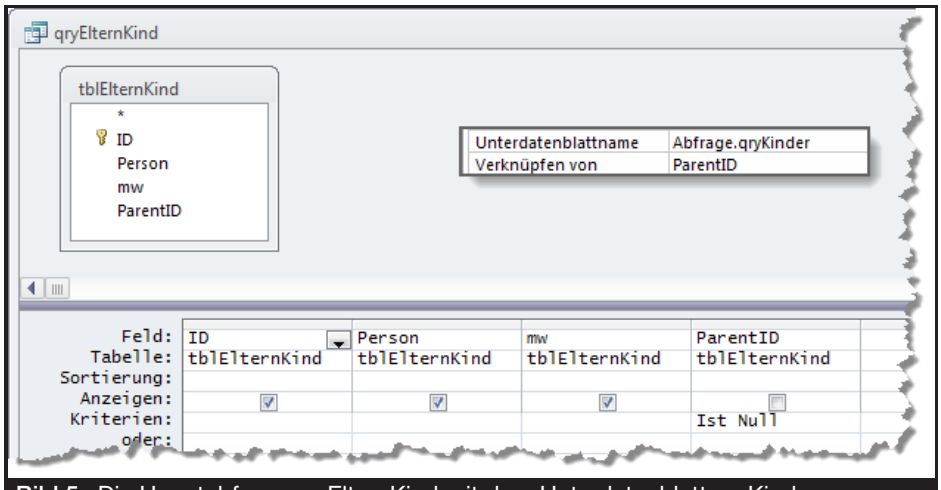


Bild 5: Die Hauptabfrage qryElternKind mit dem Unterdatenblatt qryKinder

XML-Import und -Export

Die Zahl der von Access unterstützten Formate für den Export und Import von Daten nimmt mit mit jeder neuen Version stetig ab. So strich man zuletzt etwa den dBase-Austausch. Anders beim XML-Format, das Microsoft offenbar wichtig genug erschien, um die Export-Optionen gar weiter auszubauen. In diesem Beitrag daher ein Rundblick über die Austauschmöglichkeiten...

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1509_XML.zip**.

Wozu XML?

Beim Import und Export von Datenbankobjekten haben Formate, wie **Excel** oder **dBase**, den Nachteil, dass sie auf proprietären Binärstrukturen aufsetzen, mit denen andere Anwendungen unter Umständen nichts anzufangen wissen. Bei XML ist das anders.

Es ist standardisiert, gut strukturiert, und lässt sich als Textformat grundsätzlich lesen. Aus diesem Grund hat XML zunehmend Bedeutung erlangt. Microsoft speichert etwa selbst Word- oder Excel-Dateien seit der Office-Version 2007 in diesem Format ab, erkennbar am Anhang: **docx**, **xlsx**. Auch zahlreiche Einstellungen der Oberfläche der Programme werden in XML-Strukturen untergebracht. Also ist auch der Austausch von Datenobjekten per XML eine Angelegenheit, die einigen Sinn macht, sei es für den Transfer von Objekten in andere Datenbanken, oder Backups für die aktuelle Datenbank, oder gar für die Weiterverarbeitung in Fremdanwendungen.

Hier geht es in erster Linie um den Austausch von Tabellen **innerhalb** von Access. Der Import von XML-Quellen aus anderen Anwendungen wäre ein weiteres Thema, das noch deutlich komplexer ausfiele, in diesem Rahmen aber nicht weiter diskutiert wird.

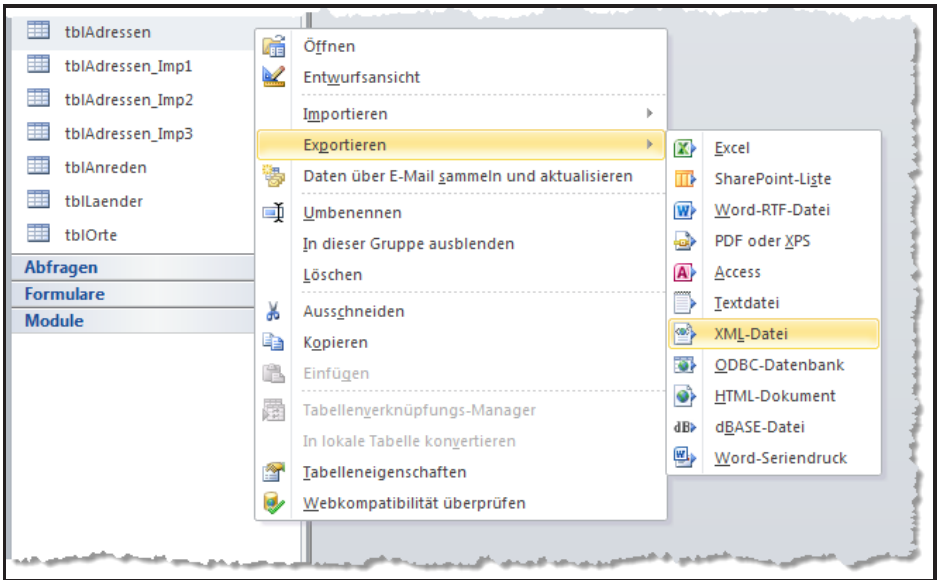


Bild 1: Exportieren einer Tabelle über das Kontextmenü im Navigationsbereich

Export über die Oberfläche

Zum Exportieren eines Datenbankobjekts in eine XML-Datei bedienen Sie sich entweder des Menübands oder des Kontextmenüs. Aktivieren Sie den Registerreiter **Externe Daten**: Dort findet sich eine mit **XML-Datei** beschriftete Schaltfläche in der Gruppe **Exportieren**. Sie ist immer dann aktiviert, wenn im Navigationsbereich ein Objekt markiert ist, welches eine Datenquelle enthält oder enthalten kann. Also nicht nur bei Tabellen und Abfragen, sondern auch bei Formularen und Berichten. Das Gleiche gilt für den gleichlautenden Kontextmenüeintrag, welcher, wie in Bild 1, verschachtelt zum Vorschein kommt.

Der Anschein der Möglichkeit, Formulare und Berichte ebenfalls exportieren zu können, trügt allerdings. Was bei diesem Vorgang ausgegeben wird, ist nicht der komplette Aufbau etwa eines Formulars, sondern lediglich die ihm zugrundeliegenden Daten, die ja aus einer Tabelle oder Abfrage stammen. Versuchen Sie das Formular **frmIntro** der Beispieldatenbank zu exportieren, so quit-

tiert Access dies mit der Bemerkung, dass keine Formulare ohne Datenbindung für diesen Vorgang zugelassen seien. Denn allein die Datenquelle interessiert den XML-Export.

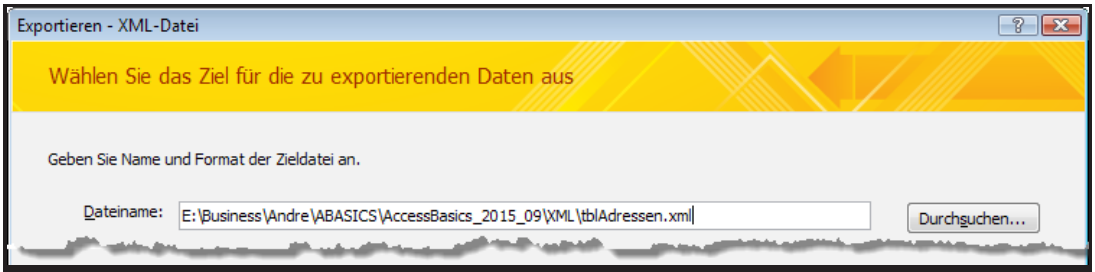


Bild 2: Ausschnitt des Access-Dialogs zum Export eines Objekts in eine XML-Datei

Markieren Sie also die Tabelle **tblAdressen** im Navigationsfenster und klicken Sie auf den Export-Button **XML-Datei** im Menüband. Das öffnet den Export-Dialog, wie in Bild 2. Access trägt in das Feld für den Dateinamen selbständig eine Bezeichnung ein, die sich aus dem Namen des Datenbankobjekts und der Endung **.xml** bildet. Der Pfad variiert dagegen je nach zuletzt getätigtem Ex- oder Import. Deshalb wählen Sie einen geeigneten Ausgabeort über den Dateiauswahl-Button **Durchsuchen...** rechts neben dem Feld aus.

Die Schaltfläche **Weiter** führt Sie zu einem Dialog, der einige Optionen für den Export abfragt, wie in Bild 3. Voreingestellt sind hier **Daten** und **Schema**. **Daten** exportiert den Inhalt der Datensätze, **Schema** erzeugt zusätzlich eine **XSD-Datei** im selben Verzeichnis, welche dann den gleichen Namen trägt, wie die XML-Datei. Diese **XML-Schema-Datei** beschreibt den Aufbau der Tabelle, sowie aller Felder, genau und verewigt hier auch alle Eigenschaften, wie Datentypen, Default-Werte oder Beschriftungen. Für den ersten Test **deaktivieren** wir diese Option und benennen nach dem Export die XML-Datei in **tblAdressenOhneSchema.xml** um. Der Export startet sofort bei Klick auf den **OK-Button** des Dialogs. Die Option zu **Präsentation der Daten (XSL)** lassen wir hier und auch im weiteren Verlauf des Beitrags außen vor. Sie hat für den reinen Datenaustausch keinerlei Bedeutung.

Wiederholen Sie den Vorgang und aktivieren nun die **Schema-Option**, wobei Sie es beim Namen **tbladressen.xml** für die exportierte Datei belassen.

Ziel dieses Unterfangen ist es, die XML-Dateien nachher zu importieren und dann zu begutachten, ob sich Unterschiede durch die Exportoptionen ergeben haben.

Ein Wort zur hier in allen Beispielen eingesetzten Tabelle **tblAdressen**, die auch schon in anderen Beispieldatenbanken für **Access Basics** zu finden ist. Diese wurde nämlich hier etwas aufgebohrt und mit weiteren Feldern versehen, die den Export etwas erschweren sollen. Da ist etwa das zusätzlich Feld **Geburtsdatum**, damit die Tabelle auch ein Feld des Datentyps **Datum** enthält. Und **Vermoegen** wurde, obgleich weitgehend sinnfrei, eingebaut, um ein **Währungsfeld** zu testen. Weiter ist das Feld **Alter** vom Typ **Double**. Schließlich hoffen wir den Export ins Schleudern bringen zu können, indem ein **Anlagefeld Bild** hinzugefügt wurde. Da es sich bei einer **Anlage** um eine obskure Angelegenheit mit verborgenden Detailtabellen handelt, ist der Export in ein einfaches Textformat, wie es bei XML vorliegt, nicht selbstverständlich. Wir sind gespannt, wie Access damit umgehen wird.

Außerdem gibt es drei Felder, die über eine **ID** auf andere Tabellen verweisen: Die Felder **IDAnrede**, **IDOrt** und **IDLand** sind **Nachschlagefelder** für die Tabellen **tblAnreden**, **tblOrte** und **tblLaender**. In der Datenblattansicht werden die entsprechenden Werte im jeweiligen Kombinationsfeld aus diesen Tabellen bezogen und korrekt angezeigt. Was macht der XML-Export daraus?

Bild 3: Abfrage der grundlegenden Export-Optionen

Für den letzten Export-Test klicken Sie im Optionen-Dialog auf die Schaltfläche **Weitere...**, die zusätzliche Exporteinstellungen im nun erscheinenden

Dialog erreichbar macht, wie in Bild 4 und Bild 5.

Unter **Zu exportierende Daten** gibt es da einen kleinen Baum, der unter der natürlich aktivierten **tblAdressen** noch den Eintrag **Daten nachschlagen** verzeichnet. Und unter diesem sind pfiffig alle Tabellen aufgelistet, auf die die Haupttabelle über die Felder **IDAnreden** bis **IDLaender** zugreifen muss, um deren Nachschlagefelder füllen zu können.

Aktivieren Sie nun alle drei Detailtabellen im Dialog.

Eine andere Codierung vorzusehen, als **UTF-8** – **UTF-16** wäre die andere Option –, ist überflüssig. Das wäre nur dann sinnvoll, wenn die XML-Datei später von anderen Programmen bearbeitet werden sollte, die mit **UTF-8** nichts anfangen können.

Auf der zweiten Registerseite des Dialogs (siehe Bild 5) geht es um den Export des Schemas der XML-Datei. Hier sind automatisch die beiden ersten Optionen aktiviert. **Primärschlüssel- und Indexinformationen einschließen** schreibt alle Indizes der Tabelle und deren Eigenschaften ins Schema.

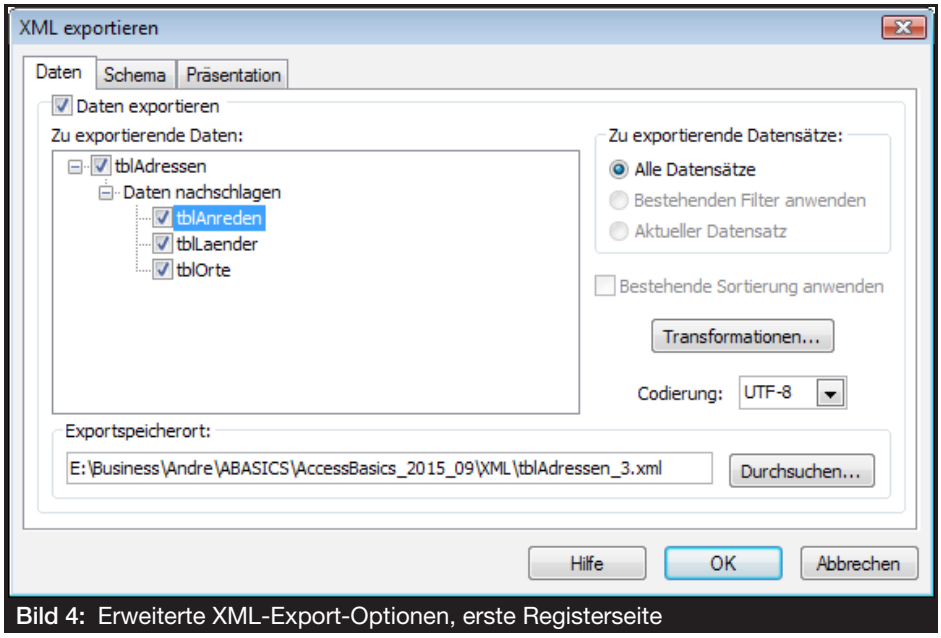


Bild 4: Erweiterte XML-Export-Optionen, erste Registerseite

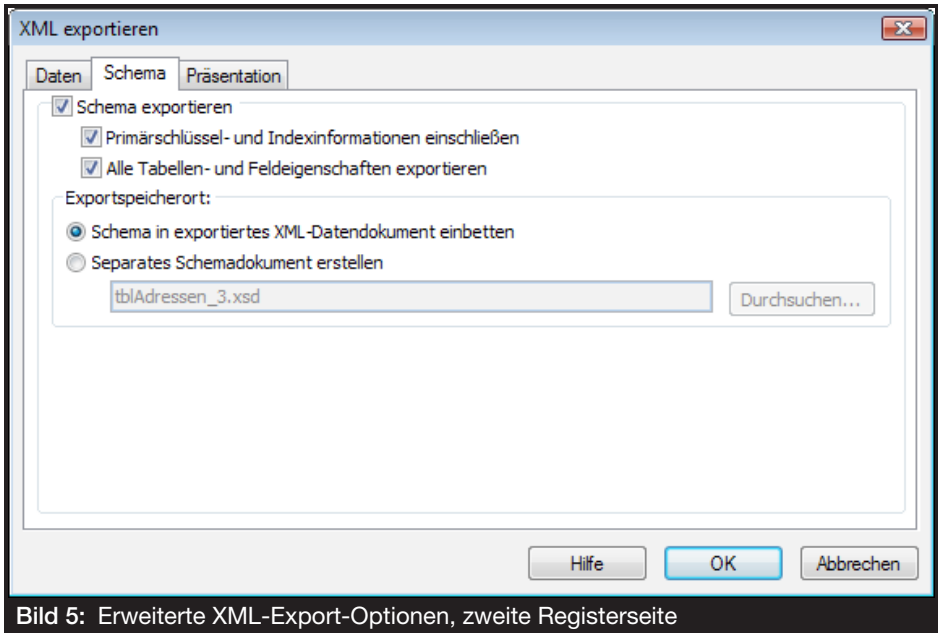


Bild 5: Erweiterte XML-Export-Optionen, zweite Registerseite

Alle Tabellen- und Feldeigenschaften exportieren bewirkt, dass zusätzlich nicht nur die SQL-relevanten Informationen mit eingeschlossen werden – also Datentypen, **Default-Werte**, Einschränkungen, et cetera –, sondern sogar Ansichtseigenschaften, wie Feldbeschreibungen, Schriftart der Datenblattansicht, oder deren Farben. Lassen Sie beide Kontrollkästchen aktiv.

Interessant ist die letzte Option **Schema in XML-Datei einbetten**. Markieren Sie diese, so unterbleibt die Ausgabe der separaten **XSD-Datei** und deren Inhalt wird stattdessen in den Kopf der XML-Datei aufgenommen. Das ist unter Umständen praktischer, als die Aufteilung in zwei Dateien.

Nach dem Einstellen aller Optionen klicken Sie auf **OK**, wodurch Sie nicht etwa zum vorigen Dialogfenster zurückkehren, sondern der Export unmittelbar ausgeführt wird. Benennen Sie die exportierte XML-Datei diesmal in **tblAdressen_PlusSche-**

ma.xml um. Nebenbei: Der Button **Hilfe** des Dialogs hat keine praktische Bedeutung. Wieder einmal öffnet sich damit die Access-Hilfe ohne erkennbaren Inhalt. Aber daran ist man ja mittlerweile gewöhnt...

Ergebnisse des XML-Exports

Wenn Sie sich die Mühe sparen möchten, die beschriebenen Exportvorgänge alle selbst durchzuführen, so finden die exportierten XML-Dateien auch begleitend zur Beispieldatenbank im gleichen Verzeichnis.

Zwar möchten wir die Dateien in erster Linie wieder reimportieren, aber ein erster Blick in ihr Innenleben kann nicht schaden. Dabei stellt sich die Frage, welches Instrument uns hier zur Verfügung steht?

Da es sich um Textdateien handelt, könnte **Notepad** das Mittel der Wahl sein. Mit so großen Dateien – 3 bis 4 MB immerhin – hat es in der Regel jedoch einige Mühe. Erheblich besser geeignet ist etwa das freie **Notepad++**, welches außerdem automatisch Unterstützung für XML-Dateien mitbringt und **XML-Tags** und -Inhalte farblich markiert. **Plug-Ins** ermöglichen zudem speziellere XML-Aufgaben, wie das Einrücken der Tags. Auch der für Dateien mit der Endung **.xml** ohnehin häufig voreingestellte **Internet Explorer** kommt infrage, der ebenfalls die Struktur der XML-Dateien farblich aufwertet und einen hierarchischen Baum darstellt.

Am besten fährt man natürlich mit speziellen XML-Editoren, die allerdings in guter Qualität nicht kostenlos zu haben sind. Der Autor verwendet etwa die **Stylus Studio XML Suite**, welche für einige **Screenshots** zur Anwendung kam.

```
<?xml version="1.0" encoding="UTF-8"?>
<dataroot xmlns:od="urn:schemas-microsoft-com:officedata"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="tblAdressen.xsd"
          generated="2015-09-28T15:31:11">
        <tblAdressen>
                <ID>17</ID>
                <IDAnrede>6</IDAnrede>
                <Nachname>Fetzer</Nachname>
                <Vorname>Armin</Vorname>
                <Telefon>+49 710461725</Telefon>
                <Email>Armin.Fetzer@yahoo.de</Email>
                <Geburtsdatum>1982-01-14T00:00:00</Geburtsdatum>
                <Alter>33.67</Alter>
                <Vermoegen>16000</Vermoegen>
        </tblAdressen>
</dataroot>
```

Listing 1: Ein Datensatz der XML-exportierten Tabelle tblAdressen

```
<?xml version="1.0" encoding="UTF-8"?>
<dataroot xmlns:od="urn:schemas-microsoft-com:officedata"
                generated="2015-09-28T17:08:28">
        <tblAdressen>
                <ID>2575</ID>
                <IDAnrede>6</IDAnrede>
                <Nachname>Steglich</Nachname>
                <Vorname>H.</Vorname>
                <Strasse>Flachsrotten</Strasse>
                <NrStrasse>21</NrStrasse>
                <PLZ>18528</PLZ>
                <IDOrt>292</IDOrt>
                <IDLand>8</IDLand>
                <Telefon>+49 6214446</Telefon>
                <Email>H..Steglich@t-online.de</Email>
                <Geburtsdatum>1945-12-10T00:00:00</Geburtsdatum>
                <Alter>69.75</Alter>
                <Vermoegen>71000</Vermoegen>
        </tblAdressen>
</dataroot>
```

Listing 2: Weiterer Datensatz in der XML-Datei ohne Schema-Referenz

Nehmen wir uns ein Extrakt der **tblAdressen.xml** vor, nämlich einen exportierten Datensatz in ihr, wie in Listing 1. Der Hauptknoten des Dokuments trägt immer den Namen **dataroot**, in dem als Attribute allerlei **Namespace**-Verweise (**xmlns**) eingetragen sind, wobei auffällt, dass der Dateiname der Schemadatei **tblAdressen.xsd** explizit mit aufgenommen ist. Zum Vergleich ein Datensatz aus der Datei **tblAdressenOhneSchema.xml** in Listing 2. Hier fehlen dann tatsächlich alle weiteren **Namespace**-Angaben im Kopf, bis auf das Standardschema **officedata**, auf welches allerdings im weiteren Dokument keinerlei Bezug genommen wird. Auf die Folgen kommen wir noch zu sprechen.

Der Aufbau beider Dateien ist ansonsten gleich. **tblAdressen** ist der erste Unterknoten des Dokuments, der gleichzeitig jeweils einen Datensatz darstellt. Ihm folgen die einzelnen Felder, wobei auffällt, dass sich deren Anzahl unterscheidet. Tatsächlich werden nur jede Felder in einen XML-Datensatz aufgenommen, die nicht leer waren. Damit Sie auch einen Datensatz mit bestücktem Anlagefeld zu Gesicht bekommen, steht ein solcher in Listing 3. Dem **XML-Tag Bild** für das Anlagefeld folgen weitere Unterknoten, wobei die eigentlichen Bilddaten in **FileData** stehen. Es handelt sich hier um eine **base64**-kodierte Darstellung des binären Inhalts, welche im Listing deutlich gekürzt wurde.

```
<tblAdressen>
        <ID>13</ID>
        <IDAnrede>12</IDAnrede>
        <Nachname>Derfert</Nachname>
        <Vorname>Daniela</Vorname>
        <Strasse>Neckarstr.</Strasse>
        <NrStrasse>22</NrStrasse>
        <PLZ>33613</PLZ>
        <IDOrt>331</IDOrt>
        <IDLand>10</IDLand>
        <Telefon>+49 775522</Telefon>
        <Geburtsdatum>1949-12-22T00:00:00</Geburtsdatum>
        <Alter>65.75</Alter>
        <Vermoegen>30000</Vermoegen>
        <Bild>
                <FileData>FAAAAAEAAAAEAAAAagBwAGcAAAD/2P/gABBKRklGAA
ECAABkAGQAAP/sABFEdWNreQABAAQAAAAaAAD/7gAOQWRvYmUAZMAAAAAB/9sAhA
IiIiInNjAzLCwsMzA7OzY2OztBQUFBQUFBQUFBQUH/wAARCAANAA0DASIAAh
EBAxEB/8QAZwAAAwEAAAAAAAAAAAAAAAAAAAgMEBQEBAQAAAAAAAAAAAAAAAAAw
xCYKVuAYAJgU5F4a+lOF1OCEqpTmFXSAXxlKZnjNhA9itX7Jk4Dbkcq Ai93d7Sqv
b0NnWEA95n/9k=
                </FileData>
                <FileName>vert.jpg</FileName>
                <FileType>jpg</FileType>
        </Bild>
</tblAdressen>
```

Listing 3: Ein Datensatz mit Anlage in der exportierten XML-Datei

Soweit die Textform der XML-Dateien. Um Ihnen eine Vorstellung zu geben, was ein XML-Viewer aus diesen Daten macht, findet sich in Bild 6 ein Screenshot aus Stylus Studio, der die Schema-Struktur eines Datensatzes aufzeigt. Sie sieht nicht viel anders aus, als in der Textform; aber das ist ein gutes Zeichen dafür, dass auch andere Programme mit dem Inhalt der XML-Datei zurechtkommen und sie nicht falsch interpretieren. Denn es gibt viele verschiedene Möglichkeiten, um tabellarische Daten in XML darzustellen.

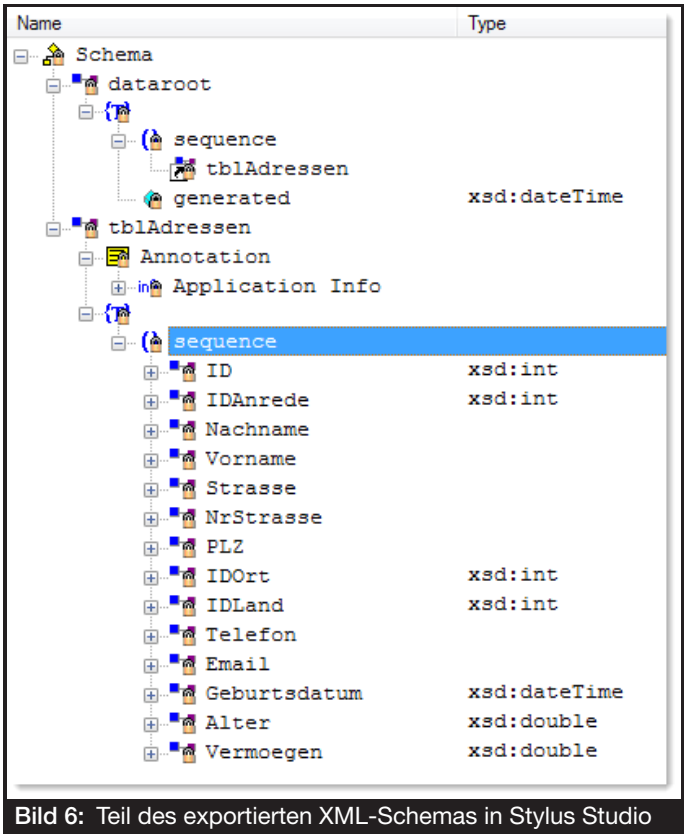


Bild 6: Teil des exportierten XML-Schemas in Stylus Studio

Kommen wir zum gegebenenfalls beim Export generierten Schema, das entweder als separate Datei vorliegt, oder in die XML-Datei integriert wurde. Inspizieren Sie die **tblAdressen.xsd** mit einem Texteditor. Ein Abdruck ist wegen ihres Umfangs hier nicht möglich.

Wesentlich ist, dass als **Schema-URN** der Ausdruck

```
xmlns:od="urn:schemas-microsoft-com:officedata"
```

im Kopf zu finden ist. Im Folgenden beziehen sich fast alle Elementdefinitionen auf diesen **Schema-Namespace** und beginnen deshalb mit dem Präfix **od:**. Und damit haben wir

es mit einem Knackpunkt zu tun: Kein XML-Editor weiß mit diesem **Namespace** etwas anzufangen, da keine entsprechende Schema-Datei zu ihm vorhanden ist. Nur MS-Office-Anwendungen selbst kennen die Bedeutung der einzelnen **XML-Attribute**, wie etwa:

```
od:tableProperty ...
od:tableProperty ...
... type="3"
... od:jetComplexType="MSysComplexTy
pe_Attachment"
```

Da es sich hier überall um Typen handelt, die nicht in **w3.org** standardisiert sind, bleibt die Interpretation der XSD-Datei ausschließlich Access selbst vorbehalten. Wir werden sehen, dass es beim Re-Import tatsächlich regen Gebrauch vom Schema macht. Für die Weiterverwendung der XML-Dateien in anderen Programmen jedoch können Sie getrost auf die Ausgabe des Schemas verzichten. Mit einer Ausnahme: auch die anderen MS-Office-Programme kennen das Schema von **officedata**. An verschiedenen Stellen lassen sich XML-Daten auch unter Excel und Word einlesen. Mit den von Access exportierten kommen sie dabei bestens klar!

Import über die Oberfläche

Sie stoßen den Import einer XML-Datei an, indem Sie entweder im Menüband den Button **Externe Daten | Importieren | XML-Datei** anklicken, oder im Navigationsbereich, Bereich **Tabellen**, das Kontextmenu und **Importieren** aufrufen. Der Import-Dialog lässt Sie dann die Datei auswählen und führt Sie im weiteren Verlauf zum Dialog in Bild 7, dem offensichtlich bereits eine Analyse des XML-Inhalts vorausging. Denn hier zeigt der Baum gleich die Tabellen namentlich an, die der Import in ihr zu finden glaubt. Korrekt wird die Tabelle tblAdressen angegeben und außerdem alle in ihr enthaltenen Felder.

Dieser Baum dient nur zur Kontrolle. Eine Aus- oder Abwahl einzelner Tabellen oder Felder ist hier nicht vorgesehen. Sie können als einzige Option aber einstellen, ob nur die Tabellenstruktur importiert werden soll, also die dann angelegte Tabelle leer sein wird, oder ob auch die Daten eingelesen werden sollen. Diese können optional aber ebenso an eine bestehende Tabelle angehängt werden. Das allerdings setzt voraus, dass eine Tabelle exakt gleichen Namens schon vorhanden ist und diese auch dieselben Felder aufweist. Die Datentypen dieser Felder müssen jedoch nicht zwingend genau gleich sein. Der Import schafft ohne weiteres auch die Konvertierung etwa von String zu Long und umgekehrt. Dass eine Textzelle nicht einem Anlagefeld zugeordnet werden kann, versteht sich allerdings von selbst. Bei Inkompatibilitäten bricht der Import nicht ab, sondern führt alles, soweit möglich, durch und gibt stattdessen zum Schluss die Meldung aus, dass ein Teil der Aufgabe nicht erfolgreich war und weitere Informationen in der erzeugten Tabelle **Importfehler** zu finden seien. In den Datensätzen dieser Tabelle ist dann protokolliert, welche Schwierigkeiten aus welchem Grund auftraten.

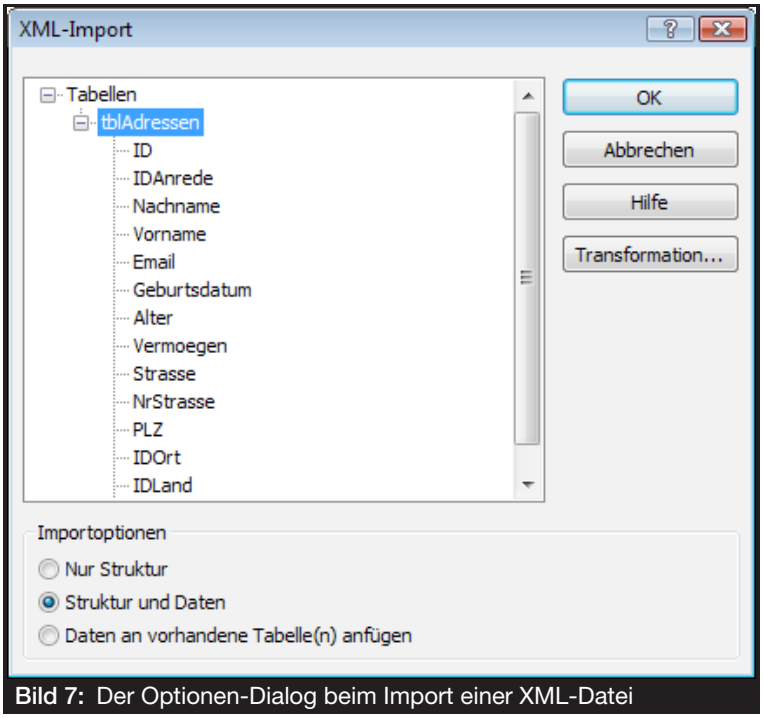


Bild 7: Der Optionen-Dialog beim Import einer XML-Datei

Machen Sie einmal den Import-Test mit der XML-Datei **tblAdressenOhneSchema.xml**. Der Dialog aus Bild 7 zeigt nun interessanterweise nicht nur eine Tabelle an, sondern zwei: Außer der **tblAdressen** verzeichnet er auch eine vermeintliche Tabelle **Bild**. Hier wusste Access sich nicht anders zu helfen und konnte mangels Schema den komplexen Datentyp nicht anders auflösen. Fahren Sie dennoch fort. Als Ergebnis erhalten

Sie eine leere Tabelle Bild, die immerhin die drei internen Anlagefelder anlegte. In der Tabelle **Importfehler** sind allerlei ominöse Gründe für den Fehlschlag zu finden, die sich alle jedoch nur auf diese Tabelle Bild beziehen. Korrekt angelegt hingegen wurde die Tabelle **tblAdressen1** – da eine Tabelle gleichen Namens bereits existierte, benannte der Import sie automatisch um. Sie müssen sich also nicht um das Überschreiben vorhandener Daten sorgen.

Öffnen Sie diese Tabelle, so sieht die Sache auf den ersten Blick schon mal nicht schlecht aus. Alle Spalten und Datensätze sind da. Doch bei genauerem Hinsehen stimmt etwa das Geburtsdatum nicht. Es weist ein Format auf, das exakt dem Inhalt der jeweiligen XML-Zelle entspricht, aber kein gültiges Access-Datum darstellt.

Also öffnen Sie die Tabelle im Entwurf, um festzustellen, dass alle Felder samt und sonders im Textformat angelegt sind. Der Import konnte rein aus den XML-Feldinhalten nicht ermitteln, um welche Datentypen es sich handelt! Ohne Schema wäre hier Spekulieren angesagt.

Führen Sie den Test abermals mit der Datei **tblAdressen.xml** durch, die ja eine begleitende Schema-Datei besitzt. Hier läuft alles glatt. Nicht nur, dass die importierte Tabelle alle Felder mit korrekten Datentypen anzeigt und die gleichen Indizes anlegte, wie sie in der Originaltabelle vorhanden sind, auch die Ansicht des Datenblatt hat genau die gleiche Gestalt. Schriften oder Hintergründe stimmen mit der Originaltabelle überein.

Der Austausch von Tabellen über XML hat also einige Vorteile, weil hier wirklich eine 1:1-Kopie vorgenommen wird. XML-exportierte Tabellen können Sie so auch in andere Datenbanken transferieren, soweit sie nicht von anderen Tabellen abhängen.

Das allerdings ist bei unserer Adresstabelle durchaus der Fall. Benötigt sie doch für die korrekte Anzeige von Anreden, Orten und Ländern noch die drei Lookup-Tabellen. Aber diese sollten ja ebenfalls in der XML-Datei gespeichert sein, falls die Option **Daten nachschlagen** beim Export aktiviert wurde.

Machen Sie also einen weiteren Test mit der Datei **tblAdressen_PlusSchema.xml**, die das Ergebnis dieser Einstellung war. Das Dialogfenster zeigt beim Import nun tatsächlich alle vier Tabellen an und importiert sie in der korrekten Struktur. Über eine einzige XML-Datei können demnach mehrere Tabellen in eine andere Datenbank verfrachtet werden. Einziger Wermutstropfen: Die etwaigen Beziehungen zwischen den Tabellen legt der Import nicht an. Denn über diese steht weder in der XML-Datei, noch im Schema etwas.

Zu guter Letzt der Versuch, die Daten an die vorhandene Tabelle **tblAdressen** anzuhängen. Haben Sie diese Option beim Import aktiviert, so dürfen Sie sich auf eine längere Denkpause von Access gefasst machen. Denn kein einziger Datensatz wird importiert. Stattdessen finden Sie in der Tabelle **Importfehler** 9706 Gründe, warum es nicht ging: Der Primärindex **ID** der Tabelle verhinderte, dass Daten des gleichen Werts angefügt werden. Wäre die Zieltabelle leer gewesen, stünde einem Import der Datensätze hingegen nichts im Wege.

Testweise können Sie die **tblAdressen** leeren; sie haben ja eine Sicherung in Form der **tblAdressen.xml** vorliegen. Sie werden nun feststellen, dass selbst der Import aus der XML-Datei ohne Schema erfolgreich verläuft und die erforderliche Datenkonvertierung gewährleistet wird – mit Ausnahme des Anlagefelds natürlich...

Abfragen exportieren und importieren

Nicht nur Tabellen können nach XML exportiert werden. Mit Abfragen klappt es ebenfalls.

Die Abfrage **qryAdressen** der Beispieldatenbank fasst die Tabelle **tblAdressen** und ihre drei **Lookup-Tabellen** zusammen. Der Aufbau ergibt sich aus Bild 8 ersichtlich. Statt der Schlüssel-IDs für **Anrede**, **Ort** und **Land** werden direkt die passenden Werte im Ergebnis eingesetzt. Schauen wir, was der XML-Export damit anstellt.

Der Ablauf des Exports ist prinzipiell der gleiche, wie bei Tabellen. Im erweiterten Exportdialog fehlt allerdings die Option **Daten nachschlagen**. Ob Sie die Einstellung zu **Indizes einbeziehen** aktivieren, ist ohne Belang. Eine Abfrage weist solche nicht auf. Die Tabellen- und Feldeigenschaften sollten jedoch ausdrücklich aktiv sein. Auch das Schema sollte exportiert werden.

Recordset-Typen

Wenn Sie lesend oder schreibend auf die Daten von Tabellen und Abfragen unter VBA zugreifen, so steht fast immer ein DAO-Recordset im Mittelpunkt des Geschehens. Dass es von diesem unterschiedliche Varianten gibt, wird dabei gern übersehen. Untersuchen wir hier, welche Typen existieren und wie sie sich in ihrem Verhalten unterscheiden.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1510_DAO_Recordsets.accdb**

Öffnen eines Recordsets

Der Zugriff auf die Daten einer Tabelle oder Abfrage geschieht unter **DAO** und **VBA** immer über ein **Recordset**, welches auf ein **Database**-Objekt und seine Methode **OpenRecordset** geöffnet wird. Access stellt eine Instanz der aktuellen Datenbank über seine Funktion **CurrentDb** bereit. Über sie greifen Sie etwa auf eine Tabelle **tblAdressen** zu, indem Sie deren Namen der **OpenRecordset**-Funktion als Parameter übergeben. Im einfachsten Fall sieht das so aus:

```
Dim rs As DAO.Recordset
Set rs = CurrentDb.OpenRecordset_
                    ("tblAdressen")
```

Statt direkt eine Tabelle anzugeben, kann auch ein **SQL-String** eingesetzt werden:

```
Set rs = CurrentDb.OpenRecordset_
         ("SELECT * FROM tblAdressen")
```

Das Ergebnis ist dasselbe. Ist der **SQL-String** sehr komplex und die Schreibarbeit unter VBA damit zu aufwändig, so kommt auch das Öffnen eines Recordset über eine Abfrage in Betracht:

```
Dim qdf As QueryDef
Dim rs As DAO.Recordset
Set qdf = CurrentDb.QueryDefs_
                    ("qryAdressen")
Set rs = qdf.OpenRecordset()
```

Hier befände sich der **SQL-String** in der Auswahlabfrage **qryAdressen**. Das Ergebnis ist auch hier identisch mit dem Ergebnis des über einen **SQL-String** direkt geöffneten Recordset-Objekts, wie zuvor.

Was bei der ganzen Sache unterschlagen wird, ist der zweite Parameter, der der Methode **OpenRecordset** übergeben werden kann. Er gibt an, welcher **Recordset-Typ** von der Methode zurückgegeben werden soll. Dies könnte etwa der Typ **Dynaset** sein, welcher in der Folge das Bearbeiten der Daten erlaubte:

```
Dim rs As DAO.Recordset
Set rs = CurrentDb.OpenRecordset_
         ("tblAdressen", dbOpenDynaset)
```

Was jedoch passiert, wenn Sie diesen Parameter weglassen? Welcher **Recordset-Typ** kommt dabei heraus?

Das können Sie ermitteln, indem Sie anschließend die **Type**-Eigenschaft abfragen:

```
Debug.Print rs.Type
```

Sie erhalten damit eine Zahl, die einen der Elemente von **RecordsetTypeEnum** wiederspiegelt. Öffnen Sie den **VBA-Objektkatalog**, stellen das obere Kombinationsfeld auf **DAO** ein und navigieren in der Klassenliste zu dieser Enumeration.

Rechts im Methodenfenster finden Sie nun die fünf möglichen Typen. Beim Klick auf einen der Einträge sehen Sie im Beschreibungsfenster ganz unten den Wert der Konstanten. Für unser erstes Beispiel wird der Wert **1** und damit der Typ **dbOpenTable** ermittelt.

Doch die Annahme, dass bei Weglassen des optionalen Typ-Parameters für **OpenRecordset** das Ergebnis-Recordset immer vom Typ **Table** ist, trügt leider. Denn für das zweite Beispiel mit dem **SQL-String** bekommen Sie den Typ-Wert **2**, was dem **Dynaset** entspricht. Gleiches gilt für das auf die Abfrage geöffnete Recordset. Und auch bei **verknüpften Tabellen**, die sich in einer anderen Datenbank befinden, stellt **DAO** den Typ automatisch auf **Dynaset** ein.

Aus diesem Grund ist es ratsam, den Typ-Parameter grundsätzlich anzugeben. Denn das Verhalten der Recordset-Varianten unterscheidet sich durchaus, wie wir gleich sehen werden.

Recordset-Typ Table

Dieser über den Parameterwert **dbOpenTable** erhaltene Typ lässt sich ausschließlich für die in der aktuellen Datenbank befindlichen Tabellen verwenden. Verknüpfte Tabellen und Abfragen kommen hier nicht infrage. Andernfalls käme es beim Aufruf zu einer Fehlermeldung. Seine Datensätze sind aktualisierbar, was bedeutet, dass Datenwerte geändert und -zeilen gelöscht oder hinzugefügt werden können.

Da auch der Typ **Dynaset** dies unterstützt, fragt sich, wofür dieser Typ überhaupt gut ist. Es gibt genau einen Grund: Nur dieser Typ erlaubt den Aufruf der Methode **Seek** zum schnellen Suchen von Datensätzen auf Felder, die indiziert sind. Dazu später mehr.

Recordset-Typ Dynaset

Mit dem Parameter **dbOpenDynaset** wird von **OpenRecordset** dieser Typ zurückgegeben. Auch hier sind die Datensätze bearbeitbar. Allerdings nicht nur für in der aktuellen Datenbank befindliche Tabellen, sondern auch für verknüpfte Tabellen und Abfragen. Bei Letzteren ist allerdings Voraussetzung, dass sie selbst einen aktualisierbaren Aufbau besitzen. Öffnen Sie die fragliche Abfrage einfach als Datenblatt und versuchen Sie darin Datensätze zu bearbeiten. Klappt das, so kann auf die Abfrage mit diesem Recordset-Typ zugegegriffen werden. Doch auch **nichtaktualisierbare Abfragen** verweigern diesen Parameter **dbOpenDynaset** nicht. Allerdings wird in der Folge beim Editieren von Datensätzen eine Fehlermeldung ausgegeben.

Recordset-Typ Snapshot

Geben Sie **dbOpenSnapshot** als Typ-Parameter für das Recordset an, so einsteht eine Datenquelle, deren Datensätze nicht bearbeitet werden können. Sie können so sicherstellen, dass nicht aus Versehen Daten von Tabellen verändert werden. Das jedoch ist auch schon der einzige sinnvolle Grund, diesen Typ einzusetzen. Denn die Performance dieses Typs ist, wie wir noch sehen werden, recht mangelhaft

Unter der Haube geschieht technisch nämlich dies: Bei einem **Snapshot** werden alle Datensätze der Tabelle oder Abfrage komplett in den lokalen Speicher der **Daten-Engine** eingelesen, während bei einem **Dynaset** zunächst nur ein Bruchteil ermittelt wird und die weiteren Datensätze erst auf Anfrage eintrudeln. Diese Anfragen entstehen erst durch die **Move**- und **Find**-Methoden des Recordsets. Damit wird auch klar, weshalb die Funktion **RecordCount** des Recordsets, welche die Zahl der Datensätze zurückgibt, bei beiden Typen unterschiedliche Ergebnisse zeigt. Beim **Snapshot** ist der Wert korrekt, da bereits alle Datensätze im Speicher liegen. Beim **Dynaset** jedoch erhalten Sie zunächst immer die **1**, weil noch kein weiterer Zugriff auf die Datensätze stattfand. Erst nach einem Aufruf von **MoveLast** stimmt die Zahl.

Besonders katastrophal verhält sich das **Snapshot** beim Zugriff auf große verknüpfte Tabellen, die in einer **Backend-Datenbank** auf einem **Netzwerk-Share** liegen. Da hier die gesamte Tabelle oder das gesamte Abfrageergebnis übertragen werden muss, kann es zu deutlichen Verzögerungen und Performance-Einbrüchen kommen.

Lange Rede, kurzer Sinn: Selten gibt es einen vernünftigen Grund, diesen Typ einzusetzen.

Recordset-Typ ForwardOnly

Das über den Typ-Parameter **dbOpenForwardOnly** erhaltene Recordset weist ähnliche Einschränkungen auf, wie ein **Snapshot**. Die Datenquelle ist nicht aktualisierbar. Zu allem Überfluss lassen sich darauf noch nicht einmal die **Find-Methoden** (**FindFirst**, **FindLast**, et cetera) anwenden. Allerdings liegt es in der Performance in der Region eines **Dynaset**, weil hier ebenfalls zunächst nur der erste Datensatz vorliegt und nicht alle Datensätze auf einen Schwung übertragen werden müssen. Man kann sich innerhalb der Datensätze nur in eine Richtung bewegen: vorwärts. Das bedeutet, dass lediglich die Methode **MoveNext** zur Navigation erlaubt ist, während **MovePrevious** und sogar **MoveLast** fehlschlagen.

Mit all diesen Einschränkungen empfiehlt sich der Einsatz dieses Typs überhaupt nicht. Theoretisch könnte die Performance bei Durchlaufen der Datensätze hier die beste sein, das jedoch bestätigt sich unter keinen Umständen.

Der Übersicht halber finden Sie in Bild 1 eine Matrix, die die einzelnen **Recordset-Typen** und deren erlaubte Methoden auflistet. **Read** bedeutet dabei, dass Werte aus den Daten-

sätzen ausgelesen werden können. Das ist selbstverständlich bei allen Typen der Fall. Beim Modifizieren von Datensätze (**Edit**- und **Update**-Methoden) sieht das schon anders aus. Die **Find**-Methoden **FindFirst**, **FindNext**, **FindPrevious** und **FindLast** spendieren nur **Dynasets** und **Snapshots**. Die recht spezielle **Seek**-Methode schließlich erlaubt nur das **Table-Recordset**.

Recordset-Typ \ Methoden	Read	Edit	Find	Seek
dbOpenDynaset	X	X	X	
dbOpenTable	X	X		X
dbOpenSnapshot	X		X	
dbOpenForwardOnly	X			

Bild 1: Recordset-Typen und ihre erlaubte Zugriffsmethoden

Aufsuchen von Datensätzen mit Seek

Während das Suchen innerhalb der Datensätze eines Recordsets über dessen Find-Methoden bereits in einigen Ausgaben von Access Basics behandelt wurde, soll die **Seek**-Methode hier etwas ausführlicher vorgestellt werden.

Während die **Find**-Methoden den Inhalt der Datensätze nacheinander Schritt für Schritt mit dem Suchausdruck vergleichen, bis Übereinstimmung festgestellt wird, geht Seek einen anderen Weg. Hier wird direkt der **Index** eines Feldes herangezogen, über den sehr schnell ein Datensatz identifiziert ist.

Damit sind schon drei Voraussetzungen gegeben:

- Das Recordset muss über den Parameter **dbOpenTable** geöffnet worden sein.
- Das Feld, auf welches die Suche sich bezieht, muss in der Tabelle einen **Index** aufweisen. Er muss nicht unbedingt **eindeutig** sein.
- Es kann über **Seek** nur ein einziges Feld abgefragt werden, da ja auch nur ein Index angesprochen wird. Die Suche nach mehreren Kriterien entfällt hier also.

Diese Voraussetzungen sind in der Tabelle **tblAdressen** der Beispieldatenbank etwa für das Feld **Nachname** erfüllt. Um den Datensatz mit einem bestimmten Namen per **Seek** aufzufinden, gehen Sie so vor:

```
Dim rs As DAO.Recordset
Set rs = CurrentDb.OpenRecordset_
        ("tblAdressen", dbOpenTable)
rs.Index = "Nachname"
rs.Seek "=", "Schmidt"
If Not rs.NoMatch Then
    Debug.Print rs!ID, rs!Vorname
End If
```

Nach dem Öffnen des Recordset vom Typ **Table** wird der Eigenschaft **Index** der Name des abzufragenden Index übergeben. Achtung! Es handelt sich nicht um den Namen des Felds! Im Beispiel ist der Index-Name zwar gleichlautend, kann in anderen Fällen aber abweichen. Den Index-Namen können Sie im Tabellenentwurf ja einstellen, wenn Sie dort den Indizes-Dialog öffnen.

Für die **Seek**-Methode geben Sie mindestens zwei Parameter an. Der erste ist ein String mit dem Vergleichsmodus – hier ein Gleichheitszeichen. Als zweiten Parameter geben Sie den Vergleichsausdruck an – hier **Schmidt**. Wichtig dabei: der **Datentyp** dieses Ausdrucks muss gleich sein, wie das abgefragte Feld! Bei **Nachname** haben wir es mit Text zu tun, und daher ist der Vergleichsausdruck ein **String**. Würden Sie in einem **Datumsfeld** suchen, so müsste der Vergleichsausdruck den Datentyp **Date** aufweisen.

Außerdem ist es möglich mehr, als einen Vergleichsausdruck, anzugeben:

```
rs.Seek "=", "Schmidt", "Müller"
```

Bis zu 13 solcher Ausdrücke nimmt Seek entgegen.

Ob ein entsprechender Datensatz gefunden wurde, offenbart die Eigenschaft **NoMatch** des Recordsets, ähnlich, wie bei den **Find**-Methoden. Steht sie auf **True**, dann wurde kein Datensatz gefunden.Bei **False** hingegen schon. Die Routine gibt diesen Datensatz im Beispiel dann per **Debug.Print** im **VBA-Direktfenster** aus.

Neben dem Gleichheitszeichen als Vergleichsoperator können Sie auch andere einsetzen, wie **>=**, **<=**, **>**, **<**. Konsultieren Sie die Hilfe zu **DAO**, um mehr über deren Bedeutung zu erfahren.

Nach Drücken von **F1** nachdem Sie die **Seek**-Methode im Objektkatalog oder im VBA-Code markiert haben, sollten sich die weiteren Erläuterungen zur Anweisung öffnen.

DAO-Objekte und -Auflistungen

Abseits der wohlbekannten Objekte Database, der Recordsets und Fields, sowie TableDefs und QueryDefs, finden Sie noch etliche andere Klassen in der Bibliothek DAO für den Datenzugriff unter VBA, die deutlich seltener zur Anwendung kommen. Im Überblick stellen wir hier die wichtigsten Vertreter vor und erläutern die grundlegenden Programmierschritte und Verwendungszwecke.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1510_DAOObjekte.accdb**

Auflistungen und Elemente

Unter dem Terminus **Auflistung** versteht man unter VBA eine Klasse, die selbst nur einen Behälter für einzelne **Klassenobjekte**, dessen Elemente, bereitstellt. Sie ähnelt insofern einem **Array**, kommt technisch gesehen aber als **VBA-Collection** daher. Die wesentlichen Methoden sind damit die Eigenschaft **Count**, welche die Zahl der Elemente der **Collection** zurückgibt, die Eigenschaft **Item**, über die auf ein einzelnes Element zugegriffen werden kann, und die Methoden **Add**, sowie **Remove**, mit denen Elemente der Auflistung hinzugefügt oder aus ihr gelöscht werden können. Zudem lassen sich die Elemente der Auflistung grundsätzlich über eine **For-Each-Schleife** durchlaufen. Die Schleifenvariable gibt dabei jeweils ein Item der Auflistung zurück.

Während die **Items** einer allgemeinen **Collection** keinen besonderen Typ aufweisen, sieht das unter den **DAO-Auflistungen** anders aus. Hier entspricht ein **Item** immer einer speziellen **DAO-Klasse** und spiegelt deren **Objektinstanz** wieder. So sind die Elemente einer Auflistung **Fields** vom Objekttyp **Field**, die **Recordsets** eines **Database**-Objekts haben den Typ **Recordset**. **TableDefs** enthalten **TableDef**-Objekte, **QueryDefs**-Auflistungen **QueryDef**-Objekte, und so fort. Die Auflistungs-Collection ist in der Regel schlicht dadurch gekennzeichnet, dass an den Elementtypnamen ein **s** angehängt wird: **Error** und **Errors**, **Parameter** und **Parameters**.

Die **DAO-Auflistungen**, um die es aber in diesem Beitrag gehen soll, sind diese:

- **Containers**
- **Documents**
- **Properties**

Containers, Container

Sie gelangen an die **Containers**, indem Sie die gleichnamige Eigenschaft eines **Database**-Objekts abfragen. Also etwa so:

```
? CurrentDB.Containers.Count
```

Das gibt die Zahl der **Container** aus, die die aktuelle Datenbank enthält. In der Regel sind das neun, wenn die Datenbank keine Besonderheiten, wie etwa **Replikation**, aufweist. Die einzelnen **Container** durchlaufen Sie mit einer Routine, wie in Listing 1. Hier wird die Objektvariable **dbs** auf die aktuelle Datenbank gesetzt, und in der **For-Each-Schleife** werden die Elemente der Auflistung **Containers** abgearbeitet. Die Enumerationsvariable ist mit **daoCont** benannt. Ausgegeben werden die Namen der

```
Sub GetContainers()
    Dim dbs As Database
    Dim daoCont As DAO.Container

    Set dbs = CurrentDb
    For Each daoCont In dbs.Containers
        Debug.Print daoCont.Name
    Next daoCont
End Sub
```

Listing 1: Prozedur zum Durchlaufen der DAO-Auflistung Containers

Container im VBA-Direktfenster. Als Ergebnis erhalten Sie diese Liste:

```
DataAccessPages
Databases
Forms
Modules
Relationships
Reports
Scripts
SysRel
Tables
```

Sie werden daraus ablesen können, dass es sich dabei um alle Objekte handelt, die die Datenbank enthält. Tatsächlich ist die **Containers**-Auflistung der virtuelle Speicher für alles, was es in der Datenbank gibt. **Forms** etwa ist die Gesamtheit der Formula-

re der Datenbank. **Reports** verweist auf die Berichte. **Modules** bezieht sich auf alle VBA-Module, **Scripts** auf die Makros. Bei **Relationships** handelt es sich um die Beziehungen zwischen den Tabellen, falls solche festgelegt wurden, während **SysRel** das grafische Layout der Beziehungen, so, wie es im Beziehungsfenster angezeigt wird, wiedergibt. Bei **DataAccessPages** handelt es sich um eine obsolete Auflistung, die die **Datenzugriffsseiten** enthält, welche früher (**Office 2000**) zum Entwickeln von datengebundenen HTML-Seiten verwendet werden konnten, heute aber nicht mehr unterstützt werden. **Tables** ist der **Container** für alle Tabellen und Abfragen der Datenbank. Access macht hier keinen Unterschied – beide geben eben eine Datenmenge zurück.

Irritierend schließlich ist der **Databases-Container**. Kann eine Datenbank nicht nur ein **Database**-Objekt enthalten? Nein, denn in VBA können Sie über die **OpenDatabase**-Funktion eine weitere Datenbank in der aktuellen Sitzung öffnen, die sowohl auf die gerade geöffnete Datei verweisen kann, wie auch auf eine andere. Zudem listet der Container interessanterweise auch noch die Datenbankeigenschaften (**Summary**) als eigene Datenbank auf, wie auch die Werteliste, die man in den Datenbankeigenschaften im **Backstage** selbst anlegen kann (**UserDefined**).

Documents, Document

Gut zu wissen, dass etwa der Container **Forms** alle Formulare der Datenbank verzeichnet. Doch wie kommt man nun an die einzelnen Formulare? Zunächst verweisen Sie auf den Container über eine Hilfevariable:

```
Dim dbs As Database
Dim daoCont As DAO.Container

Set dbs = CurrentDb
Set daoCont = dbs.Containers("Forms")
```

Sie erhalten also einen bestimmten Container, indem Sie seinen Namen als Index der **Collection** angeben. Alternativ könnte man auch die Ordinalzahl einsetzen, doch deren Wert kann sich von Datenbank zu Datenbank unterscheiden. Elementar ist nun die Methode **Documents** eines **Containers**.

Sie gibt als Auflistungseigenschaft seine einzelnen Dokumente wieder. Der Begriff mag etwas irreführend klingen, denn gewöhnlich spricht man bei Formularen von **Datenbankobjekten**, nicht von **Datenbankdokumenten**. Wie auch immer, die Dokumente können Sie auf die gleiche Weise durchlaufen, wie das bei der **Containers**-Auflistung der Fall war:

```
Dim daoDoc As DAO.Document
For Each daoDoc in daoCont.Documents
    Debug.Print daoDoc.Name
Next daoDoc
```

Als Ergebnis dieser Routine bekommen Sie die Namen aller Formulare der Datenbank.

Kombinieren Sie die Enumeration der **Containers** und der **Documents** miteinander, so kommen Sie zu einer Prozedur, wie **GetDBDocuments** in Listing 2. Hier werden alle Objekte der Datenbank ausgegeben, da in der äußeren Schleife alle **Container**, in der inneren alle **Document**-Elemente dieser durchlaufen werden.

Über das Objektmodell von Access selbst wäre das nicht möglich. Die **Forms**-Auflistung von Access bezieht sich etwa nur auf geöffnete, nicht auf alle gespeicherten Formulare. Zwar gibt es auch noch die Auflistung **CurrentProject.AllForms**, doch die ist eben speziell nur für Formulare gemacht, während die **Containers** allgemeinerer Natur sind. Mit keiner anderen VBA-Lösung kommen Sie so schnell und kurz an die Datenbankobjekte, wie mit der im Listing.

Allerdings kann man damit auch nicht viel mehr ermitteln, als die Namen der Objekte. Denn ein Document-Objekt gibt leider nicht viel her. Neben der Eigenschaft Name

```
Sub GetDBDocuments()
    Dim dbs As Database
    Dim daoCont As DAO.Container
    Dim daoDoc As DAO.Document

    Set dbs = CurrentDb
    For Each daoCont In dbs.Containers
        Debug.Print daoCont.Name
        If daoCont.Documents.Count = 0 Then
            Debug.Print , "-"
        Else
        For Each daoDoc In daoCont.Documents
            Debug.Print , daoDoc.Name
        Next daoDoc
        End If
    Next daoCont
End Sub
```

Listing 2: Durchlaufen der Auflistung Documents

verzeichnet es noch die Datumseigenschaften **DateCreated** und **LastUpdated**, die sich auf Erstelldatum und das letztes Änderungsdatum eines Objekts beziehen.

Das war's auch schon! Doch immerhin finden Sie noch die Methode **Properties** in der Objektklasse, die die Eigenschaften eines Dokuments widerspiegelt. Vielleicht erfahren wir über diese mehr über die einzelnen Objekte?

Properties, Property

Properties sind Auflistungseigenschaften, die unter **DAO** nicht nur einem **Document** innewohnen, sondern grundsätzlich allen Objektklassen. Ein **Container** hat die **Properties**-Methode, ein **Recordset**-, ein **TableDef**-, ein **Field**-, oder ein **DbEngine**-Objekt ebenso. Sie sind also die zentrale Informationsquelle für die Objekte.

Durchlaufen wir einmal die Eigenschaften eines Formularobjekts über dessen **Document**-Element, wie das in Listing 3 für das Formular **frmIntro** der Beispieldatenbank geschieht. Auch hier kommt es zu einer Enumeration über eine **For-Each-Schleife**. Die verwendete Auflistungseigenschaft sind die **Properties** des zuvor festgelegten Documents **daoDoc**.

Was uns hier natürlich interessiert, sind nicht nur die Namen der Eigenschaften, sondern auch deren Werte. Neben **Name** wird deshalb auch noch die Eigenschaft **Value** der **Properties** ausgegeben.

```
Sub DAOFormProperties()
    Dim dbs As Database
    Dim daoCont As DAO.Container
    Dim daoDoc As DAO.Document
    Dim daoPrp As DAO.Property

    Set dbs = CurrentDb
    Set daoCont = dbs.Containers("Forms")
    Set daoDoc = daoCont.Documents("frmIntro")
    For Each daoPrp In daoDoc.Properties
        Debug.Print daoPrp.Name, daoPrp.Value
    Next daoPrp
End Sub
```

Listing 3: DAO-Properties eines Formulars ermitteln

Am **Dim-Block** im Kopf der Prozedur können Sie gut nachvollziehen, wie die Objekte von **Database** bis **Property** hierarchisch voneinander anhängen. Um das nochmals besser zu verdeutlichen, finden Sie in Bild 1 auf der nächsten Seite das prinzipielle **Objektmodell** dieses Systems.

Wenn Sie sich die **Properties** des Formulars ausgeben lassen, werden Sie möglicherweise darüber enttäuscht sein, wie wenig Infos die Routine zutage fördert. Tatsächlich beschränkt sich die Ausgabe im Wesentlichen auf die Eigenschaften, die sich bereits aus den **Document**-Methoden selbst ablesen ließen. So gibt es hier etwa den **Property**-Namen **DateCreated**, der auch eine Eigenschaftsmethode von **Document** ist. Grundsätzlich finden Sie in den **Properties** eines DAO-Objekts immer auch seine Eigenschaftsmethoden. Die Auflistung **muss** sich aber nicht darauf beschränken, wie wir noch sehen werden. Immerhin gibt die Routine etwa auch die Eigenschaft **ThemeResourceName** aus, die den Wert **[Use DB Theme]** aufweist, was besagt, dass das grafische Layout-Thema des Formular sich vom Thema ableitet, das in den Optionen der Datenbank generell eingestellt wurde.

Da **Properties** die Methode eines jeden DAO-Objekts ist, können die Eigenschaften auch in einer gesonderten Prozedur ermittelt werden, welche Listing 4 zeigt. Übergeben Sie der Prozedur als Parameter das DAO-Objekt der Begierde.

Sie enumeriert dann dessen Eigenschaften und gibt neben Name und Wert der Eigenschaft jeweils auch den Datentyp über die Methode **Type** aus. Das ist eine Zahl, die einer der Enumerationskonstanten aus **DataTypeEnum** einspricht. **10** etwa ist der Wert von **dbText**, was einem String entspricht, der 255 Zeichen lang sein kann. **12** entspräche **dbMemo** für einen **Memo-String**, **4** **dbLong** für einen **Long-Wert**.

```
Sub ObjectProperties(obj As Object)
    Dim daoPrp As DAO.Property

    Debug.Print TypeName(obj)
    On Error Resume Next
    For Each daoPrp In obj.Properties
        Debug.Print , daoPrp.Name & _
            " = " & daoPrp.Value & _
            " (" & daoPrp.Type & )"
    Next daoPrp
End Sub
```

Listing 4: Allgemeine Hilfs-Routine zur Ausgabe aller Properties eines DAO-Objekts

Sie können die Routine auch aus dem VBA-Direktfenster aufrufen, zum Beispiel:

Datenbanken über Verknüpfung starten

In der Regel starten Sie Ihre Datenbanken, indem Sie auf ACCDB-Dateien doppelklicken oder, indem Sie Access öffnen und eine Datei über das Backstage-Menü laden. Andere Anwender der Datenbank finden das vielleicht weniger komfortabel und wünschen sich eine Desktop-Verknüpfung, über die sich die Anwendung schneller starten lässt. Über dieses Thema gibt es mehr zu sagen, als man auf den ersten Blick denkt ...

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1510_KommandoZeile.accdb**

Verknüpfungen anlegen
Der Zweck einer **Windows-Shell-Verknüpfung** – im Folgenden auch einfach **Link** genannt – ist die Entbindung eines Dokuments oder einer Anwendung von deren Speicherverzeichnis. Einmal angelegt kann sie an beliebigem Ort untergebracht oder verschoben werden. Zusätzlich können in der Verknüpfung noch Startargumente oder ein Icon der Wahl angegeben werden. Außerdem lässt sich in den Kompatibilitätsoptionen noch festlegen, ob die Anwendung **als Administrator** oder im Modus einer früheren Windows-Version gestartet werden soll.

Vollziehen wir das einmal mit der Beispieldatenbank zum Beitrag. Rechtsklicken Sie auf Ihren Desktop und wählen aus dem Kontextmenü den Eintrag **Neu | Verknüpfung**. Damit öffnet sich der Assistent zum Anlegen eines neuen Links. Über **Durchsuchen...** manövrieren Sie zum Verzeichnis, in dem Sie die Beispieldatei **Kommandozeile.accdb** abgelegt hatten und wählen diese aus. Mit der anschließenden Angabe der Verknüpfungsbezeichnung und Klicken auf **Fertigstellen** hat sich die Sache dann schon.

Weitere Parameter können erst eingestellt werden, nachdem die angelegte Verknüpfung über Rechtsklick und **Eigenschaften** in den Bearbeitungsmodus versetzt wurde, wie in Bild 1.

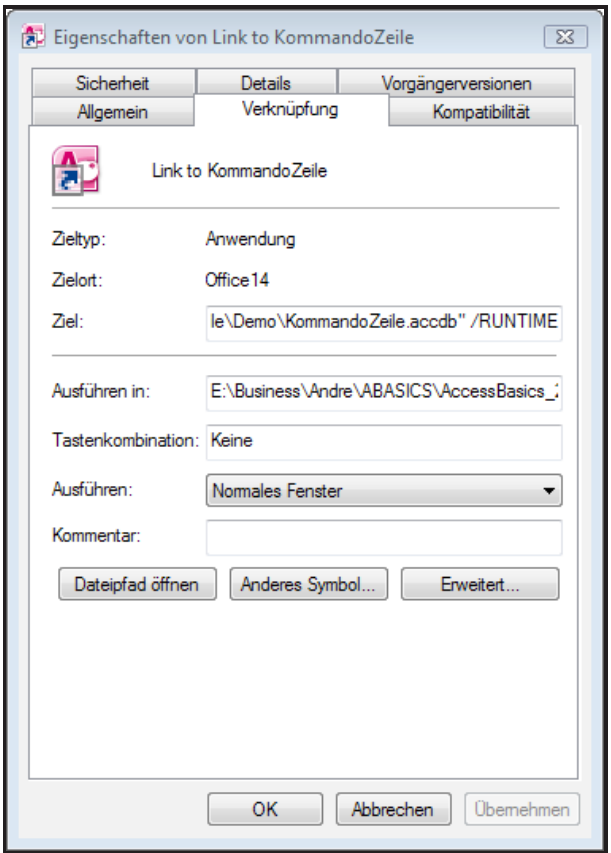


Bild 1: Verknüpfungseigenschaften für einen Link zur Datenbank Kommandozeile.accdb mit Startparameter

Windows verwendet automatisch das Access-Symbol, weil dies die zugeordnete Anwendung für **accdb-Dateien** ist. Sie können nun über **Anderes Symbol** ein alternatives Icon aus Ihrem Fundus auswählen. Das Arbeitsverzeichnis (**Ausführen in**) spielt weiter keine Rolle. Unter Umständen ändern Sie aber den Startmodus für das Access-Fenster, indem Sie unter **Ausführen** etwa den Eintrag **Maximiert** einstellen. Eine globale **Tastenkombination** hingegen werden Sie selten für eine Access-Datenbank hinterlegen.

In der Abbildung wurde im Textfeld **Ziel** an den Pfad zur **accdb-Datei** noch der **Startparameter /RUNTIME** gehängt. Auf die Bedeutung dieses und der weiteren möglichen Startparameter kommen wir noch ausführlich zu sprechen. Diese **Kommandozeilenoption** soll Access eigentlich anweisen, die Datenbank so auszuführen, als würde sie unter der **Access-Runtime** geöffnet. Starten Sie sie über den Link, so ist jedoch kein Unterschied zum normalen Start zu erkennen. Grund dafür ist, dass Windows der **msaccess.exe** über die Verknüpfung zwar den Pfad der Datenbankda-

tei übergibt, nicht jedoch weitere Parameter. Imgrunde wäre das zwar möglich, doch dafür wäre die Registry abweichend von den Einstellungen der Office-Installation zu modifizieren. Diese Option schließen wir aus.

Das Ganze klappt erst dann, wenn Sie in der Verknüpfung nicht nur den **accdb-Pfad** eingeben, sondern auch noch den Pfad zur **msaccess.exe**:

```
Ziel:
"c:\Program Files\Microsoft Office\ _
   Office14\msaccess.exe" _
  "c:\xyz\kommandozeile.accdb" _
  /RUNTIME
```

Hier ist also etwas Schreibarbeit angesagt. Und Sie müssen zunächst ermitteln, wo sich die **msaccess.exe** befindet, was sich von System zu System und je nach Office-Version unterscheiden kann. Die Anführungszeichen sollten Sie grundsätzlich um die Pfade legen, damit sich Windows nicht an etwaigen Leerzeichen verschluckt.

Dieser Aufwand ist, wie erwähnt, nur dann notwendig, wenn Sie die Datenbank mit einem Kommandoparameter starten möchten.

Access-Kommandozeilenparameter

Es gibt eine ganze Reihe von möglichen Startparametern, die man Access verabreichen kann, von denen in der freien Wildbahn einige mehr, andere wiederum weniger Bedeutung haben. Die meisten sind von Microsoft in der Access-Hilfe dokumentiert worden. Einige allerdings versteckte man und sie sind das Forschungsergebnis von Tüftlern. Außerdem gibt es Parameter, die unter früheren Access-Versionen noch funktionierten und Sinn machten, mittlerweile zwar noch entgegengenommen, jedoch von Access ignoriert werden.

Wir besprechen hier die wichtigsten Parameter. Ersetzen Sie einfach den Ausdruck **/Runtime** des vorigen Beispiels durch die folgenden Parameter. Sie können auch mehrere Startparameter kombinieren. Soll die Datenbank etwa exklusiv und im Runtime-Modus gestartet werden, so geben Sie an:

```
"<ACCDB-Pfad>" /EXCL /RUNTIME
```

Trennen Sie die einzelnen Parameter jeweils durch ein Leerzeichen. Manche Anweisungen benötigen noch weitere Angaben, also Parameter für den Startparameter. Hängen Sie auch diese jeweils durch Leerzeichen getrennt an, etwa:

```
/EXCL /X MeinStartmakro /CMD Irgendwas
```

Startparameter /WRKGRP

In früheren Zeiten wurden Access-Datenbanken nicht selten über eine **Arbeitsgruppendatei** (**MDW**) abgesichert. Über diese in Kombination mit Benutzername und Passwort ließen sich Berechtigungen und der Zugang zur Datenbank selbst einstellen.

Der Support für diese Methode wurde, warum auch immer, seit **Access 2007** eingestellt. Allerdings funktioniert sie immer noch, auch wenn Access über seine Oberfläche keine Werkzeuge mehr bereitstellt, um die Arbeitsgruppendateien und Benutzerberechtigungen zu verwalten.

Haben Sie eine solche durch Arbeitsgruppendatei abgesicherte Datenbank vor sich, so kann der Pfad zur notwendigen Datei über den Kommandozeilenschalter **WRKGRP** in der Verknüpfung angegeben werden:

```
"<msaccess.exe>" "<accdb>" _
    /WRKGRP "c:\xyz\sicherung.mdw"
```

Damit wird Access angewiesen, die angegebene Datei als Systemdatenbank für die **Access-Engine** zu benutzen, statt der **Standard-MDW**, die normalerweise in Gestalt der **system.mdw** zum Einsatz kommt und sich irgendwo in den Untiefen der Windows-Benutzerverzeichnisse befindet.

Startparameter /USER

Diesen Parameter benötigen Sie nur, wenn auch die **WRKGRP**-Option angegeben wurde. In der Arbeitsgruppendatei finden sich die angelegten Benutzer, welche Zugriff auf die Datenbank erhalten können. Mit **USER** können Sie direkt einen Benutzer namentlich festlegen, der Zugang zur Datenbank erhalten soll:

```
"<msaccess.exe>" "<accdb>" _
    /WRKGRP "c:\xyz\sicherung.mdw"
    /USER Minhorst
```

Startparameter /PWD

Die Absicherung einer Datenbank über eine Arbeitsgruppendatei führt zu einem Anmeldedialog, der Benutzer und Passwort abfragt. Haben Sie per **USER** den Benutzer schon als Startparameter festgelegt, so

erscheint er voreingestellt im Anmeldedialog. Fehlt noch das Passwort. Und das können Sie ebenso in der Kommandozeile über **PWD** angeben:

```
"<msaccess.exe>" "<accdb>" _
    /WRKGRP "c:\xyz\sicherung.mdw"
    /USER Minhorst
    /PWD geheim0789
```

In dieser Kombination kommt es **nicht** zur Anzeige des Anmeldedialogs. Die Datenbank öffnet sich unmittelbar. Übrigens können Sie den übergebenen Benutzer in der Datenbank mit einer Zeile VBA ermitteln:

```
Msgbox CurrentUser
> "Minhorst"
```

Startparameter /EXCL

Das weist Access an, die Datenbank im exklusiven Zugriffsmodus zu öffnen. Das lässt sich auch über den Datei-Öffnen-Dialog aus Access heraus bewerkstelligen, wenn Sie im Auswahldialog das Kombinationsfeld **Öffnen** auf **Exklusiv öffnen** stellen. In diesem Modus haben Sie allein die Gewalt über die Datenbank. Weitere Anwender, die sie zu öffnen versuchen, werden daran über die Meldung **Datenbank wird verwendet** gehindert. Sinn macht diese Einstellung in der Regel nur, wenn die Datenbank sich im Netzwerk befindet und mehrere Benutzer auf sie zugreifen können. Allerdings können auch Sie dann keine zweite Instanz von Access mit dieser Datenbank auf Ihrem Rechner öffnen.

Startparameter /RO

Dieser ist mit dem vorigen verwandt und steht für **Read Only**. Die Datenbank wird im schreibgeschützten Modus geöffnet, was dem Laden einer Datenbank aus Access heraus mit der Dateiauswahloption **Exklusiv schreibgeschützt öffnen** entspricht.

Zwar lassen sich in diesem Modus Formulare öffnen und Daten bearbeiten, doch die Objekte der Datenbank verweigern sich einer Modifizierung und Speicherung, also einer Entwicklung. Dafür jedoch können auch weitere Benutzer auf sie zugreifen. Liegt die Datenbank in einem Netzwerk, so stellen Sie damit immerhin sicher, dass niemand in ihr Unfug anstellt.

Startparameter /SAFE

Hier startet die Datenbank im **Abgesicherten Modus**. Access verhält sich dann ähnlich, wie Windows im abgesicherten Modus. Zahlreiche Einschränkungen kommen zum Tragen. **Add-Ins** sind deaktiviert, das Speichern ist limitiert, und vieles mehr. Informieren Sie sich bitte in der Access-Hilfe über die lange Liste der Einschränkungen.

Man wird diesem Modus höchstens testweise während der Entwicklung benötigen. Im produktiven Einsatz ist er witzlos.

Startparameter /Repair

Wenn Sie diesen Parameter setzen, so wird die angegebene Datenbank nur repariert. Access blitzt kurz auf und schließt wieder. Danach befindet sich die Datenbank in repariertem und wahrscheinlich komprimiertem Zustand (siehe folgenden Parameter).

Der Vorgang entspricht der Funktion **Datenbank komprimieren und reparieren**, die in Access über den Ribbon aufgerufen werden kann.

Startparameter /Compact

In alten Access-Versionen gab es noch einen Unterschied zwischen dem Reparieren und dem Komprimieren einer Datenbank. **Compact** komprimierte sie, ohne zu reparieren, und umgekehrt. Seit Generation gibt es diesen Unterschied in der **Datenbank-Engine** nicht mehr. Auch, wenn es nicht ausdrücklich dokumentiert ist, kann man davon ausgehen, dass die beiden Schalter **/Compact** und **/Repair** genau denselben Vorgang auslösen.

Startparameter /NEWDB

Dieser Parameter ist von Microsoft nicht dokumentiert worden. Sie brauchen in der Verknüpfung keine Datenbankdatei anzugeben, sondern nur die **msaccess.exe**, gefolgt vom Parameter. Tun Sie es dennoch, so öffnet sich die Datei normal und ignoriert den Parameter.

Mit **NEWDB** startet Access und zeigt unmittelbar einen **Datei-Speichern-Unter-Dialog** an, der Sie auffordert, den Namen und Ort einer neu zu erstellenden **ACCDB** zu bestimmen. Über eine solche Verknüpfung ersparen Sie sich einen Arbeitsschritt. Allerdings können Sie ja auch über das Kontextmenü des Explorers und den Neu-Eintrag eine Datenbank anlegen. Der Nutzen des Startparameters erschließt sich also nicht wirklich.

Startparameter /Convert

Haben Sie eine ältere **MDB-Datei** vorliegen, die Sie in das aktuelle **Access 2007-Format** konvertie-

ren möchten, so bemühen Sie für diesen Vorgang üblicherweise das **Backstage**-Menü **Datei | Speichern und Veröffentlichen | Datenbank speichern als... Access-Datenbank (*.accdb)**, nachdem Sie die **MDB** geladen hatten.

Convert macht das auch über die Kommandozeile möglich. Setzen Sie den Parameter ohne weitere Angabe ein, so startet Access mit einem Datei-Speichern-Dialog, der Namen und Ort der konvertierten Datenbank abfragt. Direkt nach dem Konvertieren schließt sich Access wieder.

Setzen Sie den Dateipfad hingegen direkt hinter dem Kommando ab, so läuft der Vorgang ohne weitere Interaktion ab:

```
"<msaccess.exe>" "c:\vwx\test.mdb" _
    /Convert "c:\xyz\test.accdb"
```

Startparameter /RUNTIME

Wer bereits einmal Datenbanken auf einer installierten **Runtime-Version** von Access laufen ließ, der weiß, dass sich die Gestalt der Oberfläche damit deutlich ändert. Sämtliche Elemente, die für die Entwicklung von Bedeutung sind, fehlen in der Oberfläche. Das schließt das Standard-Menüband ein.

Die kostenlose **Runtime-Version** kommt in der Regel dann zum Einsatz, wenn auf Firmenrechnern keine Vollversion von Access vorhanden ist. Aber auch Sicherheitserwägungen könnten ein Kriterium sein, denn in der **Runtime-Version** ist Ihre Datenbank vor jeglichen Eingriffen abseits der Daten gefeit.

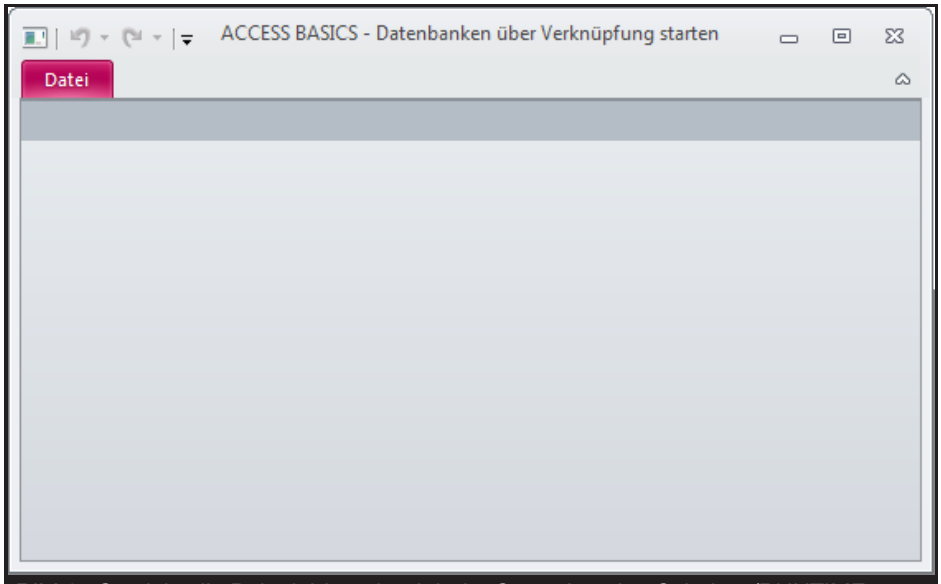


Bild 2: So sieht die Beispieldatenbank beim Start über den Schalter /RUNTIME aus

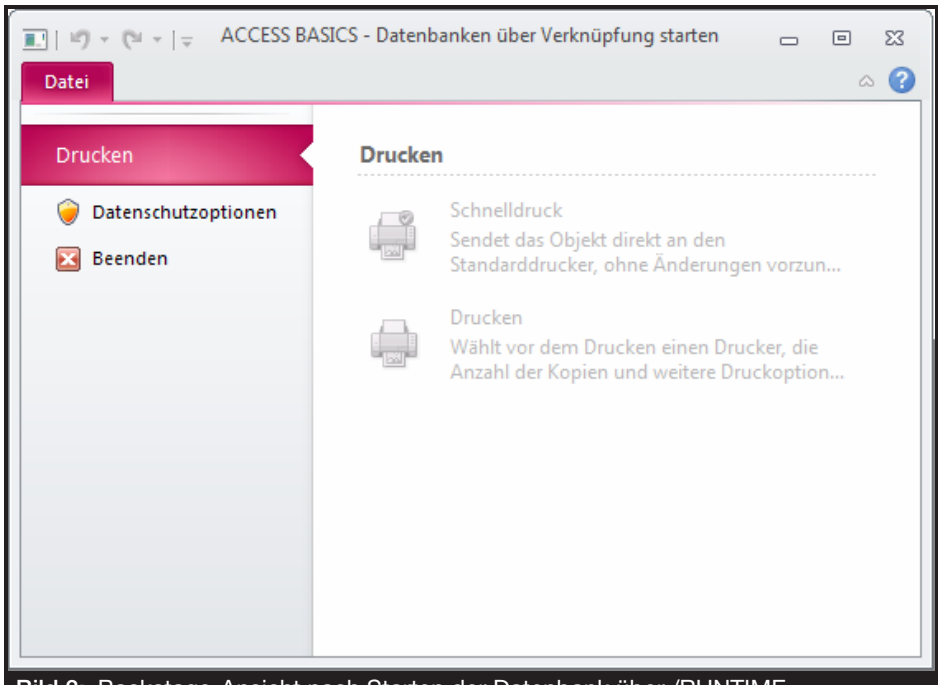


Bild 3: Backstage-Ansicht nach Starten der Datenbank über /RUNTIME

Der Schalter **RUNTIME** gestattet die Simulation diese Modus. Wie die Beispieldatenbank dann aussieht, zeigt Bild 2. Nur das **Datei**-Menü ist vorhanden, der **Ribbon** fehlt. Ein Klick auf dieses führt zum nun extrem ausgedünnten **Backstage**, wie in Bild 3. Auch die Schnellzugriffsleiste lässt sich über den Pfeil oben nicht modifizieren, weil alle Einträge in der Menüliste deaktiviert sind.

Verwenden Sie diesen Startmodus, um bei Entwicklung die Kompatibilität Ihrer Datenbank mit der **Runtime-Version** zu testen, falls Sie unter

DoCmd- und RunCommand-Anweisungen

Aktionen, die Sie über die Oberfläche von Access durchführen, lassen sich fast alle auch über VBA anstoßen. Für die hinter Menüs, Kontextmenüs und Ribbon liegenden Befehle gibt es Pendants, welche sich in zahlreichen Methoden vor allem des DoCmd-Objekts und der RunCommand-Anweisung verstecken. Über diese VBA-Funktionen lassen sich komplexe Aufgaben mit wenig Aufwand erledigen. Das ist Grund genug für eine ausführlichere Übersicht.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1511_DocmdAnweisung.accdb**

Unterschiede zwischen DoCmd und RunCommand
Zunächst handelt es sich bei **RunCommand** um eine direkte Methode der **Access.Application**-Instanz. **DoCmd** hingegen ist ein Eigenschaftsobjekt von **Application**, das generell verfügbar ist:

```
Application.RunCommand acCmd-Konstante
Application.DoCmd.<Methodenausdruck>
```

RunCommand nimmt grundsätzlich nur einen einzigen Parameter entgegen, der aus der Enumeration **acCommand** stammen muss. Das sind Konstanten, die bestimmten Aktionen entsprechen. Wollen Sie etwa ein Fenster, sei es ein Formular oder ein Bericht, maximieren, so lautet die entsprechende Konstante **acCmdDocMaximize**:

```
RunCommand acCmdDocMaximize
```

Die Zahl dieser Konstanten in den aktuellen Access-Versionen liegt bei etwa 600.

Beim **DoCmd**-Objekt sind es deutlich weniger Methoden, circa 70 nämlich, die dafür aber eine variable Zahl an Parametern aufweisen, mit denen die Aktionen gesteuert werden können. Ein Beispiel für eine Anweisung ohne Parameter wäre

```
DoCmd.Maximize
```

Auch dieser Befehl maximiert ein Formular- oder Berichtsfenster, welches aktuell den Fokus besitzt. Eine Methode mit mehreren Parametern wäre

```
DoCmd.Close acForm, _
    "frmAdressen", acSaveYes
```

Dies schließt das Formular **frmAdressen** und und speichert seinen Zustand, also vor allem seine Abmessungen, explizit. Auch ein im Entwurfsmodus befindliches Formular wird damit geschlossen und die Änderungen abgespeichert. Ohne den optionalen Parameter **acSaveYes** würden Sie mit dem Bestätigungsdialog zum Speichern konfrontiert.

Sowohl **RunCommand**-, wie **DoCmd**-Anweisungen spiegeln das wieder, was Sie auch über Schaltflächen der Oberfläche erreichen. Es handelt sich deshalb auch immer um **Sub**-Prozeduren, die keinen Rückgabewert haben. Generell könnte man formulieren, dass **RunCommand** Anweisungen entspricht, die keine weitere Interaktion erfordern, während die **DoCmd**-Methoden eher mit den Access-Assistenten zu vergleichen wären, die über mehrere Schritte Angaben abfragen, bevor die eigentliche Aktion ausgeführt wird. Diese vom Assistenten gesammelten Informationen entsprechen dann den Parametern, die die **DoCmd**-Methode erwartet. Ein Beispiel hierfür wäre der Text-**Export-Assistent**, der sein Spiegelbild in der Anweisung **DoCmd.TransferText** hat.

Während die **DoCmd**-Anweisungen in der Hilfe von Access alle hinreichend dokumentiert sind, sind Sie bei den **RunCommand**-Konstanten auf den sprechenden Namen oder Experimente angewiesen.

Microsoft selbst hat über alle Versionen von Access hinweg keine Übersicht über die Bedeutung der Konstanten veröffentlicht. Im Zweifel finden Sie Im Internet auf Seiten Dritter weitere Erläuterungen.

Im Folgenden geht es in erster Linie um die Möglichkeiten, die **DoCmd** eröffnet. Nicht alle Methoden sind aufgeführt, da viele nicht wirklich relevant sind oder nur im Umfeld von Access-Projekten (SQL-Server) Be-

deutung haben. Die Methoden sind thematisch zusammengefasst.

Objekte öffnen

Dass sich ein Formular über die **DoCmd**-Anweisung OpenForm laden lässt, ist Ihnen sicher schon untergekommen. Aber für alle anderen Access-Objekte gibt es ebenfalls äquivalente Methoden, deren wesentliche Vertreter Sie in Listing 1 finden. Interessant sind dabei vor allem die Parameter, welche das Laden steuern.

```
Sub TestOpenObjects()
    DoCmd.OpenModule "mdlDoCommands", "TestOpenObjects"
        Msg "Modul und Prozedur geöffnet"
    DoCmd.OpenQuery "qryAdressenG", acViewNormal, acReadOnly
        Msg "Abfrage geöffnet"
    DoCmd.OpenReport "rptAdressen", acViewPreview, "Vorname Like 'B*'", acDialog
        Msg "Bericht geöffnet"
    DoCmd.OpenForm "frmAdressen", , , , acFormReadOnly, acDialog, "Test1"
        Msg "Formular war geöffnet"
    DoCmd.OpenTable "tblLaender", acViewNormal, acReadOnly
        Msg "Tabelle geöffnet"
End Sub
```

Listing 1: Die relevanten Öffnen-Methoden des DoCmd-Objekts

OpenModule ist für den Laufzeitbetrieb einer Datenbank überflüssig. Die Anweisung öffnet ein VBA-Modul, das Sie namentlich angeben. Zusätzlich kann mit weiterem Parameter auch eine Prozedur des Moduls spezifiziert werden, auf die beim Öffnen der Cursor positioniert wird. Die Beispieldatenbanken von **Access Basics** verwenden den Befehl etwa hin und wieder im Intro-Formular, um beim Start sogleich ein Modul und damit den VBA-Editor zu öffnen, welches von besonderem Interesse sein kann.

OpenQuery öffnet eine Abfrage. Über den **View-Parameter** steuern Sie dabei, in welcher Gestalt die Abfrage erscheinen soll. Möchten Sie die normale Datenblattansicht, so setzen Sie **acViewNormal** ein. Für den Entwurfsmodus steht **acViewDesign**. Weniger bekannt ist, dass **acViewPreview** die Abfrage in die Seitenansicht für den Druck bringt, womit etwa kontrolliert werden kann, ob das Ergebnis auf eine Seite passt. Diese **View**-Parameter teilt sich die Anweisung mit den anderen Öffnungsmethoden. Nur sind bei Abfragen die Konstanten **acViewReport** (Berichtsansicht) und **acViewLayout** (Layout-Ansicht) nicht erlaubt und führen zum Fehler.

Handelt es sich bei der Abfrage um eine Aktionsabfrage, so hat der View-Parameter keinerlei Bedeutung, denn ein Fenster öffnet sich hier nicht.

Interessant ist der letzte optionale Parameter **DataMode**, mit dem Sie bestimmen können, ob das Abfrageergebnis editierbar sein soll. **acReadOnly** führt nämlich zu einer schreibgeschützten Ansicht, während ein **acEdit** (Vorgabe) die Datensätze bearbeitbar macht, falls die Abfrage selbst das unterstützt.

Die gleichen Parameter, wie für **OpenQuery**, finden Sie auch bei der Anweisung **OpenTable**, mit der Sie eine Tabelle öffnen. Auch hier können Sie über **DataMode** entscheiden, ob die Ansicht schreibgeschützt oder editierbar sein soll.

Umfangreich sind die Parameter, die die zum Laden eines Formulars dienende Anweisung OpenForm erwartet. Bis auf den Namen des Formulars sind diese allerdings alle optional und deshalb auch nicht so bekannt.

Die Ansichtsart steuern Sie wie bei **OpenQuery** und **OpenTable**, nur, dass nun auch **acViewLayout** gestattet ist. Standardmäßig ist **acViewNormal** voreingestellt. Als zweiten Parameter, im Listing nicht verwendet, können Sie den Namen einer Filterabfrage angeben, die die angezeigte Datensatzmenge einschränkt. Dieses Verfahren ist weitgehend obsolet. Verwenden Sie besser den dritten Parameter für die **WhereCondition**. Hier können Sie einen SQL-Ausdruck einsetzen, der als Filter dient, etwa:

```
..., "Nachname LIKE 'H*`'
```

Das führte dazu, dass nur Datensätze im Formular erschienen, die im Nachnamenfeld mit **H** beginnen. Die Verwendung dieses Parameters entspricht damit dem Setzen der Formulareigenschaft **Filter** inklusive dem Aktivieren von **FilterOn**.

Auch beim Formular kann das Editieren ausgeschaltet werden, indem Sie die Konstante **acFormReadOnly** für **DataMode** einsetzen. Und schließlich lässt sich über **WindowMode** ange-

ben, wie das Formular geladen wird. Der Standard ist **acWindowNormal**. Nehmen Sie **acIcon**, so öffnet es sich minimiert. Mit **acHidden** bleibt es sogar unsichtbar. Schließen können Sie es dann nur noch über VBA. Am interessantesten aber ist der über **acDialog** festgelegte Dialog-Modus. In diesem zeigt sich das Formular auf den ersten Blick in normalem Gewand. Tatsächlich aber ist es **modal**, was bedeutet, dass der Rest der Access-Anwendung gesperrt ist. Nur das geöffnete Dialogformular lässt sich bearbeiten. Zudem stoppt auch VBA nach dem Ausführen der aufrufenden Code-Zeile. Erst nach dem Schließen fährt es mit den weiteren Zeilen der Prozedur fort. Man kann sich dies etwa zunutze machen, um Eingaben im Formular auszulesen. Denn auch dann, wenn das Formular über die **Visible**-Eigenschaft im Formular-Code unsichtbar gemacht wird, übergibt VBA die Kontrolle an die aufrufende Prozedur. Nun könnte diese etwa Felder des Formulars auslesen, da es inzwischen zwar noch existiert, aber nicht mehr sichtbar ist. Das bedeutet allerdings auch, dass die Routine selbst das Formular über eine Close-Anweisung schließt, denn sonst irrt es auf Ewigkeiten im Speicher umher.

Als letzten Parameter finden Sie für **OpenArgs** im Listing den String **"TEST1"**. Hier können Sie beliebige Daten, also auch Zahlenvariablen oder Objekte einsetzen. Das hat auf die Gestalt des geladenen Formulars keinen Einfluss. Über dessen Eigenschaft **OpenArgs** kann aber an beliebiger Stelle im Formular-Code auf diese Angabe Bezug genommen werden. So etwa im **Beim Laden**-Ereignis des Formulars:

```
Sub Form_Load()
    Msgbox Me.OpenArgs
End Sub
```

Hier würde **TEST1** in einem Meldungsfenster erscheinen, sobald das Formular gestartet wurde.

Die Parameter für die Anweisung **OpenReport** zum Laden eines Berichts gleichen jenen von **OpenForm**. Hier ist zusätzlich die Ansichtsart **acViewReport** (Berichtsansicht) möglich. Ansonsten lassen sich hier ebenfalls Filter setzen, der **WindowMode** einstellen und **OpenArgs** übergeben. Es ist jedoch eine Besonderheit zu beachten: **acViewNormal** öffnet den Bericht nicht etwa in der normalen Ansicht, sondern schickt ihn zum Drucker! Wenn Sie also bei **OpenReport** lediglich den Namen des Berichts angeben und auf weitere Parameter verzichten, so wundern Sie sich nicht, dass scheinbar nichts passiert! Erst im Taskbar von Windows stellen Sie dann fest, dass ein Drucker in Aktion gesetzt wurde. Für die normale Ansicht unter Access setzten Sie daher **acViewPreview** ein.

Über VBA mit **DoCmd** geöffnete Objekte schließen Sie mit einer weiteren **DoCmd**-Anweisung:

```
DoCmd.Close acForm, "frmAdressen"
DoCmd.Close acModule, "mdlDoCommands"
DoCmd.Close acQuery, "qryAdressenG"
```

Formular- und Datensatzsteuerung

Ist ein Formular erst über **OpenForm** geöffnet, dann können Sie es über zahlreiche **DoCmd**-Methoden steu-

```
Sub TestFormObject()
    DoCmd.OpenForm "frmAdressen"
    DoEvents
    DoCmd.SelectObject acForm, "frmAdressen"
    DoCmd.ApplyFilter , "Nachname LIKE 'Wag*'"
'    DoCmd.SetFilter , "Nachname LIKE 'Wag*'","
    DoCmd.GoToRecord acDataForm, "frmAdressen", acLast
    DoCmd.FindRecord "Gudrun", acEntire, False, acSearchAll, False, acAll, True
    DoCmd.SetOrderBy "Vorname", "sfmKinder"
    DoCmd.FindRecord "Gisela", acEntire, False, acSearchAll, False, acAll, True
    DoCmd.FindNext
    DoCmd.SetProperty "txtNachname", acPropertyForeColor, RGB(64, 64, 255)
    DoCmd.Requery "txtVorname"
    DoCmd.ShowAllRecords
    DoCmd.Echo False
    DoCmd.GoToRecord acDataForm, "frmAdressen", acLast
    DoCmd.GoToRecord acDataForm, "frmAdressen", acFirst
    DoCmd.Echo True
    DoCmd.SearchForRecord acDataForm, "frmAdressen", acFirst, "Nachname='Hückmann'"
    If MsgBox("Formular schließen?", vbYesNo) = vbYes Then
        DoCmd.Save acForm, "frmAdressen"
        DoCmd.Close acForm, "frmAdressen", acSaveYes
    End If
End Sub
```

Listing 2: DoCmd-Anweisungen für verschiedene datensatzbezogene Aktionen

ern. Oder genauer gesagt: dessen Datensätze steuern.

Listing 2 zeigt eine gekürzte Version der Prozedur, welche Sie in der Beispieldatenbank finden. In der Originalversion ist nach jeder **DoCmd**-Zeile eine **Msgbox** eingebaut, die ausgibt, was gerade vonstattenging. Ohne diese Meldungsfenster würde der Code der Prozedur ja in Sekundenbruchteilen durchlaufen sein, ohne dass Sie die Auswirkungen der Anweisungen begutachten könnten.

SelectObject setzte den Fokus auf ein Access-Objekt und/oder markiert dieses. In unserem Fall handelt es sich um das Formular **frmAdressen**. Sinn der Angelegenheit ist es, das entsprechende Objekt zu aktivieren. Denn die folgenden Anweisungen wirken immer auf das geöffnete aktive Objekt. Sind etwa mehrere Formulare gleichzeitig geöffnet, so steuern Sie mit diesem Befehl, welches den Fokus bekommen soll.

Nach dem Aktivieren kann per **ApplyFilter** eine Filterbedingung auf die angezeigten Datensätze angewandt werden. Das entspricht dem Vorgang, wenn Sie manuell über das Menüband oder über einen Spaltenkopf in Datenblattansicht einen Filter bestimmen. Hier werden nur jene Datensätze angezeigt, die im Nachnamefeld mit **Wag** beginnen. In der Navigationsleiste wird das deutlich, weil sich hier die Zahl der Datensätze verringert hat und vom Ausdruck **(Gefiltert)** begleitet wird. Statt **ApplyFilter** können Sie auch die Anweisung **SetFilter** mit gleicher Syntax benutzen. Der Unterschied besteht darin, dass beim Schließen des Formulars und gleichzeitigem Speichern der Filterausdruck dann in der Eigenschaft **Filter** verewigt bleibt.

Einen gesetzten Filter entfernen Sie einfach wieder mit einem Aufruf von **DoCmd.ShowAllRecords**.

GotoRecord erlaubt das Springen zu einem bestimmten Datensatz. Hier kann dezidiert angegeben werden, welches Formular zu steuern ist. Für den letzten Parameter gibt es verschiedene Steuerkonstanten. **acLast** springt zum letzten Datensatz, **acFirst** zum ersten, und so weiter. Interessanter ist da schon **acGoto**. In diesem Fall muss noch ein weiterer Parameter angehängt werden, der den Offset zum aktuellen Datensatz angibt. Möchten Sie etwa zum genau fünften Datensatz des Formulars springen, so verwenden Sie zwei Zeilen:

```
DoCmd.GoToRecord  , , acFirst
DoCmd.GoToRecord  , , acGoto, 4
```

GotoRecord wirkt nicht nur auf Formulare. Auch Tabellen oder Abfragen in Datenblattansicht, wie auch Berichte lassen sich damit steuern.

FindRecord spiegelt den **Suchen-Dialog** von Access wieder und weist deshalb genauso viele Parameter auf, wie jener an Einstellmöglichkeiten anbietet. Bild 1 zeigt die Analogie zwischen Dialog und möglichen Parametern. Diese Funktion ist durchaus mächtig. Wollten Sie sie durch eine andere VBA-Programmierung ersetzen, so käme dabei eine ziemlich umfangreiche Routine heraus, die allerlei Operationen rund um das Formular-Recordset enthielte. Vor allem aber ist sie unabhängig vom Objekt, das den Fokus besitzt. Die eine gleiche **DoCmd**-Zeile lässt sich auf verschiedene Formulare, Berichte oder Datenblätter anwenden.

Möchten Sie weitere Vorkommen des Suchausdrucks finden, so muss die ganze Zeile nicht erneut aufgerufen werden, Stattdessen setzen Sie einfach ein folgendes **DoCmd.FindNext** ein.

Eine Alternative zu **FIndRecord** ist das SQL-nähere **SearchForRecord**.

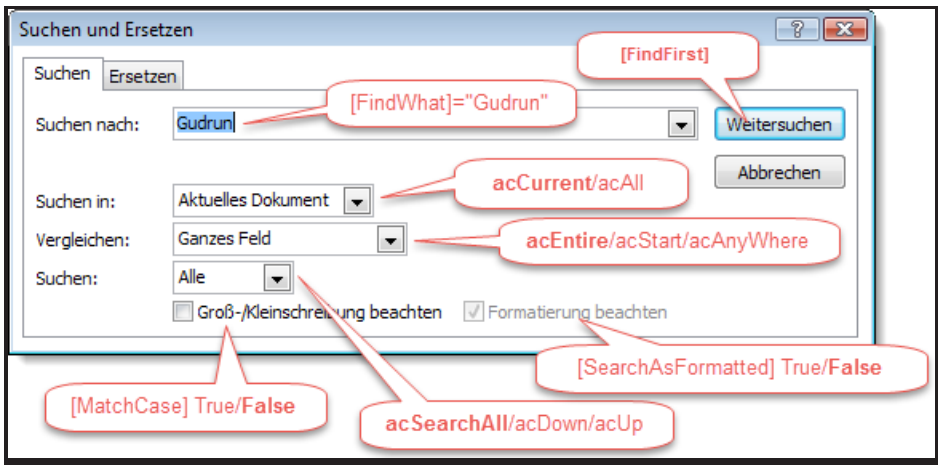


Bild 1: Der Suchen-Dialog mit der FindRecord-Analogie und deren den Parametern

Als Parameter geben Sie den Objekttyp und den Objektnamen an. Dann noch, ob nach dem ersten Vorkommen (**acFirst**) oder nach einem weiteren (**acNext**) oder früheren (**acPrevious**) gesucht werden soll. Daran schließt sich der Suchausdruck an, der der Syntax für eine **WhereCondition** (Filter) entspricht. Im Gegensatz zu **FindRecord** sind Sie hier also nicht auf einen einfachen Feldwert beschränkt, sondern können mehrere Kriterien festlegen und zusammensetzen!

Ähnlich, wie für das Filtern, gibt es auch eine **DoCmd**-Anweisung für das Sortieren von Datensätzen. **SetOrderBy** erwartet als Parameter den Namen des Felds, nach dem sortiert werden soll, und optional den Namen des Formulars oder Datenblatts – selbst Unterformulare sind möglich! Ohne diese Angabe wird das aktuelle Hauptobjekt sortiert. Sie können die Sortierreihenfolge festlegen, indem Sie an das gewünschte Feld noch den Ausdruck **ASC** (Aufsteigend) oder **DESC** (Absteigend) anhängen:

```
DoCmd.SetOrderBy "Vorname DESC"
```

Damit entspricht die Anweisung dem Setzen einer Kombination der Formulareigenschaften **OrderBy** und **OrderByOn**.

Die Formatierung von Feldern eines Datenblatts oder auch Formulars nehmen Sie in der Regel über den entsprechenden Abschnitt des Ribbon vor. Doch selbst dies lässt sich in VBA über den **DoCmd**-Befehl **SetProperty** erledigen. Dazu geben Sie den Namen des Steuerelements,

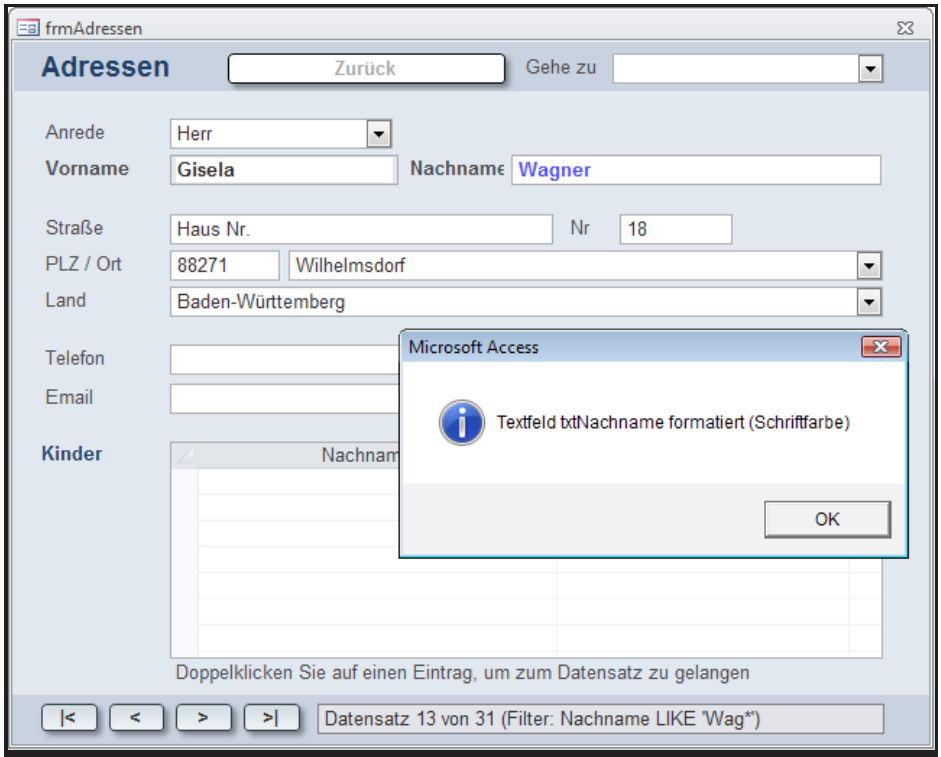


Bild 2: Das Adressformular der Beispieldatenbank nach dem Setzen einer Formatierung auf das Feld Nachname über die Anweisung DoCmd.SetProperty

welches formatiert werden soll, als Parameter an. Was formatiert werden soll, legen Sie mit einer der **acProperty**-Konstanten fest. Die Schriftfarbe etwa entspricht der Konstanten **acPropertyForeColor**. Weitere Angaben zu den Konstanten finden Sie in der Access-Hilfe zu **SetProperty**. Der letzte Parameter enthält schließlich den Wert für die Eigenschaft. Für Farben wäre dies ein **RGB**-Wert, für JA/NEIN-Eigenschaften entweder **-1** oder **0**. Im Beispiel wird die Schrift des Felds **txtNachname** blau eingefärbt (Bild 2).

Manchmal kommt es vor, dass sich die einem Formular zugrundeliegende Datenherkunft durch **DAO**-Operationen verändert hat. Nicht immer geben die Textfelder diese Änderung gleich wieder. Absichern können Sie das aber, indem Sie die Methode **DoCmd.Requery** aufrufen. Sie können ihr den Namen eines Felds als Parameter mitgeben. Nur dieses wird dann aktualisiert. Ohne Angabe werden alle Felder auf den neuesten Stand gebracht.

Die **DoCmd**-Operationen verhalten sich genauso, wie die analogen manuell angestoßenen. Springen Sie per **GotoRecord** etwa zum ersten Datensatz des Formulars und dann zu einem bestimmten Offset, so vollzieht sich das auch unmittelbar in der Anzeige. Das sieht nicht schön aus und kostet zudem Performance. Verhindern können Sie das mit der **DoCmd**-Anweisung **Echo**. Wird ihr als Parameter ein **False** übergeben, so friert die Oberfläche von Access ein. Dennoch werden alle VBA-Operationen weiter ausgeführt. Mit einem **True** wird der Zustand aufgehoben.

Navigation in und zwischen Formularen

Wie gelangt man in einem Formular intuitiv zu einem bestimmten Datensatz? Wie kann man zum zuletzt angezeigten Datensatz zurückspringen? Auf welche Weise lässt sich zu einem Detaildatensatz ohne Umwege ein Hauptformular öffnen? Antworten auf diese Fragen versucht dieser Beitrag mit einigen trickreichen Routinen zu geben, die teilweise die Navigation auf Webseiten nachbilden.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1511_FormNavigation.accdb**

Datenmodell

Die Beispielanwendung verwendet ein Datenmodell (Bild 1), welches sich um Adressen, Kunden, Bestellungen und Artikel rankt. Dabei handelt es sich nicht um eine ausgefeilte Vertriebslösung, sondern um ein Demonstrationsobjekt, das auf die wesentlichen Teile reduziert ist.

Zwei Hauptformulare speisen sich dann aus den Tabellen des Modells – so viel sei schon an dieser Stelle verraten. Das eine verwendet nur die Tabelle **tblAdressen** und deren Nachschlagetabellen **tblAnreden**, **tblOrte** und **tblLaender**.

Das andere simuliert eine Kunden- und Bestellverwaltung, wobei hier die Adresstabelle als Kundenbasis zweckentfremdet wird. An ihr hängen die Bestellungen, Bestelldetails und Bestellartikel.

Die Verknüpfung zwischen Kundenadresse und Bestellung geschieht über die **ID** der Primärtabelle und **KundeID** der Fremdschlüsseltabelle. Ähnlich sieht die Beziehung zwischen Bestellung und Bestelldetail aus. Die Tabelle **tblArtikel** ist hingegen eher wieder eine Nachschlagetabelle. Zu einer Kundenadresse kann es also mehrere Bestellungen geben und zu jeder Bestellung wiederum mehrere Bestellartikel in den Bestelldetails. Damit die Artikeltabelle etwas aufgepeppt wird, verweist ein Feld **Her-**

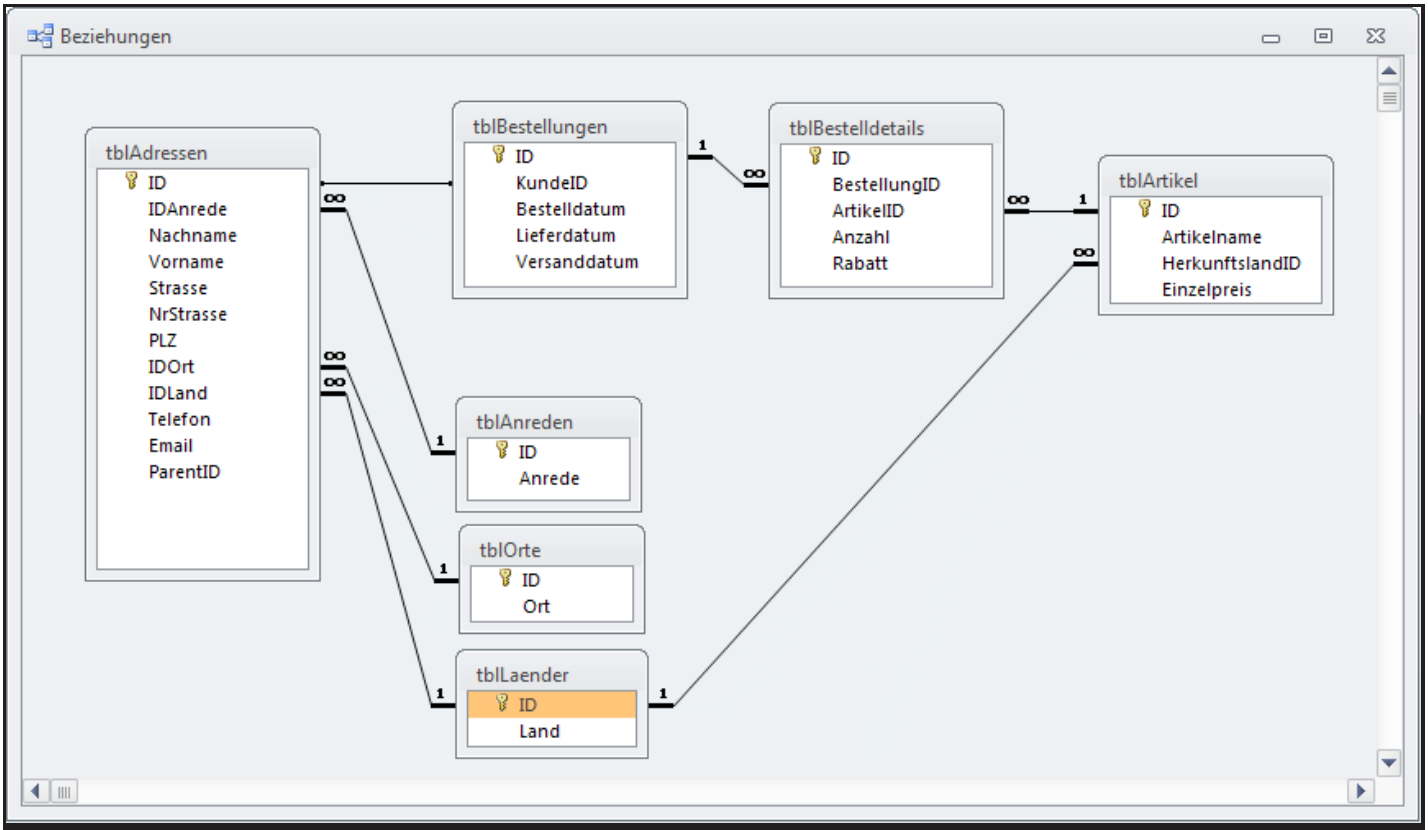


Bild 1: Datenmodell der Beispieldatenbank mit den Beziehungen zwischen Adressen, Bestellungen und Artikeln

kunftsland zusätzlich auf die Tabelle **tblLaender**.

Adressen verwalten

Über das Formular frmAdressen in Bild 2 werden nun die Adressen angezeigt und bearbeitet. Es enthält Steuerelemente für alle Felder der Tabelle, wobei die drei Nachschlagefelder **IDAnrede**, **IDOrt** und **IDLand** als Kombinationsfelder daher kommen und sich in ihrer Datenherkunft direkt aus den jeweiligen Tabellen speisen. Gebunden sind diese Felder an die erste unsichtbare Spalte **ID** der Detailtabellen. Die Datenquelle des Formulars selbst ist ebenfalls direkt die Tabelle **tblAdressen**. Soweit weist das Formular keinerlei Besonderheiten auf.

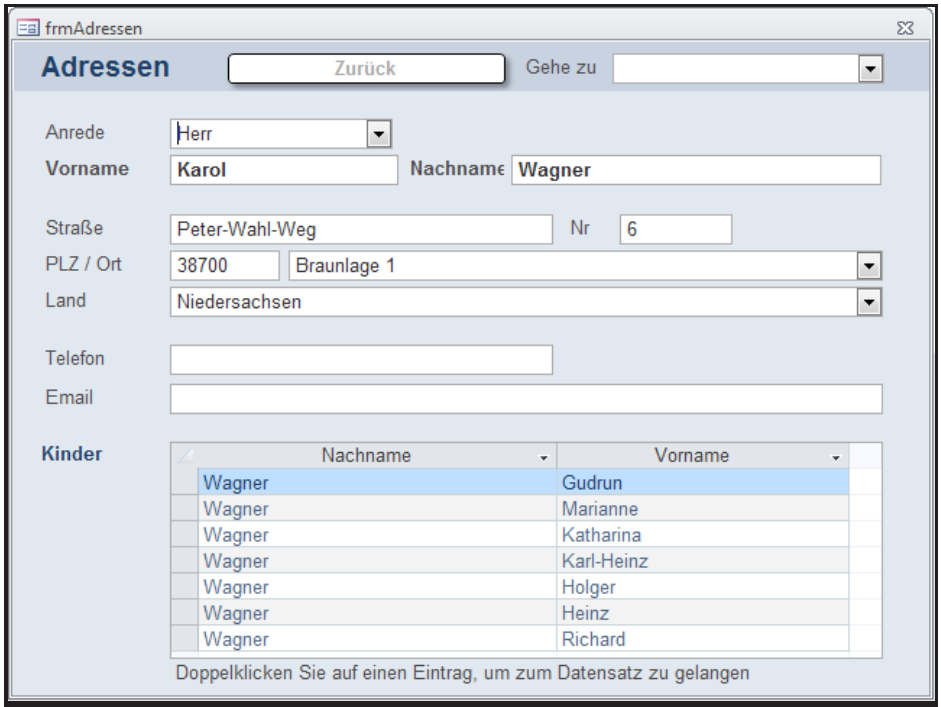


Bild 2: Das Adressformular frmAdressen der Beispieldatenbank

Zusätzlich sind ist aber ein Unterformular **sfrmKinder** integriert, welches ebenfalls auf der Adresstabelle basiert, jedoch deren Felder über eine Abfrage auf **Nachname** und **Vorname** beschränkt. Das sollen Kinder des angezeigten Hauptdatensatzes sein. Die Verbindung kommt über das unscheinbare Feld **ParentID** der Tabelle zustande. Ein Datensatz, der einen Wert in diesem Feld enthält, verweist damit auf den Elterndatensatz mit der verwiesenen **ID**. In der letzten Ausgabe von Access Basics lernten Sie diese Methode bereits im Beitrag zu **Rekursiven Tabellen** kennen. Beim Navigieren durch die Hauptdatensätze des Formulars werden also im Unterformular die zugehörigen Kinder angezeigt.

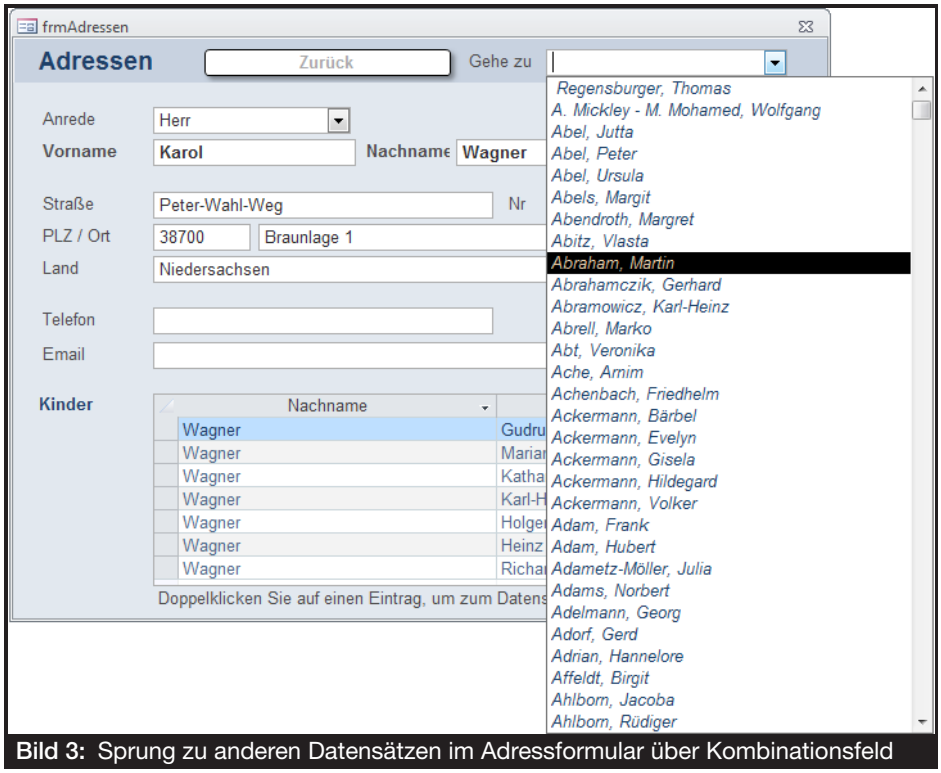


Bild 3: Sprung zu anderen Datensätzen im Adressformular über Kombinationsfeld

Es gibt nun drei Spezialelemente im Formular. Zum einen listet eine **Combobox** rechts oben im Kopfbereich alle Adressen namentlich auf. Durch Auswahl eines Eintrags gelangen Sie zum entsprechenden Adressdatensatz, wie Bild 3 demonstriert. Deshalb ist die Navigationsleiste des Formulars auch ausgeblendet. Denn

diese ist in den meisten Fällen ohnehin überflüssig. Wer sucht Datensätze schon über Vor- und Zurücksteppen oder über die Eingabe eines Zahlenwerts heraus?

Ebenfalls im Kopfbereich untergebracht ist die Schaltfläche **Zurück**. Sie ist zunächst deaktiviert. Sobald Sie aber über die Navigations-Combo zu einem anderen Datensatz springen, wird sie aktiv und zeigt die eben verlassene Adresse an, etwa den **Karol Wagner** (Bild 4). Mit Klick auf diesen Button gelangen Sie wieder zurück zum aufrufenden Datensatz.

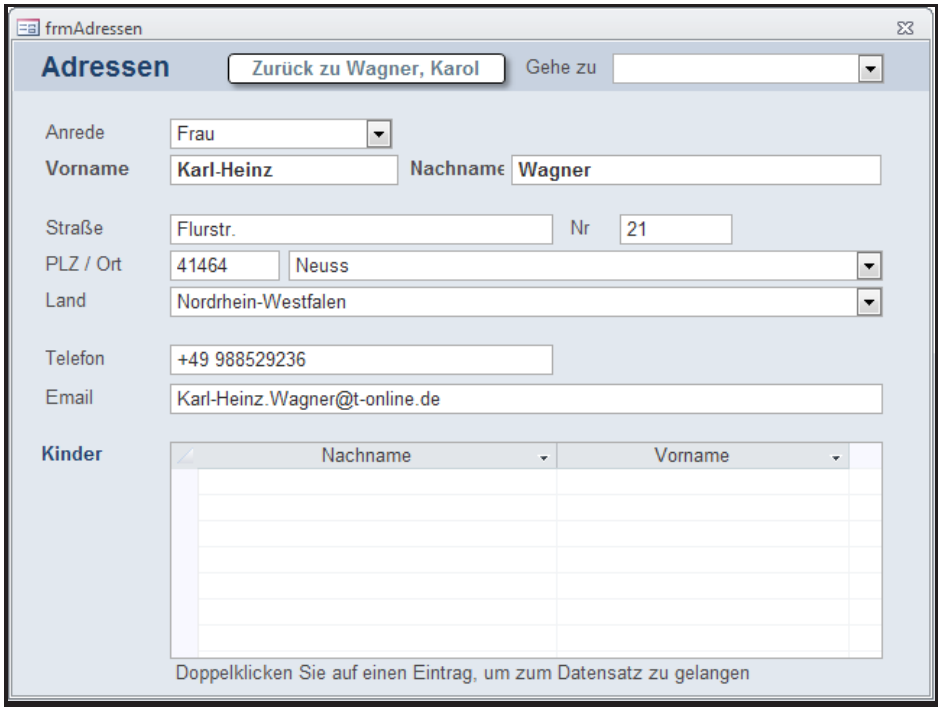


Bild 4: Die Zurück-Schaltfläche oben zeigt die zuletzt verlassene Adresse an

Wie der Text unter dem Unterformular bereits erläutert, reagieren die Datenzeilen der Kinder auf einen Doppelklick. Der löst nämlich aus, dass zum Adressdatensatzes dieses Kindes gesprungen wird. Auch in diesem Fall wird die **Zurück**-Schaltfläche aktiv, damit Sie schnell zum Elterndatensatz zurückgelangen können.

Eingebaut ist hier also eine **History** ähnlich der des Webbrowsers. Dass all diese Navigationsfunktionen mit nur sehr wenig Aufwand zu realisieren sind, zeigt ein Blick in das Code-Modul des Formulars. Listing 1 gibt den Kopf und die Initialisierung wieder.

```
Private arrUndo() As String
Private lUndo As Long

Private Sub Form_Load()
    ReDim arrUndo(1, 30)
    lUndo = -1
    Me.Recordset.FindFirst "ID=19"
End Sub

Private Sub Form_Current()
    Me!cbFind = Null
End Sub
```

Listing 1: Erster Teil des Formularmoduls von frmAdressen

Dort ist ein Array **arrUndo** deklariert, welches der **History**-Funktion dient. In diesem Array werden zuletzt verlassene Datensätze anhand ihrer **ID** und des Adressnamens gespeichert. In der Ereignisprozedur **Beim Laden** (**Form_Load**) wird das Array auf zwei Ebenen mit 30 Elementen dimensioniert. **30** ist ein willkürlich gewählter Wert, der die maximale Zahl der Rücksprünge darstellt. Er sollte ausreichen. Die erste Ebene des Arrays enthält die **IDs** der Datensätze, die zweite die Adressnamen. Wir kommen noch darauf zu sprechen, wie und wo es befüllt wird. Jedenfalls könnte ein Array-Datensatz so aussehen:

```
arrUndo(0,12) = 78
arrUndo(1,12) ="Karol, Wagner"
```

Die Variable **lUndo** ist ein Positionszeiger für das Array. Wäre die aufgeführten Elemente eben erst gespeichert worden, so enthielte **lUndo** den Wert **12**. Im **Load**-Ereignis jedoch wird **lUndo** zunächst auf **-1** gesetzt, weil es ja noch keinen Array-Eintrag gibt. Nur zur Demonstration wird in diesem Ereignis auch noch der Datensatz mit der **ID 19** angesprungen. Diese Zeile ist elementar für das Navigieren zu bestimmten Datensätzen. Statt sich etwa der

DAO-Objekte und -Auflistungen, Teil II

Waren Containers, Documents und Properties Thema der letzten Ausgabe über DAO-Objekte, so geht es diesmal abschließend um Relations und Workspaces. Dabei handelt es sich zum einen um die Abbildung und Manipulation von Tabellenbeziehungen im Objektmodell, zum anderen um die sogenannten Arbeitsbereiche der Access Database Engine. Erfahren Sie, wie Tabellenbeziehungen einer Datenbank ermittelt und neu angelegt werden können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1511_DAORelations.accdb**

Beziehungsbeispiel

Die Beispieldatenbank enthält vier uns mittlerweile wohlbekannte Tabellen, die über einige Beziehungen miteinander verknüpft sind. Zu den Feldern **IDAnrede**, **IDOrt** und **IDLand** der Tabelle **tblAdressen** existieren drei Nachschlagetabellen, die mit ihr auf unterschiedliche Weise in Beziehung stehen (siehe Bild 1). Für unsere Zwecke wurden die Beziehungen zum Testen auf etwas eigentümliche Weise modifiziert, die so allerdings keinen rechten Sinn mehr ergeben.

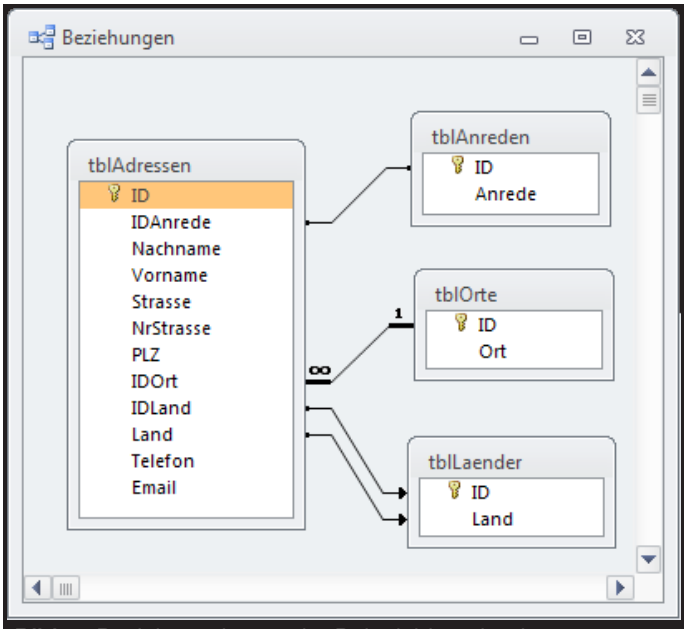


Bild 1: Beziehungslayout der Beispieldatenbank

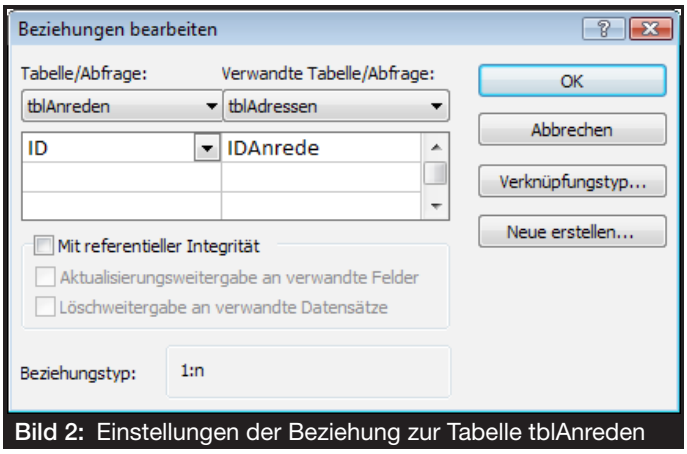


Bild 2: Einstellungen der Beziehung zur Tabelle tblAnreden

Bild 2 zeigt die Einstellungen für die Beziehung zwischen Adressen- und Anreden-Tabelle. Die **Referenzielle Integrität** wurde ausgeschaltet, aber der Typ mit **1:n** beibehalten. Die Einstellungen für die Nachschlagetabelle zu Orten findet sich in Bild 3. Hier wurde nichts verändert. Schließlich stellt Bild 4 die Beziehung für die Länder-Tabelle dar, bei der jeweils zwei Felder verknüpft wurden. Neben dem eigentlichen Primärschlüsselfeld **ID** zu **IDAnrede** ist zusätzlich der Ländername **Land** selbst in der zweiten Zeile **1:n**-verknüpft. Außerdem ist der Typ der Beziehung, erreichbar über die Schaltfläche **Verknüpfungstyp...**, auf den dritten Modus eingestellt, was sie zu einer **Left-Join-Beziehung** macht (Bild 5) Mit diesen Vorgaben lässt sich **Referenzielle Integrität** ohnehin nicht mehr festlegen, weil keine Eindeutigkeit mehr gegeben ist.

Beziehungen auslesen

Für die Beziehungen einer Datenbank ist über das VBA-Objekt **Database** deren Auflistungseigenschaft **Relations** zuständig. Die kurze Routine **GetRelations** in Listing 1 macht deutlich, wie die einzelnen **Relation**-Objekte in einer **For-Each-Schleife** durchlaufen werden können.

Zunächst wird die **Database**-Variable **dbs** über die Funktion **CurrentDb** auf eine aktuelle Instanz der Datenbank gesetzt. Als Schleifenvariable kommt die Variable **rel** vom Typ **DAO.Relation** zum Einsatz. Im VBA-Direktfenster wird dann lediglich der **Name** der

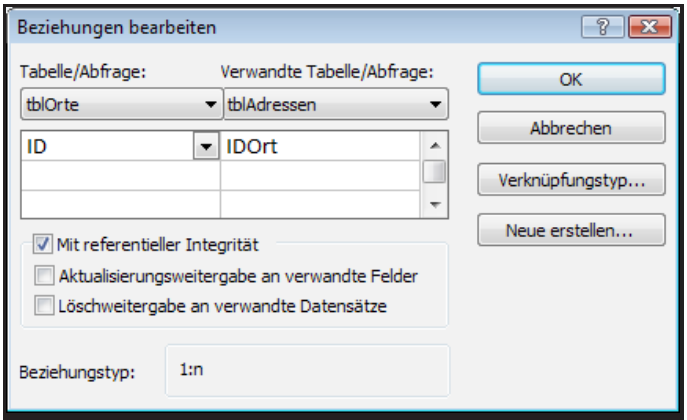


Bild 3: Einstellungen für die Beziehung zur Tabelle tblOrte

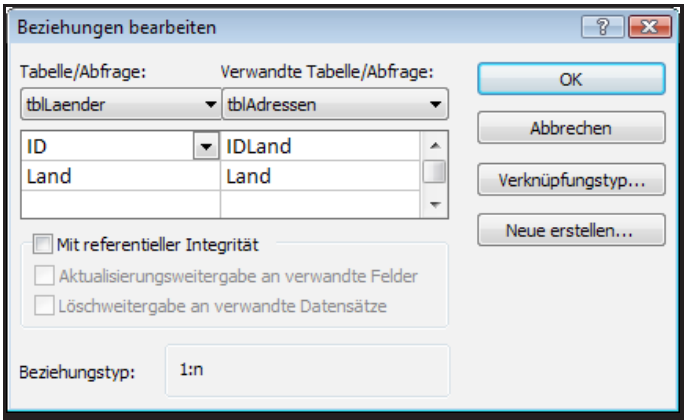


Bild 5: Einstellung für die Beziehung zur Tabelle tblLaender

Beziehung ausgegeben. Als Ergebnis erhalten Sie dies:

```
MSysNavPaneGroupCategoriesM-
SysNavPaneGroups
MSysNavPaneGroupsMSysNavPa-
neGroupToObjects
tblAnredentblAdressen
tblLaendertblAdressen
tblOrtetblAdressen
```

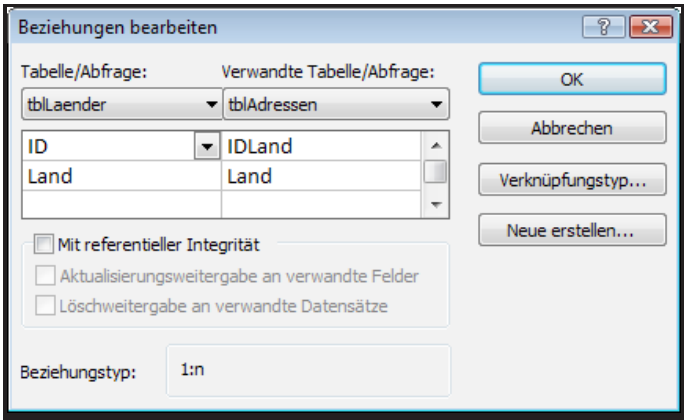


Bild 4: Verknüpfungstyp der Beziehung zu tblLaender

Dass Beziehungen überhaupt Namen haben, ist nicht so selbstverständlich, denn diese tauchen in der Oberfläche von Access nirgendwo auf. Im Beziehungsfenster sind sie auch in den Eigenschaftsdialogen nicht angegeben. Wie Sie sehen können, leiten sich die Namen aus den an der Beziehung beteiligten Tabellen ab. Die ersten beiden oben angegebenen Beziehungen sind übrigens in jeder Datenbank im **Access 2007-Format** vorhanden. Access legt sie selbst an.

Diese speziellen **MSys-Tabellen** dienen der Verwaltung und Steuerung des Navigationsbereichs.

```
Sub GetRelations()
    Dim dbs As Database
    Dim rel As DAO.Relation

    Set dbs = CurrentDb
    For Each rel In dbs.Relations
        Debug.Print rel.Name
    Next rel
End Sub
```

Listing 1: Prozedur zum Durchlaufen der DAO-Auflistung Relations

```
Sub GetRelationsTables()
    Dim dbs As Database
    Dim rel As DAO.Relation

    Set dbs = CurrentDb
    For Each rel In dbs.Relations
        Debug.Print rel.Name
        Debug.Print , rel.Table
        Debug.Print , rel.ForeignTable
        Debug.Print , rel.Attributes
    Next rel
End Sub
```

Listing 2: Ausgabe weiterer Beziehungseigenschaften (Tabellen)

Weiten wir nun die einfache Prozedur aus **Listing 1** einmal ab, um mehr über die Beziehungen zu erfahren. Die Routine in Listing 2 hat den gleichen Grundaufbau, gibt aber neben den Namen auch noch die an der Verknüpfung beteiligten Tabellen über die Eigenschaften **Table** und **ForeignTable** der jeweiligen **Relations-Objekte** wieder. Das Ergebnis, hier unter Ausschluss der **MSys**-Beziehungen:

```
tblAnredentblAdressen
            tblAnreden
            tblAdressen
             2
tblLaendertblAdressen
            tblLaender
            tblAdressen
             33554434
tblOrtetblAdressen
            tblOrte
            tblAdressen
             0
```

Table ist also jene Tabelle, welche den Primärschlüssel in der Beziehung aufweist, **ForeignTable** die Tabelle mit dem Fremdschlüsselfeld.

Die Zahl, die Attributes enthält, gibt darüber hinaus Aufschluss auf die Art der Beziehung. Es handelt sich hierbei um eine Kombination von Konstanten der Enumeration **RelationAttributeEnum** von DAO – siehe Objektkatalog. Der Wert 0 besagt dabei, dass die Beziehung **Referenzielle Integrität** fordert. Ist dies nicht der Fall, so steht hier eine 2. Die Zahl 33554434 ergibt sich aus der booleschen Addition von **dbRelationRight** (33554432) mit **dbRelationDontEnforce** (2), was eine **Right-Verknüpfung** ohne **Referenzielle Integrität** symbolisiert. Damit Sie den Typ nicht langwierig über die Analyse der Konstanten im Objektkatalog ermitteln müssen, gibt es im Modul **mdlDAORelations** die Funktion **AttributesString**, der Sie die **Attributes**-Konstante als Parameter übergeben und als Rückgabe eine Textrepräsentation des Typs erhalten:

```
? AttributesString(2)
> 1:n,Keine Ref. Integrität
? AttributesString(33554434)
> 1:n,Right Join,Keine Ref. Integrität
```

```
Sub GetRelationsProps()
    Dim dbs As Database
    Dim rel As DAO.Relation
    Dim daoPrp As DAO.Property

    Set dbs = CurrentDb
    For Each rel In dbs.Relations
        Debug.Print rel.Name
        For Each daoPrp In rel.Properties
            Debug.Print , _
                daoPrp.Name, _
                daoPrp.Value
        Next daoPrp
    Next rel
End Sub
```

Listing 3: Durchlaufen der Properties-Auflistungen der Relations

Sie können, statt die **Relation**-Methoden explizit anzugeben, auch deren **Properties** auslesen (Listing 3), und kommen damit zum gleichen Resultat:

```
tblAnredentblAdressen
    Name tblAnredentblAdressen
    Table tblAnreden
    ForeignTable tblAdressen
    Attributes 2
    PartialReplica Falsch
tblLaendertblAdressen
    Name tblLaendertblAdressen
    Table tblLaender
    ForeignTable tblAdressen
    Attributes 33554434
    PartialReplica Falsch
tblOrtetblAdressen
    Name tblOrtetblAdressen
    Table tblOrte
    ForeignTable tblAdressen
    Attributes 0
    PartialReplica Falsch
```

Hier schleicht sich noch die Eigenschaft **PartialReplica** ein, die uns nicht weiter interessiert, weil sie nur für replizierte Datenbanken von Belang ist, die ohnehin seit geraumer Zeit nicht mehr direkt von Access unterstützt werden.

Nun haben wir zwar die Tabellen einer Beziehung ermittelt, wissen aber noch nicht, welche Felder genau aus ihnen miteinander verknüpft sind. Die **Fields**-Auflistung eines **Relation**-Objekts jedoch gibt uns darüber Auskunft. Sie kann auf analoge Weise durchlaufen werden, wie die **Properties** (siehe Listing 4). Für jedes Feld **fld** eines **Relation**-Objekts werden hier die Eigenschaften **Name** und **ForeignName** ausgelesen

```
Sub GetRelationsFields()
    Dim dbs As Database
    Dim rel As DAO.Relation
    Dim fld As DAO.Field

    Set dbs = CurrentDb
    For Each rel In dbs.Relations
        Debug.Print rel.Name
        For Each fld In rel.Fields
            Debug.Print , _
                rel.Table & "." & _
                fld.Name & " > " & _
                rel.ForeignTable & _
                "." & fld.ForeignName
        Next fld
    Next rel
End Sub
```

Listing 4: Durchlaufen der Fields-Auflistungen der Relations

und mit den jeweiligen Tabellennamen kombiniert, so dass sich dieses Ergebnis zeigt:

```
tblAnredentblAdressen
  tblAnreden.ID > tblAdressen.IDAnrede
tblLaendertblAdressen
  tblLaender.ID > tblAdressen.IDLand
  tblLaender.Land > tblAdressen.Land
tblOrtetblAdressen
  tblOrte.ID > tblAdressen.IDOrt
```

Am zweiten Eintrag zur Länder-Beziehung wird deutlich, dass zwei Felder verknüpft wurden, die **Fields**-Auflistung also zwei Elemente enthält. Obwohl übrigens das Field-Objekt als **DAO.Field** ausgewiesen ist, kennt es nur genau diese zwei Eigenschaften **Name** und **ForeignName**. Alle anderen Methoden des Objekts schlagen hier fehl.

Umgekehrt kann ein **Field**-Objekt etwa eines **Recordsets**, wiewohl vom gleichen Typ, mit diesen beiden Eigenschaften nichts anfangen – eine etwas seltsame Implementation des DAO-Objektmodells.

Kommunikationsdaten als Bericht

Wie Sie Kommunikationsdaten wie Telefon- und Faxnummern oder E-Mail-Adressen komfortabel verwalten, haben Sie bereits im Artikel **Kommunikationsdaten im Griff** in Ausgabe 2/2012 erfahren. Nun wollen wir diese Daten aufgreifen und sie in einem Bericht ausgeben. Klar: Heutzutage hat man normalerweise alle Telefonnummern im Handy gespeichert und wählte diese direkt an. Es gibt aber sicher noch viele Menschen, die gern eine Telefonliste auf dem Schreibtisch liegen haben.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1511_Kommunikationsdatenbericht.accdb**.

Ausgangsdaten

Die Beispieldatenbank enthält die Tabellen aus Bild 1. Neben der Tabelle **tblKunden**, die über das Fremdschlüsselfeld **AnredeID** mit der Tabelle **tblAnreden** verknüpft ist, gibt es zwei weitere Tabellen, die für uns interessant sind.

Die erste heißt **tblKommunikationsarten** und enthält alle möglichen Kommunikationsarten wie Telefon dienstlich, Telefax dienstlich, Mobil privat und so weiter (siehe Bild 2).

Die zweite Tabelle heißt **tblKommunikationsdetails** und verknüpft die Kunden aus der Tabelle **tblKunden** mit den Datensätzen der Tabelle **tblKommunikationsdetails**.

Sie dient also als Verknüpfungstabelle für eine m:n-Beziehung zwischen Kunden und Kommunikationsdaten und steuert noch Detaildaten zu jeder Verknüpfung bei – wie etwa die Telefonnummer oder Mailadresse (siehe Bild 3).

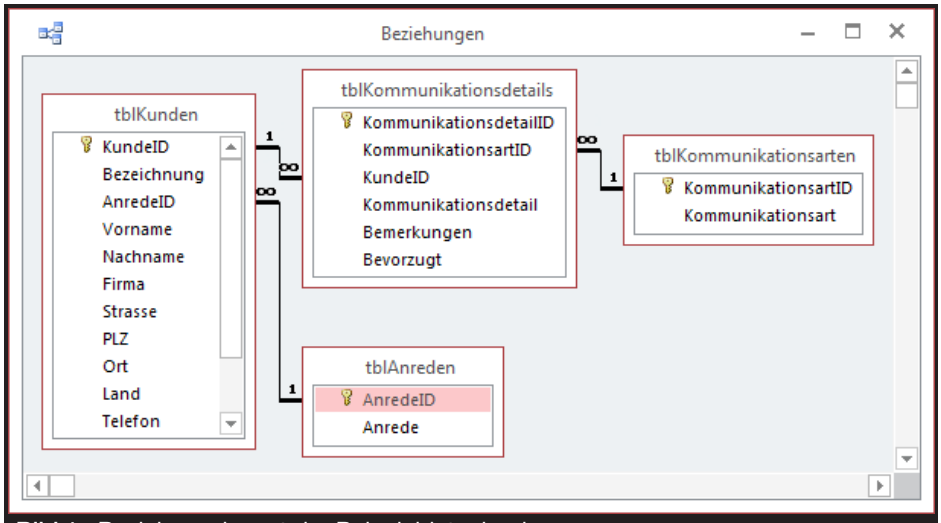
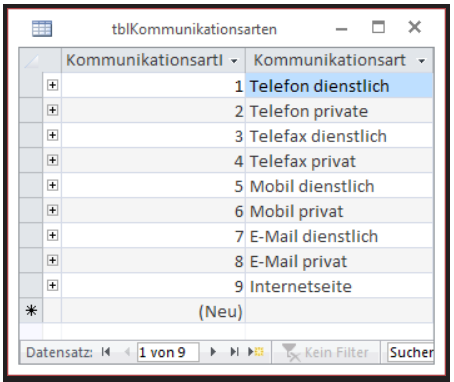


Bild 1: Beziehungslayout der Beispieldatenbank

Bericht erstellen

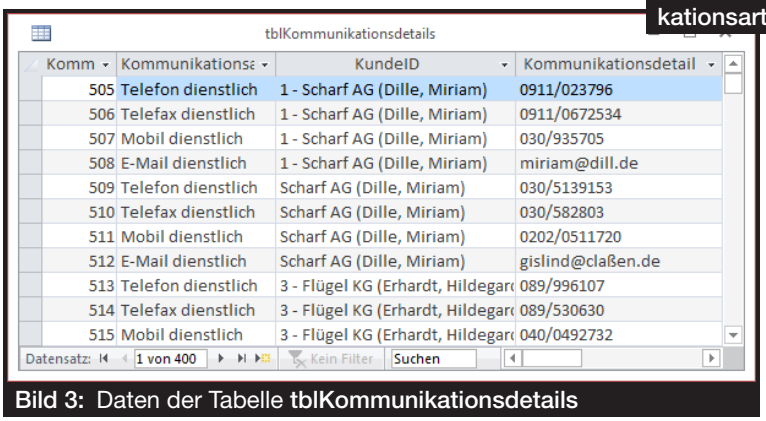
Der Bericht zur Darstellung der Kommunikationsdaten soll Daten aus allen im Datenmodell enthaltenen Tabellen außer der Tabelle **tblAnreden** anzeigen. Dazu gehören **Vorname**, **Nachname**, **Kommunikationsdetail** und **Kommunikationsart**. Also stellen wir zunächst eine entsprechende Datenherkunft in Form einer Abfrage zusammen, die wie in Bild 4 aussieht und die wie unter dem Namen **qryKommunikationsdaten** speichern.


tblKommunikationsarten

KommunikationsartI	Kommunikationsart
1	Telefon dienstlich
2	Telefon private
3	Telefax dienstlich
4	Telefax privat
5	Mobil dienstlich
6	Mobil privat
7	E-Mail dienstlich
8	E-Mail privat
9	Internetseite
(Neu)	

Datensatz: 1 von 9 Kein Filter Sucher

Bild 2: Daten der Tabelle **tblKommunikationsarten**


tblKommunikationsdetails

Komm	Kommunikationsa	KundeID	Kommunikationsdetail
505	Telefon dienstlich	1 - Scharf AG (Dille, Miriam)	0911/023796
506	Telefax dienstlich	1 - Scharf AG (Dille, Miriam)	0911/0672534
507	Mobil dienstlich	1 - Scharf AG (Dille, Miriam)	030/935705
508	E-Mail dienstlich	1 - Scharf AG (Dille, Miriam)	miriam@dill.de
509	Telefon dienstlich	Scharf AG (Dille, Miriam)	030/5139153
510	Telefax dienstlich	Scharf AG (Dille, Miriam)	030/582803
511	Mobil dienstlich	Scharf AG (Dille, Miriam)	0202/0511720
512	E-Mail dienstlich	Scharf AG (Dille, Miriam)	gislind@claßen.de
513	Telefon dienstlich	3 - Flügel KG (Erhardt, Hildegar	089/996107
514	Telefax dienstlich	3 - Flügel KG (Erhardt, Hildegar	089/530630
515	Mobil dienstlich	3 - Flügel KG (Erhardt, Hildegar	040/0492732

Datensatz: 1 von 400 Kein Filter Suchen

Bild 3: Daten der Tabelle **tblKommunikationsdetails**

Nun können wir bereits den Bericht erstellen und diesen unter dem

Namen **rptKommunikationsdaten** speichern. Dem Bericht weisen Sie dann für die Eigenschaft **Datensatzquelle** die soeben erstellte Abfrage **qryKommunikationsdaten** zu (siehe Bild 5).

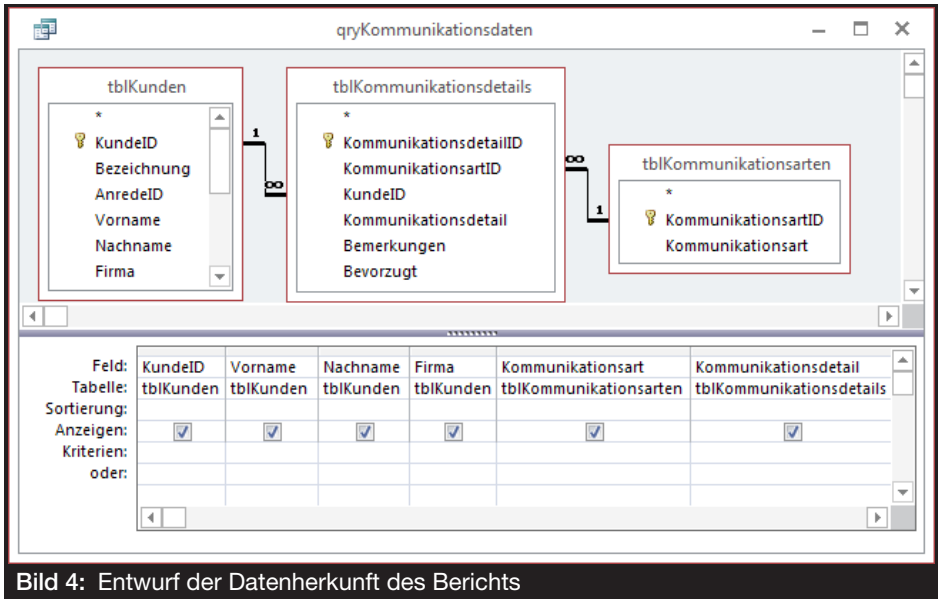


Bild 4: Entwurf der Datenherkunft des Berichts

Danach wollen wir uns Schritt für Schritt dem perfekten Kommunikationsdaten-Bericht annähern. Als Erstes tragen wir im Detailbereich einfach die benötigten Daten in tabellarischer Form ein – also die Felder **Vorname**, **Nachname**, **Firma**, **Kommunikationsart** und **Kommunikationsdetail**. Außerdem benötigen wir natürlich entsprechende Spaltenüberschriften. Wo sollen diese erscheinen? Am besten auf jeder Seite. Dann können wir den Berichtsbereich namens Seitenkopf für die Spaltenüberschriften nutzen.

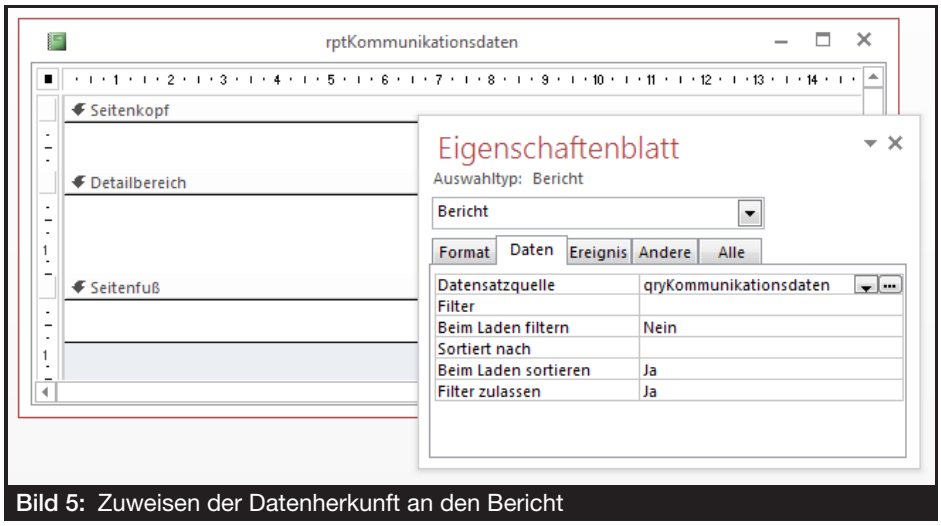


Bild 5: Zuweisen der Datenherkunft an den Bericht

Unter den moderneren Access-Versionen, also ab Version 2007, ziehen Sie die Felder einfach wie gewohnt aus der Feldliste in den Detailbereich des Formulars. Diese sind nun noch alle markiert (falls nicht, markieren Sie alle Steuerelemente durch Aufziehen eines entsprechenden Rahmens).

Tabellarisches Layout einrichten

Anschließend wechseln Sie im Ribbon zum Tab **Anordnen**. Dort finden Sie links in der Gruppe **Tabelle** die Schaltfläche **Tabelle** (siehe Bild 6).

Ein Klick auf diese Schaltfläche ordnet alle markierten Steuerelemente so an, dass die Bezeichnungsfelder als Spaltenköpfe in den Seitenkopf-Bereich verschoben werden. Die gebundenen Felder verbleiben im Detailbereich, werden dort aber ebenfalls nebeneinander angeordnet, und zwar genau unter den jeweiligen Bezeichnungsfeldern. Der Clou ist, dass Sie nun die Breite der

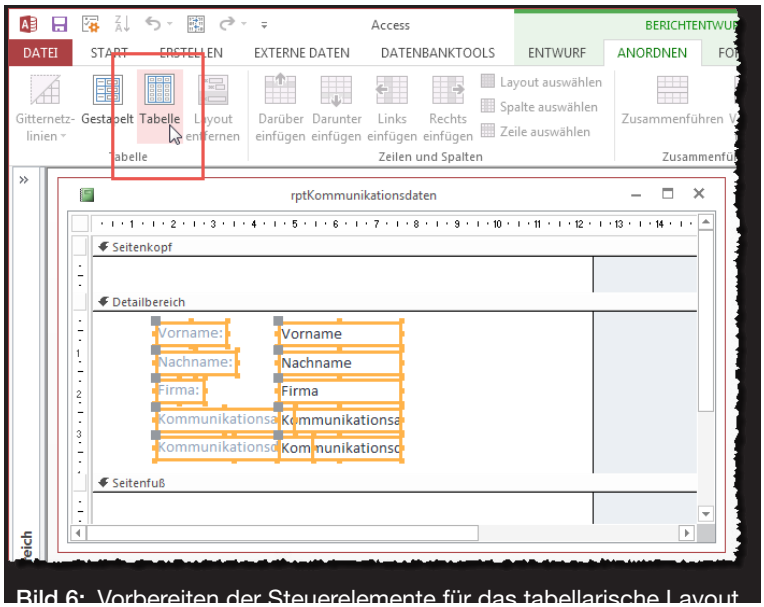


Bild 6: Vorbereiten der Steuerelemente für das tabellarische Layout

Steuerelemente anpassen können und dabei die Bezeichnungsfelder ihre Breite entsprechend ändern. Die gebundenen Steuerelemente werden im Detailbereich gegebenenfalls etwas zu weit unten angeordnet. Wenn Sie diese en bloc nach oben verschieben wollen, markieren Sie einfach nur diese Steuerelemente. Ziehen Sie dann an der oberen Markierung eines der Steuerelemente nach oben und vergrößern Sie so die Höhe, bis der gewünschte Abstand zum oberen Rand des Detailbereichs erreicht ist. Dann ziehen Sie den unteren Rand nach oben, bis Sie die vorherige Höhe wieder hergestellt haben (siehe Bild 7).

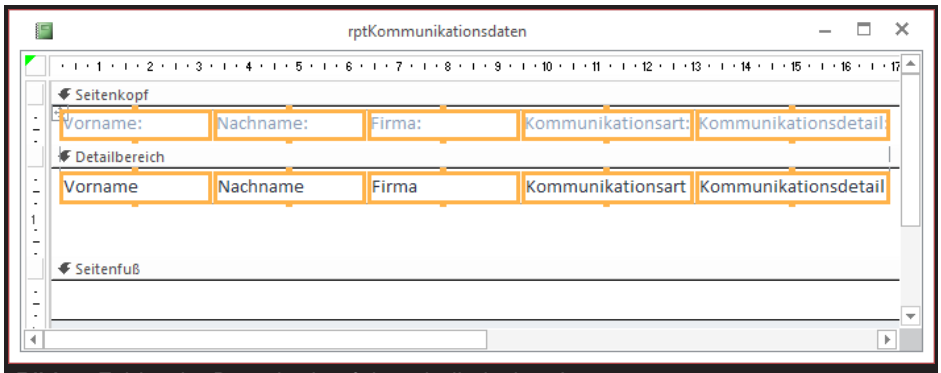


Bild 7: Felder der Datenherkunft im tabellarischen Layout

Optische Anpassungen

Wenn Sie nun in die Seitenansicht wechseln, erhalten Sie den Bericht aus Bild 8. Hier fallen zunächst die alternierende Hintergrundfarbe und die Rahmen um die einzelnen Steuerelemente negativ auf. Kein Problem: Um die Rahmen zu entfernen, wechseln Sie in die Entwurfsansicht, markieren mit **Strg + A** alle Steuerelemente und stellen die Eigenschaft **Rahmenart** auf **Transparent** ein. Um die Hintergrundfarbe komplett weiß zu halten, markieren Sie den Detailbereich und stellen die Eigenschaft **Alternative Hintergrundfarbe** auf den Wert der Eigenschaft **Hintergrundfarbe** ein (beispielsweise **Hintergrundfarbe 1**).

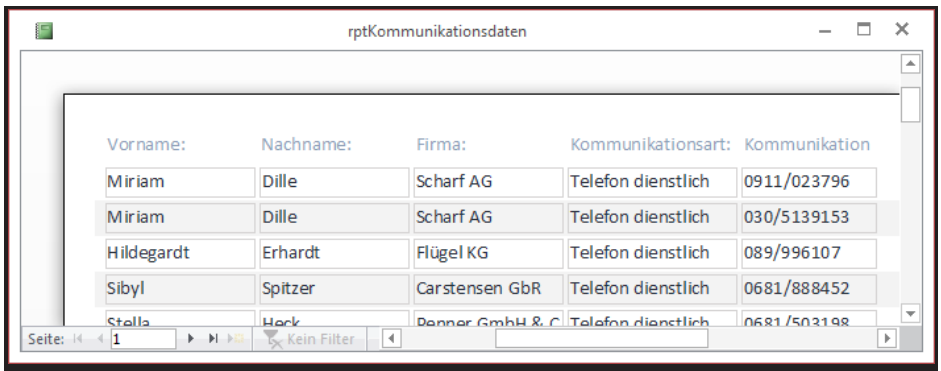


Bild 8: Erster Blick auf die Seitenansicht

Außerdem sind die Spaltenbreiten noch nicht optimiert. Dies ändern wir am einfachsten in der Layoutansicht (ab Access 2007 verfügbar). Hier stellen wir zunächst die Spaltenüberschrift **Kommunikationsart** auf den Text **Kontakt unter:** ein und leeren die Spaltenüberschrift für die Spalte **Kommunikationsdetail**. Außerdem passen Sie hier, da es nach wie vor ein Raster für die Anordnung der Steuerelemente gibt, durch Einstellen der Breite der Spaltenüberschrift oder eines der gebundenen Steuerelemente die Breite aller Steuerelemente dieser Spalte an.

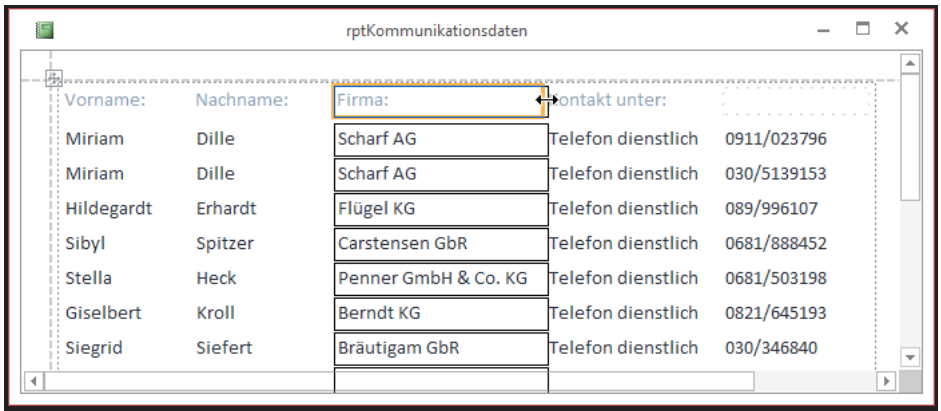


Bild 9: Anpassen der Spaltenbreiten in der Layoutansicht

Das Ergebnis sieht dann etwa wie in Bild 9 aus.

Mehrfachnennungen ausblenden

Nun zeigt der Bericht die Daten erstens nicht nach Nachname und Vorname getrennt an und zweitens werden die übrigen Daten auch noch für jede Kommunikationsart wieder-

Mehrspaltiger gruppierter Bericht

Im Artikel **Kommunikationsdaten als Bericht** in Ausgabe 11-12/2015 haben Sie erfahren, wie Sie mehrere Datensätze mit teilweise gleichen Daten ohne Wiederholungen ausgeben. Dies wollen wir in diesem Artikel verfeinern, indem wir die wiederholt auftretenden Daten gleich in einen Gruppenkopf und nur die eindeutigen Daten im Detailbereich anzeigen. Das ganze verteilen wir noch auf mehrere Spalten und fügen ein paar schicke Icons hinzu – fertig ist die perfekte Telefonliste!

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1511_MehrspaltigerGruppierterBericht.accdb**.

Beispieldaten

Die im Bericht darzustellenden Daten stammen aus den drei Tabellen **tblKunden**, **tblKommunikationsdetails** und **tblKommunikationsarten** der Beispieldatenbank, die wir für den Bericht in einer Abfrage wie in Bild 1 zusammengefasst haben (mehr Informationen zu den Basistabellen und dieser Abfrage finden Sie im Artikel **Kommunikationsdaten als Bericht**).

qryKommunikationsdaten

KundeID	Vorname	Nachname	Firma	Kommunikationsart	Kommunikationsdetail
5	Stella	Heck	Penner GmbH	E-Mail dienstlich	stella@heck.de
5	Stella	Heck	Penner GmbH	Telefax dienstlich	0681/6063927
5	Stella	Heck	Penner GmbH	Telefon dienstlich	0681/503198
5	Stella	Heck	Penner GmbH	Mobil dienstlich	0611/0201310
6	Giselbert	Kroll	Berndt KG	Telefon dienstlich	0821/645193
6	Giselbert	Kroll	Berndt KG	E-Mail dienstlich	giselbert@kroll.de
6	Giselbert	Kroll	Berndt KG	Mobil dienstlich	030/305190
6	Giselbert	Kroll	Berndt KG	Telefax dienstlich	0821/9802497
7	Siegrid	Siefert	Bräutigam GbR	Telefax dienstlich	030/339481
7	Siegrid	Siefert	Bräutigam GbR	Mobil dienstlich	0681/857935
7	Siegrid	Siefert	Bräutigam GbR	Telefon dienstlich	030/346840
7	Siegrid	Siefert	Bräutigam GbR	E-Mail dienstlich	siegrid@siefert.de

Datensatz: 15 von 400 | Kein Filter | Suchen

Bild 1: Herkunftsdaten für den Bericht

Grundlage: Gruppierung

Die Basis für eine solche Aufteilung ist das Hinzufügen einer Gruppierung inklusive Anzeige eines Gruppenkopf-Bereichs. Nachdem Sie die oben beschriebene Abfrage als Datenherkunft eines neuen, leeren Berichts hinzugefügt haben, legen Sie die benötigte Gruppierung im unteren Bereich an.

Aufbau des Berichts

Der Bericht soll nicht, wie in oben genanntem Artikel, jeweils eine Zeile pro Kunde plus Kommunikationsart und Kommunikationsdetails (also Telefonnummer, Faxnummer, E-Mail-Adresse et cetera) aufweisen, sondern direkt Name und Firma des Kunden in einer Kopfspalte darstellen und die Daten zu den verschiedenen Telekommunikationsdaten in darunter liegenden Zeilen.

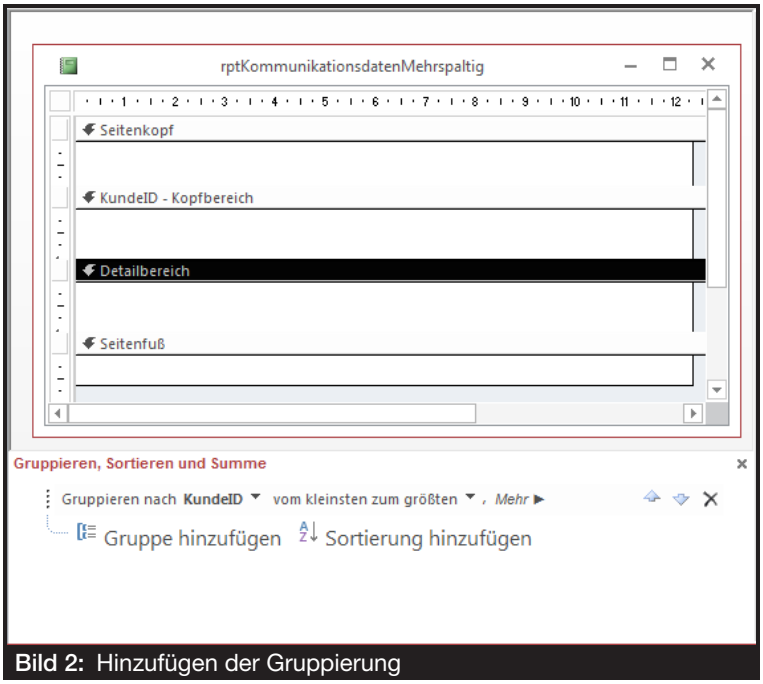


Bild 2: Hinzufügen der Gruppierung

Dazu aktivieren Sie den Gruppierungsbereich, sofern noch nicht sichtbar, mit dem Eintrag **Sortieren und Gruppieren** des Kontextmenüs, das beim Rechtsklick auf einen der Bereichsköpfe erscheint.

Ein Klick auf die Schaltfläche **Gruppe hinzufügen** fügt die Gruppierung hinzu und aktiviert gleichzeitig den Gruppenkopfbereich für das Feld **KundeID** (siehe Bild 2).

Gruppenkopf füllen

Damit wir die Spalten recht schmal halten können, wollen wir die Inhalte der Felder **Nachname** und **Vorname** sowie die **Firma**, soweit vorhanden, untereinander darstellen. Dazu legen wir im Kopfbereich ein Textfeld namens **txtKontakt** an, das den folgenden Steuerelementinhalt aufweisen soll:

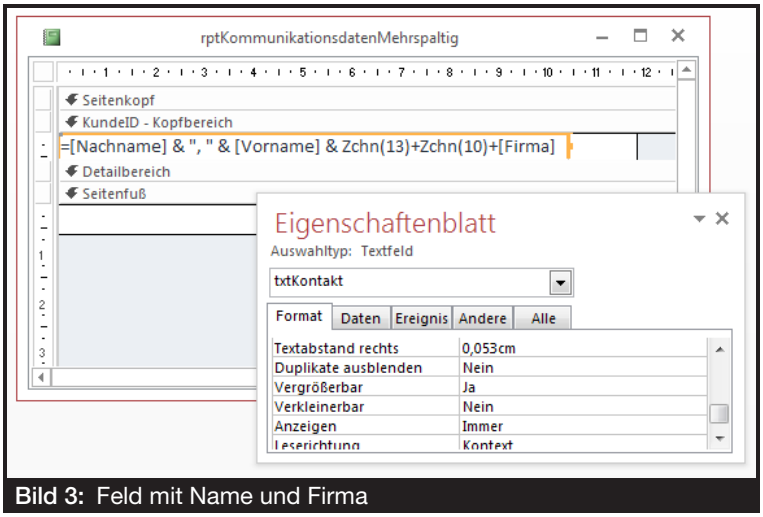


Bild 3: Feld mit Name und Firma

```
=[Nachname] & ", " & [Vorname] &
Zchn(13) + Zchn(10) + [Firma]
```

Warum arbeiten wir hier teilweise mit dem Kaufmanns-Und **(&)** und teilweise mit dem Plus-Zeichen **(+)**, um die einzelnen Elemente zu verbinden? Weil das Kaufmanns-Und alle Elemente verknüpft, auch wenn eines der Elemente den Wert **Null** aufweist, und das Plus-Zeichen immer das Ergebnis **Null** liefert, wenn eines der verknüpften Elemente den Wert **Null** aufweist.

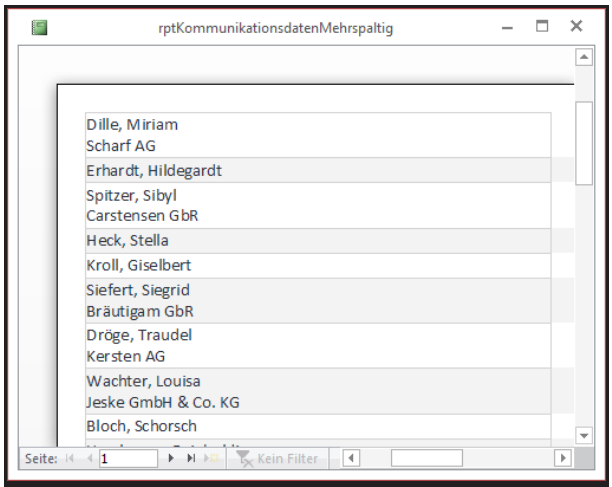


Bild 4: Nachname, Vorname und Firma erscheinen fast wie gewünscht

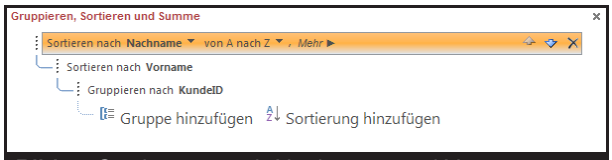


Bild 5: Sortierung nach Nachname und Vorname hinzufügen

Der Ausdruck **Zchn(13) + Zchn(10)** entspricht einem Zeilenumbruch, der dafür sorgt, dass die erste Zeile den Nachnamen und den Vornamen liefert und eine weitere Zeile die Firma.

Sollte die Firma nicht angegeben sein, benötigen wir den Zeilenumbruch nicht. Dadurch, dass wir diesen mit dem Plus-Zeichen mit dem Feld **Firma** verknüpfen, entfällt der auch der Zeilenumbruch, sollte das Feld **Firma** den Wert **Null** enthalten.

Vergrößerbares Textfeld

Wir legen die Höhe des Textfeldes zunächst so an, dass dieses genau eine Zeile aufnimmt. Damit auch die zweite Zeile angezeigt wird, falls eine Firma angegeben wurde, müssen wir die Eigenschaft **Vergrößerbar** des Textfeldes auf **Ja** einstellen. Stellen Sie bei dieser Gelegenheit auch sicher, dass diese Eigenschaft für den Kopfbereich der Gruppierung nach dem Feld **KundeID** ebenfalls den Wert **Wahr** aufweist (siehe Bild 3).

Ein Wechsel in die Seitenansicht liefert den Bericht aus Bild 4. Das sieht schon fast gut aus – mit kleinen Fehlern: Erstens werden die Einträge noch nicht nach dem Nachnamen und dem Vornamen sortiert, zweitens sollten alle Namen grau hinterlegt werden, damit sich diese von den gleich noch einzufügenden Kommunikationsdaten abheben, und drittens benötigen wir keine Rahmenlinien.

Die Rahmenlinien um die Textfelder entfernen wir, indem wir die Eigenschaft **Rahmenart** für das Textfeld **txtKontakt** auf **Transparent** einstellen. Für eine einheitliche Hintergrundfarbe sorgen wir, indem wir die Eigenschaft **Hintergrundfarbe** auf den gleichen Wert einstellen wie die Eigenschaft **Alternative Hintergrundfarbe** (unter Access 2013 etwa auf den Wert **Hintergrund1, Dunkler 5%**).

Fehlt noch die Sortierung, die wir wieder über den Bereich **Gruppieren, Sortieren und Summe** hinzufügen.

Hier klicken Sie zwei Mal auf **Sortierung hinzufügen**, wählen für die erste Sortierung das Feld **Nachname** und für die zweite das Feld **Vorname** aus und verschieben die beiden neuen Sortierungen dann so nach oben wie in Bild 5 dargestellt.

Bild 6 liefert dann die korrekte Sortierung.

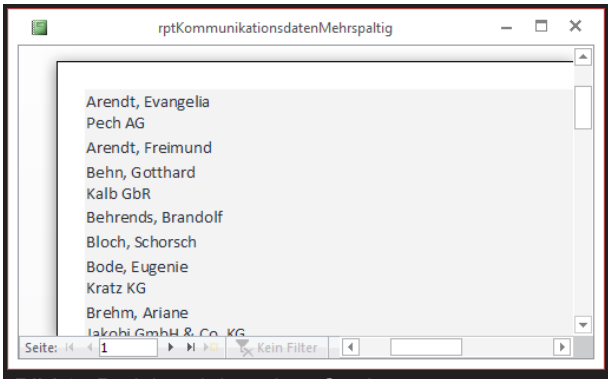


Bild 6: Bericht mit korrekter Sortierung

Kommunikationsdaten hinzufügen

Damit kommen wir zu den Telefon- und Maildaten, die ja im Detailbereich zu jedem Kunden dargestellt werden sollen. Wir vergrößern also zunächst die Höhe des Detailbereichs und fügen dann die beiden Felder **Kommunikationsart** und **Kommunikationsdetail** zum Detailbereich hinzu (siehe Bild 7). Wenn Sie die Felder aus der Feldliste in den Entwurf ziehen, können Sie die automatisch hinzugefügten Bezeichnungsfelder gleich wieder entfernen, denn diese benötigen wir nicht mehr.

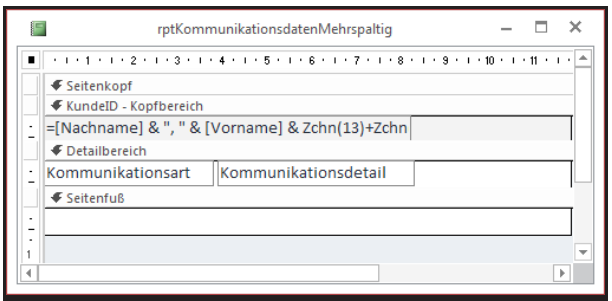


Bild 7: Hinzufügen der Kommunikationsdaten

Hier stellen wir gleich die Eigenschaft **Alternative Hintergrundfarbe** des Detailbereichs auf den Wert der Eigenschaft **Hintergrundfarbe** ein (zum Beispiel auf **Hintergrundfarbe 1**) sowie die Eigenschaft **Rahmenart** der Textfelder auf den Wert **Transparent**.

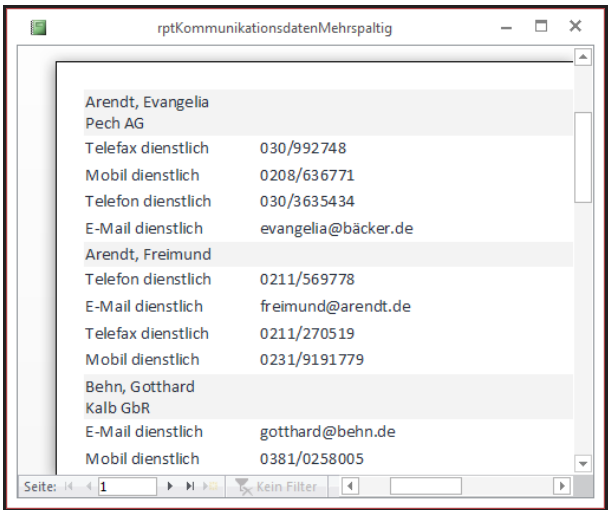


Bild 8: Kommunikationsdaten zu den einzelnen Kunden

Das Ergebnis sieht bereits recht gut aus – siehe Bild 8. Allerdings könnten wir, wenn wir die Kommunikationsarten so beibehalten wollen, auch noch die Firma in einer Zeile mit dem Kundennamen darstellen – die Spaltenbreite würde dies jedenfalls hergeben.

Aber wir wollen gleich die Kommunikationsarten ja auch noch durch geeignete Icons ersetzen, um etwas Platz zu sparen und nicht so viele sich wiederholende Texte anzuzeigen.

Icons hinzufügen (ab Access 2010)

An dieser Stelle haben wir eine neue Version des Berichts unter dem Namen **rptKommunikationsdatenMehrspaltigIcons** erstellt.

Bevor wir Icons zum Bericht hinzufügen können, müssen wir diese zur Access-Datenbank hinzufügen, welche diese in der Tabelle **MSysResources** speichert. Die Bilder müssen Sie zunächst zu einem Formular oder einem Bericht hinzufügen, und zwar über den Ribbon-Eintrag **Entwurf|Steuerelemente|Bild einfügen|Durchsuchen**.

Dies erledigen wir am besten direkt in unserem Bericht, denn dieser soll ja ohnehin ein Bildsteuerelement für die Anzeige der Icons aufnehmen.

Nun ist es kein Problem, in den gängigen Icon-Sammlungen (wir verwenden übrigens die von **www.iconexperience.com**) passende Icons für Handy, Telefon, Fax oder Mail zu finden.

Schwierig wird es jedoch bei der Unterscheidung zwischen dienstlich und privat. Bevor wir hier verschiedenen Icons zusammenfügen, suchen wir die einfachere Variante und

Bericht als Unterformular

Seit Access 2010 können Sie Berichte als Unterformular in einem Hauptformular anzeigen. Dabei lässt sich dieser zur Laufzeit beispielsweise filtern. In diesem Artikel zeigen wir, wie Sie einen Adressbericht als Unterformular in ein Formular einbinden und die angezeigten Adressen über Schaltflächen im Hauptformular filtern können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1511_BerichtAlsUnterformular.accdb**.

Beispieldaten

Die Beispieldaten entsprechen denen des Artikels Mehrspaltiger gruppierter Bericht. Diesem entnehmen wir auch den hier als Unterformular verwendeten Bericht **rptKommunikationsdatenMehrspaltigIcons**.

Filterformular

Das fertige Formular samt Unterbericht soll wie in Bild 1 aussehen. Hier sehen Sie bereits die Schaltflächen mit den Buchstaben von A bis Z, mit denen die Daten des Berichts nach dem jeweiligen Anfangsbuchstaben des Nachnamens gefiltert werden sollen. Ein Klick auf die Schaltfläche **Alle anzeigen** soll schließlich den Filter wieder aufheben.

Entwurf des Formulars

Das Formular sieht in der Entwurfsansicht wie in Bild 2 aus. Die Schaltflächen der obersten Reihe sind durchgehend mit **cmdA**, **cmdB**, **cmdC** und so weiter benannt. Die Schaltfläche darunter heißt **cmdAlle**. Den Unterbericht ziehen Sie einfach wie beim Hinzufügen eines Unterformulars aus dem Navigationsbereich in den Entwurf des Formulars.

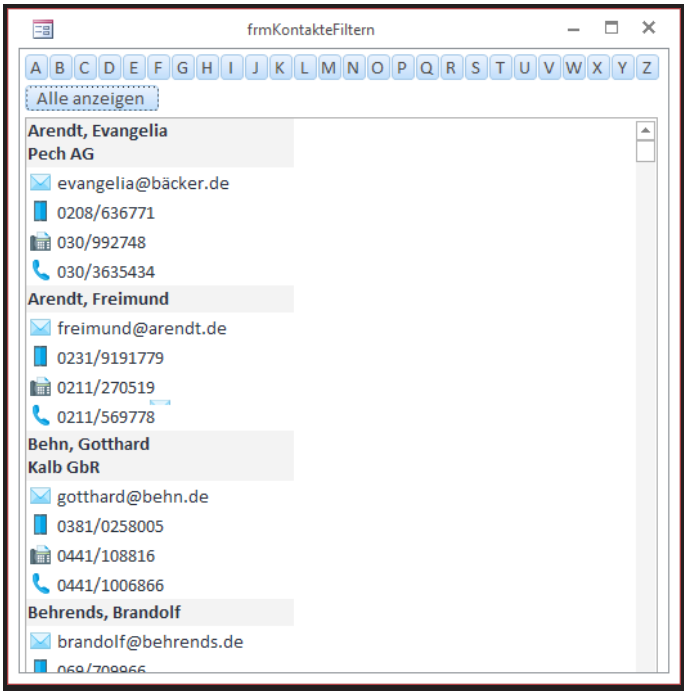


Bild 1: Herkunftsdaten für den Bericht

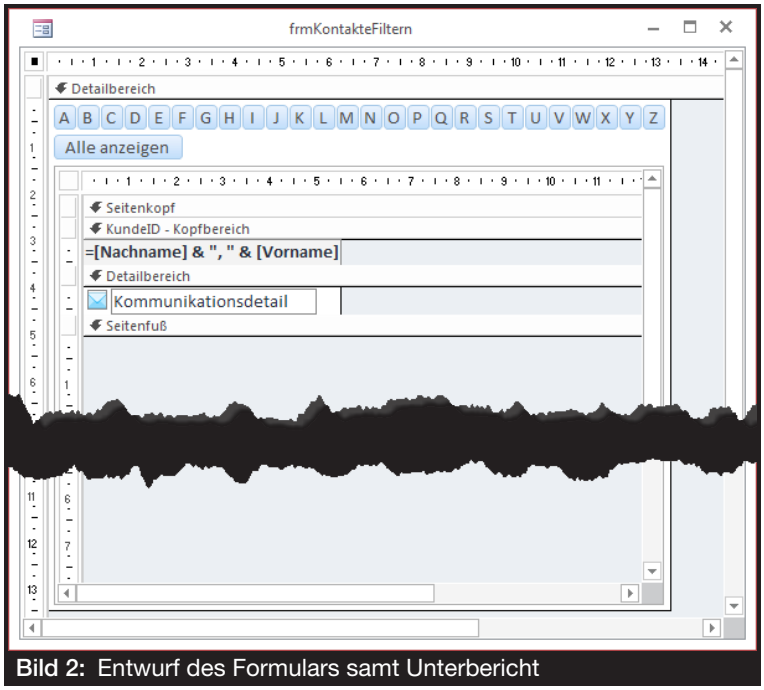


Bild 2: Entwurf des Formulars samt Unterbericht

Da das Hauptformular selbst keine Daten anzeigen soll, stellen Sie die Eigenschaften **Datenblattmarkierer**, **Navigationsschaltflächen**, **Bildlaufleisten** und **Trennlinien** auf **Nein** ein sowie **Automatisch zentrieren** auf **Ja**.

Code des Hauptformulars

Das Hauptformular enthält einige Ereignisprozeduren, die vollständig durch die **Beim Klicken**-Ereignisse der verschiedenen Schaltflächen ausgelöst werden. Den Code finden Sie in Listing 1.

Die Funktion der Schaltflächen zum Filtern

```
Private Sub FilternNach(strVergleichswert As String)
    If Not Nz(DLookup("KundeID", "qryKommunikationsdaten", "Nachname LIKE '" & strVergleichswert & "*'"), 0) = 0 Then
        Me!rptKommunikationsdatenMehrspaltigIcons.Visible = True
        With Me!rptKommunikationsdatenMehrspaltigIcons.Report
            .Filter = "Nachname LIKE '" & strVergleichswert & "*'"
            On Error Resume Next
            .FilterOn = True
            On Error GoTo 0
        End With
    Else
        Me!rptKommunikationsdatenMehrspaltigIcons.Visible = False
    End If
End Sub

Private Sub cmdAlle_Click()
    Me!rptKommunikationsdatenMehrspaltigIcons.Report.Filter = ""
    Me!rptKommunikationsdatenMehrspaltigIcons.Visible = True
End Sub

Private Sub cmdA_Click()
    FilternNach "A"
End Sub

Private Sub cmdB_Click()
    FilternNach "B"
End Sub

Private Sub cmdC_Click()
    FilternNach "C"
End Sub

... weitere Schaltflächen
```

Listing 1: Code, der den ersten Datensatz einer Gruppe ausblendet

der angezeigten Daten nach dem Anfangsbuchstaben des Nachnamens schauen wir uns anhand des Buchstabens **A** an. Ein Klick auf die Schaltfläche **cmdA** löst die Prozedur **cmdA_Click** aus. Diese enthält lediglich eine einzige Anweisung, die eine weitere Prozedur namens **FilternNach** aufruft und dieser den gewünschten Anfangsbuchstaben übergibt.

Diese Prozedur nimmt diesen Buchstaben mit dem Parameter **strVergleichswert** entgegen. Sie prüft zunächst per **DLookup**-Funktion, ob die als Datenherkunft des Unterberichts verwendete Abfrage mindestens einen Datensatz liefert, dessen Nachname mit dem angegebenen Buchstaben beginnt. Ist dies der Fall, blendet die Prozedur den Unterbericht ein und stellt die **Filter**-Eigenschaft auf einen Ausdruck wie **Nachname LIKE 'A*'** ein.

Unter Deaktivierung der Fehlerbehandlung stellt sie noch die Eigenschaft **FilterOn** auf den Wert **True** ein. Dies liefert interessanterweise bei wiederholten Aufruf einen Fehler, was wohl ein Bug ist.

Liefert die Datenherkunft keine passenden Datensätze, blendet die Prozedur das Unterformular-Steuerelement mit dem Bericht einfach aus.

Auf die gleiche Weise arbeiten auch die anderen Schaltflächen zum Filtern der Kontakte nach einem bestimmten Anfangsbuchstaben.

Die Schaltfläche **cmdAlle** hingegen leert die Eigenschaft **Filter** und blendet das eventuell ausgeblendete Unterformular-Steuerelement mit dem Bericht wieder ein.

Zusätzliche Steuerelemente mit MSForms

Zur Entwicklung von Oberflächen für Ihre Datenbanken über Formulare stellt Access an sich ausreichend Steuerelemente zur Verfügung. In manchen Fällen sind jedoch speziellere Elemente gefragt, die man sich dann umständlich aus den vorhandenen zusammenbasteln und programmieren muss. Dabei gibt Access uns die Möglichkeit, seine Oberfläche über den Schatz an ActiveX-Steuerelementen zu erweitern. Die Steuerelemente von MSForms wären dafür ein Beispiel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1601_MSFormsCtls.accdb**

Die Microsoft-Forms-Bibliothek

Wer auch unter **Excel** entwickelt, der wird mit der **MSForms-Bibliothek** sicher schon Bekanntschaft gemacht haben. Für die Interaktion über Dialoge setzt man dort **UserForms** ein, auf denen man Steuerelemente auf ähnliche Weise unterbringen kann, wie in Access-Formularen. Für diese **UserForms**, ihre Steuerelemente und Programmierung ist die **MSForms-Bibliothek** zuständig. Sie besteht aus einer Sammlung von **ActiveX-Steuerelementen**, die grundsätzlich mit **MS Office** installiert werden. Infolgedessen müssen Sie sich über deren Vorhandensein und Installation, im Unterschied zu anderen **ActiveX-Bibliotheken**, keine Gedanken machen. **MSForms** kommt automatisch im Verein mit VBA, in dem es fest verankert ist.

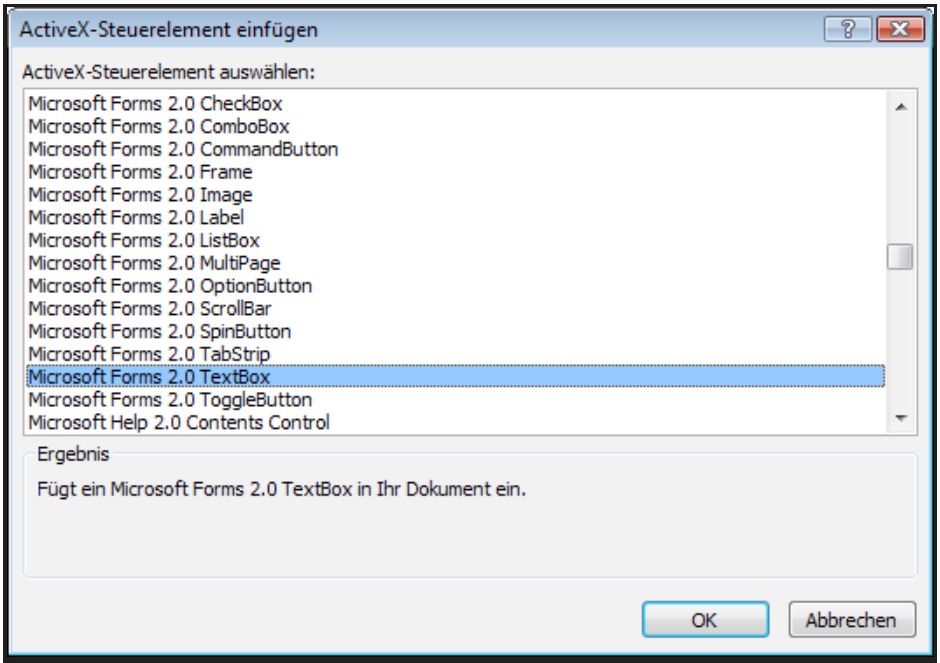


Bild 2: Die Steuerelemente der **MSForms**-Bibliothek Im ActiveX-Auswahldialog

Tatsächlich aber lassen sich die Steuerelemente nicht nur auf **UserForms** platzieren, sondern genauso gut auch direkt auf Access-Formularen oder -Berichten. Dazu öffnen Sie ein etwa ein Formular im Entwurfsmodul, aktivieren den Reiter **Entwurf** im Ribbon, klappen dessen Steuerelementebereich über den Button rechts unten auf, und klicken auf den Eintrag **ActiveX-Steuerelemente** (siehe Bild 1).

Damit öffnet sich ein Dialog zur Auswahl eines ActiveX-Steuerelements (Bild 2). Die Elemente der **MSForms-**

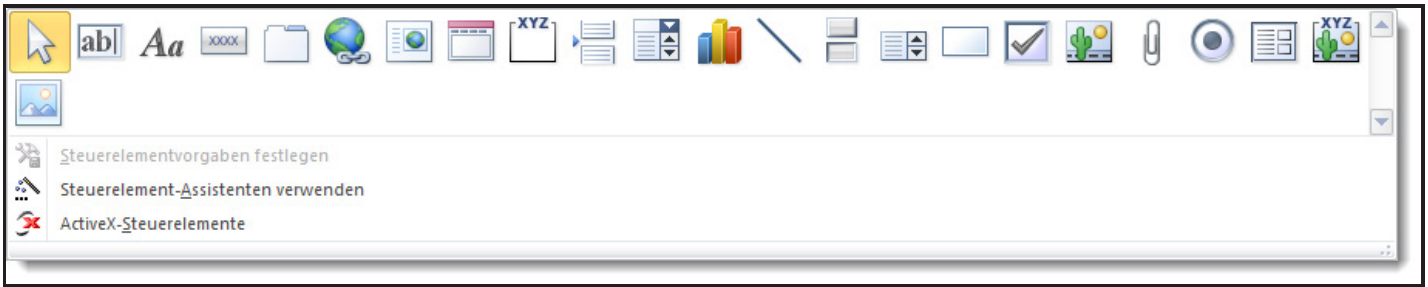


Bild 1: Hinzufügen von ActiveX-Steuerelementen zu einem Formular über den Ribboneintrag **Entwurf|Steuerelemente**

Bibliothek beginnen alle mit dem Präfix **Microsoft Forms**. Markieren Sie dann etwa die **Textbox**, klicken auf **OK**, und haben damit das entsprechende Control im Formular.

Eine Ansammlung der wichtigsten MSForms-Steuerelemente finden Sie im Formular **frmMSForms** der Beispieldatenbank, welches in Bild 3 im Entwurf zu sehen ist. Die Gestaltungsmöglichkeiten für die Steuerelemente sind im Vergleich zu den in Access eingebauten recht bescheiden und etwas altbacken, haben diese Elemente doch schon beinahe 20 Jahre auf dem Buckel. Wer hier keine größeren Ansprüche hat, den wird das nicht stören. Wie dem auch sei, die Eigenschaften der Elemente lassen sich über das **Eigenschaftenblatt** und **Format** regeln, aber auch spezieller über den Reiter **Andere**.

Kleine Erläuterung am Rande: Ein ActiveX-Steuerelement in einem Access-Formular besteht immer aus zwei Teilen. Der eine ist das eigentliche ActiveX-Steuerelement, welches auch im Entwurf seine korrekte Gestalt zeigt. Es liegt jedoch nicht direkt auf dem Formular, sondern auf einem **Container-Steuerelement**, welches Sie im Entwurf weder manuell hinzufügen, noch überhaupt zu Gesicht bekommen können. Im VBA-Objektkatalog ist dieses Container-Element ein Control des Typs **CustomControl**. Und über die **Format**-Seite des Eigenschaftenblatts können Sie dessen Formatierung einstellen. Geben Sie für die **MSForms-Textbox** etwa den **Spezialeffekt Graviert** an, so wirkt sich das tatsächlich auf den Container aus. Die Eigenschaften

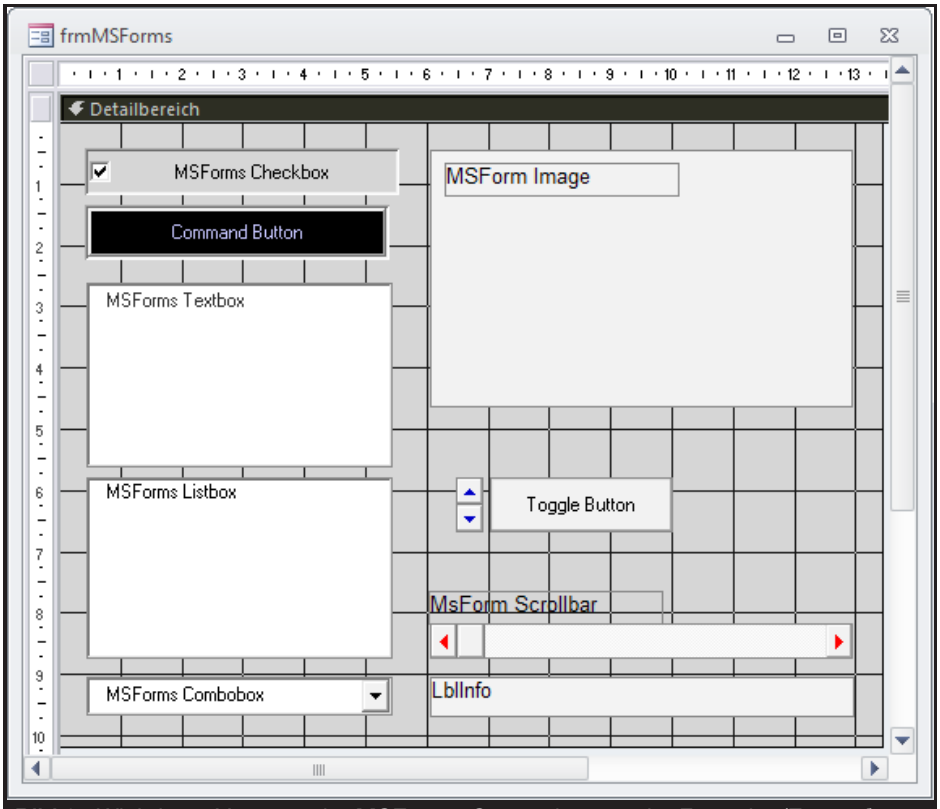


Bild 3: Wichtigste Vertreter der **MSForms**-Steuerelemente im Formular (Entwurf)

der Textbox hingegen finden sich alle unter dem Reiter **Andere**.

Leider sind hier nicht alle Eigenschaften verzeichnet, die einstellbar sind. Microsoft hat im Objektmodell der **MSForms** nur manche Eigenschaften als **Published** deklariert. So lässt sich etwa die Schriftart der Steuerelemente nie über die Eigenschaftenseite **Andere** festlegen. Das geht hier leider nur über VBA-Code. Um die Schriftgröße der **MSForms-Textbox**, welche im Formular den Namen **ctlTextbox** trägt, auf 10 einzustellen, verwenden Sie diese Zeile:

```
Me!ctlTextbox.FontSize = 10
```

Das klappt in den meisten Fällen, jedoch nicht immer. Denn **FontSize** ist eine Eigenschaft des MSForms-Steuerelements, während **Me!ctlTextbox** tatsächlich auf den Container, das **CustomControl** verweist.

Access schleift die Eigenschaft zwar gleichsam durch, doch besser ist es, direkt auf das ActiveX-Steuerelement Bezug zu nehmen. Das ist durch die Eigenschaft **Object** des **CustomControls** gegeben:

```
Me!ctlTextbox.Object.FontSize = 10
```

Möchten Sie hintereinander mehrere Eigenschaften per Code festlegen, so empfiehlt sich ein **With**-Block:

```
With Me!ctlTextbox.Object
    .FontName = "Calibri"
    .FontSize = 10
    .FontBold = True
End With
```

Welche Eigenschaften ein MSForms-Steuerelement beherbergt, ersehen Sie aus dem VBA-Objektkatalog. Bringen Sie diesen zur Ansicht und wählen im Kombinationsfeld oben links die Bibliothek **MSForms** aus. In der Liste links navigieren Sie zur **Textbox**, um auf der rechten Seite deren Methoden zu begutachten. Glücklicherweise hat Microsoft den Elementen eine Hilfedatei spendiert, die erstaunlicherweise noch immer funktioniert. Markieren Sie also eine Eigenschaft der **Textbox** im Objektkatalog und drücken **F1**, so öffnet sich die Hilfe dazu. Alles zu **MSForms** findet sich unter dem Zweig **Microsoft Forms Referenz** (Bild 4). Alternativ finden Sie eine **CHM**-Hilfedatei im Explorer unter dem Pfad **C:\Program Files\Common Files\microsoft shared\VBA\VBA6\1031\fm20.chm**. (Der Pfad ist unter **Windows 64bit** natürlich etwas anders.) Diese ist noch etwas übersichtlicher aufgebaut, als die Ausführungen in der Access-Hilfe.

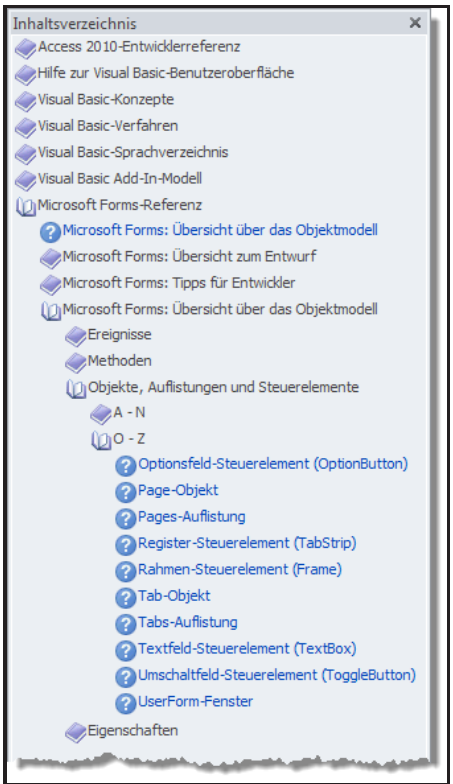


Bild 4: Die **MSForms**-Bibliothek und ihre Steuerelemente in der Access-Hilfe

Das programmatische Setzen der Eigenschaften erledigen Sie am besten gleich beim Öffnen des Formulars. In Listing 1 ruft die Ereignisprozedur **Beim Laden** des Formulars nacheinander die Formatierungsprozeduren auf. **FormatButton** etwa stellt die Schriftart ein und verabreicht der **MSForms-Schaltfläche** außerdem ein Bildchen, das über die Eigenschaft **Picture** gesetzt wird. Das Bild in Gestalt des Icons **info.ico** befindet sich parallel im Pfad zur Datenbankdatei und kann mit **LoadPicture** geladen werden. Wo das Bildchen auf dem Button angezeigt wird, lässt sich übrigens direkt über die Eigenschaftenseite **Andere** unter **PicturePosition** angeben. Hier existieren 13 Positionierungsmöglichkeiten. Im Entwurf kann das Aussehen des **Buttons** so leider nicht kontrolliert werden, es sein denn, Sie würden etwa diese Zeile im Direktfenster von VBA absetzen, während sich das Formular im Entwurf befindet:

```
Set Forms!frmMSForms!ctlCmtButton.
Picture = _
     LoadPicture(sPath & "\info.ico")
```

Zur Laufzeit präsentiert sich das Formular dann wie in Bild 5. Wenden wir

```
Dim sPath As String

Private Sub Form_Load()
    sPath = CurrentProject.Path
    FormatButton
    FormatTextbox
    FormatImage
    FormatToggle
    FormatCheckbox
    FormatListbox
    FormatCombo
End Sub

Private Sub FormatButton()
    With Me!ctlCmdButton.Object
        .FontName = "Arial"
        .FontSize = 10
        .FontBold = True
        Set .Picture = LoadPicture(sPath & "\info.ico")
    End With
End Sub

Private Sub FormatTextbox()
    With Me!ctlTextBox.Object
        .FontSize = 11
        .FontName = "Calibri"
    End With
End Sub
```

Listing 1: Die Formatierung der einzelnen **MSForms**-Steuerelemente geschieht aus dem **Load**-Ereignis des Formulars heraus

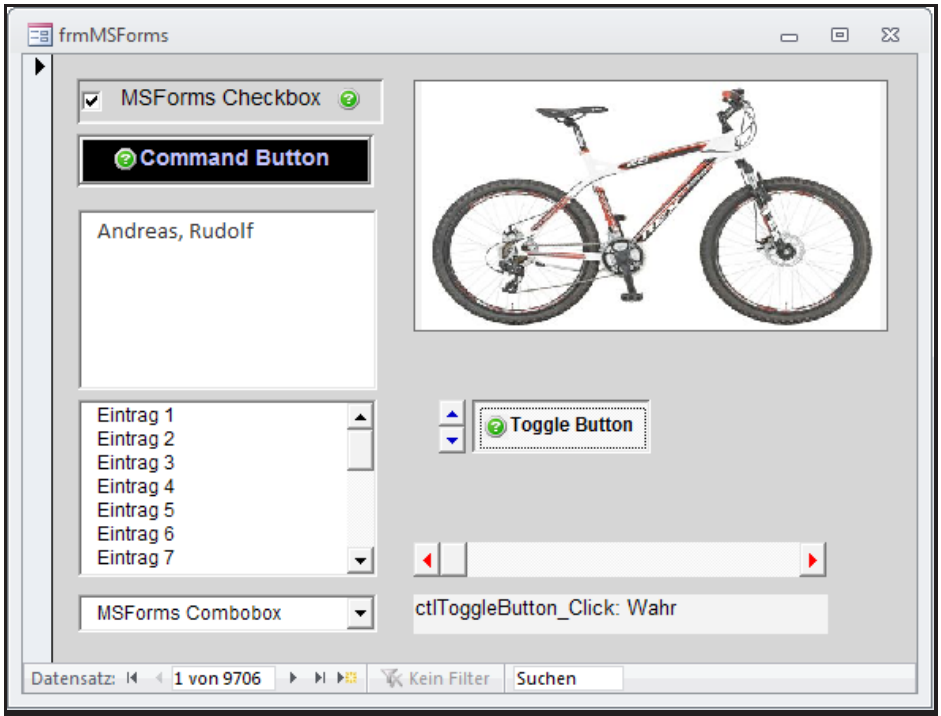


Bild 5: Das Formular **frmMSForms** in der Formularansicht zur Laufzeit

uns den wichtigsten Eigenschaften und der Programmierung der Steuerelemente zu.

Interessant ist, dass alle die Eigenschaft **Value** aufweisen, die den Wert des Elements zurückgibt oder ihn bestimmen kann. Der Datentyp unterscheidet sich dabei natürlich. Bei einer **Textbox** wäre ein String zuzuweisen, bei einer **Checkbox** ein **Boolean**-Wert. Sie müssen diesen Wert jedoch nicht zwingend per Code übergeben. Denn fast alle im Bild vereinten Steuerelemente, mit Ausnahme des **Labels** rechts unten und des **MSForms-Image** oben, sind datensensitiv! Sie erkennen das im Entwurf, wenn Sie ein Steuerelement markieren und im Eigenschaftenblatt den Reiter **Daten** inspizieren. So oben die Zeile **Steuerelementinhalt** existiert, ist eine Bindung an die Datenherkunft des Formulars möglich. **Andreas, Rudolf** kommt so durch die Bindung der **Textbox** an das Feld **Adresse** der Abfrage zustande, auf welcher das Formular basiert. Die Abfrage besteht übrigens aus diesem SQL-Ausdruck:

```
SELECT tblAdressen.ID,
    [Nachname] & ", " & [Vorname] &
    Chr(13) & Chr(10) &
    [Strasse] & Chr(13) & Chr(10) &
    [PLZ] & "  " & [Ort]
    AS Adresse
    FROM tblAdressen;
```

Die Adresse wird über die künstlichen Zeilenumbrüche so formatiert, wie es gemeinhin üblich ist.

MSForms Image

Das Bildsteuerelement der Microsoft-Forms-Bibliothek hat gegenüber dem Access-eigenen Element einen gewichtigen Vorteil: Es kennt die Objekteigenschaft **Picture**, welche vom Typ **Stdole.StdPicture** ist. Dieser Typ wird etwa von der Methode **LoadPicture** zurückgegeben und auch von zahlreichen anderen VBA-Funktionen, in denen es um Bildverarbeitung geht. Das Fahrrad wird über diese Zeile geladen:

```
With Me!ctlImage.Object
  Set .Picture = _
  LoadPicture(sPath & "\fahrrad.jpg")
End With
```

Für den **PictureSizeMode** können Sie im Eigenschaftenblatt des **Image-Controls** entweder **Clip**, **Stretch** oder **Zoom** angeben, was zu Deutsch Abschneiden, Ausdehnen oder Einpassen bedeutet. Bei **Clip** lässt sich über PictureAlignment zusätzlich bestimmen, wo das Bild platziert werden soll (Mitte, Rechts, Unten, et cetera). Und Kacheln lässt es sich auch, wenn **PictureTiling** auf **True** steht.

MSForms Command Button, Toggle Button, Checkbox

Diese drei Elemente weisen etliche Gemeinsamkeiten auf. Die Schriftart wird über die **Font**-Eigenschaften eingestellt, die Schriftfarbe über **ForeColor** und die Hintergrundfarbe über **BackColor**, welche beide einen **RGB**-Wert entgegennehmen. Daneben kann über die **Picture**-Eigenschaft noch ein Bild angezeigt werden, wobei sich hierfür vor allem **Icon**-Dateien eignen, die Transparenz aufweisen. Denn ein **PNG**-Bild etwa lässt sich mit **LoadPicture** nicht laden. Diese Methode kann nur mit **BMP**, **JPG** und **ICO** umgehen.

Ansonsten zeigen die drei Elemente keine Besonderheiten, weshalb sie

Bubble Sort

Solange Sie mit Datenzugriffsobjekten hantieren, ist das Sortieren von Daten eine einfache Angelegenheit. Diese übernehmen Sortierungen etwa über die SQL-Anweisung **OrderBy** oder dezidierte Sort-Methoden. Bei in Arrays untergebrachten Datenmengen unter VBA lässt Sie Access jedoch im Regen stehen. Hier kommen Sie um eigens programmierte Routinen nicht herum.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1601_Klassenmodule.accdb** im VBA-Modul **mdlSort**.

Abstract

Hin und wieder kommt es vor, dass Sie Daten nicht nur über **Recordsets** verwalten müssen, sondern auch über **Arrays**. Ein Beispiel wäre das Einlesen externer Textdateien, die tabellarischen Aufbau aufwiesen. In der Regel öffnen Sie solche Dateien über die **Open**-Anweisung und speichern ihren Inhalt in eine String-Variable. Die **Split-Methode** ermöglicht anschließend das Auftrennen der Zeilen in die Elemente eines String-Arrays. Möchten Sie diese Elemente nun nach einem bestimmten Kriterium sortieren, so ist eine Sortierroutine gefragt.

Es existiert ein Unmenge von Sortieralgorithmen mit Namen, wie **QuickSort**, **HeapSort**, **MergeSort**, **ShellSort** oder **InsertionSort**, die in der Regel zwar meist mit nur wenigen Zeilen Programmcode auskommen, jedoch häufig in ihrer Funktionalität nur schwer zu durchschauen sind. Anders beim **BubbleSort**-Algorithmus, der zwar in Hinsicht auf Performance der mit Abstand ineffizienteste ist, dafür jedoch umso leichter zu verstehen. Sollten Sie nicht Hunderttausende Elemente sortieren wollen, so spielt die Performance auf heutigen Rechnern in diesem Fall keine Rolle mehr.

BubbleSort

Die Sache ist recht simpel: Sie haben etwa ein eindimensionales Array mit allerlei Zeichenfolgen vor sich, die in aufsteigende Reihenfolge gebracht werden sollen. Dazu vergleicht BubbleSort jedes Element mit jedem und entscheidet, welches größer oder kleiner ist. Trifft dies zu, so vertauscht es die beiden, andernfalls nicht. Ein Beispiel-Listing sagt hier mehr, als viele Worte.

Listing 1 stellt die Routine **SortArray** vor, wie Sie sie im Modul **mdlSort** der Beispieldatenbank vorfinden. Sie übergeben der Prozedur ein Array mit Elementen, die keinesfalls nur

```
Public Enum eSortArray
    eSortAscending = 1
    eSortDescending = -1
End Enum

Sub SortArray(MyArray As Variant, Optional Order As eSortArray = eSortAscending)
    If IsEmpty(MyArray) Then Exit Sub

    Dim i As Long, j As Long
    Dim n As Long
    Dim tmp As Variant
    Dim ret As Boolean

    n = UBound(MyArray)
    For i = 0 To n - 1
        For j = i + 1 To n
            If Order = eSortAscending Then
                ret = (MyArray(i) < MyArray(j))
            Else
                ret = (MyArray(i) > MyArray(j))
            End If
            If ret = 0 Then
                tmp = MyArray(i)
                MyArray(i) = MyArray(j)
                MyArray(j) = tmp
            End If
        Next j
    Next i
End Sub
```

Listing 1: Implementation des Bubble-Sort-Algorithmus in der Prozedur **SortArray**

Klassenmodule und benutzerdefinierte Typen

Die allermeisten Prozeduren und Funktionen des VBA-Projekts Ihrer Datenbank werden Sie in ganz normalen Modulen unterbringen. Dabei wird leicht übersehen, dass es sich etwa bei Formularen um eine andere Art Module handelt, nämlich um Klassenmodule. Diese können Sie aber auch selbst anlegen. Erfahren Sie hier, wie Sie dabei vorgehen. Wir leiten das Thema zunächst jedoch mit den benutzerdefinierten Typen ein, welche Klassenmodule verständlicher machen.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1601_Klassenmodule.accdb**

Benutzerdefinierte Typen
Ein benutzerdefinierter Typ ist nichts weiter, als eine zusammengefasste Struktur von Variablen.

Interessanterweise findet man Beispiele dafür im VBA-Objektkatalog weder in der Access-Bibliothek, noch unter den VBA- oder Office-Bibliotheken. Also stellen wir gleich eine eigene Kreation vor:

```
Public Type TAdresse
    ID As Long
    Nachname As String
    Vorname As String
    Strasse As String
    PLZ As String
    Ort As String
End Type
```

Der reservierte Begriff **Type** definiert, dass unter dem folgenden Namen **TAdresse** eine Variablenstruktur folgt. Solche Typen können im Gültigkeitsbereich auch mit **Private** gekennzeichnet sein, was sie nur für das Modul zugreifbar macht, in dem sich die Deklaration befindet. **Public** bedeutet hier, dass nun das gesamte VBA-Projekt den Typen kennt und auch aus anderen Modulen heraus verwendet werden kann.

Die einzelnen Variablen des Typs werden in den folgenden Zeilen durch Namen und Variablentypen definiert. Jeglicher Datentyp ist hier möglich – etwa auch **As Object**.

Wozu ist das gut? Kann man die Elemente der Struktur nicht auch als einzelne Variablen deklarieren? Es gibt sicher viele Fälle, für die die Antwort hier Ja lautet. Doch schauen wir uns an, welche Vorteile der benutzerdefinierte Typ bietet. Denn er verhält sich genau so, wie jeder andere Datentyp. Also kann er einer Variablen zugeordnet werden:

```
Dim A As TAdresse
```

Der Zugriff auf die Elemente des Typs geschieht über Punkt und Namen:

```
A.Nachname = "Minhorst"
A.Vorname = "André"
A.Ort = "Duisburg"
Debug.Print A.Nachname
```

Der Code lässt sich noch über die **With**-Anweisung vereinfachen:

```
With A
    .Nachname = "Minhorst"
    .Vorname = "André"
    .Ort = "Duisburg"
    Debug.Print .Nachname
End With
```

Das wäre schon einmal Vorteil Nummer eins. Nach Eingabe von **A** und Punkt stellt das **IntelliSense** von VBA gleich alle Elemente des Typs in der Auswahlliste vor. Und das übrigens auch innerhalb des **With**-Blocks, wo allein die Punkt-Eingabe für das Auslösen von **IntelliSense** ausreicht.

Besonders nützlich sind benutzerdefinierte Typen jedoch dann, wenn Sie in Arrays Verwendung finden:

```
Dim Adressen() As TAdresse
ReDim Adressen(99)
Adressen(0).Nachname = "Minhorst"
Adressen(0).Vorname = "André"
Adressen(1).Nachname = "Trowitzsch"
Adressen(1).Vorname = "Sascha"
...
```

Das Array **Adressen** ist vom Typ **TAdresse**. Es wird im ersten Schritt auf Hundert Elemente dimensioniert. Danach folgt die Zuweisung der Werte einerseits über die Indizes des Arrays, andererseits über den Namen der Elemente. Wollte man das über separate Variablen erreichen, so benötigte man für jedes Element des Typs ein eigenes gesondertes Array. Das machte die Angelegenheit

deutlich unüberschaubarer, als es das Handling mit einem Array des benutzerdefinierten Typs erlaubt.

Die Sache lässt sich noch weiterspinnen. Durch den Fakt, dass ein Typelement Variablen beliebigen Datentyps aufweisen darf, kommt für dieses selbst ein benutzerdefinierter Typ infrage:

```
Public Type TAdressen
    Items() As TAdresse
End Type
```

Zwar enthält der Typ nur ein Element, welches jedoch als Array des benutzerdefinierten **TAdresse** ausgelegt ist. Damit ist nur eine Variable vom Typ **TAdressen** imstande mehrere Adressen zu speichern:

```
Dim A As TAdressen
ReDim A.Items(99)
With A
    .Items(0).Nachname = "Minhorst"
    .Items(0).Vorname = "André"
    .Items(1).Nachname = "Trowitzsch"
    .Items(1).Vorname = "Sascha"
    ...
End With
Debug.Print A.Items(1).Vorname
```

Auch hier führt der benutzerdefinierte Typ zu strukturierterer Programmierung. Treiben wir es noch einen Schritt weiter. Es lägen etwa Adressen für verschiedene Anwendungsbereiche vor, seien es Kundenadressen, Ansprechpartner, Behörden, Firmen.

Dann könnten alle Adressen zusammen wiederum in einen neuen Typ überführt werden:

```
Public Enum eAddressType
    eAddressUnspecific = 0
    eAddressCustomer = 1
    eAddressPartner = 2
    eAddressCompany = 3
End Enum

Public Type TAdressenGesamt
    AddressType As sAddressType
    AddressBlock As TAdressen
End Type
```

Die Zuordnung der Werte wird nun zwar etwas länglich, ist aber dennoch gut zu überblicken:

```
Dim A() As TAdressenGesamt

ReDim A(3)

A(1).AddressType = eAddressCustomer
Redim A(1).AdressBlock.Items(9)
With A(1).AdressBlock
   .Items(0).Nachname = "Minhorst"
   .Items(0).Vorname = "Minhorst"
   .Items(1).Nachname = "Trowitzsch"
   .Items(1).Vorname = "Sascha"
    ...
End With
A(2).AddressType = eAddressUnspecific
Redim A(2).AdressBlock.Items(9)
With A(2).AdressBlock
   .Items(0).Nachname = "Häberle"
   .Items(0).Vorname = "Adam"
   .Items(1).Nachname = "Gutschmidt"
   .Items(1).Vorname = "Andrea"
    ...
End With
...
```

Immerhin haben wir nun mehrere Adressblöcke unterschiedlichen Umfangs und Typs in einer Variablen vereint! Das einzig hinderlich ist die Notwendigkeit, die **Items**-Arrays jeweils neu zu dimensionieren. Das ließe sich mit einer festen Anzahl von Elementen umgehen:

```
Public Type TAdressen
    Items(99) As TAdresse
End Type
```

Damit läge die Maximalzahl von Adressen für einen Block bei Hundert. Probleme gibt stellen sich also ein, wenn doch mehr benötigt würden. Und bei kleinerer Anzahl würde außerdem Speicherplatz vergeudet. Dieses Beispiel sollte eigentlich nur eines demonstrieren, nämlich wie sich benutzerdefinierte Typen im Interesse von Strukturierung verschachteln lassen. Das Ganze geht in Richtung Objektorientierter Programmierung, mit der wir es später bei den Klassenmodulen noch mehr zu tun bekommen.

Tabellendaten in benutzerdefinierten Typen

Damit es nicht bei trockener Theorie bleibt, folgt ein Beispiel, das in der einen oder anderen Datenbank durchaus vertreten sein kann. Es geht hier darum, die Datensätze einer Tabelle oder Abfrage in VBA-Strukturen zu überführen.

Listing 1 zeigt das Grundgerüst mit der Prozedur **TestUDT**. Nebenbei erwähnt ist **UDT** das Kürzel für einen benutzerdefinierten Typ (**User Defined Type**). Eine Datensatzgruppe **rs** wird auf alle Datensätze der Tabelle **tblAdressen** geöffnet. In einer **Do-Loop**-Schleife werden sie alle durchlaufen und die einzelnen Feldinhalte den Elementen der Variablen **T** des Typs **TAdresse** verabreicht. Da

mache Felder der Datensätze leer sind und **Leer** nicht in einem String gespeichert werden kann – Strasse ist etwa vom Typ **String** – müssen die Werte noch einer Vorbehandlung mit der Funktion **Nz** unterzogen werden, die aus Nullwerten ordnungsgemäße Strings der Länge 0 macht.

Viel Sinn macht das Beispiel nicht. Schließlich landen alle Daten in der einen Variablen **T** und überschreiben sich damit bei jedem Schleifendurchlauf. In Listing 2 ist die Routine aufgebohrt und um den Typ **TAdressen** erweitert, welcher, wie bereits weiter oben dargestellt, mehrere Adressen aufnehmen kann. Die **Items** des Typs **T**, also die einzelnen Adressen, müssen zunächst auf die Zahl der Datensätze des Recordsets dimensioniert werden. Sie erhalten diese über die Methode **RecordCount**, nachdem Sie den Daten-Cursor über **MoveLast** einmal auf das Ende der Datensätze einstellen. Da das Array **Items** nullbasiert ist, muss die Obergrenze auf **RecordCount - 1** festgelegt werden. In der folgenden **Do-Loop**-Schleife wird auf eine Adresse nun über den Array-Index **i** Bezug genommen, wobei **T.Items(i)** als Spezifikation des **With**-Blocks dient. Die Zählervariable **i** wird nach erfolgtem **MoveNext** innerhalb der Schleife erhöht. Im Endergebnis haben Sie damit alle Datensätze der Tabelle in der einen Variablen **T**.

Was damit dann anstellen, ist ein anderes Thema. Beispiel wäre etwa das Abspeichern in eine Textdatei. Tatsächlich lassen sich Variablen eines benutzerdefinierten Typs über nur eine einzige Zeile exportieren. Sie

```
Sub TestUDT()
    Dim rs As DAO.Recordset
    Dim T As TAdresse

    Set rs = CurrentDb.OpenRecordset("SELECT * FROM tblAdressen", dbOpenDynaset)
    Do While Not rs.EOF
        With T
            .ID = rs!ID.Value
            .Nachname = Nz(rs!Nachname.Value)
            .Vorname = Nz(rs!Vorname.Value)
            .Strasse = Nz(rs!Strasse.Value)
            .PLZ = Nz(rs!PLZ.Value)
            .Ort = Nz(rs!Ort.Value)
        End With
        rs.MoveNext
    Loop
    rs.Close
End Sub
```

Listing 1: Hier werden die Daten der Tabelle **tblAdressen** hintereinander in einer Variablen **T** des benutzerdefinierten Typs **TAdressen** abgespeichert

```
Sub TestUDTArray()
    Dim rs As DAO.Recordset
    Dim T As TAdressen
    Dim i As Long

    Set rs = CurrentDb.OpenRecordset("SELECT * FROM tblAdressen", dbOpenDynaset)
    rs.MoveLast: rs.MoveFirst
    ReDim T.Items(rs.RecordCount - 1)
    Do While Not rs.EOF
        With T.Items(i)
            .ID = i
            .Nachname = Nz(rs!Nachname.Value)
            .Vorname = Nz(rs!Vorname.Value)
            .Strasse = Nz(rs!Strasse.Value)
            .PLZ = Nz(rs!PLZ.Value)
            .Ort = Nz(rs!Ort.Value)
        End With
        rs.MoveNext: i = i + 1
    Loop
    rs.Close
End Sub
```

Listing 2: Hier kommt der benutzerdefinierte Typ **TAdressen** zum Tragen

könnten die Routine von Listing 2 am Ende mit diesen Zeilen erweitern:

```
Dim F As Integer
F = FreeFile
Open "c:\daten\tabelle.txt" For Binary
As F
Put F,, T
Close F
```

Mit der dergestalt exportierten Textdatei kann allerdings kein anderes Programm etwas anfangen. Sie lässt sich aber analog in einem VBA-Programm einlesen, das den Typ **TAdressen** kennt; sinngemäß:

```
Dim T As TAdressen
Get F,, T
```

Klassenmodule

Recht grundlos fristen Klassenmodule unter VBA ein Nischendasein. Entweder wird der Umgang mit ihnen immer wieder als kompliziert dargestellt, oder aber der Nutzen in Frage gestellt. Schließlich lässt sich mit den normalen Modulen dasselbe bewerkstelligen? Dem ist aber nicht so. Grund genug, sich einmal deren Grundlagen zu widmen.

Rein mechanisch fügen Sie Ihrer Datenbank ein Klassenmodul hinzu, indem Sie den gleichnamigen Button im **Ribbon** unter der Registerseite **Erstellen** anklicken, oder, alternativ, im **Projekt-Explorer** der VBA-Umgebung und das Kontextmenü **Einfügen | Klassenmodul** betätigen. Bei Speichern vergeben Sie dem Modul einen Namen. Es hat sich eingebürgert, hier ein abweichendes Präfix zu verwenden. Versieht man normale Module meist mit dem Präfix **mod** oder **mdl**, so kommt hier eher ein **cls** oder schlicht **C** zum Einsatz. Ist das Thema des Klassenmoduls die Verwaltung von Adressen, so verwenden Sie etwa **clsAdressen**.

Vom programmtechnischen Standpunkt aus unterscheidet sich ein Klassenmodul nicht von normalen Modulen. Es existiert lediglich eine kleine Erweiterung, das Event, auf welches wir noch zu sprechen kommen. Der Unterschied besteht vielmehr darin, dass ein Klassenmodul eben eine Klasse zur Verfügung stellt, während normale Module eine Sammlung von allgemein verwendbaren Funktionen verzeichnet. Die Methoden der Klasse können aber nicht direkt aufgerufen werden. Eine Klasse ist nämlich lediglich der Bauplan für ein **Objekt**! Ein **Objekt** stellt eine **Instanz** einer Klasse dar, welche erst per **New**-Anweisung erzeugt werden muss, um an die Funktionen der Klasse zu kommen:

```
Dim C As clsAdressen
Set C = New clsAdressen
```

oder einfach

```
Dim C As New clsAdressen
```

Über **C** gelangen Sie dann an die Methoden der Klasse. Ähnlich, wie beim benutzerdefinierten Typ, zeigt **IntelliSense** die Methoden der Klasse, wenn Sie einen Punkt hinter die Variable setzen. Enthielte die Adressenklasse eine Prozedur **FindeNachnamen**, so riefen Sie jene auf diese Weise auf:

```
C.FindeNachname
```

Der Umstand, dass Klassenmodule erst auf dem Umweg über deren Instanziierung ansprechbar werden, macht sie wahrscheinlich unbeliebt, ist doch der Aufwand höher, als bei normalen Modulfunktionen.

Es gibt an anderer Stelle erschöpfende Darlegungen über Klassen, Objekte, Instanziierung und objektorientierte Programmierung. Da viele der Charakteristiken von Objektstrukturen unter VBA jedoch nicht implementiert sind, lassen wir es für den Anfang bei einigen grundlegenden Erläuterungen.

Ein Schlagwort bei Klassen ist **Kapselung**. Dadurch, dass eine Klasse erst dezidiert über eine Variable instanziiert werden muss, kapselt sich deren Funktionalität und schottet den Zugriff im Vergleich zu normalen Modulen ab. Die Methoden sind nur Teil des Objekts und damit nicht generell verfügbar.

Deshalb eignet sich eine Klasse und deren Objekt auch für die Aufnahme von Daten. Nicht zufällig hatten wir es eingangs mit benutzerdefinierten Objekten zu tun. **UDT**-Variablen sind häufig öffentlich ansprechbar und damit unsicherer, als ein Objekt.

Container-Klassen

Der benutzerdefinierte Typ **TAdresse** lässt sich problemlos in eine Klasse verwandeln und anschließend auch ähnlich handhaben. Tatsächlich haben Klassen und benutzerdefinierte Typen einige Gemeinsamkeiten. Eine Klasse kann Imgrunde als benutzerdefinierter Typ angesehen werden, der neben den statischen Variablen auch noch Methoden, also Prozeduren, aufweisen kann.

Eine Klasse **clsAdresse** sollte also die Möglichkeit bieten, den **Nachnamen**, **Vornamen** und **Ort** einer Adresse speichern und wiedergeben zu können. Das ist auf einfache Weise möglich:

```
Public ID As Long
Public Nachname As String
Public Vorname As String
Public Strasse As String
Public PLZ As String
Public Ort As String
```

Das wäre dann auch schon der komplette Code des Klassenmoduls

clsAdresse! So sprechen Sie dann die Eigenschaften an:

```
Dim C As clsAdresse
Set C = New clsAdresse
C.Nachname = "Minhorst"
C.Vorname = "André"
Debug.Print C.Nachname
```

Bis auf die Instanziierung entspricht das Verhalten hier also völlig dem benutzerdefinierten Typ.

Allerdings entspricht es nicht dem Prinzip der Kapselung, die Eigenschaften als öffentliche Variablen auszuführen. In der Regel geht man stattdessen den Weg über die sogenannten Properties und speichert indirekt in private Variablen. Properties verhalten sich nach außen wie Variablen, sind tatsächlich aber Funktionen. Hier ein Beispiel, das Sinn und Zweck verdeutlicht:

```
Private m_Nachname As String

Property Get Nachname() As String
  Nachname = m_Nachname
End Property

Property Let Nachname(Value As String)
  If Value = "Minhorst" Then
      Err.Raise 1,, _
         "Unerlaubter Nachname"
  End If
  m_Nachname = Value
End Property
```

m_Nachname ist die Variable, die tatsächlich den String speichert. Auf sie lässt sich aber nicht direkt im Adressenobjekt zugreifen, da sie als **Private** deklariert ist. Stattdessen kommt hier die Eigenschaft, das **Property**, **Nachname** zum Zug. Das **Property** teilt sich in zwei Eigenschaftenprozeduren auf, einmal die zum Zuweisen eines Werts (**Let**) und dann die zum Auslesen des Werts (**Get**). Nach außen sehen die beiden Prozeduren aus, wie eine. Voraussetzung ist allerdings, dass beide den gleichen Namen tragen und den gleichen Datentyp bearbeiten. Das **Get-Property** ist Imgrunde eine Funktion, die lediglich den Inhalt der Variablen **m_Nachname** zurückgibt. Das **Let-Property** ist eine **Sub**-Prozedur, der als Parameter der Eigenschaftswert übergeben wird. Dieser wird wiederum in der privaten Variablen **m_Nachname** abgespeichert.

Sie sprechen die Eigenschaft genau so an, wie im Beispiel oben angegeben. **C.Nachname** verhält sich wie eine Variable, die gelesen und beschrieben werden kann. Indessen wird es bei Ausführung des Codes zu einer Fehlermeldung kommen. Das jedoch nicht infolge falscher Programmierung, sondern deshalb, weil dieses Feature eingebaut wurde. Wird als Eigenschaftswert nämlich **Minhorst** übergeben, so zweigt die **Let-Prozedur** in den entsprechenden Zweig ab und ruft per **Err.Raise** ausdrücklich eine Fehlermeldung hervor. **Minhorst** lässt sich also im Klassenobjekt nicht speichern.

Hier wird deutlich, dass der Umweg über die Eigenschaftenprozeduren die Möglichkeit der Weiterverarbeitung, Überprüfung und Modifikation von Werten bietet. Der eigentliche Wert in **m_Nachname** ist eingekapselt. Das alles ist mit Variablen auf benutzerdefinierten Typen nicht möglich. Die Container-Klasse **clsAdresse** finden Sie auszugsweise in Listing 3.

```
Private m_ID As Long
Private m_Nachname As String
Private m_Vorname As String
Private m_Strasse As String
Private m_PLZ As String
Private m_Ort As String

Property Get Nachname() As String
    Nachname = m_Nachname
End Property
Property Let Nachname(Value As String)
    m_Nachname = Value
End Property

Property Get Vorname() As String
    Vorname = m_Vorname
End Property
Property Let Vorname(Value As String)
    m_Vorname = Value
End Property

Property Get Strasse() As String
    Strasse = m_Strasse
End Property
Property Let Strasse(Value As String)
    m_Strasse = Value
End Property
```

Listing 3: Klassenmodul **clsAdresse**

Enumerationsklassen

Hatten wir im Thema benutzerdefinierte Typen einen zusätzlichen Typ **TAdressen** definiert, der die Möglichkeit vorsah, mehrere Adressen in einer Variablen zu speichern, so gibt es bei der Klassenprogrammierung natürlich ebenfalls ein Pendant. Während im vorigen Fall aber **Arrays** zum Einsatz kamen, nimmt man bei Klassen eher **Collections**, um eine Enumeration oder Auflistung zu erstellen.

Listing 4 gibt die wesentlichen Teile der Klasse **clsAdressen** wieder. Hier bekommen wir es mit einem Sachverhalt zu tun, der Klassenmodule

Horoskopgenerator

Nein, um jene Horoskope, die Sie gemeinhin in Zeitschriften finden, geht es hier nicht. Wir stellen eine Datenbanklösung vor, die echte Geburtshoroskope auf astrologischer Basis erzeugen kann, wobei eine Deutung dieser allerdings nicht stattfindet. Ob Sie daran glauben, oder nicht, spielt weniger eine Rolle. Hier geht es um einige Techniken, die sich auch in anderen Datenbanken einsetzen lassen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1602_Horoskope.accdb**

Geburtshoroskope

In Bild 1 finden Sie gleich ein Beispiel für ein Geburtshoroskop, wie es das Formular **frmHoroskope** der Beispieldatenbank generiert. Im Block links oben geben Sie Daten einer Person ein, zu der das Horoskop erstellt werden soll. Das sind neben dem Namen das **Geburtsdatum** und die **Geburtszeit**, und schließlich noch der **Geburtsort**. Aus diesem berechnet die Datenbank mithilfe der Tabelle **tblOrte** den **Längen**- und **Breitengrad**, die Sie gegebenenfalls aber auch manuell eingeben können. Ein Klick auf die Schaltfläche **Update** berechnet nun die Positionen der Horoskopelemente, listet diese tabellarisch auf, und zeigt sie in der Grafik rechts im Formular an. Sollte die Materie für Sie neu sein, wovon wir einmal ausgehen, so folgen hier

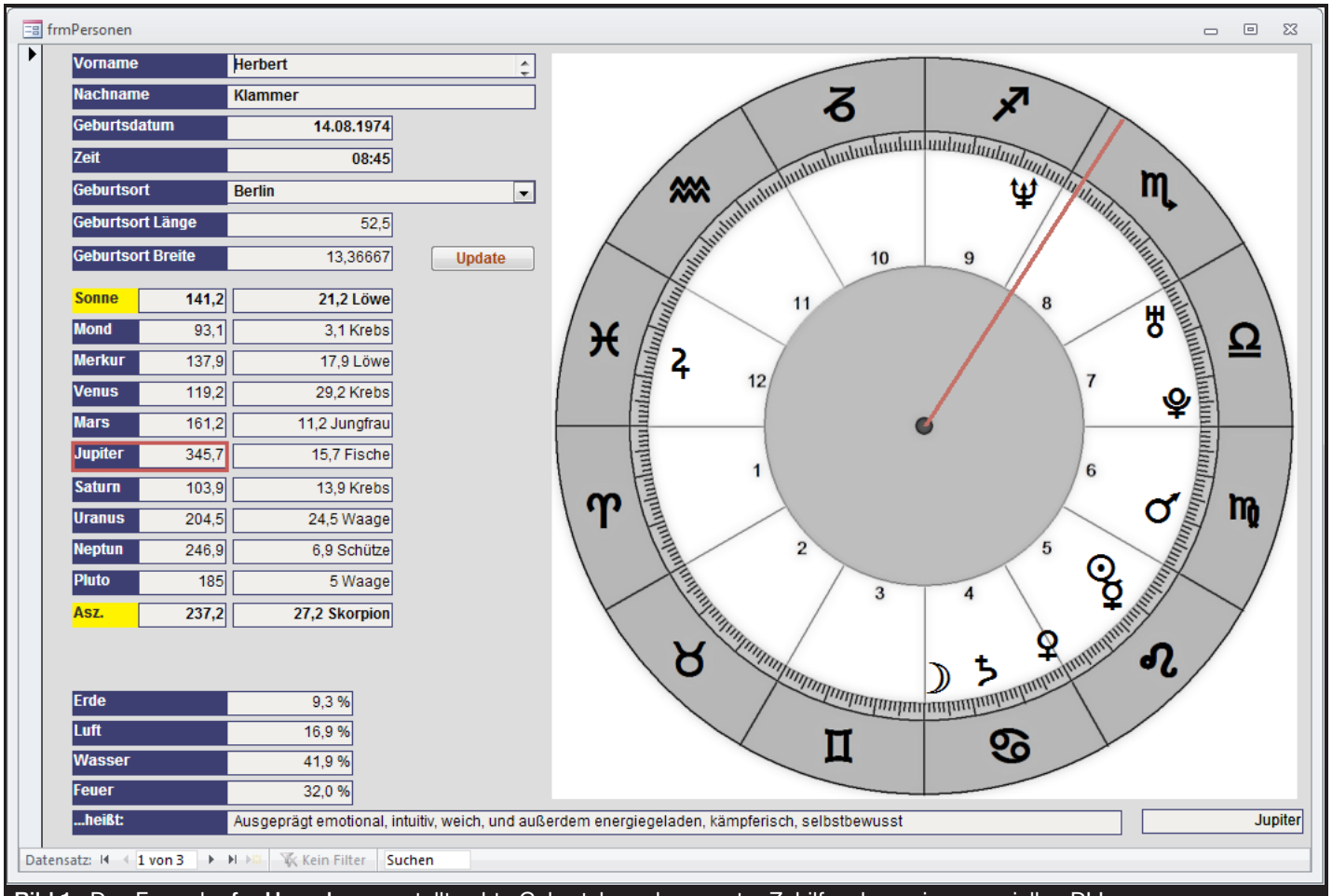


Bild 1: Das Formular **frmHoroskope** erstellt echte Geburtshoroskope unter Zuhilfenahme einer speziellen DLL

einige Erläuterungen zur Horoskopberechnung und -gestaltung.

Für ein Geburtshoroskop werden die Positionen der Planeten des Sonnensystems auf astronomischer Grundlage berechnet. Ausschlaggebend hierfür sind Geburtsdatum und -zeit, da sich die Planeten schließlich dauernd fortbewegen. Als Ergebnis erhalten Sie den Winkel, welcher sich vom sogenannten **Frühlingspunkt** als Nullkoordinate zum Planeten längs über die **Ekliptik** aufspannt. Die **Ekliptik** ist quasi jene Bahn, die um die Erde herumführt, und auf welcher sich die Planeten bewegen. Der Winkel kann daher Werte von 0 bis 360 Grad annehmen. Dieser Kreis wird linear in zwölf Bereiche unterteilt, die sich den verschiedenen **Tierkreiszeichen** zuordnen. Tatsächlich stehen in diesen Bereichen nur ungefähr die entsprechenden Sternbilder, weil auch diese sich mit der Zeit fortbewegen, die Astrologie behält die 30-Grad-Unterteilung aber stringent bei.

Die Berechnung ergibt also für jeden Planeten eine Position, die einem Tierkreiszeichen zugeordnet ist. Damit bekommt der Planet nach astrologischer Auslegung einen bestimmten Charakter. So befindet sich die **Sonne** von **Herbert Klammer** in Bild 1 etwa im Winkel von 141,2 Grad zum Ursprung und ist damit dem Zeichen **Löwe** zugeteilt. Und die Sonnenposition bestimmt nach dem Alltagsgebrauch das generelle Sternzeichen. **Herbert** ist also Löwe. Würde ein Astrologe dies deuten, so käme er zur Auffassung, dass Herbert in seinem Wesen recht selbstbewusst daher kommt, sich gerne im Mittelpunkt sieht, und auch motorisch ein gewisses Kraftpotential aufweist. Er ist nicht so leicht aus der Fassung zu bringen und das geborene Alpha-Tier.

Ähnlich werden auch die weiteren Planetenpositionen gedeutet, wobei jeder Planet für eine bestimmte Charakteristik steht. Übrigens werden Sonne und Mond in der Astrologie genau so behandelt, wie Planeten. Hinzu kommt noch als wesentlicher Bestandteil des Horoskops der **Aszendent**, welcher in der Grafik durch die rote Linie symbolisiert ist. Das ist kein Planet, sondern jene Position, welche zum Geburtszeitpunkt eben am Horizont im Osten aufging. Auch diese kann einem Sternzeichen zugeordnet werden. Der Aszendent soll, so die Astrologie, ähnlich gewichtig sein, wie die Sonnenposition, und gäbe Auskunft über die Außenwirkung einer Person.

Nicht in unsere Lösung eingebaut sind die sogenannten **Häuser** des Horoskops. Das wären, ausgehend vom Aszendenten, ebenfalls zwölf Bereiche des die Erde entlang der Ekliptik umgebenden Universums. Sie sollen Rückschlüsse über einzelne Lebensbereiche einer Person zulassen. Wir lassen das weg, weil die Lösung dann noch komplexer ausgefallen wäre, als sie es ohnehin schon ist.

Schließlich haben Sie möglicherweise auch schon von den **vier Elementen Erde**, **Wasser**, **Luft** und **Feuer** gehört. Jedes Tierkreiszeichen korreliert mit einem dieser Elemente. Befindet sich zum Geburtszeitpunkt in einem Tierkreisbereich ein Planet, so verstärkt dieser den Charakter des zugeordneten Elements. Man kann die Elemente somit über eine Gewichtungsformel aufaddieren und kommt so zu den prozentualen Anteilen, die links unten im Formular zu sehen sind. Auf sie bezieht sich die Minideutung in der untersten Zeile übrigens.

Für das Horoskop sind an sich nur die Werte der Planetenpositionen relevant. Übersichtlicher ist jedoch deren Darstellung in einer Grafik. Jeder Planet besitzt ein Symbol und jedes Tierkreiszeichen ebenso. In der Abbildung befindet sich etwa der **Jupiter** links im Sternzeichen **Fische**. Wenn Sie herausfinden möchten, welches Symbol welchem Element entspricht, so klicken Sie einfach mit der Maus auf entweder ein Sternzeichen im äußeren Ring, oder auf das Symbol eines Planeten im inneren Bereich. Rechts unten im Info-Label wird dann der Name des Elements ausgegeben.

Verwendete Techniken

Schauen wir uns im Überblick an, wie die Lösung in der Beispieldatenbank realisiert wurde.

Die Grundlage der Datenbank sind natürlich ihre Tabellen. Hier gibt es eine Tabelle **tblHoroskope**, welche die Personendaten aufnimmt – man hätte sie genauso gut **tblPersonen** nennen können. Zwei weitere Tabellen listen als Nachschlagetabellen die Planeten und Sternzeichen namentlich auf. Zwischen diesen und der Horoskoptabelle besteht

über die **n:m-Tabelle tblHoroskop-Zeichen** eine Beziehung. Schließlich vervollständigt eine Elemente-Tabelle **tblElemente**, verknüpft mit den Tierzeichen und Planeten, das Datenmodell.

Das Formular **frmHoroskope** baut auf diesen Tabellen auf, wobei nicht **tblHoroskope** die Datenherkunft darstellt, sondern ein Abfrage **qry_Horoskopzeichen**. Warum das so ist, das sehen wir noch später. Neben den Textfeldern für die Werte der zugrundeliegenden Daten finden sich noch **ungebundene Textfeldern**, deren Inhalt per VBA berechnet wird. Für die Grafik ist ein **Bildsteuerelement** verantwortlich, welches von einem roten **Liniensteuerelement** für den Aszendenten überlagert wird. Einzelne Textfelder, die einen speziellen **Astrologie-Font** als Schriftart aufweisen, werden zur Laufzeit über der Grafik positioniert, wofür die VBA-Routinen des Formularmoduls herangezogen werden.

Die Berechnung der **Ephemeriden**, also der astronomischen Planetenpositionen zu einem bestimmten Zeitpunkt, ist einen Angelegenheit, die über VBA-Code nicht zu bewältigen ist, Für die iterative Lösung des als **Mehrkörperproblem** in der Physik bekannten Themas kommen ungeheuer komplexe differenzielle Gleichungssystem zum Einsatz, für die die Performance von VBA nicht ausreicht. Zum Glück gibt es eine frei verfügbare und ständig weiterentwickelte Komponente des **Astrodienst Zürich**, die sogenannte **Swiss Ephemerisis**. Die kommt in Gestalt einer einzigen **DLL** daher, welche über **API-Programmierung** angesprochen wird. Sie berechnet mit nur einem Funktionsaufruf alle Positionen der Planeten und weitere Elemente. Wenn an die Genauigkeit dieser Berechnung höhere Anforderungen gestellt werden, so bedient sie sich optional weiterer Komponenten, nämlich einiger Datendateien, die vorberechnete Tabellen enthalten.

DLL und die für den relevanten Zeitraum verantwortlichen Dateien sind der Beispieldatenbank beigefügt.

Die Datenbank kommt ansonsten ohne weitere Bibliotheken aus. Zusätzliche Verweise sind also nicht zu setzen.

Installation

Es gibt keine Installation. Die DLL muss nicht auf dem Rechner registriert werden. Es reicht aus, dass sie sich im Ordner parallel zur **ACCDB** befindet. Lediglich die Schriftart **AstroFont99** muss installiert werden. Dazu rechtsklicken Sie die Datei **Awf99.ttf** im Explorer und wählen aus dem Kontextmenü **Installieren** aus. Die Schriftart wird damit in den **Font**-Ordner von Windows kopiert und im System registriert.

API-Programmierung

Mit diesem Thema hatten wir es in **Access [basics]** noch nicht zu tun. Grund dafür ist, dass es sich dabei um eine eher fortgeschrittene Technik handelt, vor der viele VBA-Entwickler zurückschrecken. Dabei ist die Sache einfacher, als man denkt. Zwar könnte man damit nicht nur eine komplette Ausgabe von **Access [basics]** füllen, sondern ohne weiteres ein ganzes Buch, doch die Grundlagen sind schnell erläutert.

Unter **API-Programmierung** (**Application Programming Interface**) versteht man landläufig den Zugriff auf Funktionen in externen Bibliotheken. Das sind in der Regel die System-DLLs von Windows, die manche Funktionen nach außen hin veröffentlichen. Gerade hier liegt der Hund begraben. Denn zum einen gibt es kein einfaches Verzeichnis dieser Funktionen. Es ist die **Windows-SDK**-Dokumentation der **Microsoft-MSDN**-Seiten im Internet zu Rate zu ziehen. Zum anderen ist die Zielgruppe dieser Dokumentation der **C**-Entwickler. Beispiele und Funktionssyntax in den Artikel sind allesamt in der Sprache **C** gehalten, der Inhalt auf Englisch. Verständlicherweise ist das für VBA-Entwickler schwer verdauliche Kost.

Neben den Windows-eigenen Funktionen lassen sich nach dem gleichen Prinzip jedoch auch DLLs von Drittherstellern ansprechen. Hier ist eine ausführliche Dokumentation natürlich noch wichtiger. Und vor allem benötigen Sie die Syntax der **API-Funktionen** auch in der Sprache VB(A). Denn das Übersetzen einer C-Syntax nach VBA ist eine Angelegenheit, die Experten vorbehalten ist.

Nehmen wir einfach, statt uns in abstrakten Theorien zu ergehen, ein Beispiel zur Hand. Sie möchten etwa über eine API-Funktion die Titelzeile des Access-Hauptfensters auslesen. Dafür kommt die Funktion **GetWindowText** in der **user32.dll**, beheimatet im Systemordner von Windows,

```
Public Declare Function GetWindowText _
    Lib "user32.dll" _
    Alias "GetWindowTextA" _
    (ByVal hwnd As Long, _
    ByVal lpString As String, _
    ByVal cch As Long) As Long

Sub AccessFensterTitel()
    Dim hwndAcc As Long
    Dim sTitle As String
    Dim ret As Long

    hwndAcc = Application.hWndAccessApp
    sTitle = String(255, 0)
    ret = GetWindowText(hwndAcc, _
                        sTitle, 255)
    If ret > 0 Then
        Debug.Print Left(sTitle, ret)
    End If
End Sub
```

Listing 1: API-Routine zum Auslesen des Fenstertitels von Access

infrage. Damit sie angesprochen werden kann, benötigen wir zunächst zwingend eine VBA-Deklaration dieser Funktion. Sie finden sie in der ersten Zeile von Listing 1.

Eine **API-Deklaration** findet immer im Kopf eines Moduls statt und wird durch den Ausdruck **Declare Function** oder **Declare Sub** eingeleitet. Je nachdem, ob die Funktion dann nur in diesem Modul oder von überall her aufgerufen werden soll, stellen Sie entweder ein **Public** oder ein **Private** voran.

Darauf folgt der Name der Funktion, welcher hier **GetWindowText** lautet. Würden Sie die DLL damit aufrufen, so käme es zu einer Fehlermeldung **DLL-Einsprungspunkt GetWindowText in user32.dll nicht gefunden**. Tatsächlich gibt es diese Funktion in der DLL auch gar nicht! Vielmehr existieren zwei: einmal eine für **ANSI**-Strings ausgelegte und einmal eine für **Unicode**-Strings. Deren Namen laute dann aber **GetWindowTextA** und **GetWindowTextW** – das **W** steht für **Wide Character**. Der folgende Ausdruck **Alias** trägt dem Rechnung. Zwar sprechen Sie die Funktion im Code später mit **GetWindowText** an, der Aufruf wird jedoch über den **Alias** an **GetWindowTextA** der DLL weitergeleitet. Bliebe noch zu erwähnen, dass API-Funktionen **case sensitive** sind. Sie haben also auf korrekte Groß- und Kleinschreibung der Zeichen zu achten.

Vor dem **Alias**, aber nach dem Funktionsnamen, steht über den Ausdruck **Lib** noch die Bibliothek, in der die Funktion gefunden werden kann. Hier handelt es sich um die **user32.dll**. Für den hier eingesetzten String gilt Folgendes: Entweder setzen Sie den Namen einer DLL-Datei ein, die sich im **Suchpfad** von Windows befindet, oder Sie geben den kompletten Pfad der Datei an. Die Dateierweiterung können Sie auch weglassen, wenn es sich um eine DLL handelt, da Windows dann automatisch von einer DLL ausgeht. Diese Angaben wären also alle korrekt:

```
user32
user32.dll
c:\windows\system32\user32.dll
```

Der Suchpfad von Windows umfasst im Wesentlichen die Systemordner. Befindet sich eine Bibliotheksdatei außerhalb dieses Suchpfads, so ist der komplette Pfad anzugeben – mit einer Ausnahme: DLLs, die Access für sein eigenes Funktionieren schon geladen hat, können ebenfalls ohne Ordnerpfad angesprochen werden.

Was sich nun in der umgebrochenen Deklarationszeile anschließt ist die Liste der erwarteten Parameter und der Rückgabewert der Funktion, genau so, wie es sich bei einer VBA-Funktion verhält. Wir haben hier den Parameter **hwnd** des Datentyps **Long** vor uns, **lpString** ist die String-Variable, die nach dem Aufruf den Fenstertitel erhalten soll, und **cch** des Typs **Long** gibt an, wie lange der Text maximal sein darf. Die Rückgabe der Funktion ist ebenfalls vom Typ **Long**. Bei Erfolg ist deren Wert größer, als 0, und gibt dann die Anzahl der in den String eingesetzten Zeichen wider.

Extrem wichtig ist auch der Übergabetyp der Parameter. Es ist, im Gegensatz zu den meisten VBA-Funktionen, nicht egal, ob eine **ByVal**- oder eine **ByRef**-Übergabe stattfindet. Eine Erläuterung, warum das so ist, sprengte allerdings den Rahmen dieses Beitrags.

Einmal deklariert, lässt sich die API-Funktion über deren Namen von beliebiger Stelle Ihres Codes aus aufrufen.

In der Routine **AccessFensterTitel**, die Sie etwa aus dem VBA-Direktfenster aus aufrufen, wird der Variablen **hwndAcc** zunächst das sogenannte **Fenster-Handle** von Access zugewiesen. Das steht über die Eigenschaft **hwndAccessApp** des **Application**-Objekts immer zur Verfügung. Anhand dieses Fenster-Handles berechnet die **user32.dll** schließlich den Fenstertitel in der übergebenen Variablen **sTitle**. Der Wert dieser Variablen wird davor

jedoch noch mittels **String()**-Funktion auf 255 Null-Zeichen gebracht. Hier wird bereits eine Abweichung zu VBA-Routinen sichtbar. Während ein String hier meist von der Funktion selbst eingerichtet wird, sieht das häufig bei API-Funktionen anders aus. Diese gehen davon aus, dass der übergebene String bereits alloziert ist, also schon eine feste Speichergröße aufweist. Würden Sie **sTitle** ungefüllt übergeben, so enthielte die Variable nur einen **Null-String** der Länge 0. Die API-Funktion versuchte dennoch, den Fenstertitel an die Adresse des Strings zu schreiben – mit fatalen Folgen! Es käme zum Schreiben in ungeschützte Speicherbereiche, was Access meist mit einem Absturz quittiert.

Zurück zu unserer Prozedur. Die Variable **ret** nimmt die Anzahl an Zeichen entgegen, die die API-Funktion produziert hat. Ist diese größer, als 0, dann lief alles glatt, und der relevante Teil des Strings wird über **Left** und **Debug.Print** in das VBA-Direktfenster ausgegeben.

Nun werden Sie wohl kaum die Syntax der API-Funktionen selbst anfertigen wollen. Zwar kann es nicht schaden, sich die Dokumentation etwa zu **GetWindowText** im Internet zu Gemüte zu führen, um weitere Details zu erfahren, aber das Übersetzen von **C** nach **VBA** überlassen Sie einfach Anderen. Ein Werkzeug, das sich dafür gut eignet, ist der **API-Viewer** von ActiveVB (siehe Bild 2), eine **Standalone**-Anwendung, die praktisch alle Windows-Funktionen auflistet, die Ihnen über den Weg laufen werden. Über die Suchzeile navigieren Sie zum Funktionsnamen und doppelklicken ihn, worauf die Deklaration in das rechte Fenster übertragen wird. Das können Sie mehrmals hintereinander durchführen. Zuletzt klicken Sie auf das **Copy**-Symbol, was den **API-Viewer** veranlasst, alle gewünschten Deklarationen als Text in die Zwischenablage zu kopieren. Fügen Sie diese danach in ein VBA-Modul Ihrer Wahl ein.

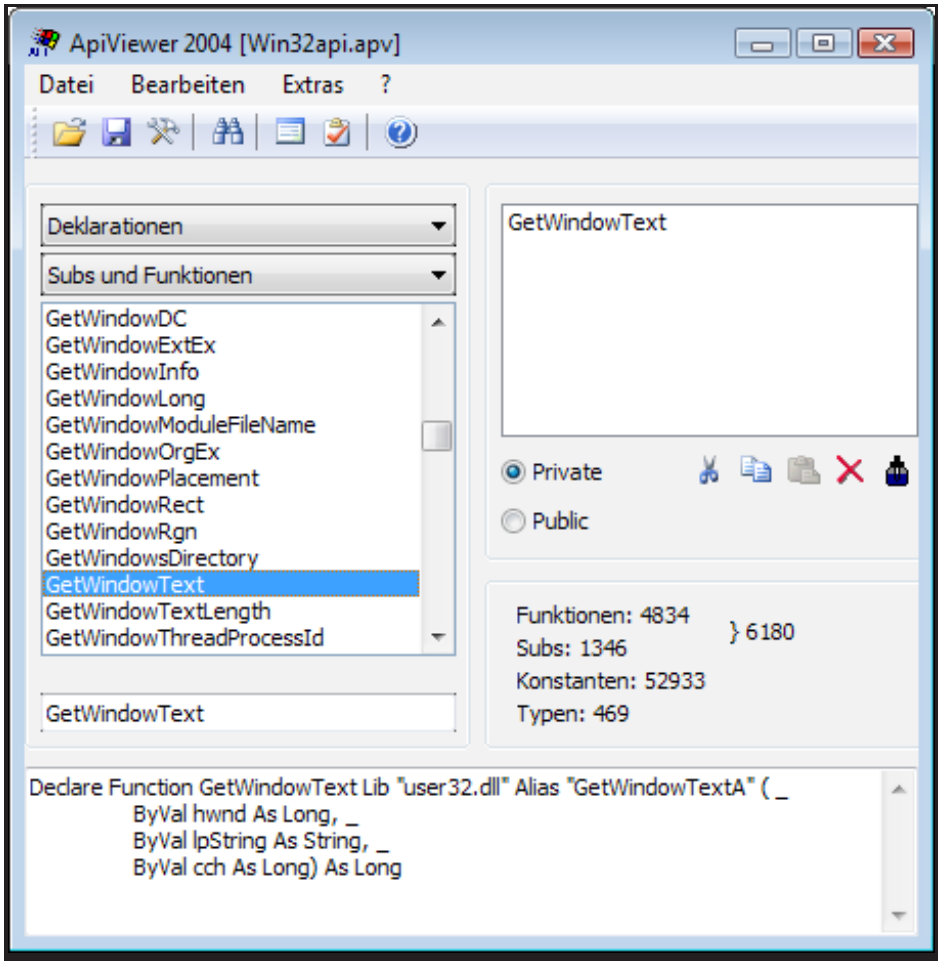


Bild 2: Der API Viewer enthält die gebräuchlichsten Windows-API-Deklarationen

Ephemeridenberechnung per API-Funktion

Die Planetenposition werden in der Beispieldatenbank mithilfe der zusätzlichen DLL **swedll32.dll** berechnet. Die von dieser Bibliothek exportierten Funktionen sind auf den Seiten der Entwickler gut dokumentiert. Auch hier haben wir es allerdings mit dem üblichen **C**-Slang zu tun. Der Hersteller lässt uns aber nicht im Regen stehen, denn es steht auch ein **VB6**-Projekt bereit, welches schon die API-Deklarationen enthält.

Sie finden diese alle im Kopf des Moduls **mdlEphemerisis** versammelt. Neben den eigentlichen API-Funktionen stehen dort noch spezielle Konstanten, die als Parameter Anwendung finden können. Obwohl hier circa 90 Deklarationen stehen, benötigen wir für unsere Zwecke aus dem Fundus nur fünf.

Wenn Sie sich die Deklarationen einmal genauer anschauen, so könnte Ihnen auffallen, dass als Bibliothek lediglich jeweils dieser Ausdruck in den Zeilen steht:

```
Lib "swedll32.dll"
```

Es fehlt also die Pfadangabe. Das würde zur Voraussetzung machen, dass sich die Datei im Suchpfad von Windows befände, etwa im Systemordner. So Sie über Administratorrechte verfügen, können Sie sie auch dorthin kopieren. Doch tatsächlich liegt sie ja im Ordner der Datenbank. Also müsste der Pfad erst ermittelt und in die Deklarationen eingebaut werden. Beim Verschieben des Datenbankordners wäre die Arbeit aber zunichte gemacht, weil der alte nun durch den neuen Pfad zu ersetzen wäre.

Dabei gibt es einen Workaround: Lassen Sie die Pfadangabe weg und laden die **swedll32.dll** selbst über eine andere API-Funktion. Da dies dazu führt, dass die DLL innerhalb des Prozessraums von Access zu stehen kommt, also quasi zu dessen geladenen Modulen gehört, findet der API-Aufruf, wie oben erwähnt, die DLL auch ohne Pfadangabe. Nötig sind dafür nur zwei **Windows-API**-Funktionen: **LoadLibrary** und **FreeLibrary**.

Die ersten Zeilen der Prozedur **BerechneHoroskop** sehen so aus, wie in Listing 2. Der API-Funktion **LoadLibraryA**, welche im Kopf deklariert ist, übergeben Sie nur den Pfad zur DLL, welche sich im Datenbankordner **CurrentProject.Path** befindet.

```
Public Declare Function LoadLibraryA Lib "kernel32.dll" _
    (ByVal lpLibFileName As String) As Long
Public Declare Function FreeLibrary Lib "kernel32.dll" _
    (ByVal hLibModule As Long) As Long

Public hMod As Long

Public Function BerechneHoroskop()
   ...
    If hMod = 0 Then
        hMod = LoadLibraryA(CurrentProject.path & "\swedll32.dll")
        If hMod = 0 Then Exit Function
    End If
    ....
End Sub
```

Listing 2: Laden der Fremdkomponente **swed32.dll** per API-Funktion **LoadLibrary**

Die Funktion gibt dabei ein **Modul-Handle** zurück, das in der projektweit gültigen Variablen **hMod** zwischengespeichert wird. Bei erneutem Aufruf von **BerechneHoroskop** ist ihr Wert nicht mehr 0, weshalb der Zweig über die **If**-Bedingung übersprungen wird, da ein erneutes Laden der Bibliothek keinen Sinn ergibt.

Im Prinzip war es das schon. Nur ist es guter Stil, eine eigens geladene DLL nach Beenden aller Aufrufe auch wieder aus dem Speicher von Access zu entfernen. Normalerweise stört es sich nicht daran, wenn dies unterbleibt, doch im Interesse von Stabilität sollten Sie es dennoch durchführen. Hier kommt **FreeLibrary** ins Spiel:

```
If hMod <> 0 Then FreeLibrary hMod
```

Die Zeile könnten Sie etwa beim Schließen des Formulars **frmHoroskop** aufrufen lassen. Da **hMod** projektweit gültig ist, lässt sich die Zeile ja von überall her aufrufen.

Kommen wir zur Prozedur **BerechneHoroskop** selbst. Ein Auflisten ist hier nicht sinnvoll, da sie doch recht umfangreich ausfällt. Zwar findet die eigentliche Berechnung der Planeten und des **Aszendenten** nur in den zwei API-Funktionen **swe_calc_ut** und **swe_houses_ex** statt, doch die der Prozedur übergebenen Daten, wie Geburtsdatum und -zeit, sowie die zum Ort, müssen etwa erst in andere Datumsformate (**Julianisch**) überführt werden, was zusätzliche Programmierung erfordert. Auch einige Einstellungen am Berechnungsmodus der DLL sind vorzunehmen. Der Code ist weitgehend aus der **VB6**-Demo entlehnt, die Sie auf den Seiten des **Astrodienst Zürich** herunterladen können.

Die Prozedur nimmt also diese Angaben entgegen und speichert das Ergebnis, die Planetenpositionen, in einem **Array result**, das der Prozedur ebenfalls als Variant zugeteilt wird, demnach also schon im aufrufenden Code definiert werden muss. Die Variable **result** enthält dann 26 Positionen in ihren **Array**-Elementen, die als **Double**-Werte für die Winkel der Planeten auf der Ekliptik stehen.

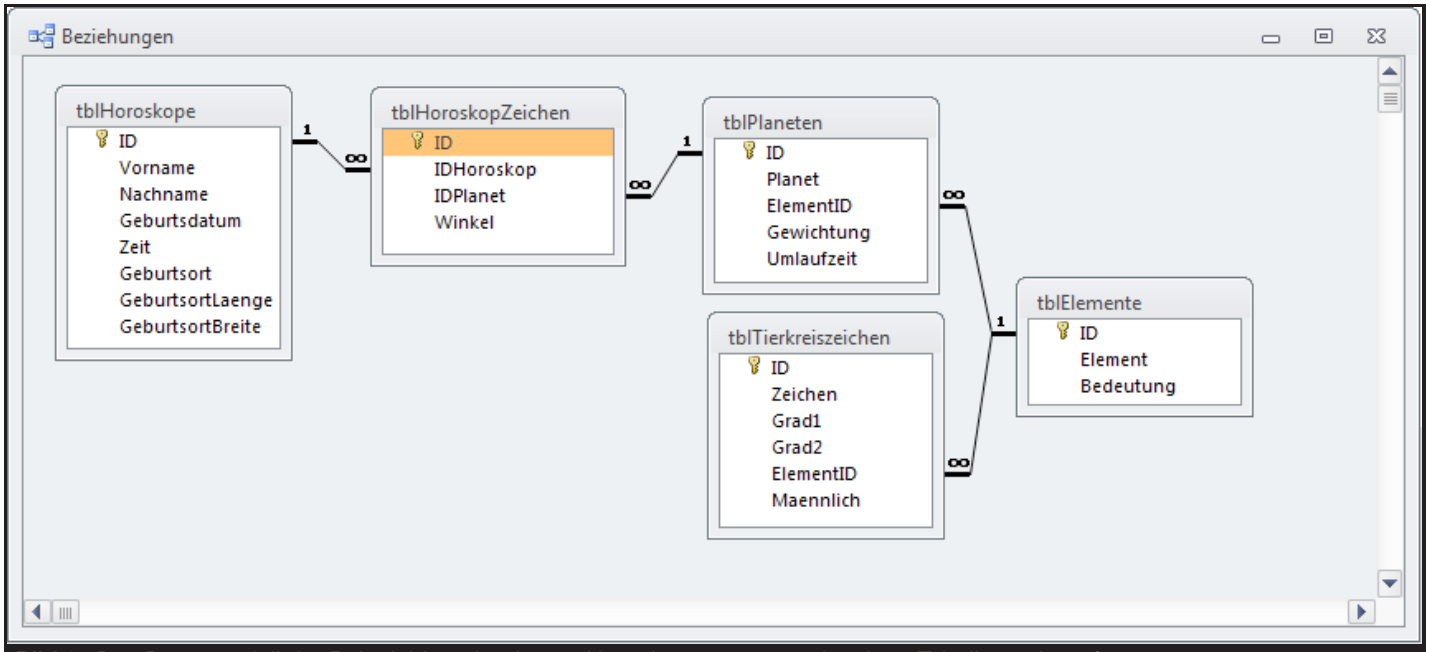


Bild 3: Das Datenmodell der Beispieldatenbank zum Horoskopgenerator mit seinen Tabellenverknüpfungen

Die Prozedur wird im Formularmodul von der Ereignisprozedur **Form_Current** aufgerufen.

Datenmodell

Bevor es an die Gestaltung des Formulars geht, beleuchten wir das Datenmodell der Beispieldatenbank noch näher. Bild 3 zeigt ihr Beziehungsfenster.

Die Tabelle **tblHoroskope** nimmt nur die Daten der Personen auf, zu denen Horoskope berechnet werden sollen. Die Planetendaten sind davon abgekoppelt, Sie stehen alle in der Tabelle **tblPlaneten**. Die Positionen der Planeten werden einem Horoskop über die **n:m-Tabelle tblHoroskopZeichen** hinzugefügt. Hier verweist einerseits **IDHoroskop** auf einen Datensatz in tblHoroskope, **IDPlanet** andererseits auf einen bestimmten Planeten. Dessen **Winkel** vervollständigt hier einen Datensatz.

Die Tabelle **tblPlaneten** ist an sich nur ein Container für alle von der DLL berechneten Positionen. Dazu gehören neben den Planeten auch der **Aszendent** und die sogenannten **Häuser**. Diese sind deshalb ebenfalls in der Tabelle aufgelistet.

Nur für die Auswertung eines Horoskops ist die Tabelle **tblElemente** gedacht. Hier stehen die Elemente **Erde**, **Wasser**, **Luft** und **Feuer**, und deren Bedeutung. Jedem Planeten und jedem Tierkreiszeichen wird ein Element zugeordnet, weshalb es auch hier eine Beziehung zu den Tabellen **tblPlaneten** und **tblTierkreiszeichen** gibt.

Horoskopformular

Dem Datenmodell entsprechend sollte man erwarten, das nicht ein einzelnes Formular für die Lösung ausreicht, sondern zwei benötigt würden. Das Hauptformular nähme die Personendaten auf und ein Unterformular, welches an **tblHoroskopZeichen** gebunden wäre, zeigte die Planetenwinkel an, indem das Hauptformular über den Schlüssel **ID** mit dem Fremdschlüssel **IDHoroskop** des Unterformulars verknüpft würde.

Normalerweise handhabt man das so. Dass es auch anders geht, zeigt indessen das Formular **frmHoroskope**, dessen Datenquelle nicht auf einer Tabelle, sondern einer Abfrage **qry_Horoskopzeichen** beruht.

Und die hat es in sich! Da das Formular neben den Personendaten auch noch die Planetenwinkel in **einem** Datensatz anzeigt, müssen alle zu einer Horoskop-**ID** gehörigen Detaildatensätze auf eine Ebene gebracht und horizontal wie Felder ausgegeben werden. Das lässt sich, außer über VBA-Programmierung, nur mit einer **Kreuztabelle** realisieren.

Alles plausibel?

Fehleingaben in Datenbanken sind, nach eigener Erfahrung, ein Quell ständiger Reibereien in Unternehmen. Oft sind die Daten redundant, mangelhaft formatiert oder lückenhaft. Nachforschungen, wer nun dies oder jenes falsch gemacht habe, führen selten zu effizienterer Arbeitsatmosphäre. Dabei könnten doch Plausibilitätsprüfungen schon bei der Eingabe in Formulare einen großen Teil dieses Missstands beheben. Wir zeigen Ihnen einige Beispiele.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1602_Plausibel.accdb**

Plausible Kunden
Anhand der schon in der Ausgabe 11/2015 vorgestellten Beispieldatenbank mit ihrem Kunden- und Bestellformular demonstrieren wir Ihnen die möglichen Fallstricke und zeigen Lösungsmöglichkeiten auf. Das Formular **frmKunden** wurde dafür mit zusätzlichem VBA-Code aufgerüstet, der die Plausibilitätsprüfung der Eingabefelder garantieren soll.

Ein immer wiederkehrender Vorgang ist die Neuanlage von Adressen, die bereits in der Datenbank vorhanden sind. Oft wird dies zu spät bemerkt, etwa dann, wenn an anderer Stelle auf die Adressen Bezug genommen und eine Verknüpfung mit anderen Daten hergestellt werden soll. Welche Adresse ist dann die richtige? Besser, Sie bauen da eine Prüfung ein, ob etwa die Kombination von Nachname und Vorname schon existiert, um dem Anwender gegebenenfalls eine Meldung zu präsentieren, wie in Bild 1. Er hat dann die Möglichkeit, die Eingabe dennoch fortzusetzen – Namen sind ja tatsächlich nicht immer eindeutig –, oder er navigiert zur bereits vorliegenden Adresse, weil er diese übersah.

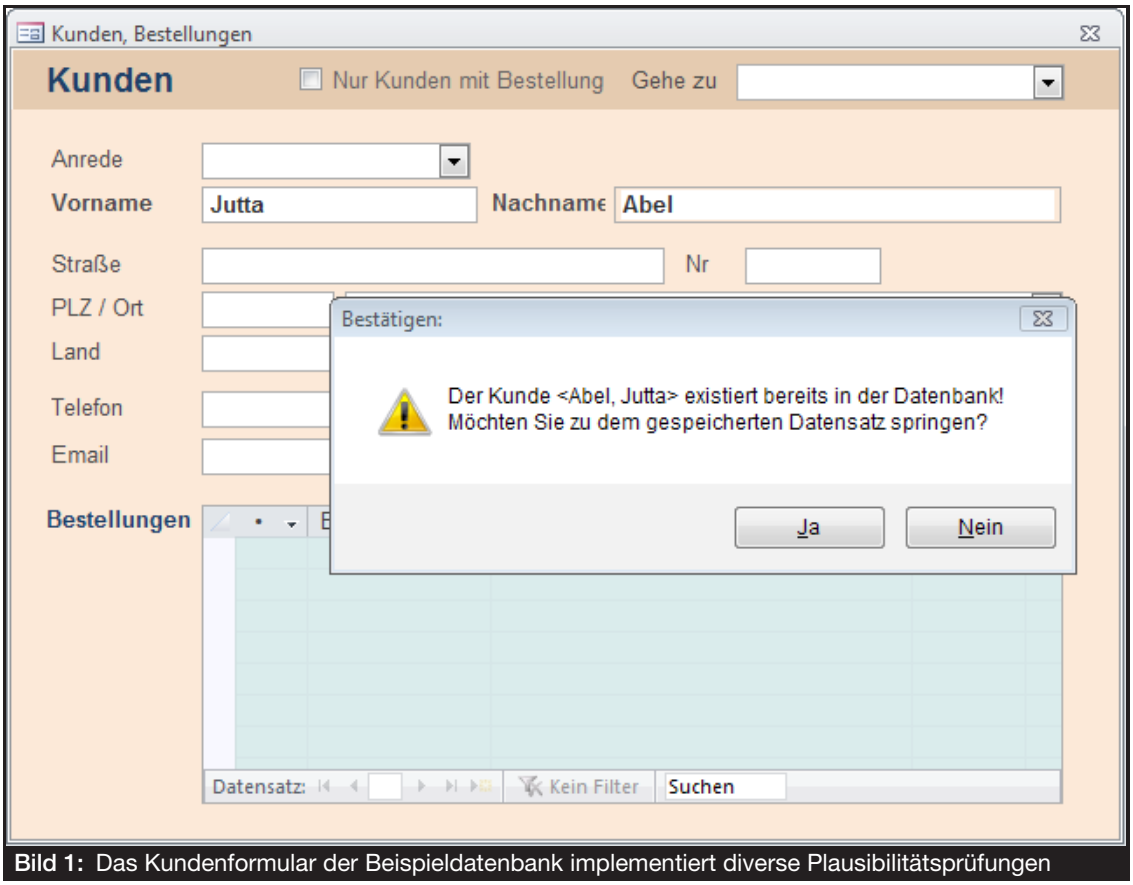


Bild 1: Das Kundenformular der Beispieldatenbank implementiert diverse Plausibilitätsprüfungen

Was hin und wieder ebenfalls passiert, ist die Eingabe überflüssiger Leerzeichen. So wird versehentlich ein Leerzeichen an den Anfang des Nachnamens gesetzt was zu "Aber, Jutta" und " Abel, Jutta" führt. In einer sortierten Adressliste bemerkte man die Doppelung gar nicht, weil der String mit dem Leerzeichen die Adresse ganz nach vor setzte. Bauen Sie eine Eliminierung überflüssiger Leerzeichen ein, so ist derartigem Unbill vorgesorgt.

Schlecht macht sich auch, wenn eine Auslieferung an den falschen Adressaten gerät, weil Ort und Postleitzahl nicht zueinander passen. Verknüpfen Sie beide miteinander über eine interne Postleitzahlentabelle, so können Sie auf Falschangaben reagieren und Korrekturen vorschlagen lassen.

Und schließlich sind E-Mail-Adressen heutzutage genauso wichtig, wie die örtlichen Angaben. E-Mails, die den Adressaten nicht erreichen, haben Verzögerungen im Workflow zur Folge. Fehler infolge inkorrekter E-Mail-Eingaben können Sie wenigstens minimieren, indem die Eingabe einer Validitätsprüfung unterzogen wird.

Formularnavigation

Bevor wir zum eigentlichen Thema kommen, beschreiben wir noch eine Erweiterung der Navigationsmöglichkeiten des Formulars. Da es keine Navigationsleiste anzeigt, muss zu einem Kunden über das Kombinationsfeld **Gehe zu** gesprungen werden. In der bisherigen Form enthält diese Combobox eine Liste aller eingegebenen Kunden. Dabei fehlt die Möglichkeit, einen Kunden zu löschen oder neu anzulegen. Zwar könnten Sie zu diesem Zweck zusätzliche Schaltflächen Löschen und Neu in das Formular einfügen, doch auch das Kombinationsfeld macht's möglich!

Die Steuerelementdaten bestanden bislang ungefähr aus dem SQL-Ausdruck

```
SELECT Nachname & ", " & Vorname
FROM tblKunden ORDER BY Nachname
```

Was wir möchten, sind die zwei zusätzlichen Einträge **Neuer Kunde** und **Kunde löschen** ganz oben in der Liste, so, wie dies Bild 2 zeigt. Da diese in der Tabelle **tblKunden** nicht existieren, müssen sie künstlich eingebaut werden. Dazu eignet sich in erster Linie eine **Union-Abfrage**. Für die **Combobox** sieht der SQL-Ausdruck nun so aus:

```
(SELECT -1 AS ID, "(Kunden löschen)"
        AS Vollname FROM tblKunden)
UNION
(SELECT 0 AS ID, " (Neuer Kunde)"
        AS Vollname FROM tblKunden)
UNION
(SELECT tblKunden.ID, [Nachname] & ",
" & [Vorname]
        AS Vollname FROM tblKunden)
ORDER BY Vollname;
```

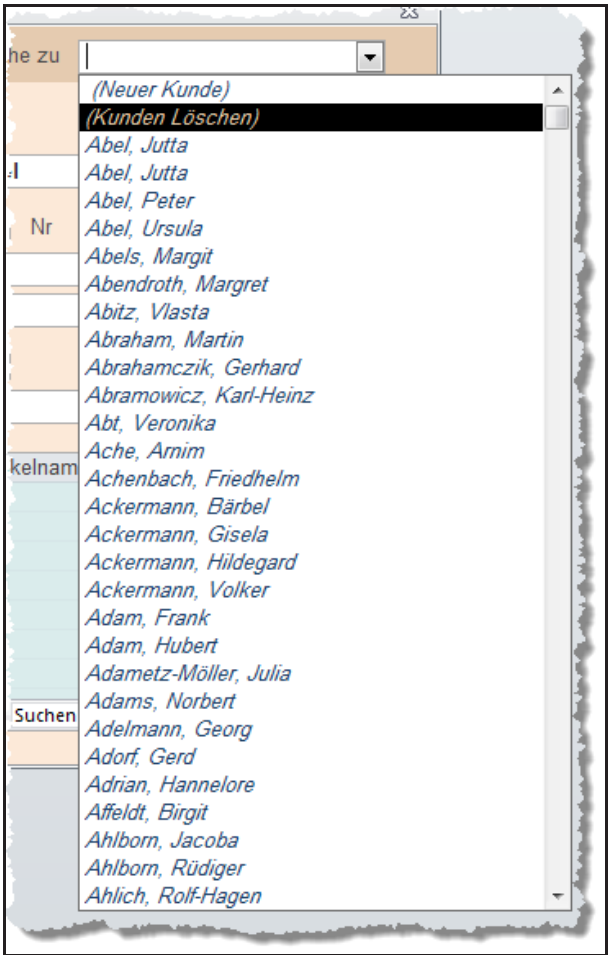


Bild 2: Die Combobox mit ihren zwei zusätzlichen Einträgen zum Neuanlegen und Löschen von Kunden

Erstaunlich, aber wahr: Nicht nur zwei, sondern auch mehrere **SELECT**-Abfragen können über die **UNION**-Klausel miteinander kombiniert werden! Die eigentliche Abfrage steht ganz unten, wobei die Kombination aus Nach- und Vorname über den Feld-**Alias Vollname** ausgegeben wird. Damit die Einzelabfragen zusammengefasst werden können, müssen sie die gleiche Zahl von Feldern und die gleichen Feldnamen ausgeben. Das erste **SELECT-Statement** setzt nun einfach den Wert **-1** als **ID** und den **Vollnamen** auf **(Kunden löschen)**. Als Tabelle im **FROM**-Teil wird ebenfalls **tblKunden** spezifiziert. Das ist aber völlig ohne Belang! Sie könnten hier genauso gut eine andere Tabelle einsetzen, weil auf deren Datensätze ja ohnehin nicht Bezug genommen wird. Nur ohne Tabelle geht es leider nicht. Von der Performance her ist es günstiger, die gleiche Tabelle zu verwenden, auf die im unteren Teil tatsächlich zugegriffen wird.

Der Inhalt der zweiten Klammer ist identisch aufgebaut. Hier steht die **ID 0** für den Eintrag **(Neuer Kunde)**. Davor steht, wohlgemerkt, ein Leerzeichen! Das führt dazu, dass

dieser Eintrag infolge der abschließenden aufsteigenden Sortierung über alle Datensätze in der **ORDER BY**-Klausel ganz oben steht.

Die Ereignisprozedur für das Kombinationsfeld **cbFind** ist nun natürlich entsprechend anzupassen. Im Ereignis **Nach Aktualisierung** kommt es zur Ausführung der Prozedur in Listing 1. Je nach Wert (**cbFind.Value**) des Kombinationsfelds verzweigt der **Select-Case**-Block in unterschiedliche Aktionen. Wurde etwa **Neuer Kunde** ausgewählt, so ist der Wert von **cbFind 0**, was die **DoCmd**-Anweisung **GotoRecord** mit dem Parameter für einen neuen Datensatz auslöst. Ähnlich bei **-1** für **Kunde löschen**, was den aktuellen Datensatz über die **RunCommand**-Anweisung löschen würde. Konjunktiv deshalb, weil davor noch eine Sicherheitsabfrage über eine **Messagebox** gestellt ist, die den Anwender auf den folgenden prekären Vorgang hinweist. Für alle anderen Werte von **cbFind** kommt es zum unteren **Else**-Teil des Blocks, der das Formular-**Recordset** auf die gewünschte **ID** einstellt und damit den Sprung zum entsprechenden Kunden im Formular erwirkt.

```
Private Sub cbFind_AfterUpdate()
    On Error Resume Next
    Select Case Me!cbFind.Value
    Case 0
        DoCmd.GoToRecord acDataForm, Me.Name, acNewRec
    Case -1
        If MsgBox("Diesen Kunden wirklich löschen?", _
            vbYesNo Or vbExclamation, "Bestätigen:") = vbYes Then
            RunCommand acCmdDeleteRecord
            Me!cbFind.Requery
        End If
    Case Else
        Me.Recordset.FindFirst "ID=" & Me!cbFind.Value
    End Select
End Sub
```

Listing 1: Das aktualisierte Kombinationsfeld **cbFind** lässt durch den **Select Case**-Zweig unterschiedliche Aktionen folgen

Unerwünschte Leerzeichen

Der einfachste Teil der Plausibilitätsprüfungen ist das Entfernen vor- und nachstehender Leerzeichen. Wird im Eingabefeld für den **Nachnamen** ein Name eingeben und ein versehentliches Leerzeichen vorangestellt, so entfernt die Ereignisprozedur **Nach Aktualisierung** des Felds dieses über die Trim-Funktion sogleich:

```
Private Sub txtNachname_AfterUpdate()
    Me!txtNachname.Value = _
            Trim(Me!txtNachname.Value)
End Sub
```

Dies geschieht also unmittelbar nach dem Verlassen des Textfelds. Der Anwender hat gar keine Möglichkeit, ein Leerzeichen hineinzuschummeln.

Auf gleiche Weise wird mit dem Vorname-Feld verfahren.

```
Private Sub txtEmail_AfterUpdate()
    Dim sMail As String

    sMail = Nz(Me!txtEmail.Value)
    If Len(sMail) = 0 Then Exit Sub
    If IsEmailValid(sMail) Then Exit Sub
    MsgBox "Die angegebene E-Mail-Adresse ist nicht korrekt!", _
                                                vbExclamation
    Me!txtEmail.SetFocus
End Sub
```

Listing 2: Überprüfung der E-Mail-Adresse auf Gültigkeit

Stimmige E-Mail-Adresse

Wird das Eingabefeld für die E-Mail-Adresse verlassen, so löst dies seine **AfterUpdate**-Prozedur aus, wie in Listing 2. Hier kann die Adresse auf Validität getestet werden. Die Adresse wird zunächst der Variablen **sMail** zugewiesen. Die **Nz**-Funktion stellt dabei sicher, dass es bei geleertem Feld nicht zu einer Fehlermeldung kommt, weil dann der Feldwert nicht ein String, sondern **Null** wäre. In diesem Fall ist die Länge des String in **sMail** gleich **0**, was zum Verlassen der Prozedur führt.

Erst anschließend ruft die Prozedur die Funktion **IsEmailValid** auf, welche im Modul **mdlEmail** steht, und verlässt die Routine ebenfalls, falls der Rückgabewert auf **True** steht. Andernfalls weist eine Meldung den Anwender darauf hin, dass die Adresse zu korrigieren wäre (Bild 3).

Die separate Funktion **IsEmailValid** finden Sie in Listing 3. Sie erzeugt über **CreateObject** erst ein **ActiveX-Objekt** des Typs **VBScript.RegExp**. Dieses Objekt ist Bestandteil von Windows und ermöglicht die Auswertung sogenannter **Regulärer Ausdrücke**. Ein Verweis auf die Bibliothek braucht auf diese Weise nicht gesetzt zu werden. Das Kernelement des Objekts ist

seine Eigenschaft **Pattern**. Die Wirkungsweise des undurchsichtigen String-Ausdrucks zu erläutern, wäre an dieser Stelle zuviel des Guten. Erwähnt sei lediglich, dass er eine E-Mail-Adresse auf verschiedene Aspekte hin durchleuchtet.

Das @-Zeichen etwa muss vorkommen, ein Punkt für die **Domain**-Erweiterung ebenfalls, die Länge darf nicht zu kurz sein, nur Zahlen und Buchstaben sind erlaubt. Nach Absetzen seiner Methode **Test** gibt das Objekt entweder **True** oder **False** zurück, je nachdem, ob die **Pattern**-Bedingung erfüllt ist.

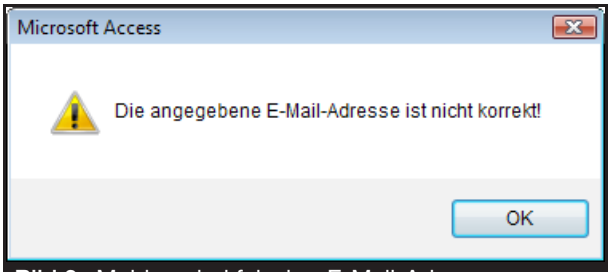


Bild 3: Meldung bei falscher E-Mail-Adresse

```
Function IsEmailValid(sEmail As String) As Boolean
    Dim objRegExp As Object

    Set objRegExp = CreateObject("VBScript.RegExp")
    With objRegExp
        .IgnoreCase = True
        .Global = True
        .Pattern = "^([a-zA-Z0-9_\-\.]+)@[a-z0-9-]" & _
            "+(\.[a-z0-9-]+)*(\.[a-z]{2,3})$"
        IsEmailValid = .Test(sEmail)
    End With
End Function
```

Listing 3: Ein regulärer Ausdruck dient als Hilfsmittel zur E-Mail-Prüfung in der Funktion **IsValidEmail**

Fehlende Angaben

Im Eifer des Gefechts kann es schon einmal vorkommen, dass manche Eingabefelder übersehen werden. Manchmal liegt die Information auch einfach nicht vor. Dennoch ist es sinnvoll auf fehlende Eingaben zu reagieren, um einem lückenhaften Datenbestand vorzubeugen.

Exemplarisch ziehen wir das **Hausnummernfeld** für diese Überprüfung heran. Wird es leer gelassen, so soll eine Meldung den Anwender darauf hinweisen. Wenig Sinn macht es indessen, dabei die Ereignisprozedur **Nach Aktualisierung** des Textfelds selbst zu verwenden.

Die Prüfung auf Vollständigkeit sollte erst erfolgen, wenn der Datensatz gespeichert wird. Sie kennen das sicher von Web-Formularen. Das geeignete Ereignis ist damit die Prozedur **Form_AfterUpdate**, also **Nach Aktualisierung** des ganzen Formulars, beziehungsweise Datensatzes. Oder doch **BeforeUpdate**, **Vor Aktualisierung**? Sehen Sie selbst...

Beides ist im Beispielformular implementiert. Listing 4 zeigt die zwei Ereignisprozeduren an. Ihr Code ist weitgehend identisch. Nach dem Test, ob das Feld **txtStrasse** einen Wert enthält über die **IsNull**-Funktion, weil nur dann die Überprüfung der Hausnummer sinnvoll ist, liest die Routine den Inhalt des Nummernfelds **txtNr**. Ist es leer, so tritt ein Meldungsfenster auf den Plan, das das Fehlen der Eingabe bemängelt. Der Wortlaut ist aber in den zwei Prozeduren unterschiedlich.

Denn zu AfterUpdate kommt es in der Regel, wenn der aktuelle Datensatz verlassen wird. Das kann nicht unterbunden werden, weshalb es für eine Korrektur an dieser Stelle zu spät ist. Die lapidare Meldung in Bild 4 ist daher das Ergebnis.

```
Private Sub Form_BeforeUpdate(Cancel As Integer)
    If Not IsNull(Me!txtStrasse) Then
        If IsNull(Me!txtNr) Then
            If MsgBox("Die Eingabe der Straßennummer fehlt noch!" & _
                vbCrLf & "Diese Nachtragen?", vbYesNo Or vbExclamation, _
                "Hinweis:") = vbYes Then
                    Cancel = True
                    Me!txtNr.SetFocus
            End If
        End If
    End If
End Sub

Private Sub Form_AfterUpdate()
    If Not IsNull(Me!txtStrasse) Then
        If IsNull(Me!txtNr) Then
            MsgBox "Die Eingabe der Straßennummer fehlt noch!", _
                vbExclamation, "Hinweis:"
        End If
    End If
End Sub
```

Listing 4: Ereignisprozeduren zu Vor/Nach Aktualisierung des Formulars

SELECT-Abfragen in Union

In der vorigen Ausgabe von Access [basics] schnitten wir das Thema UNION-Abfragen bereits an. Dort wurde ein Kombinationsfeld zur Formularnavigation mit einem UNION-SELECT-Ausdruck als Datenherkunft belegt. Deshalb nehmen wir diese SQL-Spezialität hier einmal etwas allgemeiner unter die Lupe und betrachten zusammengesetzte Abfragen im Detail.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1603_Union-Abfragen.accdb**

Daten zusammenfassen

In der Beispieldatenbank stellt sich eine Aufgabe, die so gar nicht selten zu erledigen ist: Daten verschiedener Herkunft sollen in einer Tabelle zusammengeführt werden. Es handelt sich um Adressen, die etwa aus importierten CSV-Dateien stammen und bereits in den Tabellen **tblAdressen1** bis **tblAdressen4** vorliegen.

Die erste enthält nur Adressen aus dem Postleitzahlengebiet **1*** und **2***, die nächste aus **3***, **4*** und **5***, eine weitere aus **6***, **7***, **8*** und **9***, sowie schließlich eine für jene aus **0***. Die Datensätze dieser vier Tabellen transferieren wir nun in ihrer Gesamtheit in die Tabelle **tblAdressen**.

Hierfür könnten Sie eine einfache Anfügeabfrage erstellen und nacheinander die jeweiligen Quelltabellen in sie einsetzen, um den Export zu bewerkstelligen. Das allerdings ist wenig elegant und zieht so einiges Mausgeklicke nach sich, von der Fehleranfälligkeit einmal abgesehen. Besser, Sie greifen hier zu einer UNION-Abfrage, die die Sache in einem Rutsch erledigt.

UNION-Abfragen

Eine **UNION-Abfrage** fasst zwei oder mehr einzelne Abfragen in einem Ergebnis zusammen. Während Sie fast alle Abfragetypen mit dem GUI-**Query-Editor** anlegen und bearbeiten können, sieht das bei **UNION-Abfragen** anders aus. Hier sind Sie im Entwurf auf den **SQL-Editor** beschränkt. Ohne grundlegende SQL-Kenntnisse sind sie damit also nicht im Boot.

Bild 1 zeigt, wie Sie gespeicherte **UNION-Abfragen** im Navigationsbereich von Access sofort erkennen können. Das Schnittmengensymbol macht sie deutlich.

Legen Sie Ihre erste **UNION-Abfrage** so an: Klicken Sie im Ribbon **Erstellen** auf **Abfrageentwurf**. Schließen Sie sofort den Dialog zur Tabellenauswahl. Der **Ribbon-Tab Entwurf** ist nun automatisch aktiv, und links wird auch gleich der Button **SQL** für die Ansicht eingeblendet. Klicken Sie darauf, um den **SQL-Editor** zu öffnen.

Nur als Hinweis hat Access bereits den Ausdruck **Select;** als SQL-Text im Editor eingefügt. Um nun alle Datensätze der Tabelle **tblAdressen1** im Abfrageergebnis anzuzeigen, erweitern Sie den Ausdruck in

```
SELECT * FROM tblAdressen1
```

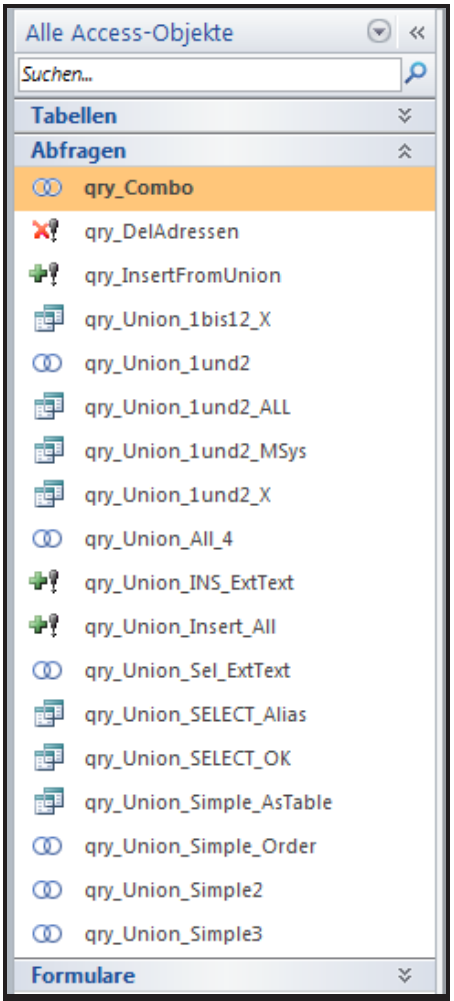


Bild 1: Die zwei Ringe stellen das Symbol für UNION-Abfragen im Navigationsbereich dar

Da die Aufgabe darin besteht, zusätzlich zu dieser Tabelle auch noch eine weitere ins Ergebnis zu holen, benötigen wir ein weiteres **SELECT-Statement**, das diesmal aber über den Ausdruck **UNION** mit dem an-

deren **SELECT-Statement** verknüpft wird:

```
SELECT * FROM tblAdressen1
UNION
SELECT * FROM tblAdressen2
```

Klicken Sie auf den **Ausführen**-Button im Ribbon, um sich davon zu überzeugen, dass diese Abfrage einwandfrei funktioniert. Speichern Sie sie unter dem Namen **qry_Union_Simple** ab. Im Navigationsbereich wird sie sogleich angezeigt und weist das entsprechende Symbol auf. Ein Rechtklick auf den Eintrag listet im Kontextmenü übrigens nun ebenfalls nur die **SQL**-Ansicht ein. Die normale **Entwurfsansicht** ist nicht erreichbar.

Eine UNION-Abfrage wird als solche allerdings nur identifiziert, wenn Sie den Aufbau von oben hat und Sie das Ergebnis nicht in Klammern einfassen. Denn auch diese Syntax ließe sich herstellen:

```
SELECT * FROM (
SELECT * FROM tblAdressen1
UNION
SELECT * FROM tblAdressen2
)
```

Die eigentliche **UNION-Abfrage** ist hier innerhalb der Klammern zu finden und wird wie eine einzelne Tabelle behandelt, indem ein weiterer **Select**-Ausdruck auf diese Unterabfrage angewandt wird. Das Anzeigeergebnis ist zwar dasselbe, die Abfrage wird dabei jedoch als normale **Auswahlabfrage** gespeichert. Damit besteht automatisch auch die Möglichkeit, sie im **Entwurfsmodus** zu öffnen. Das sieht dann etwa aus, wie

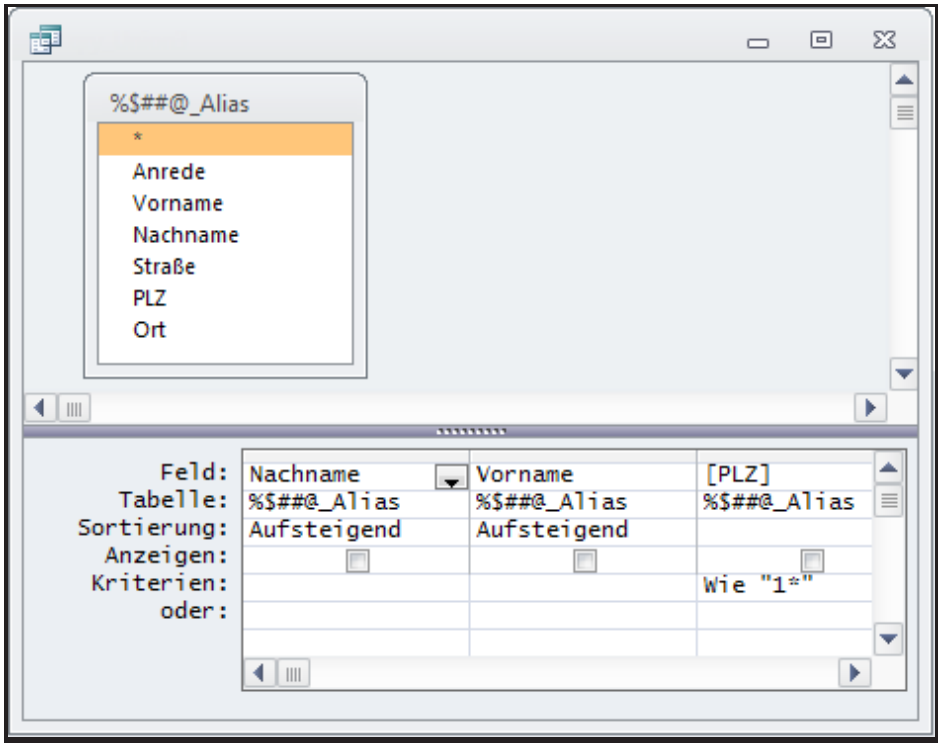


Bild 2: Dass es sich hier um eine UNION-Abfrage handelt, ist nicht gleich erkennbar

in Bild 2. In dieser Abfrage wurde das **UNION**-Ergebnis über das äußere **Select-Statement** noch einer Sortierung nach Namen unterzogen, sowie einer Filterung nach dem Postleitzahlenbereich **1***.

Das geht aus dem Entwurf zwar vor, irritierend ist aber die seltsame Tabellenbezeichnung **[%$##@_Alias]**. Access setzt diesen Ausdruck immer dann ein, wenn eine Unterabfrage nicht namentlich definiert ist. Sie können das etwa so ändern:

```
SELECT * FROM (
SELECT * FROM tblAdressen1
UNION
SELECT * FROM tblAdressen2
) As UTable
```

Hier wird das Klammerergebnis, die UNION-Abfrage, als virtuelle Tabelle **UTable** definiert, was anschließend auch im Entwurfsmodus zu sehen ist. Das **[%$##@_Alias]** macht dem **UTable** Platz. Indessen geht im Entwurf auch jetzt noch nicht hervor, dass die angezeigte Tabelle tatsächlich eine **UNION-Abfrage** darstellt. Erst das Schalten in den SQL-Modus lüftet das Geheimnis.

Sie können direkt auch mehrere Tabellen oder Abfragen in einer **UNION-Abfrage** zusammenführen.

Alle für unsere Zielabfrage benötigten Adresstabellen fügen sich so aneinander:

```
SELECT * FROM tblAdressen1
UNION
SELECT * FROM tblAdressen2
UNION
SELECT * FROM tblAdressen3
UNION
SELECT * FROM tblAdressen4
```

Im Ergebnis stehen damit alle Adressen aller Postleitzahlenbereiche bereit.

Wie viele Tabellen in einer **UNION-Abfrage** eingebaut sein dürfen, steht nicht eindeutig fest. An sich gibt es die Aussage von Microsoft, dass eine Abfrage nicht mehr als 32 Tabellen enthalten darf. Entweder ist diese Information veraltet und spiegelt nicht mehr die Eigenschaften der **Access Database Engine** wider, oder für **UNION** gelten andere Regeln. Jedenfalls gelang es uns problemlos, auch 49 Adresstabellen aneinanderzufügen. Erst bei 50 und mehr kam es zur Fehlermeldung

```
Ausdruck oder Abfrage zu komplex
```

Es dürfte auch von der Anzahl der Felder der Quelltabellen abhängen, wann diese Meldung auf den Plan tritt.

Übrigens können sie die gleiche Tabelle auch mehrmals mit sich selbst kombinieren:

```
SELECT * FROM tblAdressen1
UNION
SELECT * FROM tblAdressen1
```

Man sollte hier erwarten, dass sich die Zahl der Datensätze gegenüber **tblAdressen1** verdoppelt. Tatsächlich ist das nicht der Fall!

Der Grund dafür ist, dass Access im Ergebnis keine doppelten identischen Datensätze ausgibt, also zusätzlich ein **DISTINCT** ausführt. Der Ausdruck **UNION** bekommt damit ja auch die korrekte Bedeutung: Das Ergebnis ist die **Mengensumme** beider Tabellen! Allerdings können Sie dies aufheben, indem Sie nach **UNION** noch das Prädikat **ALL** setzen:

```
SELECT * FROM tblAdressen1
UNION ALL
SELECT * FROM tblAdressen1
```

In diesem Fall werden tatsächlich alle Ausgangsdatensätze aneinander gereiht. Vorteil dieser Variante ist die bessere Performance. Access muss hier lediglich die Datensätze summieren, während ohne **ALL** noch die Überprüfung auf Doppelungen stattfindet.

Sortierung und Filterung

Die kombinierten Datensätze können gleich innerhalb der **UNION-Abfrage** sortiert und gefiltert werden, indem die normalen **ORDER BY**-Statements oder **WHERE**-Bedingungen ans Ende des SQL-Ausdrucks gestellt werden:

```
SELECT * FROM tblAdressen1
UNION
SELECT * FROM tblAdressen2
UNION
SELECT * FROM tblAdressen3
WHERE PLZ LIKE "1*"
ORDER BY Nachname;
```

Die drei Quelltabellen werden kombiniert und das Ergebnis schließlich der Filterung nach Postleitzahlengebiet 1 unterzogen, sowie aufsteigend nach Nachname sortiert.

Welche Anforderungen bestehen an die Ausgangstabellen?

Die wichtigste Anforderung ist, dass die Zahl der Felder jeder Tabelle oder Abfrage übereinstimmen. Das gilt indessen nicht für die Namen der Felder. Auch der Datentyp spielt keine Rolle. Versuchen Sie etwa dies:

```
SELECT Nachname FROM tblAdressen1
UNION
SELECT PLZ FROM tblAdressen2
```

Selbst dann, wenn **PLZ** vom Datentyp Zahl (**Long**) wäre, käme ein gültiges, wenn auch wenig sinnvolles, Ergebnis heraus. Die eine Spalte der Abfrage wird in der Anzeige als **Nachname** ausgegeben. Im Zweifel erzeugt Access hier den Datentyp **String**. Sie erkennen nebenbei, dass es unter Umständen durch die Konvertierungen und eindeutige Kombinierung der Datensätze ganz schön zu arbeiten hat.

Völlig belanglos ist die Benennung der Felder allerdings nicht. Nehmen Sie etwa die Ausgangstabellen **tblAdressen1** und **tblAdressen4**. In letzterer gibt es das Feld **Nachname** nicht. Hier lautet es stattdessen **NName**. Dennoch funktioniert

```
SELECT * FROM tblAdressen1
UNION
SELECT * FROM tblAdressen4
ORDER BY Nachname;
```

Versuchen Sie es aber anders herum, so findet die SQL-Engine das im Sortierausdruck verlangte **Nachname** jedoch nicht mehr und gibt deshalb eine Fehlermeldung aus:

```
SELECT * FROM tblAdressen4
UNION
SELECT * FROM tblAdressen1
ORDER BY Nachname;
```

Tastensteuerung abfangen

Nicht immer sind die Steuerungen, die Access über zahlreiche Tasten und Tastenkombinationen ermöglicht, auch wirklich erwünscht. Anwender kennen diese oft gar nicht und sind dann manchmal über Reaktionen der Oberfläche irritiert, die durch versehentliche Anschläge zustande kamen. Unter Umständen kann es auch zu Datenverlust kommen. Wir stellen dar, wie Sie dem Problem mit unterschiedlichen Ansätzen beikommen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1603_Tastenhandling.accdb**

Spezialtasten

Wie viele andere Applikationen auch kennt Access eine große Zahl von Tastenkombinationen, über die sich die Oberfläche steuern lässt. Eigentlich ist das sinnvoll, da der ständige Wechsel zwischen Tastatur und Maus nicht unbedingt komfortabel ist. Voraussetzung aber ist, dass Sie diese Tastenkombinationen auch kennen.

Aus der Hilfe zu Access wurde über einige Umwege die Tabelle **AccessTasten** der Beispieldatenbank gewonnen, welche alle aktuell möglichen Tastenfunktionen auflistet (siehe Bild 1). Der Umfang ist mit etwa 450 Funktionen erschlagend. Je nach Kontext kommen bestimmte Kombinationen mehrfach vor. **ALT+F4** etwa schließt in der Regel ein Fenster oder Formular, beendet aber auch Access selbst, wenn gerade kein Fenster offen ist. Ist das erwünscht? Wenn der Anwender sich auskennt, dann schon, andernfalls ist es eine Gefahr.

AccessTasten

Tasten	
ALT	Gleichzeitiges Schließen des geöffneten Menüs und des
ALT+B	Wechseln zur vorherigen Seite des Assistenten
ALT+BILD-AB	Anzeigen des vorherigen Bildschirms links
ALT+BILD-AUF	Anzeigen des nächsten Bildschirms rechts
ALT+Buchstabentaste	Auswählen einer Option, zum Aktivieren oder Deaktivier
ALT+C	Abbrechen der Seitenansicht oder Layoutvorschau
ALT+DRUCK	Kopieren einer Abbildung des ausgewählten Fensters in
ALT+EINGABETASTE	Anzeigen eines Eigenschaftenblatts in der Entwurfsansic
ALT+EINGABETASTE	Anzeigen eines Eigenschaftenblatts
ALT+EINGABETASTE	Anzeigen des Dialogfelds Eigenschaften
ALT+F	Abschließen des Assistenten
ALT+F11	Wechseln vom Visual Basic-Editor zurück zur Entwurfsar
ALT+F11	Wechseln zwischen dem Visual Basic-Editor und dem vo
ALT+F4	Beenden von Access 2010
ALT+F4	Beenden von Access oder Schließen eines Dialogfelds
ALT+F4	Schließen eines Dialogfelds
ALT+F4	Schließen des Dialogfelds Eigenschaften
ALT+F4	Schließen des Bereichs Feldliste
ALT+F5	Wechseln im Unterdatenblatt zum Datensatznummernfe
ALT+F5	Wechseln zum Feld Seitenzahl, geben Sie dann die Seiter

Datensatz: 7 von 455 | Kein Filter | Suchen

Bild 1: Tabelle AccessTasten mit allen unter Access möglichen Tastenkombinationen

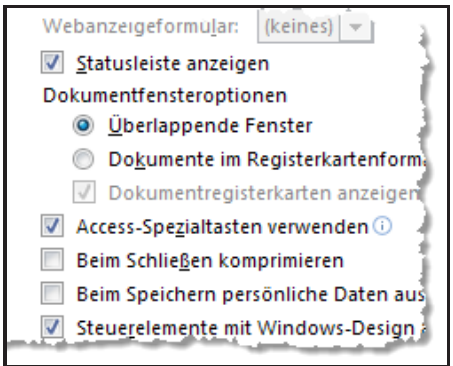


Bild 2: Spezialtasten in den Optionen der Datenbank ein- und ausschalten

Blättern Sie ruhig einmal die Tabelle durch. Sie finden in ihr garantiert viele Kombinationen, die Sie noch nicht kannten, oder über deren Funktion Sie in einem anderen Kontext erstaunt sein werden. Dem Benutzer Ihrer Datenbank werden diese Funktionen sicherlich erst recht unbekannt sein. Sie sind deshalb auf der sicheren Seite, wenn Sie die Spezialtasten von Access in den Optionen der Datenbank abstellen. Sie finden die zugehörige Checkbox in den Optionen unter **Aktuelle Datenbank** und dem Eintrag **Access Spezialtasten verwenden**, wie in Bild 2.

Entfernen Sie hier besser das Häkchen, wenn die Datenbank auch von anderen Benutzern verwendet werden soll.

Leider ist keinesfalls klar definiert, was Microsoft unter Spezialtasten versteht. Sehr viele der Kombinationen in der Tabelle funktionieren auch dann noch, wenn die Option deak-

tiviert ist. Aus diesem Grund müssen wir uns Gedanken machen, welche zusätzlichen Lösungen zum Umgang mit Tastenereignissen infrage kommen.

Autokeys

Dass ein mit **AutoExec** benanntes Makro von Access unmittelbar nach dem Öffnen einer Datenbank ausgeführt wird, wissen Sie wahrscheinlich. Das zweite automatisch ausgeführte nennt sich **AutoKeys**. So vorhanden liest Access die dort definierten Tasten und Tastenkombinationen sofort ein und belegt diese mit den von ihnen bestimmten Aktionen. Die eingebauten Funktionen werden so außer Kraft gesetzt.

Das Thema wurde bereits in der Ausgabe **09/2012** von **Access Basics** ausführlicher dargestellt. Lesen Sie sich dort ein, wenn Sie über die Anlage des Makros mehr erfahren möchten. An dieser Stelle zeigen wir Ihnen am Beispiel von Bild 3, wie unerwünschte Tasturereignisse mit dem **AutoKeys**-Makro abgefangen werden können.

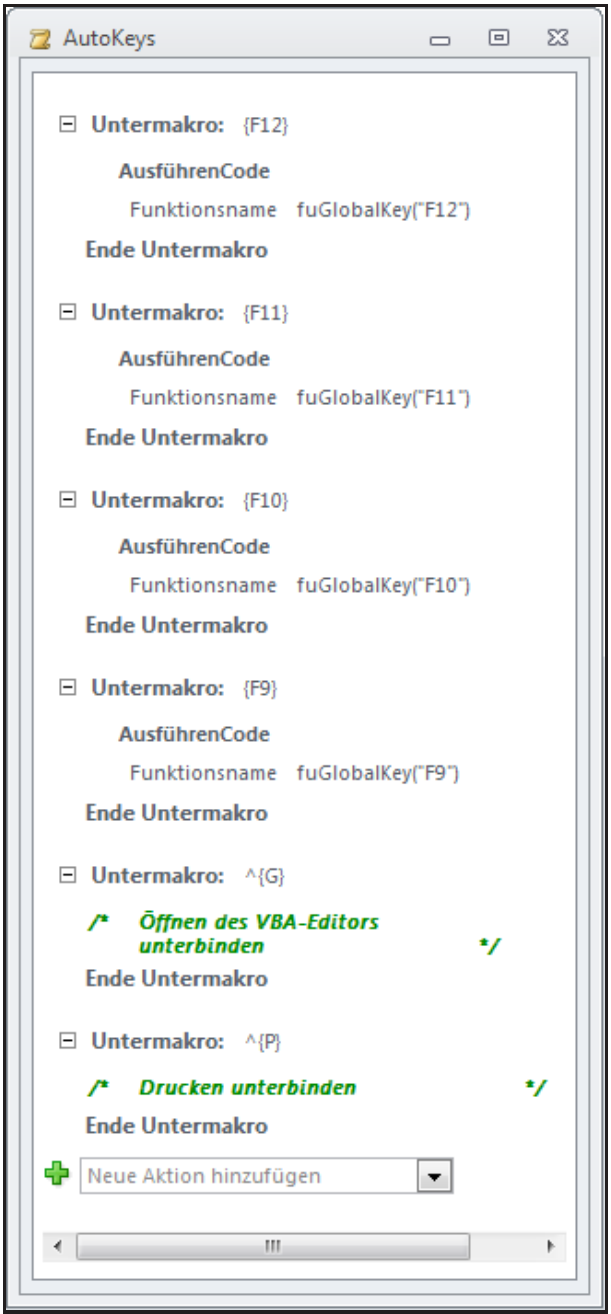


Bild 3: Beispiel eines kurzen Autokeys-Makros

Grundsätzlich muss hier für eine Taste oder eine Tastenkombination eine Aktion angegeben werden. Ohne diese ergibt eine Makrozeile ja keinen Sinn. Was aber mit den unerwünschten Tasten anfangen, damit gar nichts passiert? Am geeignetsten dafür ist die Aktion **Kommentar**. In der untersten Zeile finden Sie die Angabe von **^{P}**, was im **AutoKeys**-Dialekt der Kombination **Strg + P** entspricht. Diese Kombination führt zum Drucken des aktuell angezeigten Objekts. Wäre das ein Formular mit Tausenden von Datensätzen, so würde diese Aktion wohl eher zum Unmut des Anwenders führen, falls sie versehentlich ausgelöst worden wäre. Also ist als Reaktion auf dieses Tastenereignis ein Kommentar mit der Beschreibung **Drucken unterbinden** eingefügt, der dann in grüner Schrift erscheint. Ausgeführt wird demnach bei **Strg + P** lediglich der Kommentar – also gar nichts!

Ähnlich wird mit **Strg + G** verfahren, was üblicherweise den VBA-Editor öffnet. Auch das will man in produktiver Umgebung sicher nicht haben, wobei hier allerdings auch das Abschalten der **Spezialtasten** in den Optionen der Datenbank greifen würde.

Anstelle des kompletten Deaktivierens einer Tastenkombination über eine **Kommentar**-Aktion können Sie aber auch eigene Funktionen aufrufen lassen. Die Tasten **F9** bis **F12** im Beispiel etwa rufen alle die Funktion **fuGlobalKey** auf, wobei jeweils ein anderer Parameter übergeben wird. Diese Funktion muss öffentlich in einem Modul Ihrer Datenbank vorhanden sein. In der Beispieldatenbank finden Sie diese im Modul **mdlGlobalKeys**. Listing 1 verdeutlicht die Routine.

In der Parametervariablen **sKey** wird der im Makro angegebene String entgegengenommen und anschließend in einem **Select Case**-Block verarbeitet. Wurde **F11** gedrückt, was normalerweise den **Navigations-**

Google- und Bing-Maps im Formular

Adressen sind in Datenbanken allgegenwärtig. Access [basics] macht da keine Ausnahme. Sie möchten auf einer Karte sehen, wo sich eine Adresse befindet? Dann bedienen Sie sich etwa Google-Maps oder Bing-Maps im Webbrowser. Wenn Ihnen das aber zu umständlich ist, so bauen Sie doch einfach den Geo-Support direkt in ihre Datenbank ein! Wir zeigen im Beitrag, wie das geht, und wie die Maps unmittelbar in einem Formular angezeigt und angesteuert werden können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1603_Google-Maps.accdb**

Google- und Bing-Maps

Beide Dienste ähneln sich und weisen ähnliche Features auf. Bei der Flexibilität hat **Google** die Nase vorn, bei der Vogelperspektive und, nach unseren Erfahrungen, hinsichtlich der Performance, ist **Bing** etwas interessanter. Ansonsten ist es sicher Geschmackssache, welche Methode Sie bevorzugen.

In die Eingabefelder der Maps geben Sie die gewünschte Adresse oder Koordinaten ein. Im Erfolgsfall wird der Ort nun angezeigt. Wählen können Sie unter verschiedenen Ansichten, wie Karte, Satellit oder 3D. Features, wie Verkehr, Radrouten oder Öffentliche Verkehrsmittel sind nun zusätzlich einstellbar.

Lässt sich das fernsteuern? Sie brauchen keine Fernsteuerung des Browsers über **SendKeys** oder ähnliche fehleranfällige Programmierung, und auch keine **JavaScripte**. Denn zum Glück lassen sich verschiedene Parameter der Dienste direkt in der aufrufenden **URL** mitgeben. Dazu gehören auch die Adresse und einige Ansichtseinstellungen. Aufgabe dieses Beitrags ist die Lösung, die sattsam bekannten Adressen der Beispieldatenbank unmittelbar als Karte im Formular anzuzeigen.

Webbrowser

Da es sich um Webdienste handelt, benötigen wir natürlich zwingend einen Webbrowser zur Darstellung der Karten. Seit einigen Generationen von Access ist dieser bereits als Steuerelement in Formulare integrierbar. Es gibt zwei Alternativen: Entweder Sie verwenden des eingebaute Webbrowser-Steuerelement, oder Sie fügen den Internet-Explorer als **ActiveX**-Steuerelement in ein Formular ein (**Ribbon|Entwurf|Steuerelemente|ActiveX-Steuerelemente ... Microsoft Webbrowser**). Beides kommt so ziemlich auf das Gleiche heraus. Der Unterschied besteht darin, dass sich der eingebaute Browser direkt an ein Feld der Datenherkunft des Formulars binden lässt. Das muss ein Textfeld sein, welches den Link für die anzuwählende **URL** enthält. Beim Navigieren durch die Datensätze des Formulars wird dann automatisch jeweils die entsprechende Seite im Browser angezeigt. Andererseits legt der eingebaute Browser nicht komplett die Schnittstelle zum darunterliegenden **InternetExplorer** offen. Wenn hier mehr Interaktion mit dem Objektmodell über VBA benötigt wird, dann ist das ActiveX-Steuerelement besser geeignet. Vor allem dann, wenn auf eine Datenbindung verzichtet werden kann.

Beide Steuerelemente lassen sich ansonsten auf ähnliche Weise ansteuern, und auch das **Verankern**-Feature von Access bei Größenänderungen des Formulars funktioniert bei beiden anstandslos. Wir verwenden für dieses Beispiel das eingebaute Webbrowser-Steuerelement und zeigen kurz, wie die Sache prinzipiell läuft.

Datengebundener Browser

Die Tabelle **tblURLs** ist die Basis für das Formular **frmMaps**, welches den Webbrowser als einziges Steuerelement enthält. Sie hat einen bescheidenen Aufbau. Neben der nebensächlichen **Autowert-ID** für jeden Datensatz sind im Feld **URL** die Internetadressen gespeichert, welche vom Browser angesteuert werden sollen. Also ist im Formular der **Steuerelementinhalt** des Browsers an dieses Feld gebunden.

Die Tabelle enthält im Auslieferungszustand drei Datensätze. Die erste Internet-Adresse lautet **about:blank**. Das führt dazu, dass das Formular beim Öffnen nur eine leere Seite im

Browser-Steuerelement anzeigt. Das ist sinnvoll, denn ohne irgendeine URL zeigt der Browser eine unerwünschte Seite mit der Meldung **Adresse ungültig** an. **about:blank** vermeidet dies.

Der zweite Eintrag in der Tabelle ist

```
https://www.google.de/maps/
```

Das ist die Startseite von **Google Maps**. Öffnen Sie zum Test das Formular. Klicken Sie in der Navigationsleiste auf den nächsten Datensatz. Leider entspricht das Ergebnis nicht den Erwartungen (Bild 1). **Google** scheint sich nicht mit dem **Webbrowser-Steuerelement** anfreunden zu können. Es hat einige Nachforschungen erfordert, um diesem Verhalten, das **Microsoft Bing** übrigens nicht aufweist, auf die Schliche zu kommen. Die Ursache liegt darin begründet, dass das Webbrowser-Steuerelement, wie auch das ActiveX-Pendant, den Google-Servern die Version 7 des Internet-Explorers melden, obwohl etwa auf dem Windows-System der Internet-Explorer 9 installiert ist. Seit geraumer Zeit akzeptiert **Google Maps** diese Version jedoch nicht mehr und zeigt deshalb die angezeigte Meldung. Endstation?

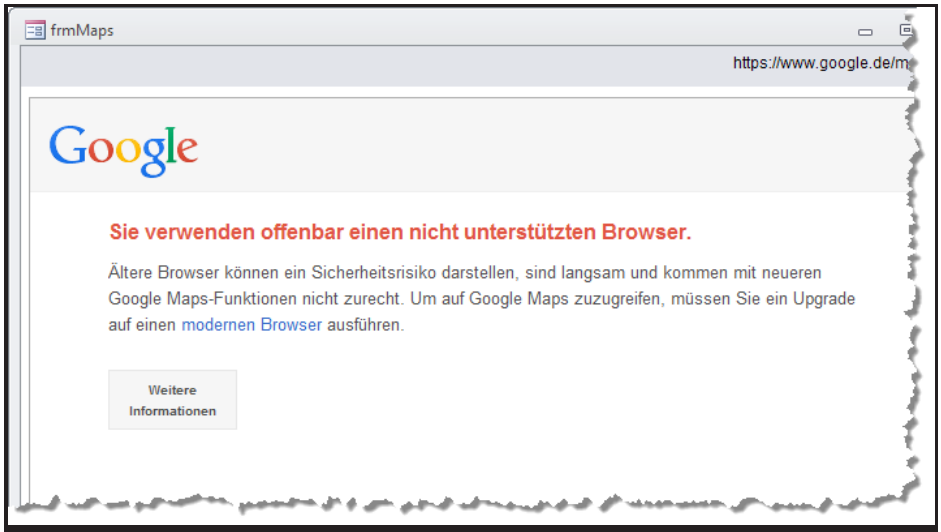


Bild 1: Google Maps akzeptiert das Webbrowser-Steuerelement nicht als Browser

Tatsächlich lässt sich über die **Registry** einstellen, wie sich das WebControl verhalten soll. Über einen bestimmten Schlüssel im Zweig **HKEY_CURRENT_USER**, also dem Benutzerzweig der **Registry**, kann angegeben werden, dass das **WebControl** immer mindestens die Version 9 ausgibt. Die Routine **SetWebControlAsIE9** im Modul **mdlWebControl** der Beispieldatenbank übernimmt das Setzen des entsprechenden Werts in der **Registry**.

Dazu sind keine Administratorrechte nötig, weil jeder Benutzer auf seinen Zweig **HKEY_CURRENT_USER** Zugriff hat. Allerdings erfordert das Ändern dieses Werts einen Neustart von Access. Auf eine Beschreibung der nicht so trivialen Prozedur sehen wir hier ab. Verwenden Sie das Modul auf analoge Weise, nachdem Sie es in eine Ihrer Datenbanken importiert haben, wenn Sie dort **Google Maps** im Formular benötigen.

In der Beispieldatenbank wird die Prozedur vom **Intro**-Formular aufgerufen. Alternativ ginge das natürlich auch über ein **AutoExec**-Makro. Zusätzlich sollte noch die Routine **ChangeWebControlFeature** durchlaufen werden, die weitere Bedingungen für das Webbrowser-Steuerelement über eine **API**-Funktion setzt, damit sich dieses genauso verhält, wie der externe Browser. **SetWebControlAsIE9** überprüft übrigens zuerst, ob der gewünschte **IE9**-Modus bereits gesetzt ist und macht dann weiter nichts. Andernfalls werden Sie über eine Meldung darüber informiert, dass ein Neustart von Access und der Datenbank nötig ist. Die Prozedur **RestartAccess** übernimmt das Neustarten selbst. Sie brauchen also nichts manuell zu veranlassen. Eine neue Instanz von Access wird in der Routine mit dem Pfad der aktuellen Datenbank per **Shell**-Anweisung gestartet, und die offene Instanz wird geschlossen. Wenn Sie die Beispieldaten-

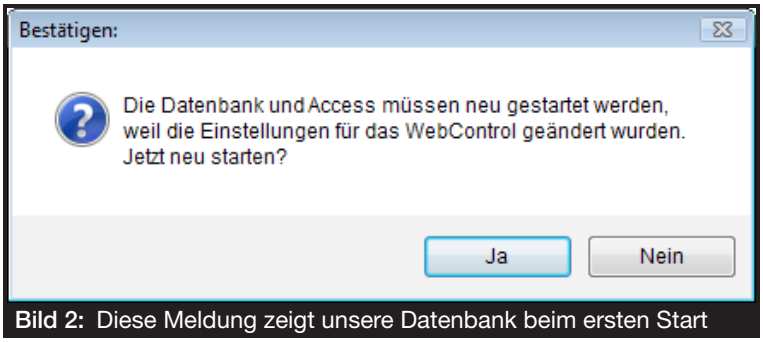


Bild 2: Diese Meldung zeigt unsere Datenbank beim ersten Start

bank zu ersten Mal öffnen, werden Sie mit einiger Sicherheit mit diesem Vorgang konfrontiert (siehe Bild 2).

Ohne diese Modifikation der **Registry** ist leider nichts zu machen. Sie brauchen aber keine Befürchtungen zu haben: Der entsprechende Eintrag ist erstens im sicheren Benutzerzweig und hat, zweitens, keinerlei Auswirkungen auf das System. Er betrifft ausschließlich das **Webbrowser-Control**.

Nachdem diese Änderungen vorgenommen wurden, klappt es auch tatsächlich mit der Anzeige von **Google Maps** im Formular, wie Bild 3 beweist.

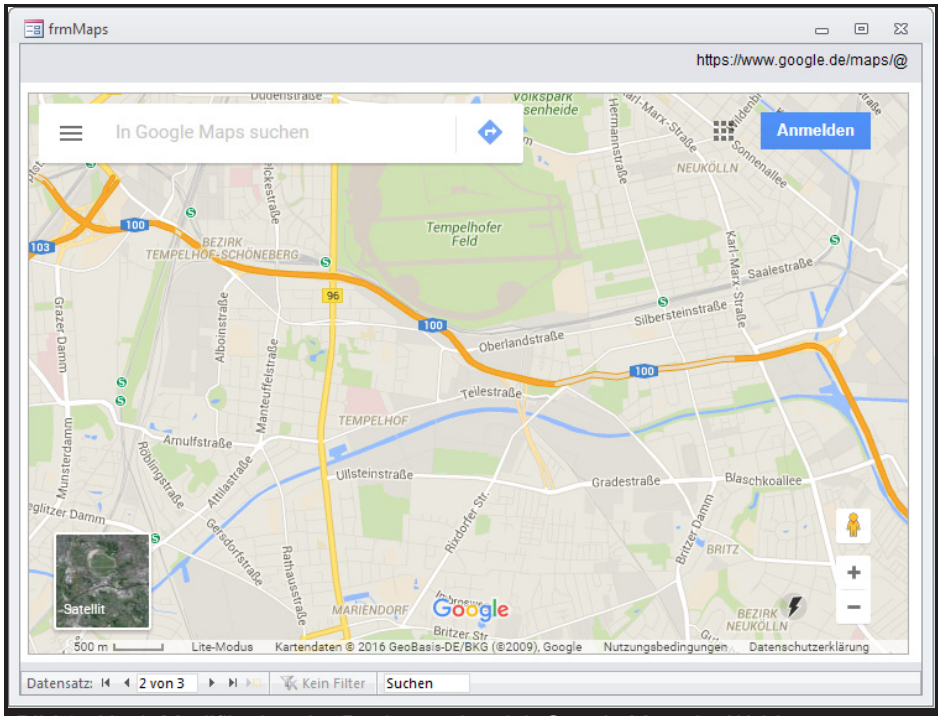


Bild 3: Nach Modifikation der Registry zeigt sich Google Maps im Webbrowser-Steuerelement ordnungsgemäß, aber etwas eingeschränkt

Beim Weiterschalten zum nächsten Datensatz kommt eine URL zum Einsatz, die dem externen Browser entstammt. Dazu wurde Google Maps aufgerufen und als Adresse Berlin eingegeben. Zumindest im Firefox ändert sich nach dem Anzeigen des Orts in der Karte die Adresszeile des Browsers. Im Internet Explorer ist das nicht so. Die so erzeugte Adresse wurde in den Datensatz kopiert. Wird sie dem **Webbrowser-Control** spendiert, so zeigt es die Karte nun auf genau die gleiche Weise. Fast!

Denn wenn Sie die Abbildung genauer inspizieren, wird Ihnen möglicherweise der unscheinbare Blitz am rechten unteren Rand auffallen. Ein Klick auf ihn fördert die Informationen in Bild 4 zutage. Google ist also noch immer nicht mit dem Browser zufrieden und benutzt den sogenannten **Lite-Modus**, in dem einige Features, wie die Fotos am unteren Rand oder **Street View** nicht verfügbar sind. Das macht aber gar nichts, weil diese für unsere Lösung zum Anzeigen von Adressen ohnehin eher stören würden. Das Verschieben, Zoomen der Karte mit der Maus, oder auch das Setzen von Markern per Rechtsklick und das anschließende Routen, funktionieren indessen problemlos. Über das Menü-Symbol können auch die Einstellungen zur Kartenansicht geöffnet werden. Das Ergebnis ist wie gewünscht. Störend sind nur die doch etwas groß geratenen Flächen zu Adresseingabe oder des Menüs. Setzen Sie deshalb die Abmessungen des **Webbrowser-Steuerelements** im Formular auf geeignete Dimensionen. Und geben Sie für die **Verankern**-Eigenschaft am besten den Modus **Quer und nach unten Dehnen** an, damit es sich beim Maximieren des Formulars entsprechend vergrößert.

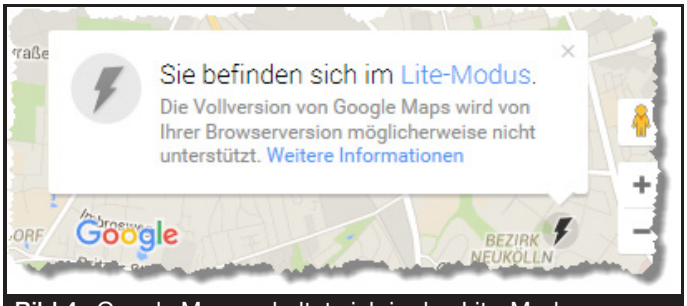


Bild 4: Google Maps schaltet sich in den Lite-Modus

URL-Parameter für Google Maps

Ziel war es ja, Google Maps nicht nur im Webbrowser-Control anzuzeigen, um dort eine Adresse für einen Ort manuell einzugeben, sondern dies über VBA-Programmierung zu erreichen. Ein Link, ähnlich, wie der im dritten Datensatz

der Tabelle **tblURLs**, muss nun über String-Verarbeitung künstlich aus einer Datenbankadresse erzeugt werden.

Das ist nicht weiter schwierig. Sie hängen die Adresse einfach an die folgende URL an:

```
https://www.google.com/maps/place/
```

Für die Suche nach dem Ort **Bayreuth, Königsallee** kommt damit dies heraus:

```
https://www.google.com/maps/place/Bayreuth, Königsallee
```

Zu früheren Zeiten scheiterte eine solche Suche, da weder Leerzeichen in einer **URL** erlaubt sind, noch Umlaute oder andere Sonderzeichen. **Google** ist aber schließlich für seine intelligenten Suchmaschinen bekannt, und so findet es die Adresse inzwischen dennoch, weil es den wahrscheinlichsten Ort heraussucht. Das klappt fast immer.

Die Reihenfolge von Ort, Land und Straße samt Hausnummer sind weitgehend egal, und das Komma als Trennzeichen kann ebenfalls meist entfallen.

Fragt sich noch, ob auch die Darstellung der Karte bereits über einen URL-Parameter zu beeinflussen ist? Das ist zu bejahen, auch wenn die Angelegenheit im Vergleich zu den früheren Versionen von **Google Maps** deutlich undurchsichtiger geworden ist und außerdem leider nicht von **Google** dokumentiert. Versuch und Irrtum machen jedoch klug, und so fanden einige Leute heraus, dass die Steuerparameter für die Darstellung sich in einem **data**-Block verstecken. Hängen Sie etwa zusätzlich diesen String an die bisherige Such-URL an:

```
/data=!5m1!1e0
```

Das führt zur normalen Kartendarstellung. Mit

```
/data=!3m1!1e3
```

jedoch kommt es zur Satellitenansicht. Voll ausgeschrieben:

```
https://www.google.com/maps/place/Bayreuth, Königsallee/data=!3m1!1e3
```

Zu weiteren Möglichkeiten später mehr.

Kundenadressen mit Landkarte im Formular

Nachdem wir die Essentials ermittelt haben, kann es an die Gestaltung des Formulars mit Kundenadressen und eingebauter **Google Maps**-Karte gehen. Bild 5 zeigt die Entwurfsansicht des Formulars **frmKunden**.

Neben den Adressfeldern und der Combobox rechts oben zur Navigation durch die Datensätze ist im unteren Bereich das Browser-Steuerelement **ctlWebX** untergebracht. Seine **Verankern**-Eigenschaft steht auf **Quer und nach unten dehnen**, damit es sich bei Größenänderungen des Formulars entsprechend ausdehnt. Der **Steuerelementinhalt** ist an das Feld **MapsURL** gebunden. Dieses Feld ist der Tabelle tblKunden

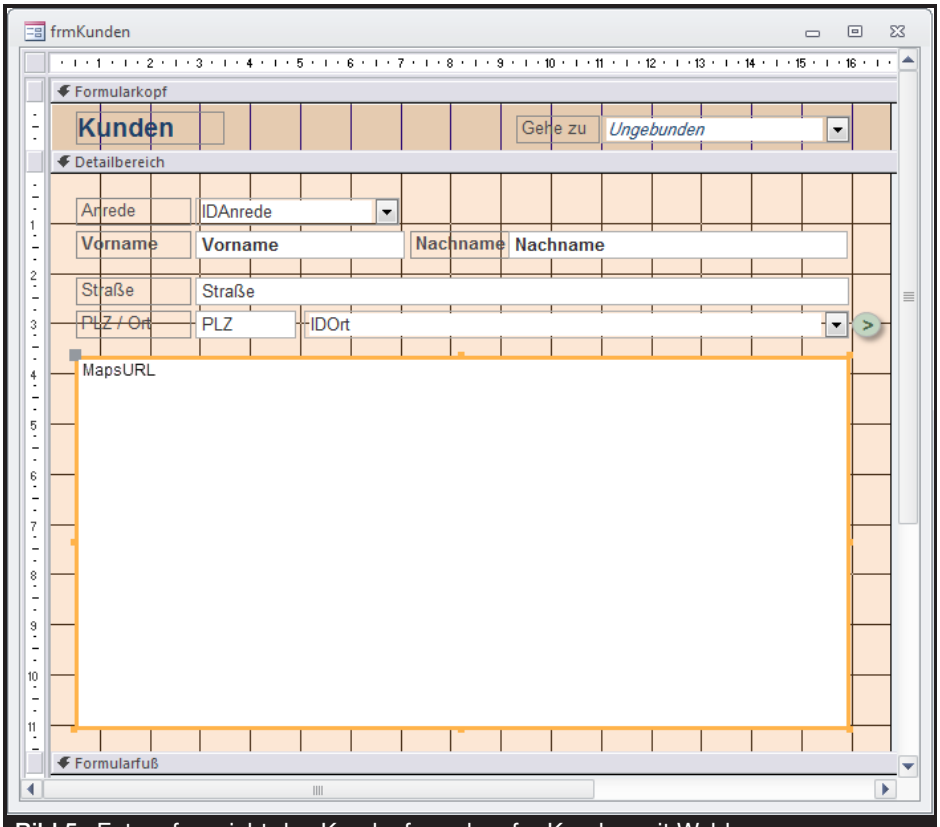


Bild 5: Entwurfsansicht des Kundenformulars frmKunden mit Webbrowser

hinzugefügt worden. Ihr Feldaufbau sieht so aus:

ID	Long (Autowert)
IDAnrede	Long
Vorname	Text
Nachname	Text
Straße	Text
PLZ	Text
IDOrt	Long
MapsURL	Text
BingURL	Text

IDAnrede ist mit der Nachschlagetabelle **tblAnreden** verknüpft, **IDOrt** mit **tblOrte**. In **MapsURL** soll später der Link zur **Google Maps**-Karte gespeichert werden, in **BingURL** der Link zur **Bing Maps**-Karte im Browser. Beide Textfelder haben den Standardwert **"about:blank"**, damit bei einem neuen Datensatz der Browser keine Fehlermeldung anzeigt, sondern eine leere Fläche. Diese Felder werden erst dann gefüllt, wenn die Maps-Dienste den Ort bereits ermittelt haben.

Das geschieht indessen nicht automatisch, sondern wird durch Auswahl des Eintrags **(Google Maps Link speichern)** in der Navigationscombo oben rechts ausgelöst. Deren Datenherkunft besteht aus der andernorts bereits beschriebenen **UNION-Abfrage**, die auf der Tabelle **tblKunden** basiert und den künstlichen Einträgen **(Neuer Datensatz)** und **(Datensatz löschen)**, dem die erwähnte Zeile mit der ID **-2** hinzugefügt wurde.

Das Erstellen des Such-Strings für Google Maps übernimmt die Ereignisprozedur **Beim Anzeigen**

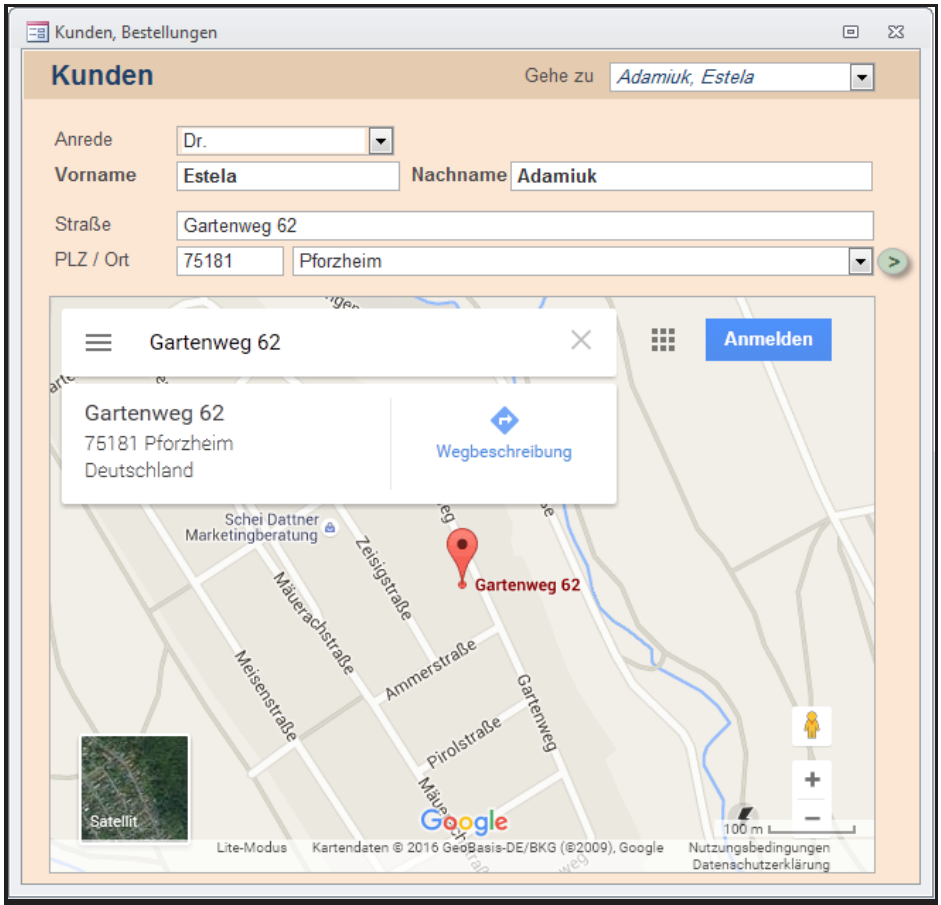


Bild 6: Das Kundenformular mit in Google Maps gefundenem Ort zur Laufzeit

(**Form_Current**) des Formulars. Das bedeutet, dass bei Navigation zu einem Kundendatensatz sogleich die **Maps-URL** produziert und dem Browser-Steuerelement zugewiesen wird. Also zeigt im Idealfall das Formular, wie in Bild 6, immer eine Karte der Adresse des Orts an. Die Ereignisprozedur testet dabei zunächst, ob das Feld **MapsURL** bereits einen gespeicherten Link enthält:

```
If Me!MapsURL <> "about:blank" Then
    ...(Such-String erzeugen)
End If
```

Da der Standardwert des Felds auf **about:blank** steht, ist das ein Kriterium dafür, ob bereits ein Link gespeichert wurde, denn dann sieht sein Inhalt anders aus. In diesem Fall tut die Routine weiter nichts, denn das Steuerelement ist ja an den Link gebunden und zeigt ihn im Browser automatisch an.

Das Zusammensetzen des Such-Strings ist als Teil der Prozedur in Listing 1 abgebildet. In **sURL** steht die Basis-URL für **Google Maps**, in **sLoc** der Adressteil des Links. Ist das Feld **IDOrt** ungleich Null, also mit einem Wert belegt, so wird der Fokus auf das Kombinationsfeld **cbOrt** gesetzt, damit sein angezeigter Text ausgelesen werden kann. Wir brauchen ja den Text und nicht die **ID** des Orts. Den Text einer Combobox aber kann man nur dann auslesen, wenn der Fokus des Formulars auf ihm

Elemente für die Bildanzeige

Der Häufigkeit von Fragen in Foren nach zu schließen, scheint der Umgang mit Bildern in Formularen und Berichten für viele Entwickler ein Thema hoher Priorität zu sein. Zwar geht es dort meist um das Laden von Bilddateien über VBA-Routinen, doch dass es neben dem Access-eigenen Bildsteuerelement noch weitere Möglichkeiten zur Darstellung gibt, wird dabei gerne übersehen. Hier ein Überblick über die Alternativen und ihre Vor- und Nachteile.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1604_ ImageCtls.zip**.

Bilddateien unter Access

Selbst **Microsoft** hat erkannt, dass Bilder in Datenbanken eine zunehmend größere Rolle spielen. Ob Fotos von Personen oder Gegenständen, ob Kataloge oder Produktdatenbanken, Privatgebrauch oder Gewerbe, überall finden sich Fälle, in denen Bilder in Datenbanken ein zusätzliches Informationselement darstellen können.

Bis zur Version von **Office 2007** beschränkten sich die Möglichkeiten auf das eingebaute **Bildsteuerelement** oder auf das **OLE-Feld** oder **-Steuerelement**. Letzteres war, wenn man von aufwändiger VBA-Programmierung absehen wollte, zugleich die einzige Lösung, um Bilddateien binär direkt in Tabellen abzuspeichern. Deren Anzeige im Formular war dann immer ein Risiko, weil der hierfür erforderliche **OLE-Server** in Gestalt eines externen Programms, wie **PaintBrush**, auf dem System vorliegen muss. Auch ein **Photoshop**-Bild lässt sich auf diesen Weg einbetten, doch das geht dann eben nur mit installierten **Photoshop** sowohl auf Entwickler-, wie Zielmaschine. Über die Performance dieser Lösung kann auch nichts Positives berichtet werden.

Darum gab es zwei Weiterentwicklungen: Zum einen lässt sich ein Bildsteuerelement nun direkt an ein Tabellenfeld binden, das die Pfade zu den Bilddateien enthält, zum anderen bietet die Kombination von neuem **Anlagefeld** und **Anlagesteuerelement** einen komfortablen Ersatz für die in die Jahre gekommene **OLE**-Lösung. Einst von vielen Entwicklern verschrien, weil Binärformate in Datenbanken angeblich nicht zu suchen hätten, ist das inzwischen ein alter Hut von gestern. Alle **DBMS-Systeme** haben längst den Support für Binärdaten eingebaut und setzen teilweise darauf noch spezielle Bibliotheken, wie jene der Geoinformationssysteme (**GIS**) mit ihren komplexen Datenformaten. Klar muss man sich durchaus noch Gedanken darüber machen, ob die großen Dateien wirklich in Tabellen untergebracht sein müssen. Schließlich weist Access ein Limit von 2 GB für eine Datenbank auf. Bei kleineren Fotos in komprimiertem Format und bei überschaubarer Datensatzzahl sollte das jedoch kein Problem darstellen.

Hier geht es nicht um die Anwendungsbereiche für Bilder in Datenbanken. Dazu fällt Ihnen sicher selbst genug ein. Wir nehmen sechs Steuerelementlösungen unter die Lupe und zeigen deren Ansteuerung, sowie deren Eigenschaften. Im Einzelnen sind dies die Folgenden:

- Das Access-**Bildsteuerelement**
- das **OLE-Feld**
- das **Anlagesteuerelement**
- das **MS-Forms Image Control**
- die Access-**Schaltfläche**
- das **Webbrowser Control**

Auf einen Blick

Im Formular **frmBilder** der Beispieldatenbank sind alle sechs Steuerelemente miteinander vereint und zeigen jeweils dasselbe Foto an. So können Sie sie direkt vergleichen und Verhalten, Features und Qualität begutachten. Die Steuerelemente haben alle die gleichen Abmessungen. Quelle für die Fotos sind Bilddateien im Datenbankverzeichnis, beim Anlage- und OLE-Steuerelement jedoch auch direkt in einer Tabelle gespeicherte Bilddaten.

Bild 1 zeigt das Formular zur Laufzeit, wobei hier der zweite Datensatz

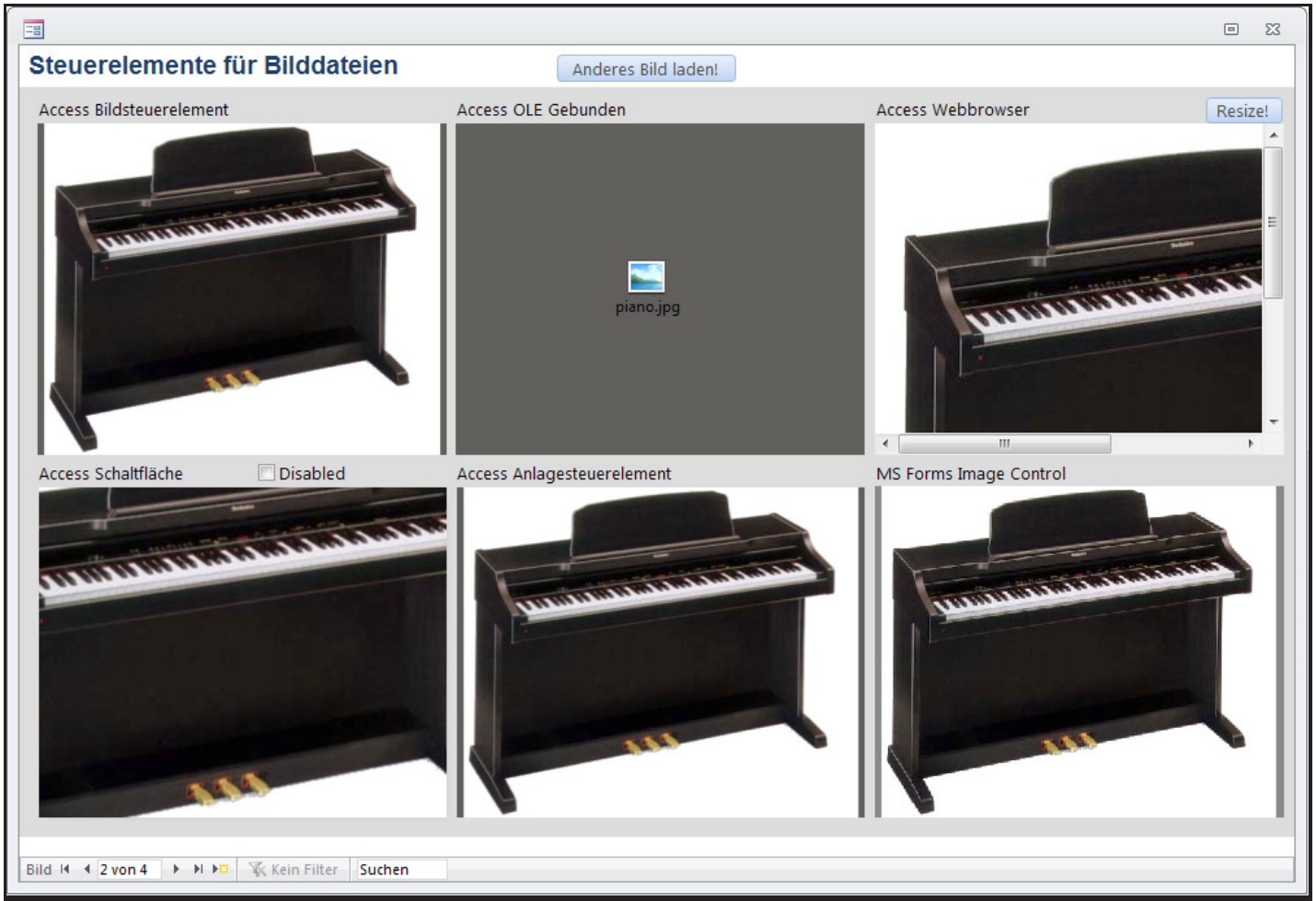


Bild 1: Die sechs Steuerelemente im Formular frmBilder zeigen alle dieselbe Bilddatei an – fast jedenfalls!

der Tabelle **tblBilder** eingeblendet ist, welcher auf der Bilddatei **piano.jpg** basiert. Bis auf das **OLE-Steuerelement** stimmt die Anzeige mit den Erwartungen überein. **Bildsteuerelement**, **Anlagesteuerelement** und **MS Forms Image** zeigen eine identische Darstellung und Skalierung, während die **Schaltfläche** und das **Webbrowser Control** offenbar die Originalgröße des Bilds wiedergeben. Bei genauerem Hinsehen unterscheidet sich allerdings auch die Anzeigequalität. **Bild**- und **Anlagesteuerelement** geben das Bild etwas unscharf wieder, was sich auf ein **Antialiasing** zurückführen ließe. Das **MS Forms Control** hingegen zeigt beim Skalieren kein **Antialiasing**, was teilweise zu verpixelten Kanten führt. Den besten Kompromiss liefert das **Webbrowser Control** ab.

Warum das gebundene **OLE-Feld** nur ein Symbol statt des Inhalts anzeigt, führen wir später aus.

Den Entwurf des Formulars erläutern wir später – vorher werfen wir einen Blick auf die Daten, die ihm zugrunde liegen.

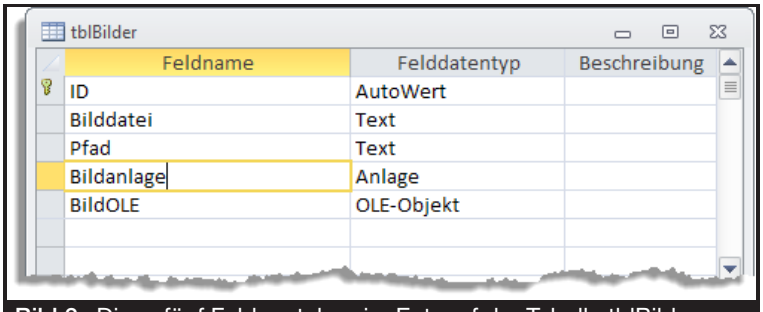


Bild 2: Diese fünf Felder stehen im Entwurf der Tabelle tblBilder

Bilddaten

Die Tabelle **tblBilder** hat den Aufbau aus Bild 2. **Bilddatei** ist ein Textfeld, welches nur den Dateinamen aufnimmt, etwa **piano.jpg**. **Pfad** enthält das Verzeichnis, in dem die Datei liegt.

Der Inhalt dieses Felds ist zunächst leer und wird erst beim Start der Datenbank über eine VBA-Anweisung im **Intro**-Formular mit dem aktuellen Verzeichnis festgesetzt:

```
CurrentDb.Execute _
   "UPDATE tblBilder SET Pfad='" _
   & CurrentProject.Path & "'"
```

Damit steht in jedem Datensatz der Tabelle in **Pfad** das Verzeichnis aus **CurrentProject.Path**.

Bildanlage ist ein **Attachment (Anlagefeld)** mit der Beschriftung **Anlagebild**, welches später das **Anlagesteuerelement** speisen soll. Und **BildOLE** schließlich ist ein Feld vom Typ **OLE-Objekt**, an das im Formular das **Gebundene OLE-Steuerelement** gehängt wird. Die Datenblattansicht der Tabelle zeigt Bild 3. Hier wurden bereits die entsprechenden Bilddateien in das Attachment geladen, und ebenso als **OLE-Objekte** in das Feld **BildOLE**. Interessant ist, was Access aus den geladenen Daten im **OLE-Feld** in der Ansicht macht. Im einen Fall zeigt es die Beschriftung **Package**, im anderen ein **Bitmap Image**. Die Vorgehensweise, falls Sie noch nie ein OLE-Objekt in eine Tabelle speicherten: Klicken Sie

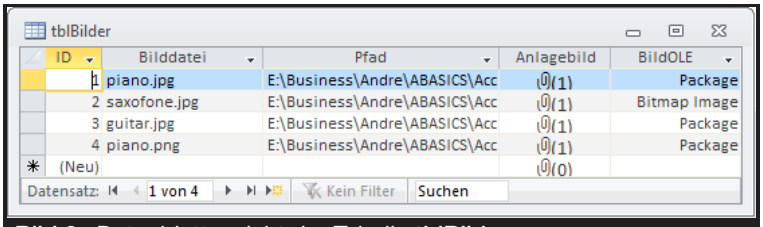


Bild 3: Datenblattansicht der Tabelle **tblBilder**

Bild 4: Kontextmenü Objekt einfügen...

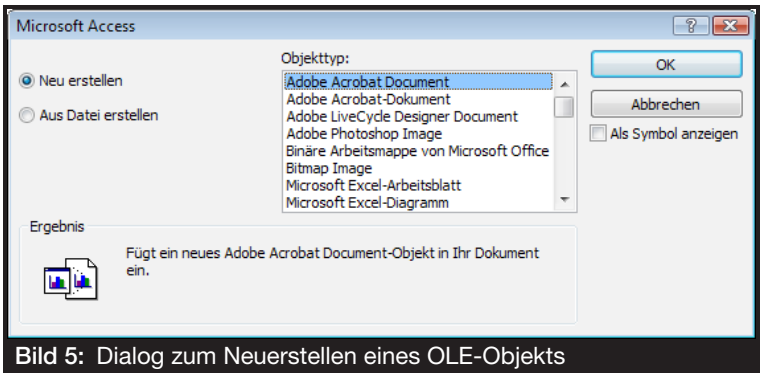


Bild 5: Dialog zum Neuerstellen eines OLE-Objekts

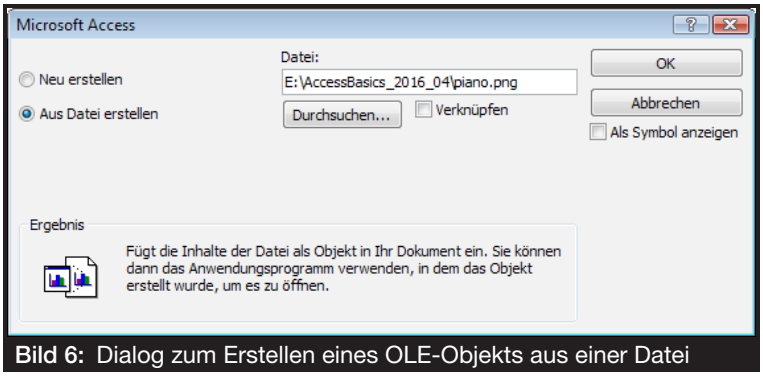


Bild 6: Dialog zum Erstellen eines OLE-Objekts aus einer Datei

rechts in die gewünschte Tabellenzelle und wählen aus dem Kontextmenü (Bild 4) den Eintrag **Objekt einfügen...**.

Access rechnet dann ein bisschen, um die im System installierten sogenannten **OLE-Server** zu ermitteln und anschließend in einem Dialog zu präsentieren (Bild 5).

Sie haben nun die Wahl, ein Dokument auf Basis der in der Listbox aufgeführten Anwendungen komplett neu zu erstellen, oder aber, nach Klick auf Datei (Bild 6), eine existierende Bilddatei in ein **OLE-Objekt** zu verwandeln. Windows entscheidet im Hintergrund, welcher **OLE-Server** dabei zum Einsatz kommt. Dies hängt von den im System eingestellten Dateitypverknüpfungen ab. Statt physischem Speichern der Datei in das Tabellenfeld könnten Sie auch ein Häkchen auf **Verknüpfen** im Dialog setzen, was dann nur eine Referenz auf die Datei und deren zuständigen **OLE-Server** hinterlässt.

Laden Sie etwa eine **JPG**-Datei in das **OLE-Feld**, so verwandelt sich diese in ein **Package**. Das Gleiche gilt für **PNG**- oder **TIFF**-Dateien. Lediglich bei **BMP**-Dateien kommt es zum Eintrag **Bitmap Image**. Und genau dies ist in der Regel auch der einzige Dateityp, der unmittelbar

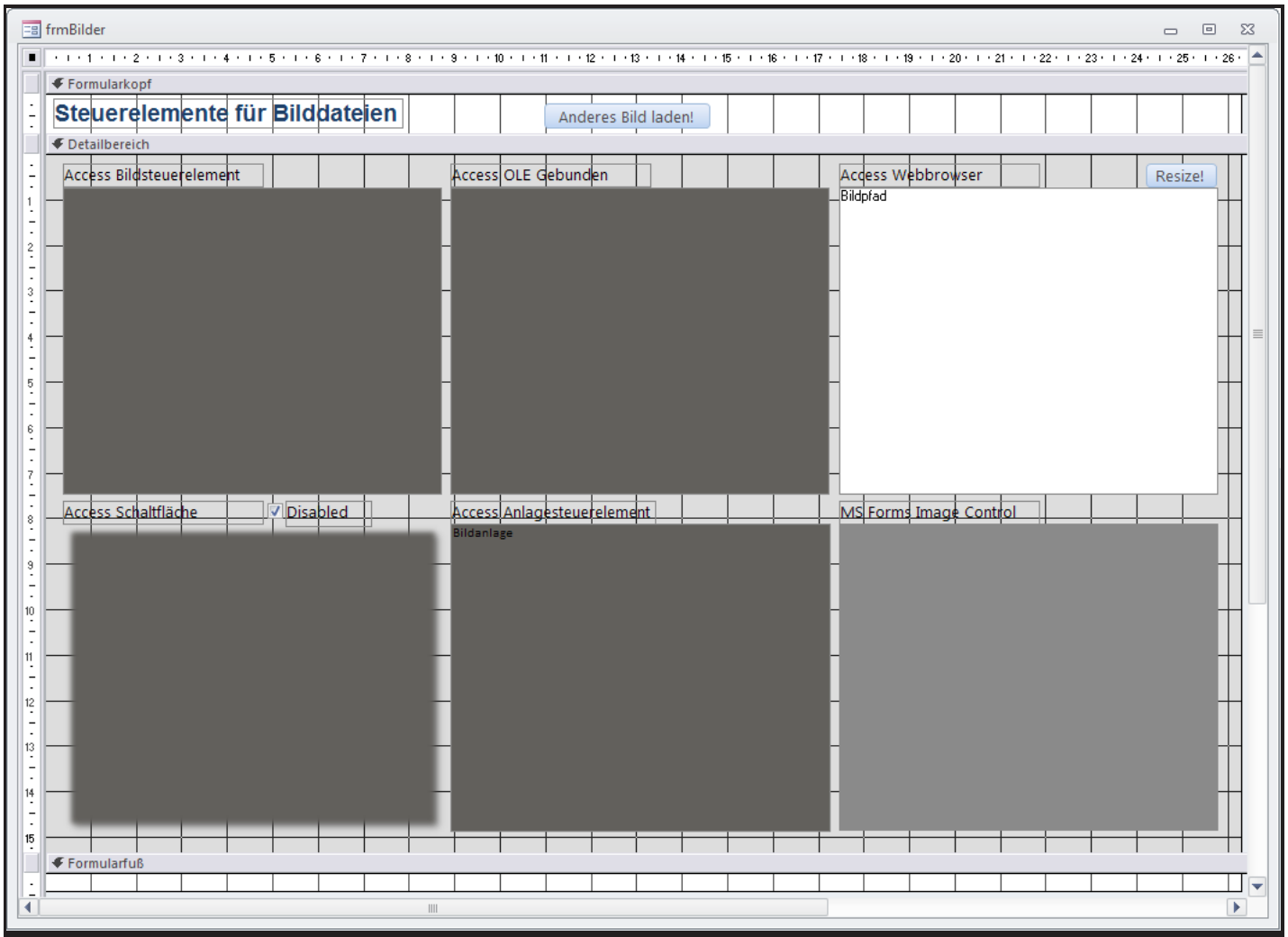


Bild 7: Der Entwurf des Formulars frmBilder mit den sechs für die Bildanzeige eingesetzten Steuerelementen

im **Gebundenen OLE-Steuerelement** angezeigt werden kann. Denn Windows verknüpft mit den meisten Grafiktypen die Windows-**Bildvorschau**, welche keinen **OLE-Server** darstellt.

Bei **BMP**-Dateien hingegen kommt, warum auch immer, **PaintBrush** zum Einsatz, welches zur Laufzeit innerhalb des **OLE-Steuerelements** dann die Grafik anzeigt. Allein damit schon scheidet das OLE-Steuerelement als sinnvolles Mittel zur Bildanzeige aus. Bitmap-Dateien haben wegen ihrer erheblichen Größe in Datenbanken nichts zu suchen!

Zurück zum Entwurf des Formulars, den Sie in Bild 7 finden. Für das **Bildsteuerelement** links oben finden Sie im **Entwurfs-Ribbon** zum Einsetzen möglicherweise nicht gleich die passende Schaltfläche. Seit **Access 2010** wird hier eine spezielle **Dropdown-Box** verwendet, die an die Tabelle **MSysResources** gebunden ist. Sie befindet sich ganz rechts und hat die Aufschrift **Bild einfügen**. Sie bietet zwei Möglichkeiten: Entweder Sie wählen ein bereits in der Datenbank befindliches Bild aus, das auch gleich als Vorschaubild erscheint, oder Sie gehen auf **Durchsuchen...** und laden eine Bilddatei. Erst dann wird das Steuerelement im Formular angelegt. Ein leeres lässt sich leider nicht erzeugen. Sie müssen gegebenenfalls ein Dummy-Bild laden und es anschließend im Eigenschaftenblatt des Steuerelements löschen (Eintrag **Bild**). Dennoch wird dieses Bild dann in der **Dropdown-Galerie** im Ribbon angezeigt, weil es bereits als Datensatz in **MSysResources** aufgenommen wurde. Dort allerdings können Sie diesen Datensatz anschließend getrost löschen. Nun ja, nicht gerade der Weisheit letzter Schluss...

Unterabfragen als Feldinhalt

Dass gespeicherte Abfragen selbst wieder als Pseudotabellen in andere Abfragen integriert und dort mit anderen Tabellen verknüpft werden können, dürfte Ihnen bekannt sein. Weniger verbreitet jedoch sind jene Abfragen, die ungespeichert als SQL-Ausdruck zur Berechnung von Feldinhalten und Bedingungen dienen. Diese mit den Schlagwörtern Unterabfrage oder Subselect bezeichneten Ausdrücke sind Gegenstand dieses Beitrags.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1604_Unterabfragen.accdb**

Unterabfragen, Subselects

Eine **Unterabfrage** (engl. **Subselect**) ist nichts weiter, als eine vollständige in SQL geschriebene Auswahlabfrage, deren Ergebnis oder Ergebnisse zur Anzeige eines einzelnen Felds oder zur Bestimmung eines Auswahlkriteriums in einer anderen Abfrage dienen. Im Folgenden verwenden wir einige Beispiele, um verschiedene Möglichkeiten im Umgang mit diesen Unterabfragen zu demonstrieren.

Grundlage für die Beispieldatenbank ist das Datenmodell in Bild 1, welches Kunden und deren Bestellungen verwaltet. Neben den Kundendaten, bestehend aus der Tabelle **tblKunden** und deren Nachschlagetabellen **tblAnreden**, **tblOrte** und **tblLaender**, sind die Bestellungen in der Tabelle **tblBestellungen** festgehalten und zu jeder Bestellposition jeweils ein Datensatz in der Tabelle **tblBestelldetails**. Die Artikel jeder Bestellposition kommen schließlich aus der Nachschlagetabelle **tblArtikel**.

Auswertungen über Subselects

Eine Anforderung an eine Auswertung könnte nun sein, die Anzahl von Bestellungen in einer Übersicht darzustellen, die jeder Kunde getätigt hatte. Die dafür geeignete Abfrage wäre eine **Gruppierungs**- beziehungsweise **Aggregatabfrage**. Bild 2 zeigt, wie Sie diese gestalten könn-

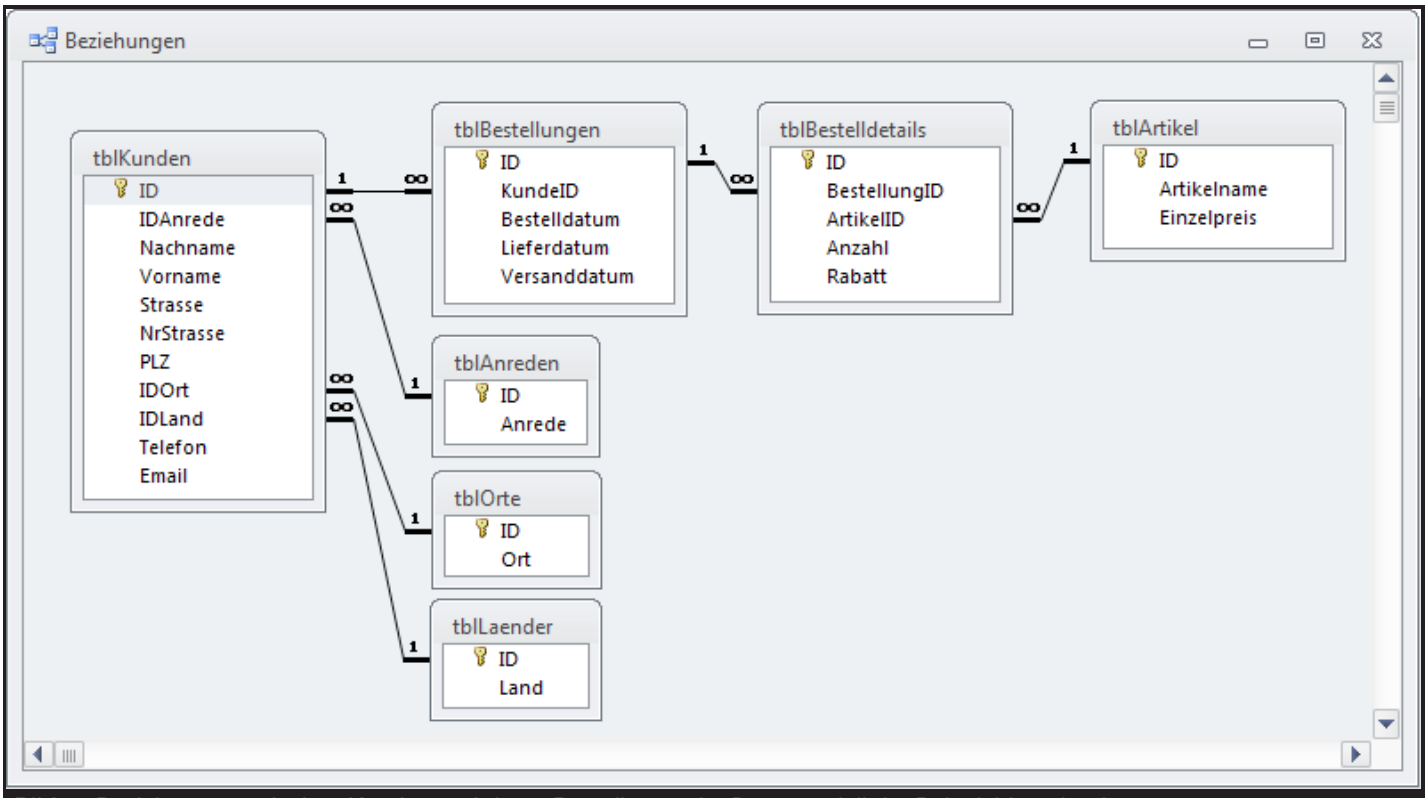


Bild 1: Beziehungen zwischen Kunden und deren Bestellungen im Datenmodell der Beispieldatenbank

ten. Aus der Kundentabelle entnehmen wir lediglich die **ID** der Datensätze und den vollständigen Namen **Kunde**, welcher sich aus **Nachname** und **Vorname** zusammensetzt:

```
...Nachname & ', ' & Vorname AS Kunde
```

Um die Anzahl von Bestellungen zu jedem Kunden zu ermitteln, wird die Tabelle **tblBestellungen** mit der Kundentabelle 1:1 verknüpft. Die Berechnung der Anzahl erfordert als Aggregat, dass Sie aus der Abfrage eine **Gruppierungsabfrage** machen, indem Sie das Summensymbol ∑ im Entwurfs-Ribbon anklicken. Für die nun eingeblendete Zeile Funktion geben Sie bei **ID** und **Kunde** die **Gruppierung** an, und beim Feld **Anzahl Bestellungen** die **ID** von **tblBestellungen**, sowie die Funktion **Anzahl**. Die Bedingung, dass diese Anzahl größer, als 0, sein muss, ist eigentlich überflüssig, denn durch die 1:1-Verknüpfung werden ohnehin alle Datensätze ausgefiltert, die keinen Bezug zwischen den Tabellen besitzen.

Im Ergebnis haben Sie nun alle Kunden, die überhaupt etwas bestellten und die Gesamtzahl ihrer Bestellungen. Sie könnten noch nach diesem Feld absteigend sortieren, um die Top-Besteller zu ermitteln.

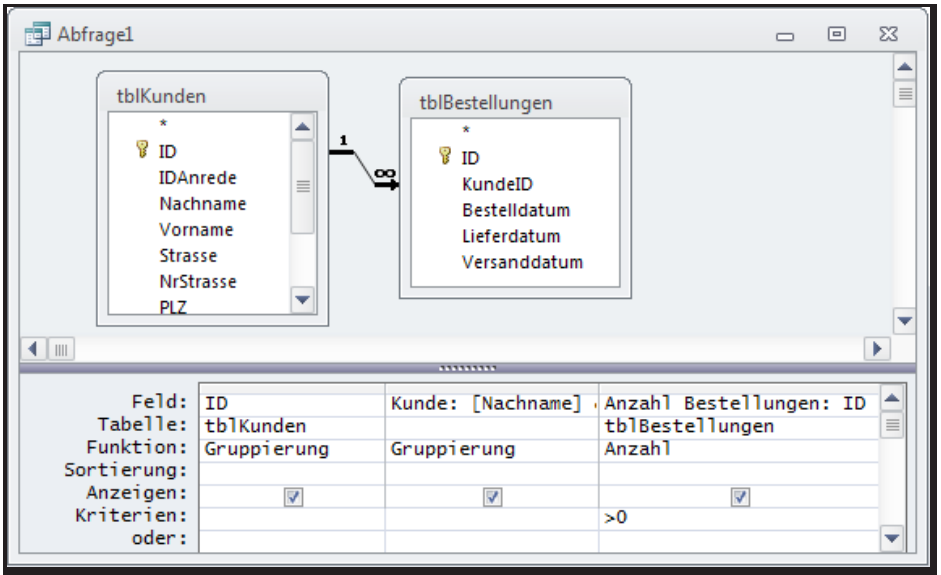


Bild 2: Die Abfrage gibt nur jene Kunden aus, die mindestens einmal bestellten

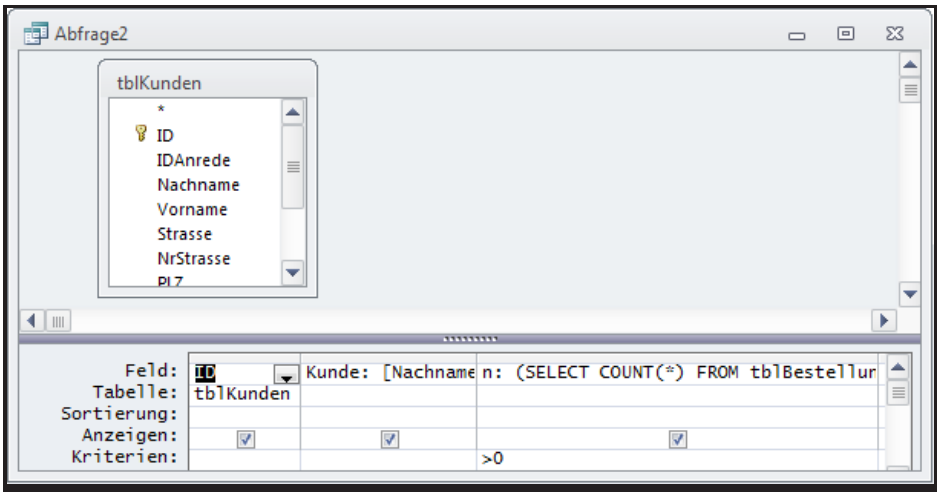


Bild 3: Auch hier zeigen sich nur Kunden mit Bestellungen, jedoch mit Unterabfrage

```
SELECT tblKunden.ID, [Nachname] & "," & [Vorname] AS Kunde,
(SELECT COUNT(*) FROM tblBestellungen As TMP WHERE (TMP.KundeID=tblKunden.ID)) AS n
FROM tblKunden;
```

Listing 1: Das ist dann der SQL-Code der Abfrage2

Diese Auswertung lässt sich aber auch anders realisieren. Statt die Tabelle **tblBestellungen** direkt in den Abfrageentwurf zu ziehen, wird sie in einen **SELECT**-Ausdruck als **Unterabfrage** integriert, der das Ausgabefeld **Anzahl Bestellungen** speist. Wie das aussieht, zeigt Bild 3. Hier wird bereits deutlich, dass solche Abfragen in der Entwurfsansicht etwas schwieriger zu handeln sind, da die SQL-Ausdrücke für die Unterabfragen relativ lang ausfallen und damit die Spaltenbreite dieser berechneten Felder ziemlich breit aufgezogen werden müsste. Entweder Sie verwenden dann die **Zoom**-Funktion für das Feld (rechter Mausklick), oder Sie schalten besser gleich in die **SQL-Ansicht**, die den Code von Listing 1 zutage fördert. Das Ergebnisfeld **Anzahl Bestellungen** ist hier mit **n** bezeichnet. Es errechnet sich über die Aggregatfunktion **COUNT**(*) auf die Datensätze von **tblBestellungen**. Da wir ja aber nicht für jeden Kunden die Gesamtzahl **aller**

Bestellungen ausgeben möchten, sondern nur die Zahl **seiner** Bestellungen, braucht es noch eine zusätzliche **WHERE**-Bedingung. Pro Kundendatensatz müssen die Bestellungen gemäß seiner **ID** gefiltert werden.

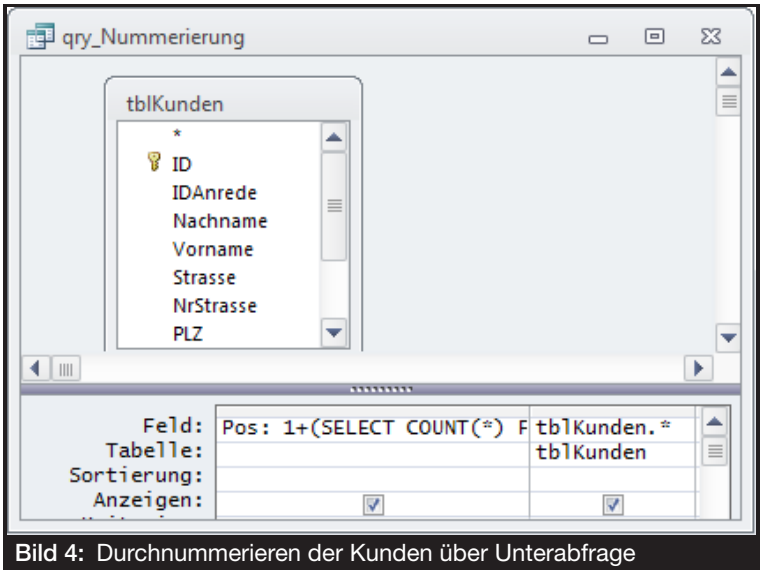


Bild 4: Durchnummerieren der Kunden über Unterabfrage

Hier kommt nun ein Trick ins Spiel: Die Tabelle **tblBestellungen** wird in den **Alias**-Namen **TMP** umbenannt. Auf ihr Feld **KundeID** können Sie somit über **TMP.KundeID** Bezug nehmen. Ihr Wert muss mit der **ID** der Tabelle **tblKunden** übereinstimmen, damit die Filterung greift. Also setzen Sie diese Bedingung ein:

```
...WHERE TMP.KundeID = tblKunden.ID
```

Das Ergebnis stimmt mit dem der Abfrage aus Bild 2 überein.

Man könnte meinen, dass die Performance dieser Abfrage darunter litte, dass für jeden Kundendatensatz die Unterabfrage zur Anzahlberechnung erneut ausgeführt werden muss. Tatsächlich unterscheidet sie sich nicht nennenswert von der gruppierten Abfrage oben, denn auch bei Aggregatabfragen muss intern für jeden Datensatz eine eigene neue Berechnung durchgeführt werden.

Nummerieren

Der prominenteste Vertreter von Unterabfragen ist die Lösung zum Durchnummerieren der ausgegebenen Datensätze. Sie möchten etwa alle Kunden der Datenbank ausgeben und die Datensätze fortlaufend nummerieren. Unter **Excel** wäre das ein leichtes Unterfangen. Sie fügten hier einfach ganz vorn eine zusätzliche Spalte ein, schrieben eine **1** und eine **2** in die beiden ersten Zellen hinein und zögen die Spalte nach Markierung dieser zwei Zellen über die ganze Kundentabelle auf.

Leider gestaltet sich das unter Access weitaus schwieriger. Eine Nummerierungsfunktion fehlt hier.

Über eine gewiefte Unterabfrage lässt sich die Nummerierung dennoch erreichen. Voraussetzung allerdings ist, dass die Tabelle einen **Primärschlüssel**, wie die **Autowert-ID** von **tblKunden**, aufweist. Ohne diesen wäre die Sache nur über eine gar nicht so einfache VBA-Funktion zu lösen.

Das Feld **Pos** in Bild 4 stellt das Nummernfeld dar. Es berechnet sich über eine Unterabfrage, die in Listing 2 besser zu überblicken ist. Zunächst ermittelt **COUNT** die Anzahl der Datensätze der Tabelle **tblKunden**. Hätte sie 9.000 Datensätze, so gäbe **Pos** in jeder Zeile eben diesen Wert aus, was nicht wünschenswert ist. Eine zusätzliche Bedingung ändert aber die Situation.

Auch hier wird die Tabelle intern über den **Alias** in **TMP** umbenannt. Anschließend filtert die **WHERE**-Bedingung diese **TMP**-Tabelle so, dass nur jene Datensätze berücksichtigt werden, deren **ID** (**Autowert**!) kleiner ist, als die aktuell im Datensatz stehende **ID** der angezeigten **tblKunden**:

```
...WHERE TMP.ID < tblKunden.ID
```

Also filtert die Abfrage zu jedem Kundendatensatz die virtuelle Kundentabelle (**TMP**) so, dass ihre Datensatzzahl der entspricht, die bis zu dieser Position erreicht wurde. Diese Anzahl wird in **Pos** als Nummerierung zweckentfremdet.

Eine absteigende Nummerierung erreichen Sie, indem Sie statt des Kleiner-Zeichens das Größer-Zeichen

```
SELECT 1+(SELECT COUNT(*) FROM tblKunden AS TMP WHERE TMP.ID<tblKunden.ID)
  AS Pos,
tblKunden.* FROM tblKunden;
```
Listing 2: SQL-Code der Abfrage zum Durchnummerieren der Kunden

in die Bedingung einsetzen. Jede Tabelle mit einem Autowert lässt sich auf diese Weise durchnummerieren. Später zeigen wir noch ein Beispiel, wie sich die Artikel jeder Bestellung durchnummerieren lassen. Die Unterabfrage dazu fällt etwas umfangreicher aus, weil sie noch weitere Bedingungen enthalten muss.

Unterabfragen für Bedingungen

Eine Unterabfrage gibt einen oder mehrere Datensätze zurück. Soll ein Feldwert ermittelt werden, so darf die Zahl dieser Datensätze naturgemäß nicht größer sein, als 1. Schließlich kann eine Feldzelle ja nicht mehr anzeigen, als einen Wert. Die Aggregatfunktion **COUNT** etwa garantiert dies, ebenso, wie **FIRST** (**Erster Wert**) oder **LAST** (**Letzter Wert**) oder **AVG** (**Mittelwert**). Verwenden Sie hier Unterabfragen, die mehrere Datensätze zurückliefern, so wird Access bereits beim Abspeichern der Abfrage meckern und Sie darauf aufmerksam machen, dass eine solche Unterabfrage nur einen Wert zurückgeben darf.

Anders bei Unterabfragen, die für Bedingungen benutzt werden. Als Beispiel möchten wir alle Kunden ausgeben, die irgendwann Bestellungen tätigten. Das Kriterium dafür ist, dass ihre Kunden-**ID** in der Tabelle **tblBestellungen** irgendwo auftaucht. Die Abfrage in Bild 5 baut auf der früheren auf, in der die Gesamtzahl aller Bestellungen eines Kunden bestimmt wurde. Auch hier findet für diese Feldberechnung eine Unterabfrage Anwendung. Der Nachteil dieser Lösung besteht darin, dass **alle** Kunden ausgegeben werden, also

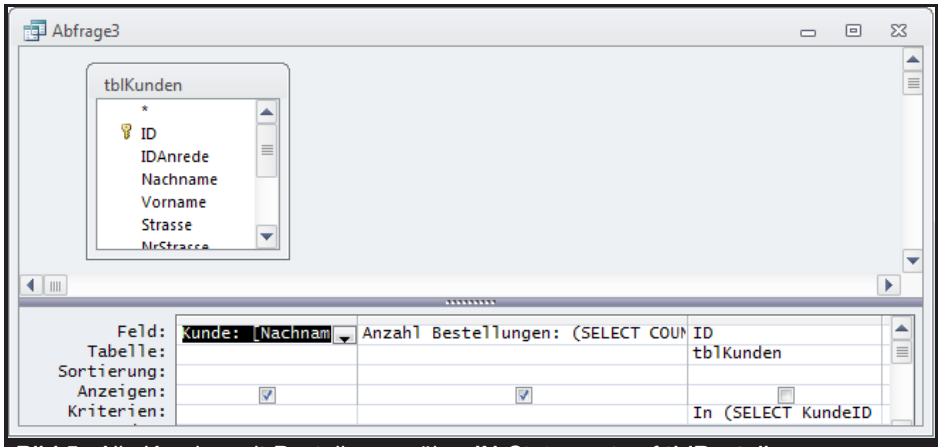


Bild 5: Alle Kunden mit Bestellungen über **IN**-Statement auf **tblBestellungen**

```
SELECT [Nachname] & "," & [Vorname] AS Kunde,
(SELECT COUNT(*) FROM tblBestellungen As TMP WHERE (TMP.KundeID=tblKunden.ID)) AS n
FROM tblKunden
WHERE [tblKunden].[ID] In (SELECT KundeID FROM tblBestellungen);
```

Listing 3: Hier der SQL-Code von **Abfrage3**

auch jene, bei denen **COUNT** den Wert **0** zurückgibt. Natürlich könnte man als Kriterium für dieses Feld den Ausdruck **>0** bestimmen. Die Abfrage müsste dann für jeden Kunden seine Bestellzahl ermitteln und aus dem Ergebnis all jene streichen, deren Wert **0** betrüge. Eine Vorfilterung macht diese Abfrage aber schneller.

Dazu wird im dritten Feld nachgeschaut, ob die Kunden-**ID** überhaupt in der Tabelle **tblBestellungen** vorkommt. Das **IN-Statement** ist dafür der geeignete SQL-Ausdruck. Sie finden ihn vollständig in der vierten Zeile von Listing 3. Nach dem Tag **IN** hat eine Auswahlliste in Klammern zu folgen. Hat die **ID** einen Wert, der mit einem der Elemente der **IN**-Klammer übereinstimmt, so ist der Rückgabewert der Funktion **True**. Die Auswahlliste kommt hier in Gestalt einer Unterabfrage daher:

```
SELECT KundeID FROM tblBestellungen
```

Eine Alternative zum **IN-Statement** ist die **ANY**-Klausel, die noch etwas schneller arbeitet. Dazu nochmals ein Beispiel, welches alle Kunden mit Bestellungen ermittelt. Die Abfrage in Bild 6 zeigt namentlich lediglich alle Kunden mit Bestellungen, wobei als Kriterium für die Kunden-**ID** hier das **ANY-Statement** herhält, dem die gleiche Unterabfrage in Klammern zugewiesen wird, wie im vorigen Beispiel.

Der Unterschied zum **IN-Statement** besteht darin, dass hier nicht eine Mengenfunktion vorliegt, sondern ein Wertvergleich, erkennbar am Gleichheitszeichen (siehe Listing 4). Intern sollte die **Access Database Engine** dabei wohl denselben Algorithmus verwenden, aus irgendeinem Grund ist das **ANY-Statement** jedoch etwas schneller. Möglicherweise cachet sie hier das Ergebnis der Unterabfrage. Microsoft schweigt sich über die Interna aus.

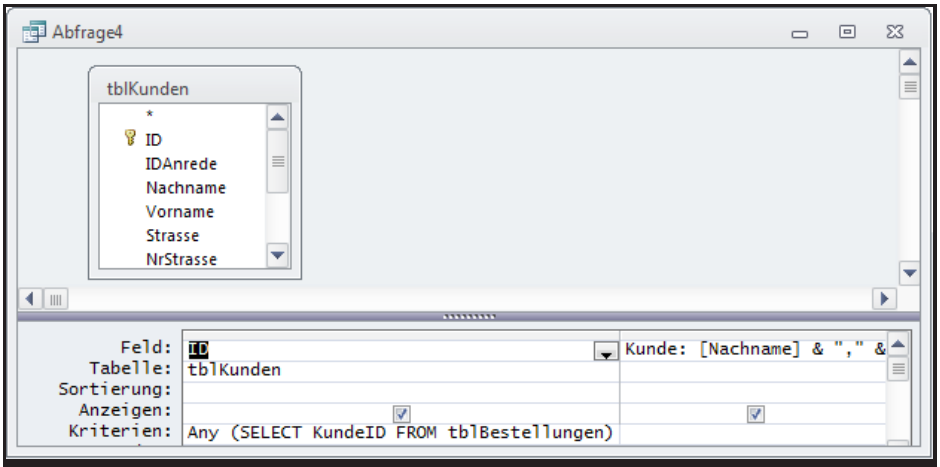


Bild 6: Kunden mit Bestellungen, hier über das **ANY**-Statement ermittelt

```
SELECT tblKunden.ID, [Nachname] & "," & [Vorname] AS Kunde
FROM tblKunden
WHERE tblKunden.ID=Any (SELECT KundeID FROM tblBestellungen);
```

Listing 4: Das ist der doch recht übersichtliche SQL-Code von **Abfrage4**

```
SELECT tblKunden.ID, [Nachname] & "," & [Vorname] AS Kunde FROM tblKunden
WHERE tblKunden.ID In (SELECT TOP 5 KundeID FROM tblBestellungen ORDER BY KundeID DESC)
ORDER BY [Nachname] & "," & [Vorname];
```

Listing 5: **Abfrage5** zeigt die ersten fünf Kunden mit Bestellungen über **IN**-Klausel

Natürlich kann man Unterabfragen auch noch beliebig weiter ausbauen. Die Abfrage in Listing 5 benutzt ebenfalls das **IN-Statement**, filtert die Auswahlmenge jedoch über das Limit **TOP 5**. Also gibt die Klammer, die Unterabfrage, immer nur fünf Datensätze zurück, wobei diese zusätzlich nach der **ID** des Kunden sortiert sind. Die Abfrage insgesamt zeigt damit ebenfalls nur fünf Kunden an, die außerdem noch nach Kundennamen sortiert sind.

Hauptabfragen kürzen

Aus Webshops kennen Sie vielleicht die Funktion **Kunden bestellten auch diese Artikel**. Zur aktuell angezeigten Ware werden in einer Liste jene Artikel aufgeführt, die häufig ebenfalls in den Warenkorb wanderten. Auch unsere Beispieldatenbank kann mit diesem Feature aufwarten. Die Abfrage in Bild 7 zeigt, wie's geht.

Gefragt ist, welche anderen Artikel außer **Gustaf's Knäckebröd** parallel bestellt wurden. Also muss als Kriterium für den Artikelnamen der ersten Tabelle **tblArtikel** der String **Gustaf's Knäckebröd** stehen. Die 1:1-Verknüpfung mit **tblBestelldetails** gewährleistet, dass nur die Bestellpositionen im Ergebnis auftauchen, in denen der Artikel vorkam. Zu jeder Bestellposition gibt nun es eine zugehörige Bestellung, die durch das

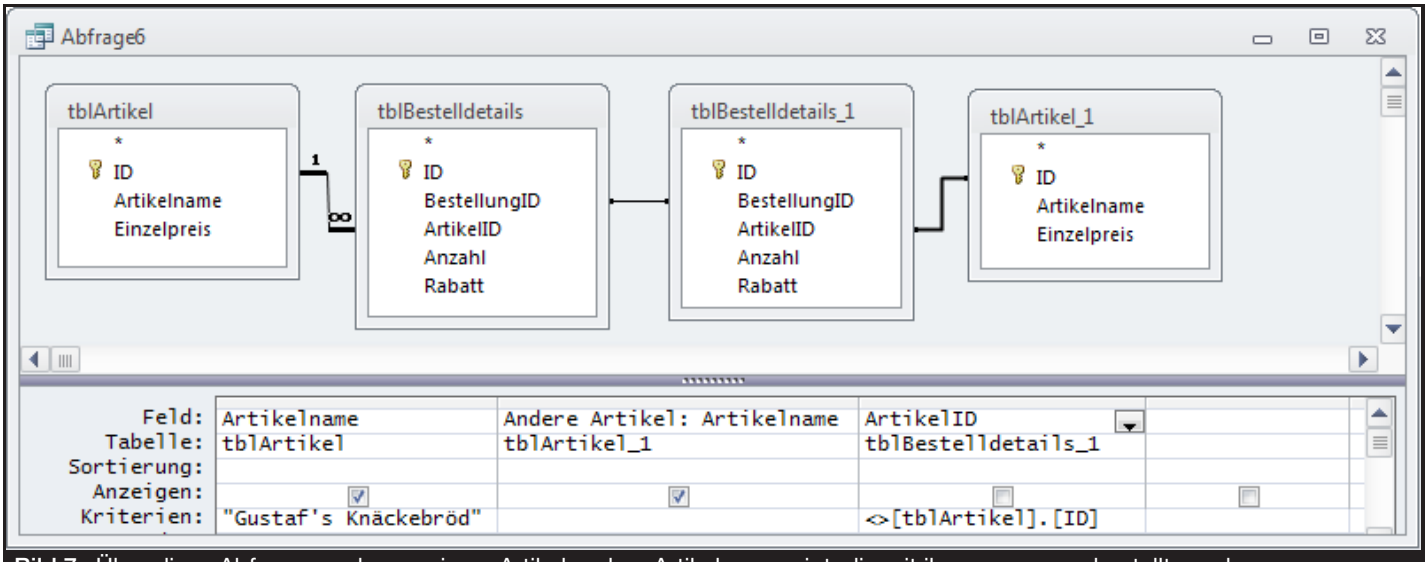


Bild 7: Über diese Abfrage werden zu einem Artikel andere Artikel angezeigt, die mit ihm zusammen bestellt wurden

Feld **BestellungID** identifiziert ist. Damit weiß man, welche Bestellungen uns im Verein mit dem **Knäckebröd** weiter interessieren. Nun wird an die Tabelle eine weitere Instanz der **tblBestelldetails** (**tblBestelldetails_1**) geknüpft, wobei die Feldinhalte von **BestellungID** übereinstimmen müssen. Bis hierher haben wir im Ergebnis also alle Bestellungen und Bestellpositionen mit der gewünschten **ID**, wobei diese nun natürlich weitere Artikel aufweisen können. Deren Name wird über eine weitere Instanz der **tblArtikel** bestimmt und als Feld **Andere Artikel** zurückgegeben. Das sieht in der Datenblattansicht aus, wie in Bild 8.

Bild 8: Ergebnis der Abfrage Andere Artikel (**Abfrage6**)

Soweit der traditionelle Entwurf der Abfrage. Die vier Quelltabellen lassen sich in der Hauptabfrage etwa auf zwei reduzieren, wenn für die **Anderen Artikel** eine Unterabfrage eingesetzt wird. Die Abfrage modifizieren wir nun so, dass statt des festen Strings **Gustaf's Knackebröd** der Parameter **P** zum Einsatz kommt (siehe Bild 9). Beim Ausführen der Abfrage werden Sie daher zunächst mit einem Dialog konfrontiert, der den Wert des Artikelnamens abfragt. Hier können Sie wieder das **Knäckebröd** hineinschreiben. Die Abfrage geht nun den gleichen Weg und ermittelt alle Bestellungen über tblBestelldetails, die den Artikel enthielten. Die über BestellungID identifizierten Bestellungen müssen nun aber als Kriterium Bestandteil der Werte der IN-Klausel sein. Dort stehen als Unterabfrage

```
SELECT DISTINCT tblArtikel.Artikelname FROM
tblArtikel INNER JOIN tblBestelldetails ON tblArtikel.ID = tblBestelldetails.ArtikelID
WHERE tblArtikel.Artikelname<>[P] AND
tblBestelldetails.BestellungID In (SELECT tblBestelldetails.BestellungID FROM
tblArtikel INNER JOIN tblBestelldetails ON tblArtikel.ID = tblBestelldetails.ArtikelID
WHERE tblArtikel.Artikelname=[P]);
```
Listing 6: Vollständiger SQL-Code von **Abfrage6b**

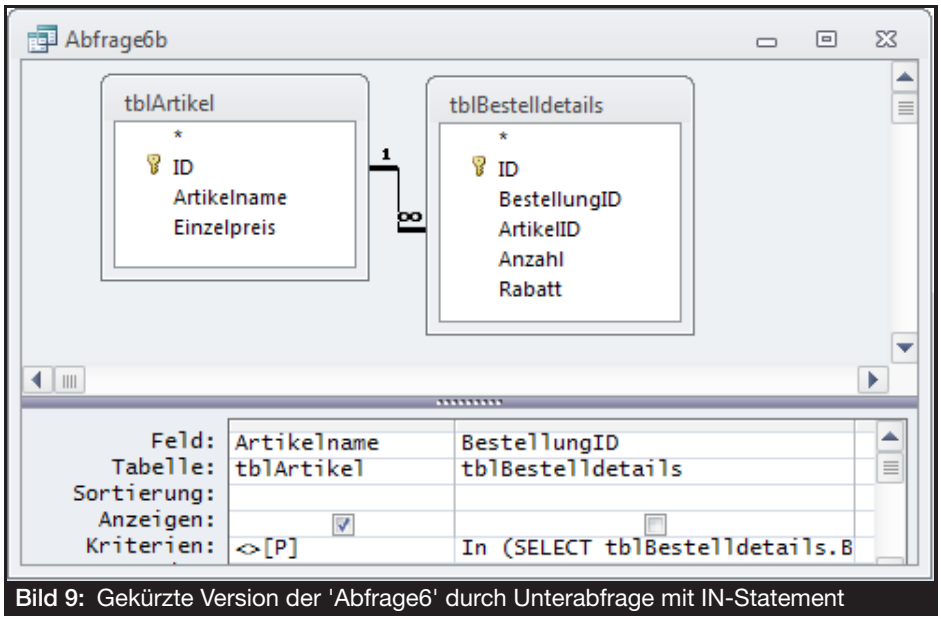


Bild 9: Gekürzte Version der 'Abfrage6' durch Unterabfrage mit IN-Statement

diesmal nicht nur die Datensätze einer Tabelle, sondern die von zwei verknüpften, nämlich abermals der tblBestelldetails und der tblArtikel. Der SQL-Code in Listing 6 macht das deutlich. Die Unterabfrage hat den Ausdruck

```
SELECT tblBestelldetails.BestellungID
FROM tblArtikel INNER JOIN
  tblBestelldetails ON tblArtikel.ID =
  tblBestelldetails.ArtikelID
WHERE tblArtikel.Artikelname=[P]
```

Imgrunde entspricht das dem rechten Teil der Abfrage in Bild 7 und

Textboxen im Detail

Das zweifelsohne prominenteste Steuerelement in Datenbanken ist, vom Label vielleicht abgesehen, das Textfeld. Fast alle Ein- und Ausgaben geschehen über datengebundene Textboxen. Der Umgang mit ihnen scheint trivial zu sein, und in der Regel ist das auch der Fall, aber hin und wieder ist genaueres Hinsehen erforderlich. Schauen wir uns einmal die spezielleren Situationen an.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1605_Textboxen.accdb**

Feldentwürfe für Text in Tabellen

Die möglichen Formen von Textfeldern demonstriert die Tabelle **tblTexte** in Bild 1 in gleich drei Variationen. Üblicherweise nimmt der Feldtyp **Text** String-Daten auf. Die von Access eingestellten Vorgaben für die Eigenschaften des Felds beim Anlegen lassen sich in der Regel unverändert übernehmen.

Sie können sie eventuell um **Gültigkeitsregeln** oder **Standardwerte** erweitern. Die Länge des Felds ist auf 255 Zeichen voreingestellt. Diesen Wert brauchen Sie nicht zu verändern. Größer geht nicht, und kleiner ist überflüssig, da dies in der Datenbank keinerlei Platz einspart. Soll die Länge begrenzt sein, wie etwa bei Postleitzahlen auf 5, so erledigen Sie dies besser über eine zusätzliche Gültigkeitsregel.

TextKurz ist also ein normales Textfeld. Soll die Kapazität des Felds größer sein, als 255 Zeichen, so nehmen Sie ein **Memofeld** (**TextLang**). Es ist in der Lage, bis circa 1 Milliarde Zeichen zu speichern. Das ist gleichzeitig die maximale Größe einer Access-Datenbank in Bytes. Denn grundsätzlich werden alle Textzeichen als **Unicode** in der Tabelle gespeichert. Somit belegt ein Zeichen zwei Bytes. Allerdings verringert sich das durch die **Unicode-Kompression** im Schnitt wieder, jedoch nie um den Faktor zwei.

Seit **Access 2007** kann direkt im Tabellenentwurf für ein Memofeld zusätzlich die Eigenschaft **Textformat** gesetzt werden (siehe **TextRich**). Neben dem normalen Format steht nun auch das **Rich-Text-Format** zur Verfügung. Intern ändert das nichts am Memofeld und dem Speichermodus, Access weiß dann aber, wie es den Text zu interpretieren hat. So ein Feld gestattet dann schon in der Datenblattansicht der Tabelle die Formatierung des Texts über einen aufpoppenden Toolbar (Bild 2). Dieser taucht nicht auf, wenn die Eigenschaft **Rich-Text** deaktiviert ist.

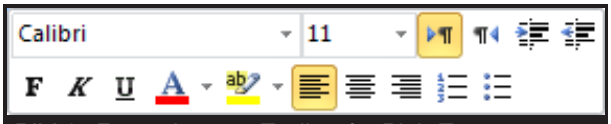


Bild 2: Formatierungs-Toolbar für Rich-Text

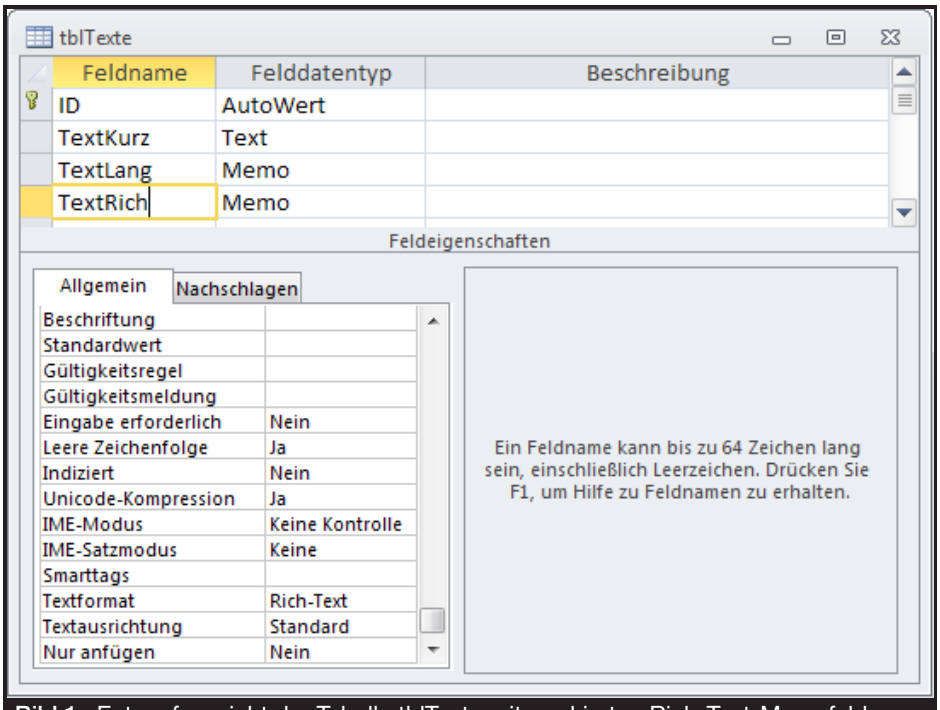


Bild 1: Entwurfsansicht der Tabelle tblTexte mit markierten Rich-Text-Memofeld

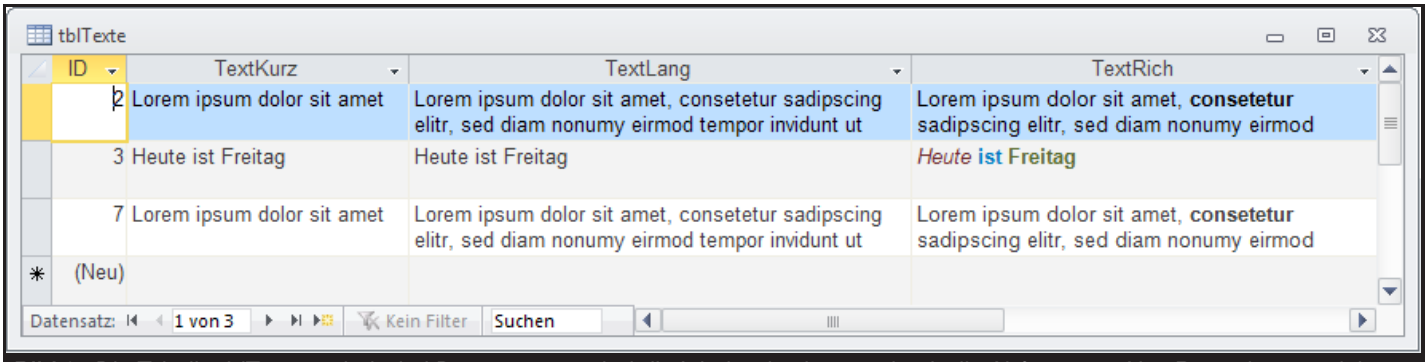


Bild 3: Die Tabelle tblTexte enthält drei Datensätze, wobei die Inhalte des letzten durch die Abfrage qry_NewRecord zustandekam

Access speichert neben den eigentlichen Textdaten dann auch HTML-Tags zur Formatierung, die Sie aber nicht zu Gesicht bekommen. Das funktioniert auch dann nicht, wenn Sie später das **Rich-Text-Format** deaktivieren. Access entfernt bei diesem Vorgang nämlich sämtliche HTML-Tags.

Die Tabelle **tblText** wurde mit zwei Datensätzen gefüllt, wobei ein **Lore-Ipsum**-Generator im Internet zu Anwendung kam (Bild 3). In den Kurztext sind nur wenige Zeichen kopiert, während **TextLang** und **TextRich** mit dem identischen Text von etwa 60.000 Zeichen versehen wurden. Für den dritten Datensatz wurde der erste zunächst kopiert. **TextLang** und **TextRich** sollten dann durch **Copy&Paste** in der Länge verdoppelt werden. Das schlägt indes fehl!

Denn für die Datenblattansicht kommt intern ebenfalls eine **Textbox** zum Einsatz, und diese erlaubt nur Bearbeitungen bis etwa 65.000 Zeichen! Andernfalls erscheint die Meldung **Text ist zu lang, um bearbeitet werden zu können**.

Wie aber bekommt man dann mehr Zeichen in das Tabellenfeld?

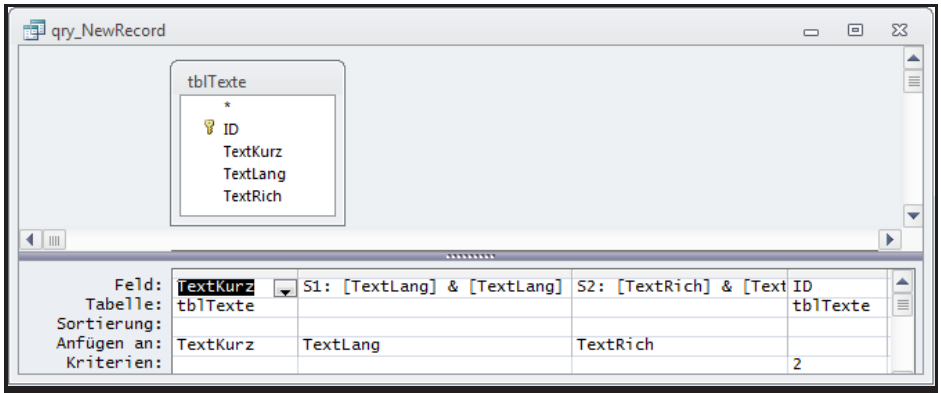


Bild 4: Anfügeabfrage qry_NewRecord zum Duplizieren eines Datensatzes innerhalb der gleichen Tabelle mit Vergrößern zweier Textfelder

Die eine Möglichkeit ist der **Import** aus einer Textdatei.

Die andere ist das Füllen des Textfelds über **VBA** und ein **Recordset**. Das ist sicher die umständlichste Variante.

Eine Lösung ist jedoch ebenfalls das Anlegen oder Modifizieren von Datensätzen über **Anfüge-** oder **Änderungsabfragen**. So geschehen in der Abfrage **qry_NewRecord** aus Bild 4. Die **Anfügeabfrage** nimmt den Ausgangsdatensatz mit der **ID 2** und fügt diesen an die Tabelle **tblTexte** selbst wieder an, dupliziert ihn also. Dabei werden die Inhalte von **TextLang** und **TextRich** aber verdoppelt:

```
[TextLang neu]=[TextLang] & [TextLang]
```

So kam der Datensatz mit der **ID 7** in Bild 2 zustande. Versuchen Sie in der Datenblattansicht nun, den Inhalt der Memofelder dieses Datensatzes zu ändern, sei es durch Einfügen oder Löschen von Zeichen, so funktioniert das abermals nicht. Hier tritt noch nicht einmal eine Meldung auf. Der Datensatz verhält sich so, als sei er gesperrt!

Um es kurz zu machen: Sie werden eine Access-Textbox, ob in der Datenblattansicht oder im Formular, durch keinerlei Tricks dazu bewegen können, Texte länger als 65.000 Zeichen zu bearbeiten. Ist das gewünscht, so kommen Sie um den Einsatz einer alternativen Textbox, etwa dem **ActiveX**-Steuerelement **MSForms-Textbox**, die solche Be-

Designs für Steuerelemente

Access 2010 räumte mit der Notwendigkeit auf, jedes Steuerelement in seinem Layout einzeln formatieren zu müssen. Neben zusätzlichen Gestaltungsmöglichkeiten wurden die sogenannten Designs eingeführt, welche als Vorlagen dienen, und einem Satz von Steuerelementen im Formularentwurf auf einen Schlag zugewiesen werden können. Weil das Ganze aber etwas undurchsichtig realisiert ist, haben wir das Wesentliche zum Thema hier zusammengetragen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1605_Control-Designs.accdb**

Designs

Als **Design** bezeichnet Microsoft unter Access einen Satz von Voreinstellungen für Farben, Schriftarten und anderen Formatierungen für Steuerelemente. Ein abgespeichertes **Design** kann markierten Elementen auf einen Schlag verabreicht werden, wobei sie dann die entsprechende Gestalt annehmen.

Das bestimmte **Design** kann aber nicht nur einzelnen Steuerelementen oder Formularen und Berichten zugewiesen werden, sondern der Datenbank global. Damit erhalten neu angelegte Elemente automatisch das gewünschte einheitliche Aussehen. Access enthält bereits eine große Auswahl von Designsätzen, die Sie um eigene Kreationen erweitern können, um Ihre **Corporate Identity** zu manifestieren.

Die Grundlagen zu Designs wurden bereits in der Ausgabe **06/2015** von **Access Basics** behandelt. Dort ging es um die Neuanlage oder Modifikation von Designs. Hier jedoch schauen wir uns näher an, welche Bestandteile ein **Design** enthält und wie sich die Formatierungen auch über VBA steuern lassen.

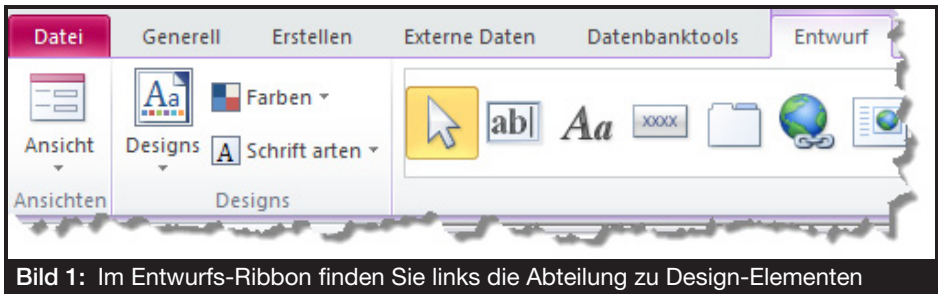


Bild 1: Im Entwurfs-Ribbon finden Sie links die Abteilung zu Design-Elementen

Designs in der Benutzeroberfläche

Etwas verwirrend mag erscheinen, dass die Funktionen zu Designs im **Ribbon** auf mehrere Orte verteilt sind. Haben Sie etwa ein Formular im Entwurf oder in der Layoutansicht geöffnet, so gibt es zwei **Tabs**, die aktiv werden. Der eine verzeichnet unter **Entwurf** die Gruppe **Designs**, der andere nennt sich **Format** und beherbergt die Anweisungen zum Gestalten der einzelnen Steuerelemente.

Bild 1 zeigt die Gruppe Designs im **Entwurf**-Reiter. Dort lassen sich **Designs** auswählen (siehe Bild 2), bearbeiten und neu anlegen. Die Ribbon-Galerien unter **Farben** und **Schriftarten** ermöglichen das Bestimmen von Farbpaletten und Schriftvorgaben. Während diese beiden nicht interaktiv mit dem Formular zusammenarbeiten, reicht in der **Designs**-

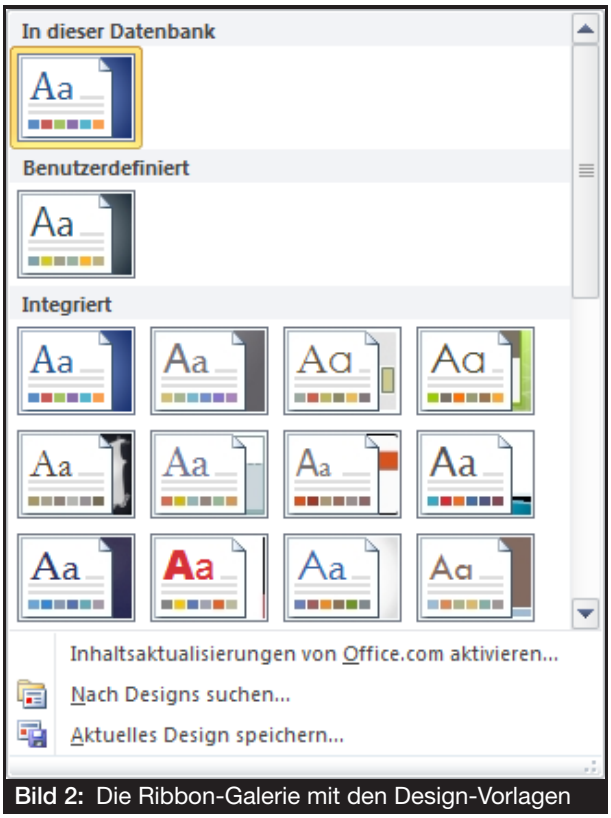


Bild 2: Die Ribbon-Galerie mit den Design-Vorlagen

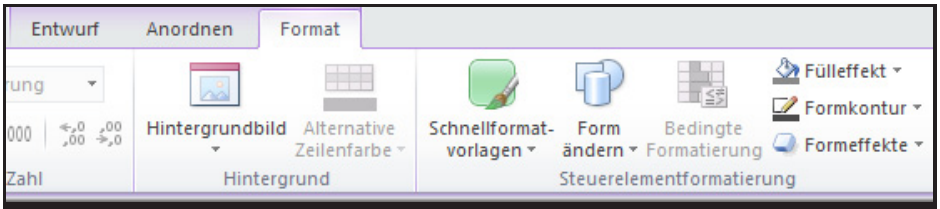


Bild 3: Die Designerabteilung Steuerelementformatierung im Format-Ribbon

Galerie bereits das Überfahren mit der Maus über die einzelnen **Designs** aus, um das neue Aussehen des Formulars im Entwurf zu begutachten. Weitere wissenswerte Informationen darüber erhalten Sie, wie erwähnt, in der Ausgabe **06/2015**.

Markieren Sie hingegen im Formular ein Steuerelement, so aktivieren sich im **Format**-Reiter verschiedene Elemente der Abteilung **Steuerelementformatierung** (siehe Bild 3). Je nach Art des Steuerelements sind die Möglichkeiten mehr oder wenig begrenzt. Bei einer **Schaltfläche** etwa sind alle Elemente aktiv, während bei einer **Textbox** nur **Fülleffekt** und **Formkontur** Änderungen zulassen. Die anderen Elemente im Ribbon sind dann deaktiviert.

Klappen Sie die **Schnellformatvorlagen** aus, so finden Sie in dieser eine Vielzahl von Voreinstellungen für die Gestaltung des Steuerelements. Das betrifft Form, Ränder, Schatten, et cetera. Leider ist diese Dropdown-Galerie nur und ausschließlich bei **Buttons** aktiv. Also bei normalen **Schaltflächen,** wie auch **Umschaltflächen**. Für alle anderen Steuerelemente gibt es keine Schnellformatvorlagen!

Das gleiche gilt für die Galerie **Form ändern**, die die Erscheinungen in Bild 4 anbietet. Die **Schnellformatvorlagen** basieren übrigens jeweils

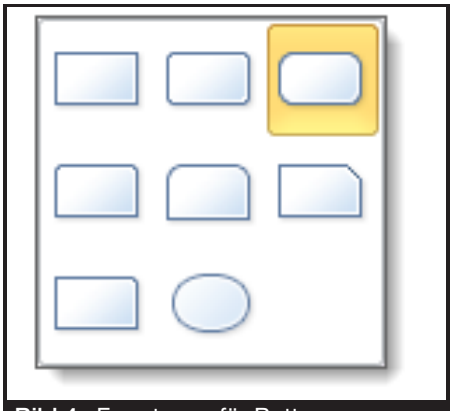


Bild 4: Formtypen für Buttons

auf den Farben des aktuell eingestellten **Designs**.

Für alle Steuerelemente jedoch können zumindest Füll- und Randfarben-

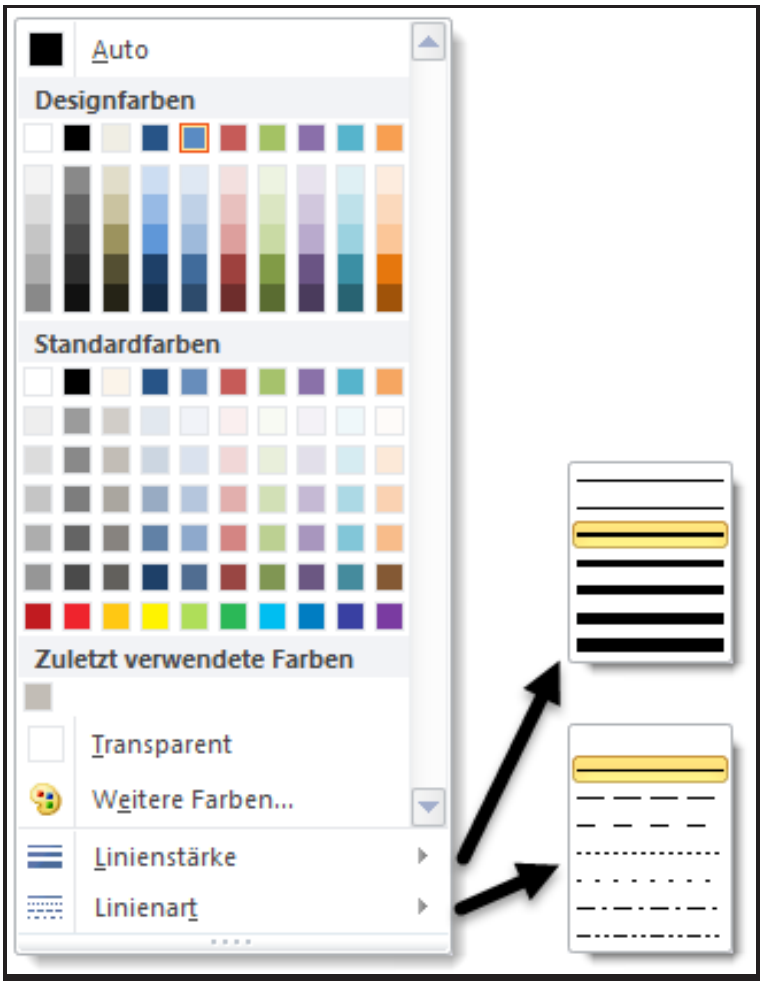


Bild 5: Randgestaltung über Farben und Linienarten

farben bestimmt werden. Die **Formkontur** im Menü betrifft die Ränder eines Steuerelements. Ausgeklappt zeigt sich die Abteilung in Bild 5. Im oberen Bereich sind die zehn **Designfarben** abgebildet, wobei diese in unterschiedlichen Helligkeitsvarianten daherkommen. Darunter findet sich die Palette der **Standardfarben**. Nach mehrmaliger Auswahl von Farben aus einem der Bereiche füllt sich die Zeile mit den **Zuletzt verwendeten Farben**. **Transparent** eliminiert den Rand und macht ihn durchsichtig, entfernt ihn quasi.

Sie sind natürlich nicht ausschließlich auf die Farben der dargestellten Paletten angewie-

sen, sondern können über **Weitere Farben** und den dann sich öffneten Farbmischerdialog beliebige Farben auswählen. Auch diese verewigen sich über die Dauer der Sitzung unter **Zuletzt verwendete Farben**, aus denen sie sich anschließend schnell wieder anwählen lassen.

Die Hintergrundfarbe eines Steuerelements nennt sich im Ribbon-Menü **Fülleffekt**. Die Galerie ähnelt mit ihren Farbpaletten jener für die Formkontur. Eine Farbe ist aber ja noch kein Effekt: Bei Schaltflächen – und nur bei diesen! – ist zusätzlich der Eintrag Farbverlauf aktiv (siehe

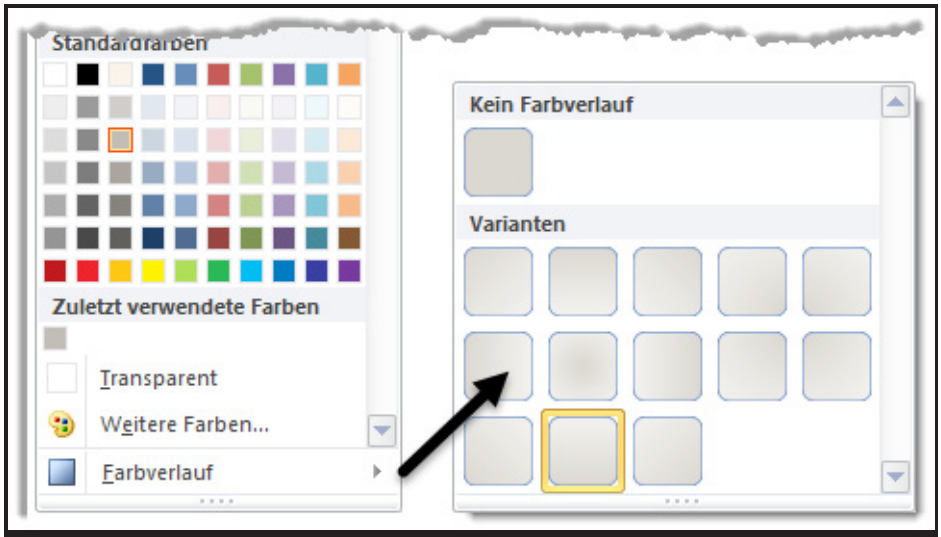


Bild 6: Bei Schaltflächen stehen neben den normalen Hintergrundfarben zur Gestaltung auch noch verschiedene einfarbige Verläufe zur Auswahl

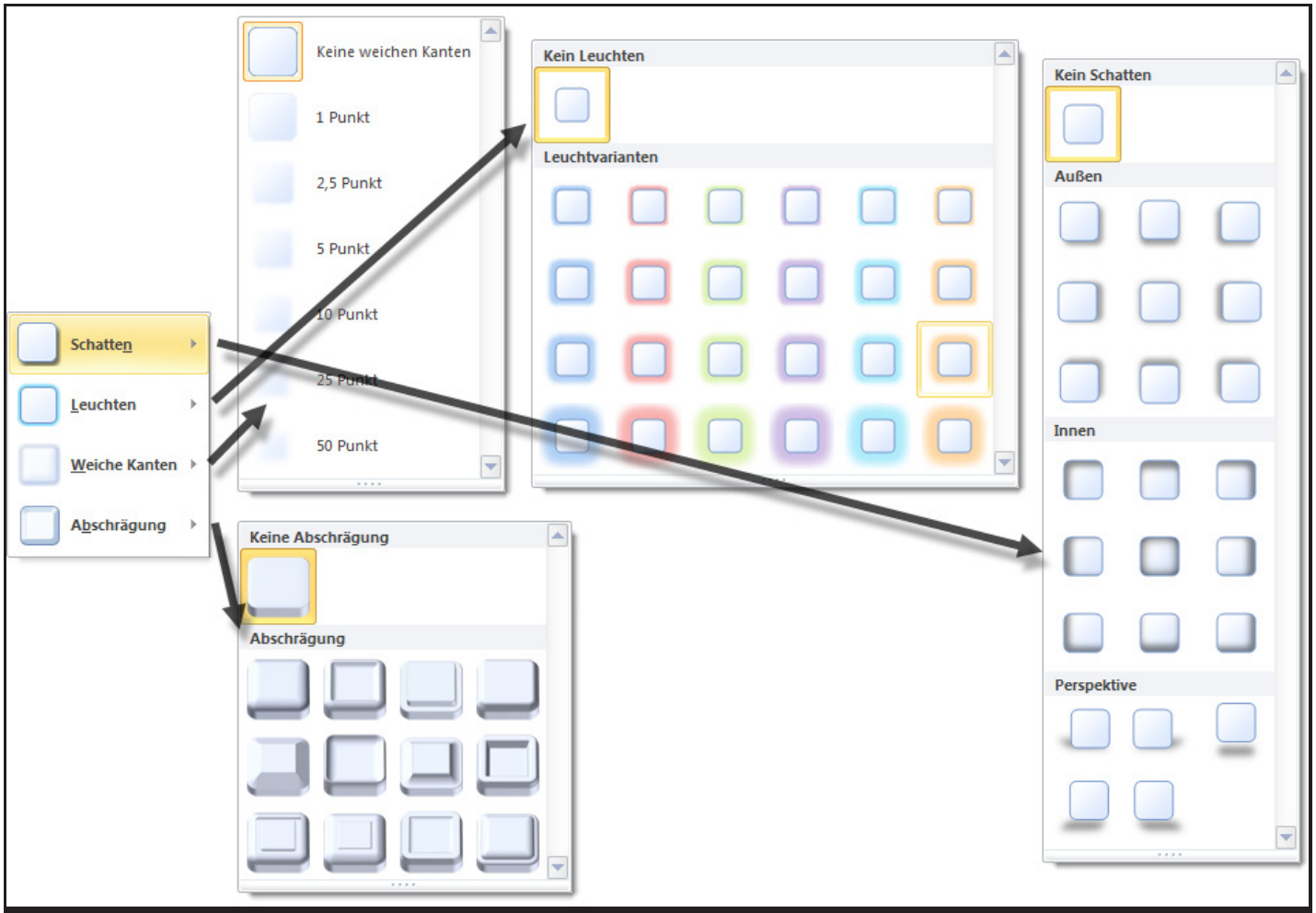


Bild 7: Alle Formeffekte in den entsprechenden Ribbon-Galerien für Schaltflächen auf einen Blick

Bild 6). Der Verlauf basiert dabei auf der zuvor ausgewählten Farbe. Der Kontrast des Verlaufs ist grundsätzlich festgelegt, kann also nicht von Ihnen bestimmt werden.

Soviel zum Layout für alle Steuerelemente. Richtig interessant wird es bei **Buttons**, wo sich im Ribbon neben dem **Formtyp** noch sogenannte **Formeffekte** auswählen lassen. Das sind vier Kategorien im Dropdown-Menü links in Bild 7. Die Pfeile verdeutlichen, welche Untergalerien sich zu diesen öffnen. Alle vier Effekte können beliebig miteinander kombiniert werden. Allerdings gibt es hier Konflikte. Ist ein Button etwa mit einem leichten **Leuchten** versehen, so verschwindet möglicherweise ein gleichzeitig aktivierter **Schatten**, weil sich beides überlagert. Experimentieren Sie hier einfach ein wenig herum, um die Priorität eines Effekts zu ermitteln. Auch die **Weichen Kanten** vertragen sich teilweise nicht mit dem **Schatten**. Grundsätzlich aber gilt dies: **Leuchten** und **Schatten** vergrößern die Fläche, die ein Button regulär einnimmt, während **Weiche Kanten** und **Abschrägung** innerhalb dieser wirken.

Bild 8 zeigt das Formular **frmDesigns** der Beispieldatenbank. Hier sind fast alle möglichen Steuerelemente

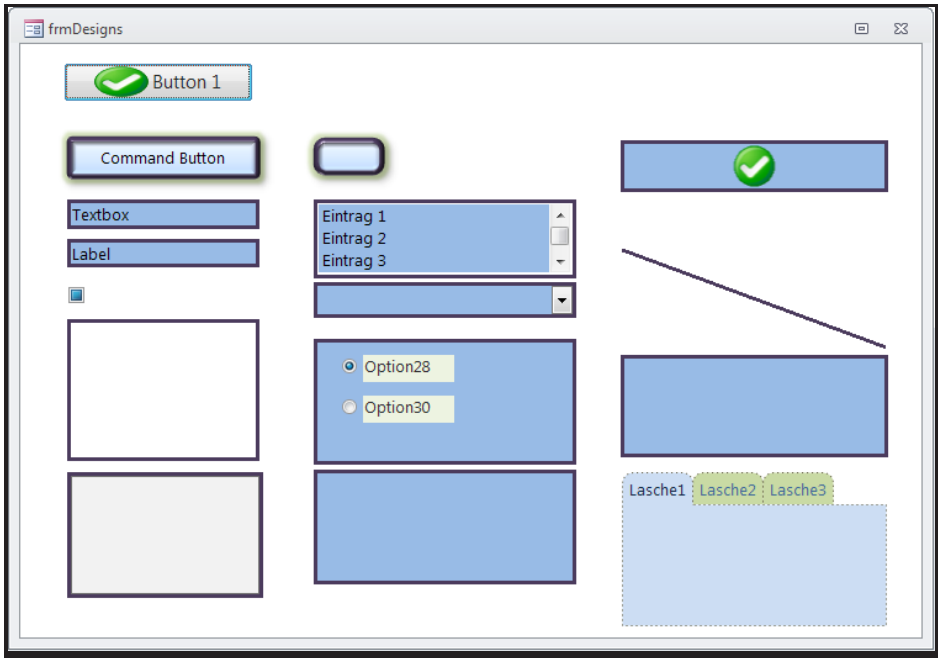


Bild 8: Im Formular frmDesigns sind alle Steuerelemente inklusive ActiveX vereint

mit einem Design versehen. Rechts oben ist übrigens ein **MSForms-Image** eingefügt, also ein **ActiveX-Steuerelement**. Beim Markieren dieser Elemente im Formularentwurf können Sie im Ribbon kontrollieren, welche Formatierungsmöglichkeiten für sie zur Verfügung stehen. Dabei fällt Ihnen eventuell auf, dass die Spezialeffekte für Schaltflächen auch beim **Registersteuerelement** verfügbar sind. Wieso das? Ganz einfach: Ein **Registersteuerelement** stellt Imgrunde die Kombination aus zwei Elementen dar. Die Laschen oben sind tatsächlich Buttons. Deutlich wird dies, wenn Sie das Element im Eigenschaftenblatt unter **Format | Formatvorlage** in den Modus **Schaltflächen** versetzen. Ein Verlauf wirkt sich dann tatsächlich auch nur auf diesen Bereich aus und dehnt sich

Bild 9: Tabellenabhängigkeiten im Datenmodell zum Bestimmen der Formatierungseigenschaften aller Steuerelemente in der Beispieldatenbank

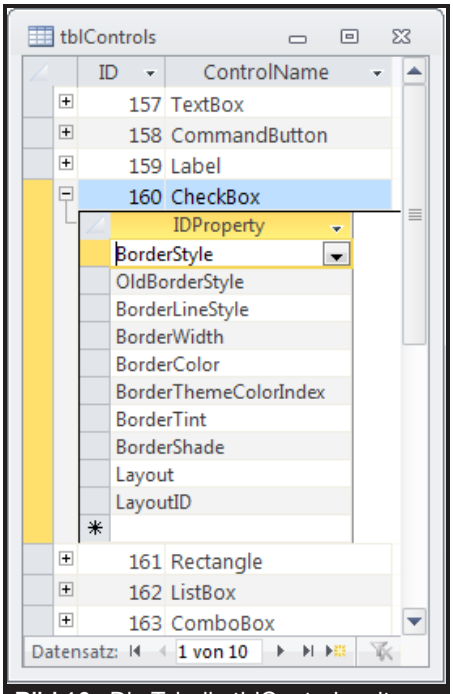


Bild 10: Die Tabelle tblControls mit ausgeklapptem Unterdatenblatt

nicht etwa auf die gesamte Fläche des Steuerelements aus.

Formatierungseigenschaften ermitteln

Damit Sie nicht auf Versuch und Irrtum angewiesen sind, ist in der Beispieldatenbank eine Routine enthalten, die die Formatierungseigenschaften aller Steuerelemente über VBA ermittelt.

Nach dem Öffnen des Formulars **frmDesigns** wird im Ereignis **Beim Laden** die Unterprozedur **EnumCtlProperties** aufgerufen. Wir geben diese nicht komplett wieder, sondern konzentrieren uns auf das verwendete Datenmodell, bestehend aus den Tabellen **tblControls**, **tblCtlProperties** und **tblProperties** (Bild 9, vorige Seite).

Die Tabelle **tblProperties** ist bereits gefüllt und enthält alle Formatierungseigenschaften mit ihrem VBA-Namen. **Hintergrundfarbe** heißt hier **BackColor**, **Leuchten Glow**, **Formtyp Shape**, et cetera. Die Routine durchläuft alle Steuerelemente des Formulars anhand seiner Controls-Auflistung und schreibt die Steuerelementbezeichnungen in die Tabelle **tblControls**:

```
For Each ctl In Me.Controls
  rs!ControlName.Value = TypeName(ctl)
Next ctl
```

Zu jedem liest sie dann den Wert aller VBA-Formatierungseigenschaften aus. Nicht unterstützte Eigenschaften zeitigen dabei einen Fehler, der über **On Error Resume Next** aufgefangen wird. Ereignet sich **kein** Fehler, so schreibt die Prozedur einen neuen Datensatz in die Tabelle **tblCtlProperties**, wobei sowohl **Control**, wie **Property**, durch die **IDs** der Nachschlagetabellen **tblControls** und **tblProperties** referenziert sind:

```
Dim vProp As Variant, lCtl As Long
lCtl = DLookup("ID", "tblControls", _
    "ControlName='" & TypeName(ctl) _
    & "'")
On Error Resume Next
vProp = ctl.Properties(sProp)
If Err.Number = 0 Then
    rs2.AddNew
    rs2!IDControl = lCtl
    rs2!IDProperty = nProp
    rs2.Update
End If
```

Die Funktion **TypeName** ermittelt dabei den VBA-Namen eines konkreten Steuerelements. Eine Schaltfläche **cmdOK** hat etwa den **TypeName CommandButton**. Nach dem Durchlaufen dieser Routine können Sie die Tabelle **tblControls** öffnen und zu jedem Steuerelement das Unterdatenblatt ausklappen. Es zeigt dann alle für dieses Element gültigen Eigenschaften an (siehe Bild 10, vorige Seite).

Wesentlich übersichtlicher wäre es, wenn Sie alle unterstützten Eigenschaften aller Steuerelemente in einer Übersicht vor sich hätten. Kein Problem! Erstellen Sie eine **Kreuztabellenabfrage**, wie in Bild 11 dargestellt. Die Steuerelementeigenschaften dienen als **Zeilenüberschrift**, die Control-Namen als **Spaltenüberschrift**. Gibt es keine Übereinstimmung zwischen Eigenschaft und Control in der Tabelle **tblCtlProperties**, so ist **IDProperty** Null, was im Ausdruck für den **Wert** der Kreuztabellenzelle ausgewertet wird und zu

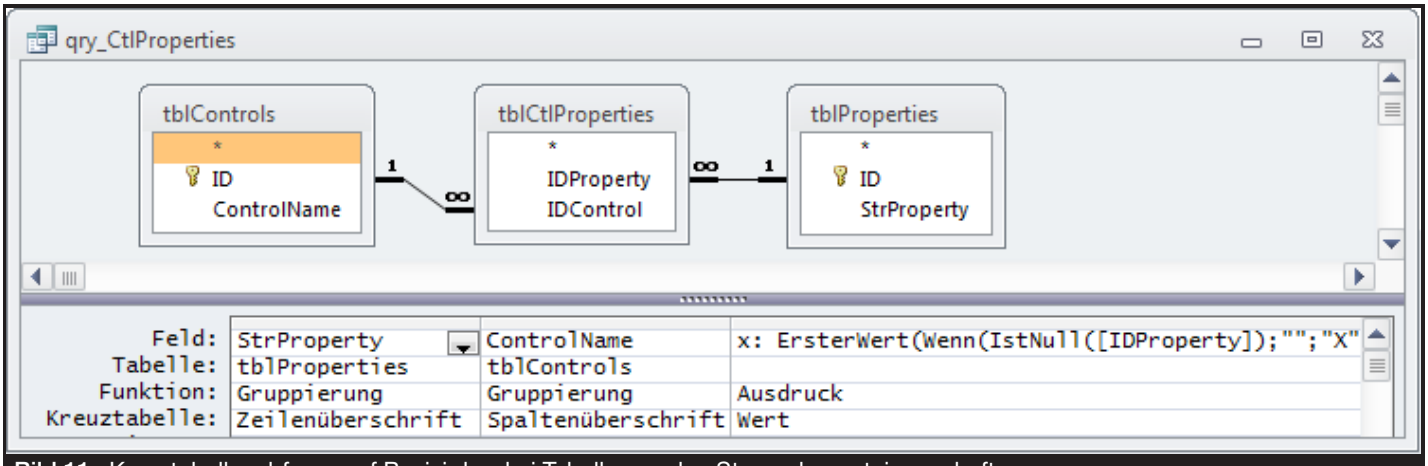


Bild 11: Kreuztabellenabfrage auf Basisi der drei Tabellen zu den Steuerelementeigenschaften

Währungen runden

Im Finanzsektor wird alles sehr genau genommen. Wird über kleinere Rundungsfehler an anderer Stelle gern einmal hinweggesehen, so sind hier Inkonsistenzen in der Regel absolut inakzeptabel. Einzelne Cent-Beträge können sich über Millionen von Datensätzen zu größeren Summen aufaddieren. Wohin Sie genau schauen müssen, um solche Umstände zu vermeiden, untersuchen die folgenden Ausführungen.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1605_Runden.accdb**

Geldbeträge im Tabellenentwurf
Für Zahlen mit Nachkommastellen kommen unter Access im Prinzip vier Datentypen infrage: **Single**, **Double**, **Currency** und **Decimal**, also Fließkommazahl einfacher und doppelter Genauigkeit, sowie Währungs- und Dezimaltyp.

Single scheidet wegen der geringen Genauigkeit aus. Hier werden zur Speicherung nur vier Bytes verwendet, während der **Double**-Typ acht Bytes benötigt. Wie groß der Unterschied letztlich ist, können Sie mit einem kleinen Test im VBA-Direktfenster ermitteln. Lassen Sie sich die Ergebnisse dieser beiden Zeilen ausgeben:

```
? CSng(Now)
? CDbl(Now)
```

Die Funktion **Now** gibt das aktuelle Datum samt Uhrzeit aus, wobei diesem **Date**-Typ tatsächlich ein **Double**-Wert unterliegt.

CSng wandelt den Wert in einen **Single**-Typ um, **CDbl** in einen **Double**-Typ. Sie erhalten nun etwa

```
42516,4 (Single)
42516,4020486111 (Double)
```

Wären dies Geldbeträge, so betrüge die Differenz bereits circa 0,2 Cent!

Die große Zahl an Nachkommastellen lässt **Double** als für Währungen geeignet erscheinen. Doch immerhin muss es einen Grund geben, warum der Typ **Currency** zusätzlich zur Verfügung steht. Einerseits führt sein Einsatz dazu, dass in Tabellen auch gleich automatisch das Währungszeichen erscheint. Zudem ist die Zahl der Nachkommastellen gleich zwei. Andererseits speichert **Currency** aber Zahlen auch nach einem anderen System. Liegt dem **Double**-Typ mathematisch ein Exponentialsystem zugrunde, hantiert **Currency** mit einer festen Zahl von ganzzahligen Stellen, nämlich **19**. Die Umwandlung in einen Fließkommatyp geschieht hier rein optisch durch Setzen des Kommas an die viertletzte Stelle, also über Division der Ganzzahl durch 10.000. Ergo stehen grundsätzlich vier Nachkommastellen zur Verfügung, während bei **Double** und **Single** die Zahl der Nachkommastellen variabel ist und von der Größe des Betrags abhängt.

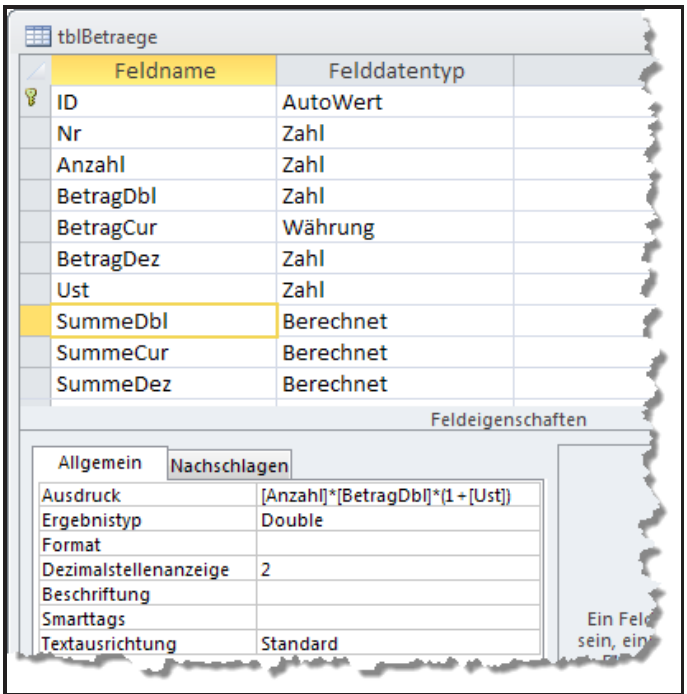


Bild 1: Entwurf der Tabelle tblBetraege mit ihren drei Währungsfeldern, sowie darauf bezogenen berechneten Feldern

Decimal schließlich ähnelt im Ergebnis **Currency**, weil der angezeigt Wert sich durch Division mit einer Zehnerpotenz ergibt. Die Genauigkeit ist sogar noch höher, als bei **Currency**. Doch auch hier gilt das gleiche, wie bei **Double** und **Single**: Die Anzahl der Nachkommastellen ist

variabel. Außerdem kann VBA mit diesem Datentyp schlecht umgehen. Eine Variable vom Typ **Decimal** lässt sich erst gar nicht deklarieren! Der Typ scheidet damit für Währungsrechnungen weitgehend aus.

Für eine empirische Studie wurde die Tabelle **tblBetraege** in der Beispieldatenbank angelegt, deren Entwurf sich aus Bild 1 ergibt. Sie simuliert eine Bestelltabelle, wo es zu jeder Position die Anzahl gibt, den eigentlichen Betrag und die Umsatzsteuer (**Ust**, Prozentzahl, **Double**). Der Betrag ist für unseren Versuch über drei Varianten abgebildet. **BetragDbl** ist ein Feld vom Typ **Double**, **BetragCur** ist ein **Currency**-Typ und **BetragDez** von Typ **Decimal**. Zur Kontrolle wurden noch drei **Berechnete Felder** eingefügt. **SummeDbl** ist dabei folgender Ausdruck hinterlegt:

```
[Anzahl] * [BetragDbl] * (1 + [Ust])
```

Ohne eine Abfrage bemühen zu müssen, wird damit die Positionssumme gleich im Datenblatt angezeigt. In produktiven Datenbanken sollten Sie von diesen **Berechneten Feldern** aber eher absehen, weil die Anzeige direkter Tabelleninhalte dort zu vermeiden ist. **SummeCur** und **SummeDez** berechnen nach der identischen Formel, beziehen sich aber jeweils auf die Felder **BetragCur** und **BetragDez**.

Um die Tabelle mit Daten zu füllen, gibt es die Prozedur **GeneriereBetraege** in der Datenbank. Sie legt 10.000 Datensätze an und schreibt nach Zufallsprinzip Zahlen in die drei Felder **Anzahl**, **Betrag...** und **Ust**. Beim Betrag wird über String-Verarbeitung darauf geachtet, dass nur exakt zwei Nachkommastellen generiert werden:

```
Dim dBetrag As Double
Dim sBetrag As String
dBetrag = 100 * Rnd
sBetrag = CStr(Val(dBetrag * 100) / 100)
rs!BetragDbl.Value = sBetrag
```

Die genaue Gestaltung der Routine interessiert hier weniger, als ihr Ergebnis, welches in Bild 2 dargestellt ist. Im Datenblatt wurde hier zusätzlich die **Aggregatzeile** eingeblendet. An die Aggregatzeile kommen Sie über diese Schritte:

- Öffnen Sie die Tabelle in der Datenblattansicht
- Aktivieren Sie im Ribbon die Schaltfläche **Start | Datensätze | Summen**. Die Aggregatzeile erscheint unten in der Ansicht.
- Gehen Sie in die Aggregatzelle einer Datenspalte und wählen Sie aus dem nun erscheinenden Kombinationsfeld die gewünschte **Aggregatfunktion** aus – hier jeweils **Summe**. Je nach Datentyp des Felds ist allerdings nur ein Teil der Aggregatfunktionen verfügbar.

Die Summen der drei Betragsspalten sind alle identisch. Das gilt auch für die Summen der berechneten Spalten, wobei hier **SummeDbl** drei Nachkommastellen zeigt, obwohl für die Formatierung des Felds dezidiert nur zwei Stellen im Entwurf festgelegt wurden. Aber die Formatierung der Aggregatzeile selbst lässt sich nun mal nicht beeinflussen.

Immerhin wird deutlich, dass sich **Double** bei Berechnungen anders verhält, als **Currency** und **Decimal**.

tblBetraege

ID	Nr	Anzahl	BetragDbl	BetragCur	BetragDez	Ust	SummeDbl	SummeCur	SummeDez
22992	2487	4	57,24	57,24 €	57,24	19,00%	272,4624	272,46 €	272,4624
22993	2487	2	89,83	89,83 €	89,83	19,00%	213,7954	213,80 €	213,7954
22994	2487	1	89,69	89,69 €	89,69	7,00%	95,9683	95,97 €	95,9683
22995	2487	5	95,01	95,01 €	95,01	19,00%	565,3095	565,31 €	565,3095
22996	2488	3	26,14	26,14 €	26,14	7,00%	83,9094	83,91 €	83,9094
22997	2488	2	55,87	55,87 €	55,87	19,00%	132,9706	132,97 €	132,9706
22998	2488	3	4,74	4,74 €	4,74	0,00%	14,22	14,22 €	14,22
22999	2489	2	61,34	61,34 €	61,34	0,00%	122,68	122,68 €	122,68
23000	2489	3	27,67	27,67 €	27,67	7,00%	88,8207	88,82 €	88,8207
*	(Neu)				0				
	Summe		499791,63	499.791,63 €	499791,63		1635471,461	1.635.471,46 €	1635471

Datensatz: 1 von 10000 | Kein Filter | Suchen

Bild 2: Das ist die Datenblattansicht der Tabelle tblBetraege, unten mit der hinzugefügten Aggregatzeile für die Summen

Dennoch kann man sich fragen, warum **Double** für Währungen nicht auch in Betracht kommt, wo doch das Summenergebnis trotz der hohen Zahl von Datensätzen nicht von der Währungsspalte abweicht?

Fügen wir der Tabelle noch zwei weitere berechnete Felder hinzu (siehe Bild 3). **Dbl_x_1000** multipliziert den Betrag mit Tausend, schneidet die Kommastellen per **Int**-Funktion ab, und dividiert wieder durch Tausend:

```
Int(1000 * [BetragDbl])/1000
```

Mit der Currency-Spalte wird analog verfahren. Da die Geldbeträge – auch intern! – nur zwei Nachkommastellen aufweisen, kann die Multiplikation mit 1000 nur zu einer Ganzzahl führen und die anschließende Division wieder nur zu zwei Nachkommastellen. Bild 4 zeigt das Ergebnis, wobei für die neuen Felder ebenfalls die Summen in der Aggregatzeile aufgerufen wurden. Und siehe da, die Summen unterscheiden sich nun plötzlich. Statt 63 Cent findet sich im berechneten Feld der Nachkommaanteil 54,9 Cent. Eine Gruppierungsabfrage auf die Tabelle würde übrigens das

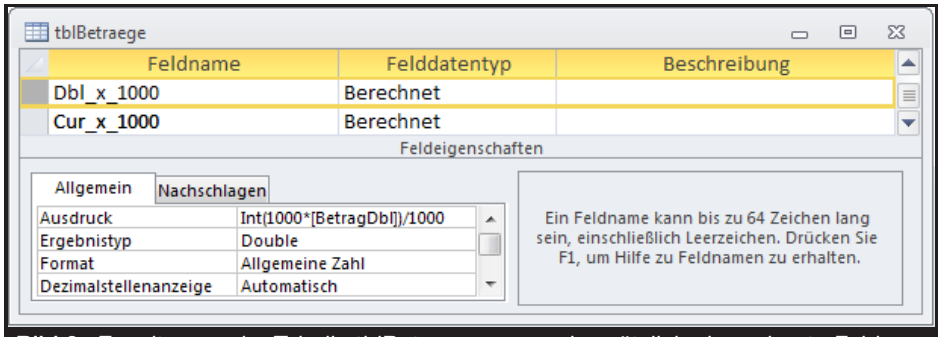


Bild 3: Erweiterung der Tabelle tblBetraege um zwei zusätzliche berechnete Felder

gleiche Ergebnis zeitigen. Die Differenz ist nicht mehr akzeptabel.

Zustande kommt sie durch die Multiplikation des Betrags mit 1000. Hierdurch verringern sich Zahl der Nachkommastellen des Exponentialtyps Double intern. In der Folge kommt es zu Ungenauigkeiten, die bei Summierung unangenehm zu Buche schlagen.

Fazit: Vermeiden Sie den Typ Double bei Geldbeträgen. Leider reicht das Konvertieren eines Double-Felds in Currency nachträglich nicht aus, um die Ungenauigkeiten zu eliminieren, da Currency intern mit vier Nachkommastellen arbeitet. Die Ungenauigkeiten bleiben deshalb unter Umständen in der dritten und vierten Stelle erhalten.

Optisches Runden von Geldbeträgen

Immer wieder gern gemacht und doch fehleranfällig: das Runden von Beträgen auf zwei Nachkommastellen. Auf den ersten Blick ist es plausibel. Da multiplizieren Sie einen Betrag mit einer Prozentzahl, wie der Umsatzsteuer, und kommen so zu mehr, als zwei, Nachkommastellen. Also werden diese mit einer geeigneten Funktion gerundet.

Spätestens beim Summieren dieser errechneten Beträge kommt es zu deutlichen Fehlern. Vor allem dann, wenn Sie von der für diesen Zweck vorgesehenen VBA-Funktion **Round** Gebrauch machen. Denn diese rundet nach mathematischen Regeln, nicht nach kaufmännischen! Ein einfaches Beispiel:

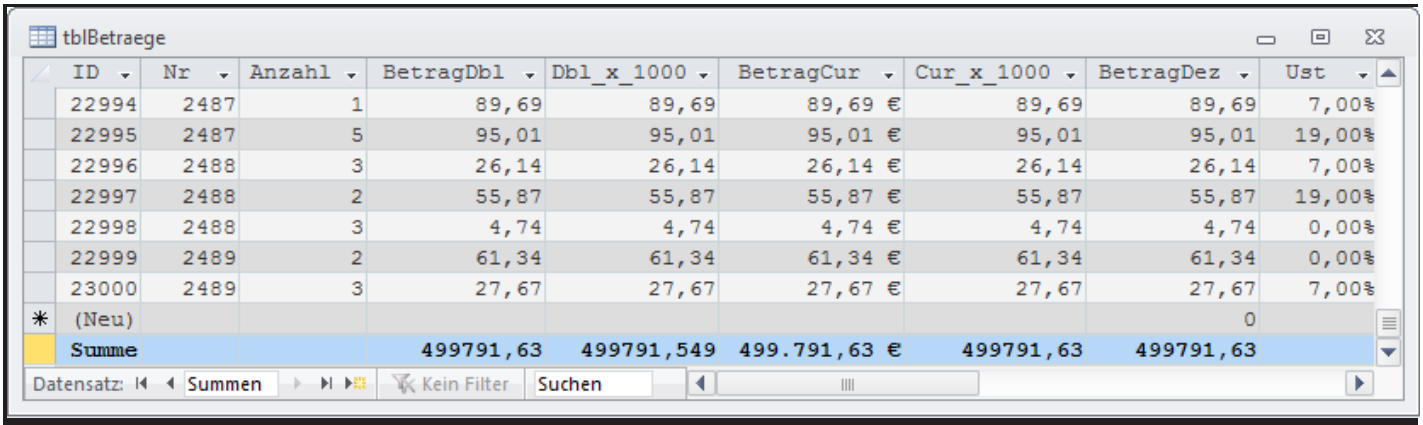


Bild 4: Die Summen geben in Dbl_x_1000 das Problem wieder, welches durch Währungsfelder des Typs Double verursacht wird

Daten sichern über XML

Für die Aufgabe, dass mehrere Tabellen einer Datenbank in eine andere verfrachtet werden sollen, weil Sie dort exakt dieselben Daten benötigen, gibt es unterschiedliche Lösungen. Je nach Situation kommt es auf die Inhalte an, auf die Beziehungen zwischen den Tabellen, oder auf die Eigenschaften und Attribute der Felder. Mit Access 2010 ist eine neue Möglichkeit hinzugekommen, die wir uns genauer anschauen.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1606_TabExport.accdb**.

Exportfunktionen
Bei Neuanlage einer Datenbank möchten Sie das Entwickeln des Datenmodells schnell erledigt haben. Sie greifen daher häufig auf existierende Datenbanken zu, die bereits die gewünschten oder ähnliche Tabellen enthalten, importieren diese entweder in die neue Datenbank, oder exportieren sie aus der bestehenden.

In der Regel werden Sie sich dabei der für diesen Zweck vorgesehenen Funktionen der Oberfläche bedienen. Das Menüband spendiert uns dafür sogar einen eigenen Reiter **Externe Daten** mit seinen Gruppen **Exportieren**, sowie **Importieren und Verknüpfen**.

Auch über VBA können Sie den Import und Export auf unterschiedliche Weise anstoßen.

In aller Kürze hier einige Möglichkeiten, auf die an anderer Stelle bereits eingegangen wurde:

```
RunCommand acCmdExportAccess
```

Ruft lediglich den Dialog zum Exportieren einer Tabelle in eine externe Datenbank auf

```
RunCommand acCmdImportAttachAccess
```

Ruft nur den Dialog zum Importieren von Elementen aus einer externen Datenbank auf. Bei diesen beiden Anweisungen sind also zusätzlich manuelle Angaben erforderlich.

```
DoCmd.RunSavedImportExport "<Name>"
```

In diesem Fall läuft der Import oder Export automatisch ab, jedoch unter einer Bedingung: Sie haben den Vorgang bereits über die Oberflächendialoge einmal durchgeführt und ihn namentlich abgespeichert.

```
DoCmd.TransferDatabase ...
```

Das ist die Hauptfunktion zum Exportieren von Tabellen. Die Funktion erwartet eine Vielzahl von Parametern, die sie steuern. Nachteil: Nur eine einzelne Tabelle kann über sie exportiert werden.

```
Application.SaveAsText acTable, _
    "<Name>", "<Dateiname>"
```

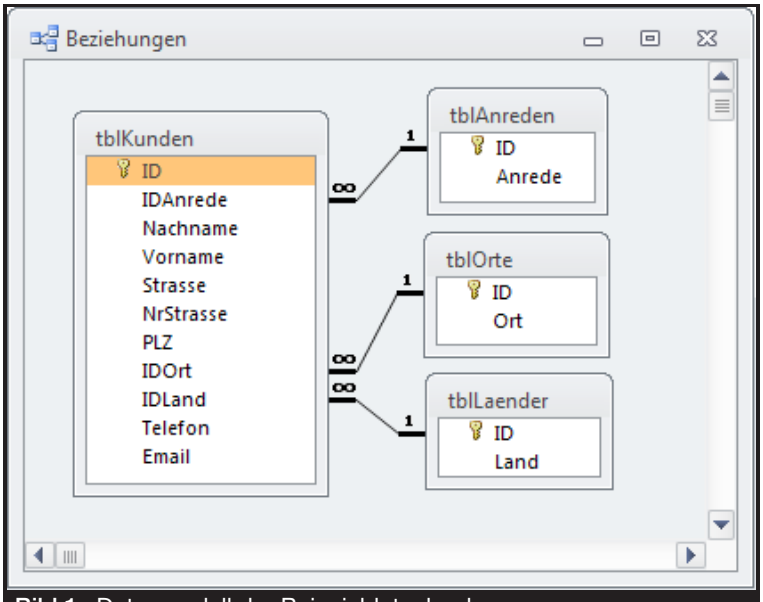


Bild 1: Datenmodell der Beispieldatenbank

Diese versteckte Funktion speichert die Tabelle komplett mit all ihren Eigenschaften in eine Datei. Das Pendant zum Wiederherstellen der Tabelle:

```
Application.LoadFromText
acTable,"<TabellenName>",
"<Dateiname>"
```

Sollen mehrere Tabellen ohne jede manuelle Interaktion exportiert werden, wie etwa jene des Datenmodells aus

Bild 1 der Beispieldatenbank, so kommt man folglich um eine ausprogrammierte Routine nicht herum. Doch seit **Access 2010** gibt es hierfür eine neue Anweisung, die die Sache deutlich einfacher macht.

Die Funktion ExportXML erwartet, ähnlich, wie **TransferDatabase**, die Übergabe einer Menge von Parametern. Der erste gibt den Objekttyp des Elements an, das exportiert werden soll. **IntelliSense** erleichtert Ihnen dabei die Auswahl der Parameterkonstanten. Darunter finden Sie mit **acExportForm** oder **acExportReport** auch die Objekttypen Formular und Bericht. Wer nun denkt, dass sich darüber der komplette Aufbau von Formularen und Berichten transferieren ließe, der irrt! Tatsächlich bezieht Access hier nur die Daten dieser Objekte in den Export ein, also im Prinzip deren Datenherkünfte. Das braucht man wohl höchst selten!

Die einzige für unseren Zweck sinnvolle Konstante ist deshalb **acExportTable**. Listing 1 zeigt, wie die Tabelle **tblKunden** über **ExportXML** in eine **XML-Datei** überführt wird. Zunächst wird der Ausgabepfad in der Variablen **sPath** zwischengespeichert. Im Beispiel ist das das Verzeichnis der aktuellen Datenbank. **ExportXML** wird als zweiten Parameter der Name der Tabelle (**tblKunden**) übergeben und als dritter der Dateiname, welcher sich aus Pfad plus **kunden.xml** bildet.

Darauf folgen mehrere Kommas, die bedeuten, dass hier vier optionale Parameter nicht angegeben werden.

```
Sub ExportTable()
    Dim sPath As String

    sPath = CurrentProject.Path & "\"
    ExportXML acExportTable, "tblKunden", sPath & "kunden.xml", _
        , , , , _
        acEmbedSchema Or acExportAllTableAndFieldProperties
End Sub
```

Listing 1: Die einfachste Prozedur zum Datenexport per Funktion ExportXML

```
Sub ExportTables()
    Dim sPath As String
    Dim AData As AdditionalData

    Set AData = CreateAdditionalData
    AData.Add "tblAnreden"
    AData.Add "tblOrte"
    AData.Add "tblLaender"
    sPath = CurrentProject.Path & "\"
    ExportXML acExportTable, "tblKunden", sPath & "kunden.xml", _
        , , , , _
        acEmbedSchema Or acExportAllTableAndFieldProperties, , AData
End Sub
```

Listing 2: ExportXML erlaubt den Export weiterer Tabellen über AdditionalData

Man braucht sie in der Regel nicht, denn sie beziehen sich alle auf optionale Zusätze der XML-Datei, wie das Schema oder die Präsentation. Wichtig hingegen ist der nächste Parameter, der die Art des Exports über wählbare **Flags** festlegt. **acEmbedSchema** sagt Access, dass das XML-Schema in der XML-Datei selbst abzulegen ist, statt separat. **acExportAllTableAndFieldProperties** gibt an, dass nicht nur Daten und Struktur ausgegeben werden sollen, sondern auch die Eigenschaften und Attribute sowohl der Tabelle, wie aller Felder. Das wären dann Beschriftungen, Gültigkeitsformeln, Standardwerte, also alles, was nicht in die **SQL-DDL**-Repräsentation passt, aber auch Spaltenbreiten und weitere Layoutangaben, wie Schriftart oder Farben. Derlei erlaubt **TransferDatabase** nicht! Die Anweisung wurde wahrscheinlich für die Automation der ab **Access 2007** neu eingeführten **Vorlagen** und **Anwendungs-Parts** eingeführt, denn die Gestalt der XML-Datei entspricht diesen **Template**-Dateien genau.

Nun kann man über die Funktion aber nicht nur eine, sondern beliebig viele Tabellen exportieren. Listing 2 zeigt, wie's geht. Alles, was man dafür braucht, ist eine Objektvariable des Typs **AdditionalData**, die man an die Funktion **ExportXML** als letzten Parameter anhängt. Im Listing nennt sie sich **AData** und wird zunächst über die Funktion **CreateAdditionalData** angelegt. Nun können an das Objekt Tabellen über die Methode **Add** angefügt werden – hier die Tabellen **tblAnreden**, **tblOrte** und **tblLaender**. Auch für diese Tabellen werden all ihre Eigenschaften in der XML-Datei verewigt. Dies lässt sich auf keine andere Weise so einfach

Fixierte Formulare

Wenn Sie Formulare nicht aus speziellen Vorlagen oder aus bereits existierenden über Kopien erstellen, so ist einige Fleißarbeit im Entwurf angesagt, bis die Voreinstellungen von Access für neue Formulare durch Ihre eigenen Designvorstellungen ersetzt sind. Das unterbleibt oft, und so fristet auch die unscheinbare Eigenschaft Verschiebbar häufig ein Schattendasein. Schade, denn sie eröffnet ganz neue Gestaltungsmöglichkeiten für Ihre Datenbank!

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1606_FixedForms.accdb**.

Verschiebbarkeit deaktivieren

Um unteren Bereich des Eigenschaftenblatts eines Formulars finden Sie den Eintrag **Verschiebbar**, den Access zunächst auf den Wert **Ja** einstellt. Ändern Sie das auf **Nein** (siehe Bild 1), so kann das Formular zur Laufzeit weder über Maus und Titelzeile auf dem MDI-Arbeitsbereich von Access verschoben werden, noch über Tastenkombinationen oder Tools. Es bleibt an Ort und Stelle. Das funktioniert auch bei Formularen, bei denen die Eigenschaft **Popup** auf **Ja** steht. **Popup**-Formulare sind eigene Fenster, die nicht der Verwaltung des MDI-Arbeitsbereichs unterliegen, und dennoch kann Access sie anweisen, am Desktop festzukleben. Sie sind etwas schwieriger zu handhaben, weil sie leicht durch andere geöffnete Anwendungen oder Access selbst in den Hintergrund geraten.

Das Formular **frmFix** der Beispieldatenbank öffnet sich gleich nach deren Start (Bild 2). In seinem Code **Beim Laden** blendet es zusätzlich den Navigationsbereich aus und außerdem den Ribbon. Von Access bleibt somit nur dessen Rahmen und Titelzeile übrig, und im Arbeitsbereich befindet sich das Formular in der linken oberen Ecke, aus der es auch nicht herausgeholt werden kann. Das Formular schaltet nach 5 Sekunden den Ribbon und Navigationsbereich wieder ein, womit sich die Darstellung aus Bild 3 ergibt. Gibt es

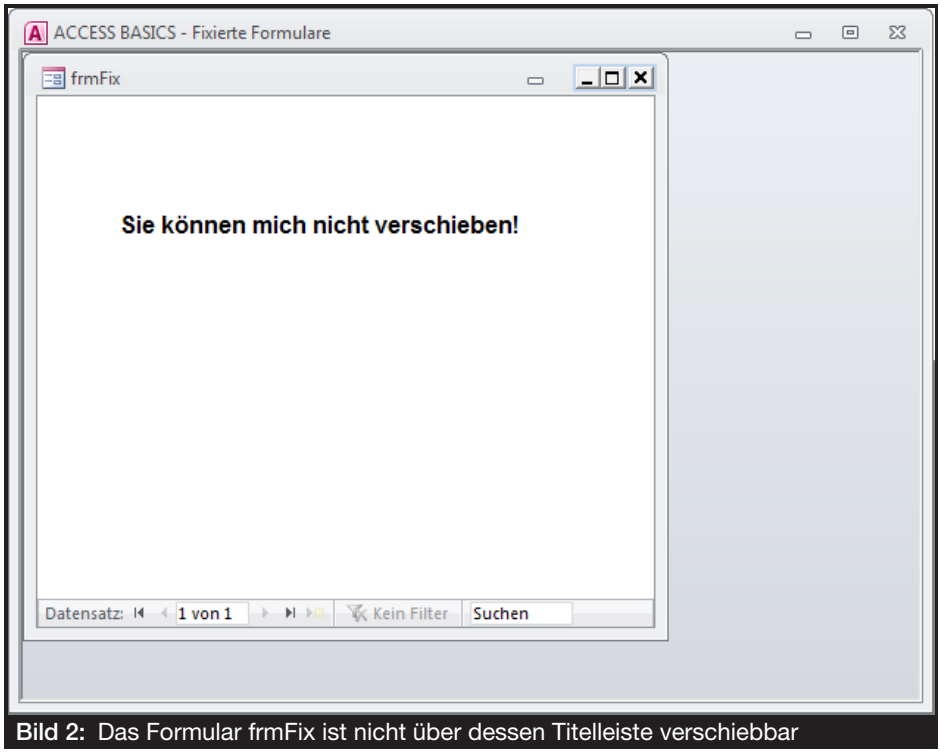


Bild 2: Das Formular frmFix ist nicht über dessen Titelleiste verschiebbar

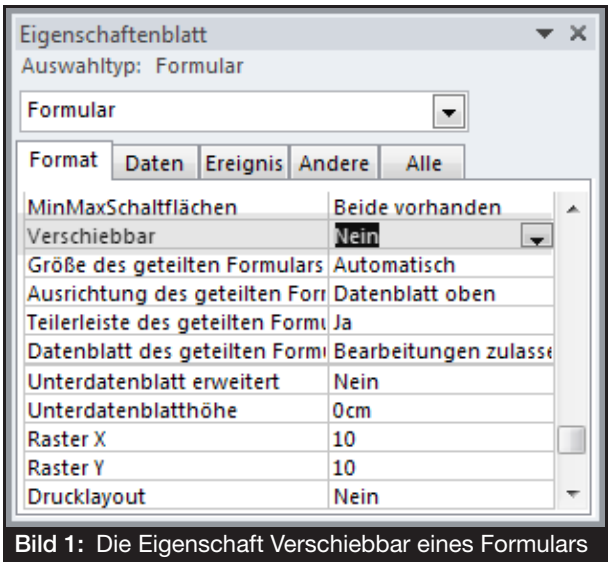


Bild 1: Die Eigenschaft Verschiebbar eines Formulars

nützliche Anwendungsgründe für diese Einstellung?

Die in der Regel verschiebbaren Formulare von Access werden spätestens dann unerfreulich, wenn mehrere Formulare in der Oberfläche geöffnet sind. Sie überlappen sich dann unter Umständen, oder ein Maximieren des einen überdeckt das andere. Das kann beim Anwender zu Irritationen führen – vor allem dann, wenn die Daten des einen mit denen das anderen in irgendeiner Weise verknüpft sind. Derlei Missverständnisse räumen Sie aus, wenn Sie Formulare an einem vorbestimmten Ort öffnen lassen und gegebenenfalls nebeneinander oder untereinander positionieren.

Sehr praktisch ist, dass die Eigenschaft **Verschiebbar** sich auch über VBA steuern lässt (**Form.Moveable**), und das auch noch zur Laufzeit des Formulars! Sie können also die Modi jederzeit ändern. Das Beispielformular **frmFix** wartet nach Einblenden des Ribbon nochmals 5 Sekunden und schaltet dann die Eigenschaft **Verschiebbar** auf **Ja**, worüber zusätzlich der Schriftzug in ihm informiert:

```
Me.Moveable = True
Me!LblInfo.Caption = _
  "Sie können mich jetzt verschieben!"
```

Falls Sie interessiert, wie der Code zum Ausblenden von Ribbon und Navigationsbereich aussieht, so sehen Sie sich die Prozedur in Listing 1 an. Übergeben Sie keinen Parameter für **UnHide**, wodurch der Wert der Variablen automatisch **False** beträgt,

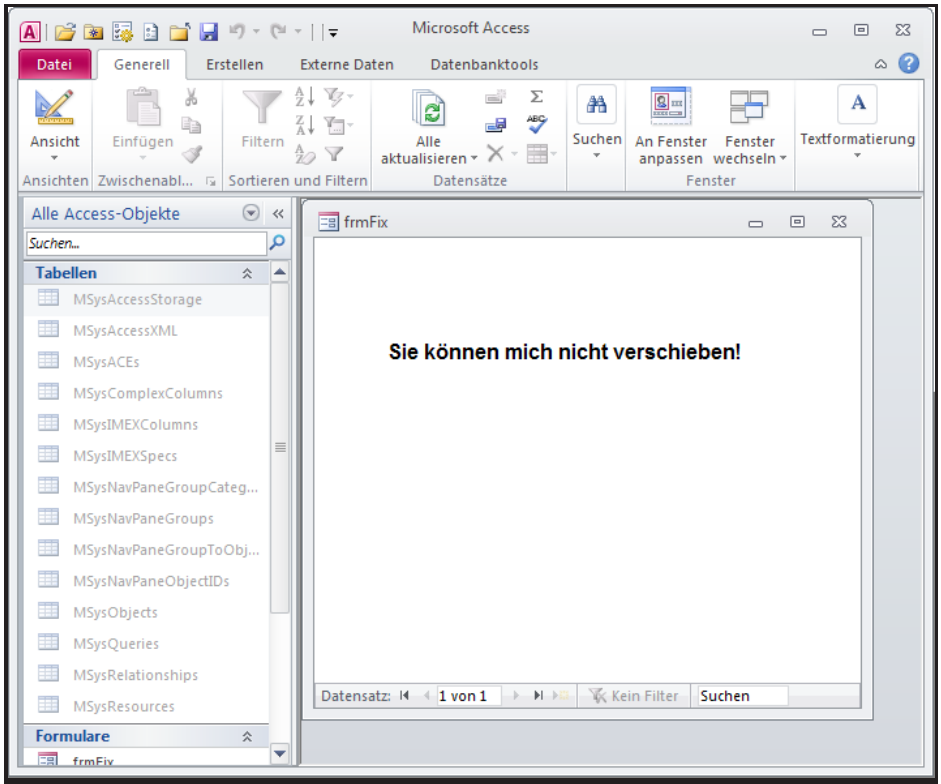


Bild 3: Am fixierten Modus des Formulars ändert sich auch bei eingeblendetem Ribbon und Navigationsbereich nichts

so werden die beiden Elemente ausgeblendet. Bei Übergabe von **True** werden sie wieder eingeblendet.

Die Routine setzt zunächst den Fokus auf den Navigationsbereich, indem sie die **DoCmd**-Methode **SelectObject** aufruft ohne ein Objekt anzugeben. Die **RunCommand**-Konstante **accCmdWindowHide** bewirkt allgemein, dass das aktive Fenster in Access ausgeblendet werden soll. Da dies im Moment das eben fokussierte Navigationsfenster ist, verschwindet dieses. Die Konstante **accCmdWindowUnhide** vollzieht den umgekehrten Vorgang.

Den **Ribbon** blendet man über die Methode **DoCmd.ShowToolbar** ein und aus, wobei als Toolbar der Name **Ribbon** übergeben werden muss,

```
Sub ShowNavigationPane(Optional UnHide As Boolean)
    On Error Resume Next
    DoCmd.SelectObject acTable, , True
    If UnHide Then
        RunCommand acCmdWindowUnhide
        DoCmd.ShowToolbar "Ribbon", acToolbarYes
    Else
        RunCommand acCmdWindowHide
        DoCmd.ShowToolbar "Ribbon", acToolbarNo
    End If
End Sub
```

Listing 1: Ein- und Ausblenden des Ribbon und Navigationsbereichs

und zusätzlich eine Konstante für den gewünschten Zustand, also entweder **acToolbarNo** oder **acToolbarYes**. Die Prozedur können Sie von beliebigem Ort Ihres VBA-Projekt aus aufrufen, und das Formular **frmFix** tut selbiges in seiner **Load**-Ereignisprozedur.

Formulare positionieren

Befindet sich ein Formular in nicht verschiebbarem Zustand, so gilt das nur für die manuellen Oberflächeninteraktionen, nicht aber für VBA. Die Methode **Move** eines Formulars ist weiterhin funktionstüchtig:

```
Forms("frmFix").Move 0,0, 4000, 3000
```

oder einfach

```
Me.Move 0,0, 4000, 3000
```

Die ersten beiden Parameter der Methode (**x**, **y**) geben an, wohin die linke obere Ecke des Formulars wandern soll. Bei normalen Formularen sind diese Werte relativ zum Arbeitsbereich von Access zu verstehen. Bei **Popup**-Formularen gilt der Desktop als Bezug.

Die beiden weiteren Parameter lassen Größenänderungen des Formulars zu. Hier übergeben Sie die Werte für Breite und Höhe in der Einheit Twips, wobei ein Twip in der Regel 1/15 eines Punkts oder 1/567 eines Zentimeters beträgt. Die angegebene Code-Zeile positioniert also das Formular exakt in die linke obere Ecke und setzt seine Größe neu auf 4000 Twips Breite und 3000 Twips Höhe.

```
Sub MoveFixedForms()
    Dim sForm As String
    Dim h As Long, w As Long

    CloseAllObjects

    sForm = "frmFixBackground"
    DoCmd.OpenForm sForm
    DoCmd.Maximize
    w = Forms(sForm).InsideWidth
    h = Forms(sForm).InsideHeight
    DoCmd.Restore
    DoCmd.MoveSize 0, 0, w, h
    DoEvents

    sForm = "frmFix2"
    DoCmd.OpenForm sForm
    Forms(sForm).Moveable = False
    Forms(sForm).Move 3000, 270, 8000, 3300

    sForm = "frmFix3"
    DoCmd.OpenForm sForm
    Forms(sForm).Moveable = False
    Forms(sForm).Move 3000, _
        1200 + Forms("frmFix2").InsideHeight, 8000, 3900
End Sub
```

Listing 2: Routine zum Öffnen dreier nicht verschiebbarer Formulare mit Positionierungen

```
Sub CloseAllObjects(Optional AlsoNavis As Boolean = True)
    Dim i As Long

    For i = Forms.Count - 1 To 0 Step -1
        DoCmd.Close acForm, Forms(i).Name
    Next i
    For i = Reports.Count - 1 To 0 Step -1
        DoCmd.Close acReport, Reports(i).Name
    Next i
    ShowNavigationPane Not AlsoNavis
End Sub
```

Listing 3: Schließen aller Objekte der Datenbank

Switchboard

Kommen wir zu einem Beispiel, dass diese VBA-Steuerungen umsetzt. **MoveFixedForms** in Listing 2 öffnet drei Formulare und positioniert sie, wobei das erste Formular **frmFixedBackground** eine besondere Bedeutung hat, auf die wir noch zu sprechen kommen.

Zuerst aber wird die Hilfsroutine **CloseAllObjects** von ihr aufgerufen, die Sie in Listing 3 finden. Sie schließt alle möglicherweise noch offenen Formulare und Berichte, indem Sie die Auflistung **Forms** und **Reports** in zwei Schleifen abwärts zählend durchläuft. Das ist notwendig, weil beim Aufwärtszählen sich die Anzahl der Objekte in **Count** durch das Schließen derselben jeweils verringern würde. Der Routine kann optional der Boolean-Parameterwert **AlsoNavis** übergeben werden. Steht er auf **True**, so wird die bereits in Listing 1 gezeigte Prozedur angestoßen, welche Ribbon und Navigationsbereich ausblendet.

Dann wird **frmFixBackground** durch OpenForm geladen und über **DoCmd.Maximize** maximiert, um anschließend die Variablen **w** und **h** mit den über **InsightWidth** und **-Height** ermittelten Formulardimensionen zu bestücken. **DoCmd.Restore** versetzt das Formular wieder in den nicht-maximierten Normalzustand, denn beim Wechsel zu einem anderen Formular würde dieses sonst ebenfalls automatisch maximiert. Schließlich übernimmt die **DoCmd**-

Seitennavigation

Nicht nur die dynamische Positionierung von Steuerelementen zur Laufzeit eines Formulars eignet sich zur Datensatznavigation, auch das Ein- und Ausblenden dieser kann manchmal nützlich sein. Wir stellen ein Anwendungsbeispiel mit einer vertikalen Suchleiste dar, die platzsparend den Sprung zu den gewünschten Datensätzen anbietet, wobei diese über einen Mausklick sichtbar wird.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1607_Sidenavigation.accdb**.

Search Bar

Bei dieser Anwendung geht es nicht mehr um ein Verschieben von Steuerelementen, sondern um das Sichtbarmachen einer Listbox, die dennoch den Eindruck einer Bewegung verursacht. Das Formular **frmKunden** der Beispieldatenbank macht es vor.

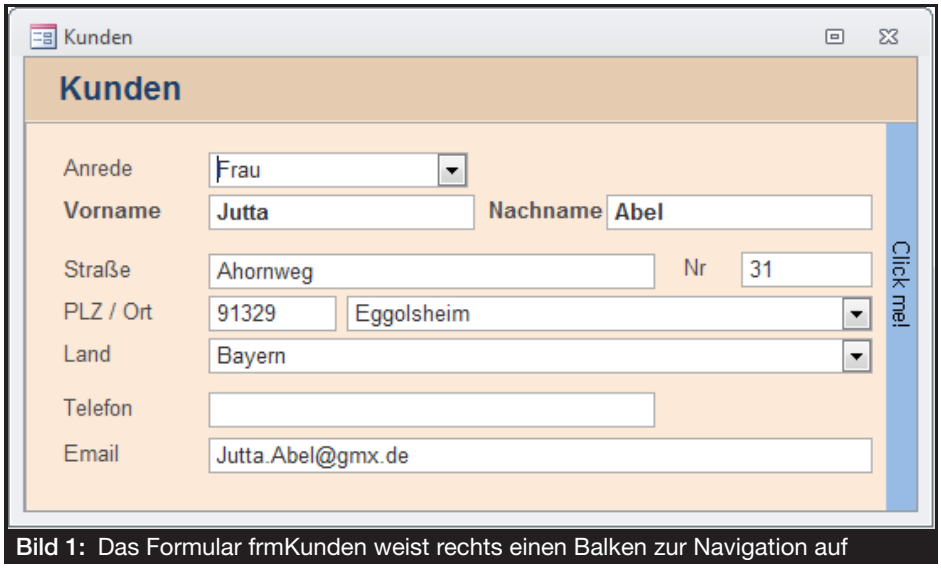


Bild 1: Das Formular frmKunden weist rechts einen Balken zur Navigation auf

Nach dem Öffnen sieht es aus, wie in Bild 1. Außer den Adressfeldern gibt es rechts einen blauen Balken, der scheinbar mit der Aufschrift **Click Me!** versehen ist. Klicken Sie ihn an, so blendet sich eine Listbox ein, wie in Bild 2. Sie enthält die Namen aller Kunden. Wahlweise kann ein Eintrag über Mausklick oder über die Pfeiltasten der Tastatur samt abschließendem **ENTER** aktiviert werden, was den entsprechenden Datensatz ins Formular bringt. Danach wird die Listbox automatisch wieder geschlossen. Sie tut das zudem auch bei der Taste **ESC** und dann, wenn Sie den Fokus über die Maus in eines der Textfelder des Formulars setzen.

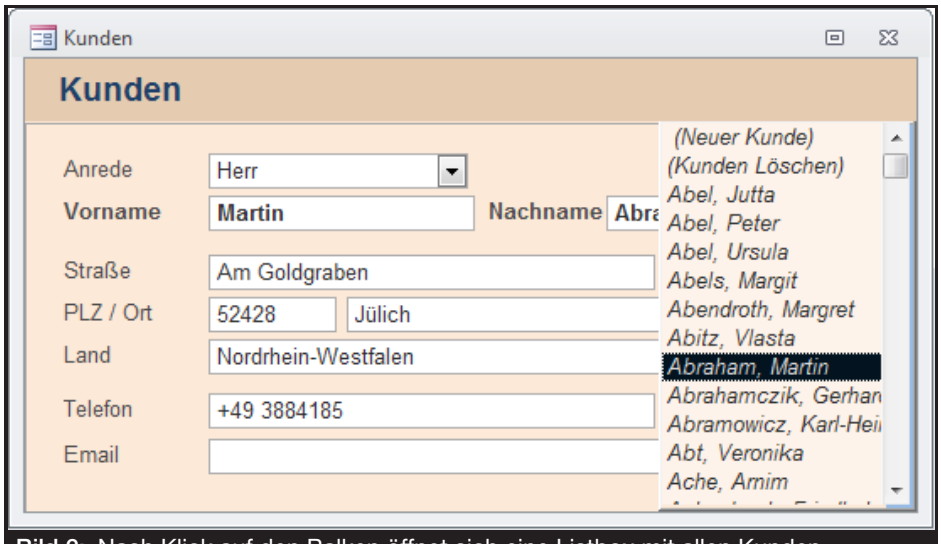


Bild 2: Nach Klick auf den Balken öffnet sich eine Listbox mit allen Kunden

Dies entspricht einem Abbruch des Such- beziehungsweise Navigationsvorgangs. Das Einblenden der Listbox wirkt so, als fahre es von der rechten Seite ins Formular. Um zu verdeutlichen, wie die Sache aufgebaut ist, betrachten wir zunächst den Entwurf des Formulars (siehe Bild 3). Hier findet sich rechts die ungebundene Listbox **lstFind**, welche über ihr Kontextmenü und den Eintrag **Position | In den Vordergrund** über zwei weitere Steuerelemente gebracht wurde. Ihr **Verankern**-Modus steht

auf **Nach unten dehnen**, damit sie Größenänderungen des Formulars vertikal mitmacht. Die überdeckten Steuerelemente sind einmal ein blaues Rechteck **rNavi** mit dem **Verankern**-Modus **Quer und nach unten dehnen**, sowie ein Label **LblClick** mit der Beschriftung **Click Me!**. Damit es diesen Text um 90 Grad gedreht ausgibt, steht die Eigenschaft **Vertikal** im Reiter **Andere** des Eigenschaftenblatts auf **Ja**. Damit beide Steuerelemente beim Formularstart zu sehen sind, ist die Eigenschaft **Sichtbar** der überlagernden Listbox auf **Nein** eingestellt. Deren Datenherkunft übrigens ergibt sich aus einer **UNION**-Abfrage, die aus der Tabelle **tblKunden** und zusätzlichen künstlichen Datensätzen zum Löschen und Hinzufügen von Kunden besteht. Wir erläuterten eine solche bereits in Ausgabe **03/2016** zum Thema **UNION-Abfragen**.

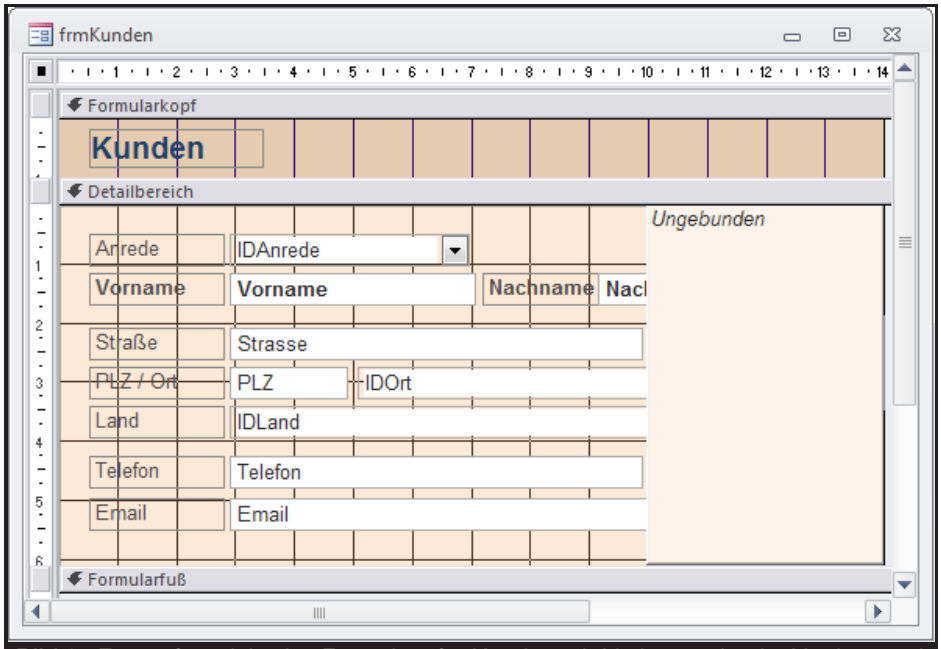


Bild 3: Entwurfsansicht des Formulars frmKunden mit Listbox rechts im Vordergrund

Schreiten wir zum Code des Formulars! Listing 1 zeigt, was beim Anklicken entweder des blauen Rechtecks **rNavi** oder des Labels **LblClick** (**Click Me!**) passiert.

LblClick_Click enthält selbst keine Anweisungen und leitet nur an die Ereignisprozedur **rNavi_Click** weiter. Dort wird der Listbox **lstFind** erst der Wert des im Formular aktuell angezeigten Datensatzes über dessen **ID** zugewiesen, damit sie den Kundeneintrag markiert. Dann wird sie sichtbar gemacht. Durch das Einblenden überdeckt sie nun Balkenrechteck und Label rechts. Schließlich wird der Fokus auf sie gesetzt, damit Maus- und Tastatureingaben sofort auf sie wirken können.

Verliert sie den Fokus, was etwa durch Setzen des Mauszeigers in ein Formularfeld geschieht, so löst dies ihr Ereignis **LostFocus** aus. Der Formularcode reagiert darauf über die Ereignisprozedur **lstFind_LostFocus** und aktiviert den Formular-Timer, welcher auf 100 ms eingestellt wird. In der folglich ausgelösten Timer-Prozedur wird der Timer durch Setzen auf **0** am Ende wieder deaktiviert. Er wird also nur für ein sehr kurzes Delay verwendet, um die ersten zwei Codezeilen verzögert auszuführen.

```
Private Sub rNavi_Click()
    Me!lstFind = Me!ID
    Me!lstFind.Visible = True
    Me!lstFind.SetFocus
End Sub

Private Sub LblClick_Click()
    rNavi_Click
End Sub

Private Sub lstFind_LostFocus()
    Me.TimerInterval = 100
End Sub

Private Sub Form_Timer()
    Me!cbAnrede.SetFocus
    Me!lstFind.Visible = False
    Me.TimerInterval = 0
End Sub
```

Listing 1: Ein- und Ausblenden der Navigations-Listbox über das Timer-Event des Formulars

Grund für diesen Umweg ist die Tatsache, dass diese Zeilen in der **LostFocus**-Prozedur nicht korrekt funktionieren würden, weil etwa die **SetFocus**-Anweisung erst ausgeführt wird, wenn das alte Ereignis abgeschlossen ist. Bei Fokusverlust wird die Listbox wieder unsichtbar gemacht, blendet sich damit aus, und das erste Formularfeld für die Anrede bekommt stattdessen den Fokus. Der Ausgangszustand ist wieder hergestellt.

Datenbanken von Altlasten säubern

Dass sich Access-Datenbanken mit der Zeit aufblähen, dürfte jedem Entwickler einmal aufgefallen sein. Beim Löschen von Daten und Objekten, dem Importieren von Objekten aus anderen Datenbanken, Modifikationen an Modulen, dem Überschrieben von Objekten, fällt Müll an, der oberflächlich nicht sichtbar ist. Über die Ribbon-Funktion 'Komprimieren und Reparieren' kann aber ein Teil dieses Mülls leider nicht entfernt werden. Mit recht überschaubarem VBA-Code ändert sich das.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1606_Saeubern.accdb**.

Temporäre Datenbankobjekte
Access-Datenbanken können sowohl Tabellen, wie Abfragen, enthalten, die Sie im Navigationsbereich nicht zu Gesicht bekommen. Auf diese möchten wir hier den Fokus setzen.

Die Beispieldatenbank implementiert vier relational verknüpften Tabellen zu Kundenadressen. Auf diese setzt das Formular **frmKunden** auf (Bild 1). Die Haupttabelle **tblKunden** steht für die Datenherkunft des Formulars, die Detailtabellen **tblAnreden**, **tblOrte** und **tblLaender** werden für die Kombinationsfelder verwendet. Die Steuerelemente kamen durch einfaches Hineinziehen der **Vorhandenen Datenfelder** zustande. Kombinationsfelder ergeben sich deshalb, weil für die Felder **IDAnrede**, **IDOrt** und **IDLand** im Entwurf von **tblKunden** jeweils **Nachschlagetabellen** angegeben wurden. Also füllt Access hier im Eigenschaftenblatt neben dem **Steuerelementinhalt** auch gleich die passenden Quelltabellen ein und verknüpft die Felder, wie dies aus der Abbildung hervorgeht.

Grundsätzlich legt Access nun beim Speichern sowohl für das Formular, wie auch für Kombinations- und Listenfelder intern Abfragen an, die weder im Navigationsbereich, noch im Formularentwurf in Erscheinung treten. Doch öffnen Sie einmal die Sys-

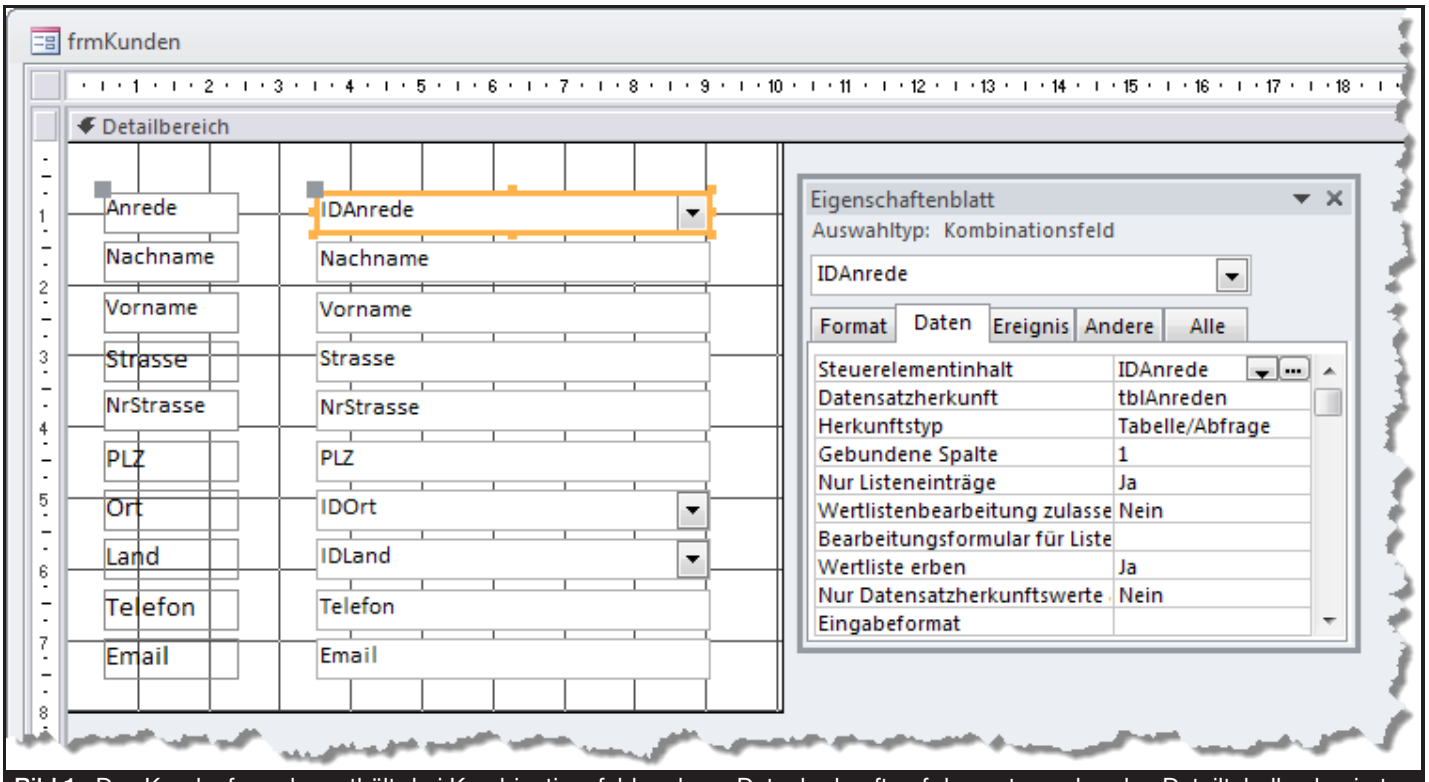


Bild 1: Das Kundenformular enthält drei Kombinationsfelder, deren Datenherkunft auf den entsprechenden Detailtabellen basiert

temtabelle **MSysObjects**, die einen Katalog aller Datenbankobjekte verzeichnet. Bild 2 zeigt, dass hier vier Abfragen zu sehen sind. Sie werden durch die Spalte **Type** identifiziert, wenn diese den Wert **5** aufweist. Für die Namen der Abfragen ist jeweils das Präfix **~sq_** spezifiziert. Folgt darauf ein **f**, so handelt es sich um die Datenherkunft eines Formulars. Access verwendet also nicht die Tabelle **tblKunden** selbst, sondern eine auf ihr beruhende Abfrage für die Daten des Formulars. Der Name des Formulars schließt sich daran an. Für die Datenherkunft von Kombinationsfeldern steht eine **~sq_c**...-Abfrage. Ihr folgt der Name des Formulars, in dem sich das Kombinationsfeld befindet, und schließlich der Name des Steuerelements, ebenfalls durch ein **~sq_c**-Präfix gekennzeichnet.

Diese Abfragen sind über die Oberfläche nicht zu erreichen, wohl aber über DAO! Geben Sie so etwa den SQL-String für die dem **Anrede**-Steuerelement zugrundeliegende Abfrage im VBA-Direktfenster aus:

```
? CurrentDB.QueryDefs _
 (" ~sq_cfrmKunden~sq_cIDAnrede").SQL
```

Sie erhalten dieses Ergebnis:

```
SELECT DISTINCTROW *
FROM tblAnreden;
```

Das Formular hat diese Datenherkunft:

```
? CurrentDB.QueryDefs _
  ("~sq_ffrmKunden").SQL
> "SELECT DISTINCTROW *
FROM tblKunden;"
```

MSysObjects

Id	LvProp	Name	ParentId	Type
-2147483648		DataAccessPages	251658240	3
-2147483647		Forms	251658240	3
-2147483646		Modules	251658240	3
-2147483645		Reports	251658240	3
-2147483644		Scripts	251658240	3
-2147483643		SysRel	251658240	3
-2147483642	.ong binary-Daten	SummaryInfo	251658242	-32757
-2147483641	.ong binary-Daten	UserDefined	251658242	-32757
-2147483640	.ong binary-Daten	frmIntro	-2147483647	-32768
-2147483639	.ong binary-Daten	frmKunden	-2147483647	-32768
-2147483638		mdlSaeubern	-2147483646	-32761
-2147483637		Admin	-2147483643	-32758
-2147483636	.ong binary-Daten	~sq_cfrmKunden~sq_cIDAnrede	251658241	5
-2147483635	.ong binary-Daten	~sq_cfrmKunden~sq_cIDLand	251658241	5
-2147483634	.ong binary-Daten	~sq_cfrmKunden~sq_cIDOrt	251658241	5
-2147483633	.ong binary-Daten	~sq_ffrmKunden	251658241	5
-2147483632		{1706D1A5-CB18-47BF-8AA1-956A(	251658243	8
-2147483631		{68EF2676-93F6-4527-8206-C88E89(	251658243	8
-2147483630		MSysNavPaneGroupCategoriesMS	251658243	8

Datensatz: 13 von 53 | Kein Filter | Suchen

Bild 2: In der Systemtabelle MSysObjects finden sich die versteckten Abfragen für die Kombinationsfelder des Formulars, beginnend jeweils mit dem Präfix ~sq_

Das **DISTINCTROW** sagt der **Database Engine**, dass die zurückgelieferten Datensätze eindeutig sein sollen. Das ist an sich überflüssig, da die Tabellen allesamt einen **Primärschlüssel** aufweisen.

Entfernten Sie diesen später aber, so könnten auch doppelte identische Datensätze in ihr abgespeichert werden. Hier bekommt das **DISTINCTROW** dann einen Sinn.

Diese internen Abfragen sind übrigens nicht schreibgeschützt. Sie können ihren SQL-Ausdruck problemlos ändern; etwa so:

```
CurrentDB.QueryDefs _
  ("~sq_ffrmKunden").SQL = _
"SELECT * FROM tblKunden"
```

Diese Änderung bleibt auch nach Modifikationen am Formularentwurf erhalten! Es versteht sich von selbst, dass Sie hier nicht unsinnige Ausdrücke einsetzen sollten, um Probleme zu vermeiden.

Was hat dies alles aber mit dem Müll zu tun, von dem eingangs die Rede war? Leider kommt es vor, dass besonders bei Formularen, die Sie aus anderen Datenbanken importierten, die auf sie bezogenen internen Abfragen nicht ordnungsgemäß gelöscht werden, wenn sie selbst entfernt werden. Damit finden sich verwaiste ~sq-Einträge in der Systemtabelle MSysObjects, und auch deren Spezifikationen in MSysQueries bleiben bestehen. Das bläht die Datenbank auf, wenn auch in bescheidenem Umfang.

Schwerwiegender jedoch ist ein anderer Sachverhalt. Löschen Sie Objekte im Navigationsbereich über die Taste ENTF, so läuft alles glatt. Verwenden Sie aber STRG-ENTF, wie Sie es eventuell aus dem Explorer gewohnt sind, um Dateien nicht

in den Papierkorb zu transferieren, sondern endgültig zu löschen, so verhält sich Access etwas eigentümlich. Scheinbar ist das Objekt nun gelöscht, findet sich aber trotzdem noch in **MSysObjects**. Das ändert sich in der Regel erst, nachdem Sie die Datenbank schließen oder komprimieren. Machen Sie dazu folgendes Experiment.

Markieren Sie drei Tabellen, wie in Bild 3. Kopieren Sie diese über das Kontextmenü oder über **STRG-C** und fügen Sie sie als Kopien wieder in die Datenbank ein. Access fragt Sie dabei nach dem Namen der einzufügenden Tabellen. Belassen Sie es bei den Voreinstellungen, wodurch es zu den Einträgen in Bild 4 kommt. Markieren Sie die neuen Tabellen und drücken nun **STRG-ENTF**. Die Tabellen scheinen nun gelöscht worden zu sein.

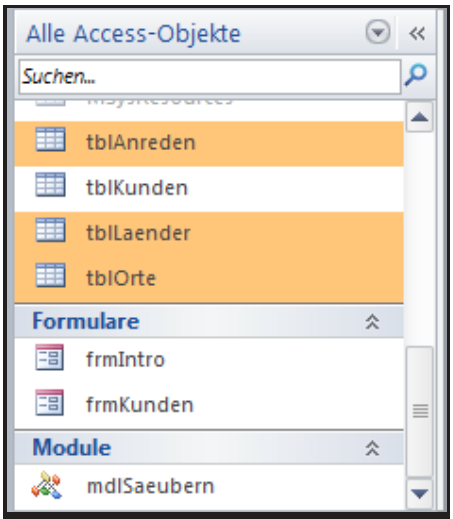


Bild 3: Markieren dreier Tabellen

Tatsächlich hat Access sie jedoch lediglich umbenannt. Ein Blick in **MSysObjects** zeigt sie mit kryptischen Namen, die alle mit **~TMPCLP** beginnen (siehe Bild 5). Die Spalte Type identifiziert sie über den Wert 1. Ihr Ursprung ist indessen nicht mehr auszumachen. Beim Schließen der Datenbank entfernt Access diese temporären Tabellen. Auch andere Objektarten erhalten übrigens dieses Präfix. Leider bleibt dieser Löschvorgang unter Umständen aus. Stürzt Access etwa beim Ausführen vom API-Anweisungen ab, so verbleiben die temporären Objekte für ewig in der Datenbank. Welche anderen Ursachen es hat, dass dieser Aufräumvorgang nicht ausgeführt wird, ist nicht dokumentiert. Tatsächlich passiert dies nach unseren Beobachtungen öfter, als nur beim Absturz von Access. Natürlich bläht sich die Datenbank nur deutlich mehr auf, als das bei den internen Abfragen der Fall ist. Auslösend für diesen Beitrag war nämlich die Tatsache, dass die Beispieldatenbank bei Entwicklung plötzlich eine Größe von 16 MB hatte. Komprimieren und Reparieren half nicht. Ursache waren Experimente mit den VBA-Anweisungen ExportXML und ImportXML, die an anderer Stelle beschrieben werden. Die Kundentabelle fand sich mehrfach als ~TMPCLP-Objekte in der Datenbank!

Temporäre Objekte löschen

Das Löschen dieser verwaisten Objekte ist denkbar einfach. Es braucht dafür nur die Routine CleanDB, die Sie im Modul mdlSaeubern der Beispieldatenbank finden. Zeigen sich bei Inspektion der Tabelle MSysObjects in Ihrer Datenbank solche verwaisten Objekte, so importieren Sie die Routine einfach in Ihre Datenbank. Bei ihrer Ausführung kann nichts kaputt gehen.

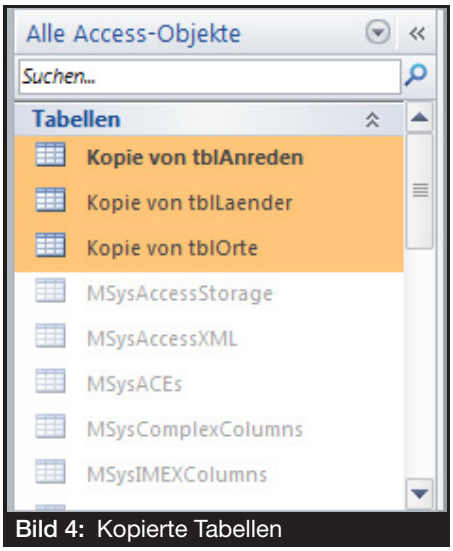
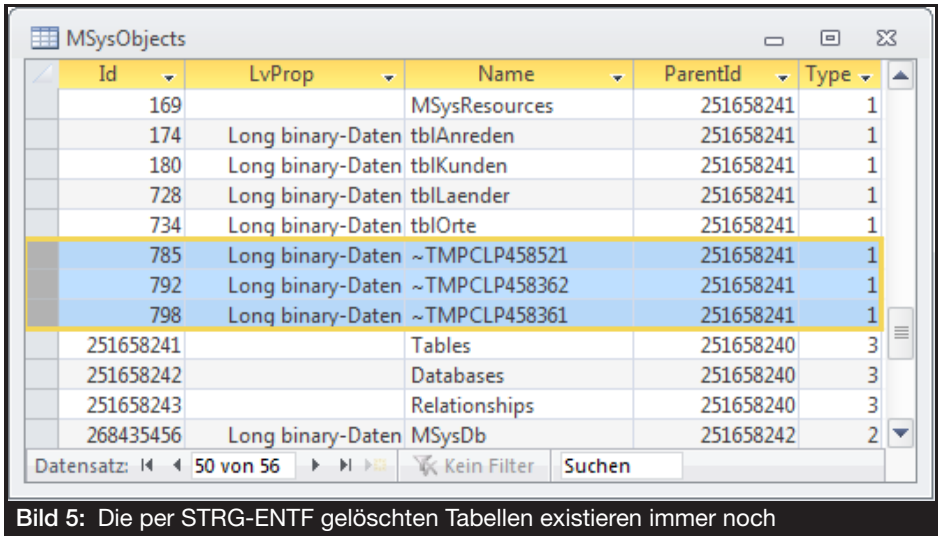


Bild 4: Kopierte Tabellen


MSysObjects

Id	LvProp	Name	ParentId	Type
169		MSysResources	251658241	1
174	Long binary-Daten	tblAnreden	251658241	1
180	Long binary-Daten	tblKunden	251658241	1
728	Long binary-Daten	tblLaender	251658241	1
734	Long binary-Daten	tblOrte	251658241	1
785	Long binary-Daten	~TMPCLP458521	251658241	1
792	Long binary-Daten	~TMPCLP458362	251658241	1
798	Long binary-Daten	~TMPCLP458361	251658241	1
251658241		Tables	251658240	3
251658242		Databases	251658240	3
251658243		Relationships	251658240	3
268435456	Long binary-Daten	MSysDb	251658242	2

Datensatz: 50 von 56 Kein Filter Suchen

Bild 5: Die per STRG-ENTF gelöschten Tabellen existieren immer noch

Flying Controls

Fliegende Steuerelemente? Üblicherweise positionieren Sie Textboxen, Labels und Kombinationsfelder fest im Entwurf eines Formulars und ändern an der ergonomischen Gestalt zur Laufzeit nichts mehr. Doch manchmal gibt es gute Gründe für ein dynamisches Layout, bei dem Steuerelemente ihren angestammten Platz verlassen. Einige Anregungen dazu liefert dieser Beitrag.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1607_FlyingCtls.accdb**.

Steuerelemente verschieben

Für die Position jedes Steuerelements sind dessen Eigenschaften **Links** und **Oben** im Formular- oder Berichtsentwurf verantwortlich, die Sie entweder durch Ziehen der Elemente mit der Maus ändern, oder durch einen direkten Eintrag im Eigenschaftenblatt. Gleiches gilt für die **Breite** und **Höhe** eines **Controls**.

Unter VBA greifen Sie auf diese Eigenschaften über die Properties **Left** und **Top** zu, sowie über **Width** und **Height**. Diese Methoden sind nicht schreibgeschützt, so dass Sie über entsprechende Wertzuweisungen die Positionen zur Laufzeit ändern können. Ein Textfeld **txtName** bringen Sie etwa so in eine andere Position und auf neue Abmessungen:

```
txtName.Left = 300
txtName.Top = 120
txtName.Width = 2100
txtName.Height = 270
```

Das Ganze können Sie auch über einen **With-Block** auf das **Control** umsetzen und setzen besser noch den Container **Me** des Steuerelements voran:

```
With Me!txtName
  .Left = 300
  .Top = 120
  .Width = 2100
  .Height = 270
End With
```

Die Einheit dieser Werte ist unter Access immer **Twips**. 15 **Twips** entsprechen in der Regel 15 Pixeln und ein Zentimeter entspricht etwa 567 **Twips**. Die Positionsangaben beziehen sich grundsätzlich auf die linke obere Ecke des Steuerelements als seinen Ursprung und sind, entgegen der Aussage in der Access-Hilfe, relativ zum jeweiligen Container. Das kann der Detailbereich sein, aber auch der Kopf- oder Fußbereich. Bei Berichten kommen noch weitere Containerflächen hinzu.

Eine Alternative zum direkten Zuweisen der Werte an **Left** und **Top** stellt die **Move**-Methode dar, die allen Steuerelementen zueigen ist. Mit ihr können Sie die vier Eigenschaften auf einen Schlag einstellen:

```
Me!txtName.Move 300, 120, 2100, 270
```

Alle vier Angaben sind optional. Sie können also mit dieser Methode etwa nur Position oder nur die Abmessungen ändern, indem Sie die anderen Parameter weglassen:

```
Me!txtName.Move 300,120
Me!txtName.Move , , 2100, 270
```

Schließlich gibt es mit der **DoCmd**-Anweisung **MoveSize** noch eine weitere Methode, die in ihren Parametern der **Move**-Methode der Controls entspricht. Sie wirkt allerdings nicht nur auf Steuerelemente, sondern genauso auf Formularfenster. Das macht zur Bedingung, dass das zu positionierende Element zum Zeitpunkt ihres Aufrufs den Fokus besitzt:

```
Me!txtName.SetFocus
DoCmd.MoveSize 300, 120, 2100, 270
```

Auch hier sind alle Parameter optional. Die Methode entkoppelt die Anweisung vom Steuerelement und eignet sich daher nur für Fälle, in denen das **Control**-Objekt ungewiss ist.

Fliegende Steuerelemente

Das weitgehend sinnfreie Formular **frmFlying** der Beispieldatenbank demonstriert die Verschiebeeffekte zur Laufzeit (Bild 1). In einer VBA-Schleife werden über den Timer des auf Adressdatensätzen basierenden Formulars sämtliche Steuerelemente fortlaufend verschoben. Damit sie nicht über den Rand hinaus geraten, was Access ohnehin mit einer Fehlermeldung quittieren würde, reflektieren sie quasi billardmäßig von den

Seiten. Den Code, der beim Starten des Formulars zunächst ausgeführt wird, finden Sie in Listing 1. Er ist auf den ersten Blick nicht selbsterklärend.

Modulweit gelten die Variablen **colCtl** (**Collection**-Objekt) und das Array **arrXY**. Die **Collection** versammelt indiziert die Namen aller Controls des Formulars, **arrXY** ist zweidimensional und speichert deren Positionen. Die **For-Each**-Schleife durchläuft alle Steuerelemente über die Iteratorvariable **ctl**. Das können Textboxen, Comboboxen oder Labels sein. In **colCtl** werden per **Add**-Methode die Indizes (**i**) der Controls aufgenommen, wobei als Schlüssel jeweils der Steuerelementname herhält. Das Array **arrXY** wird über **ReDim** in der zweiten Dimension auf die Zahl der Steuerelemente dimensioniert (**Controls.Count**) und andererseits auf **1** in der ersten Dimension, was bedeutet, dass zu jedem Array-Element zwei Werte abgespeichert werden können: ein **X**- und ein **Y**-Wert für die Positionsverschiebungen. Die **Collection**

```
Private colCtl As New VBA.Collection
Private arrXY() As Long

Private Sub Form_Load()
    Dim ctl As Access.Control
    Dim i As Long

    ReDim arrXY(1, Me.Controls.Count - 1)
    For Each ctl In Me.Controls
        arrXY(0, i) = 1 + 5 * (Rnd - 0.5)
        arrXY(1, i) = 1 + 5 * (Rnd - 0.5)
        colCtl.Add i, ctl.Name
        i = i + 1
    Next ctl
    Me.TimerInterval = 20
End Sub
```

Listing 1: Prozedur beim Laden des Formulars frmFlying (Initialisierung)

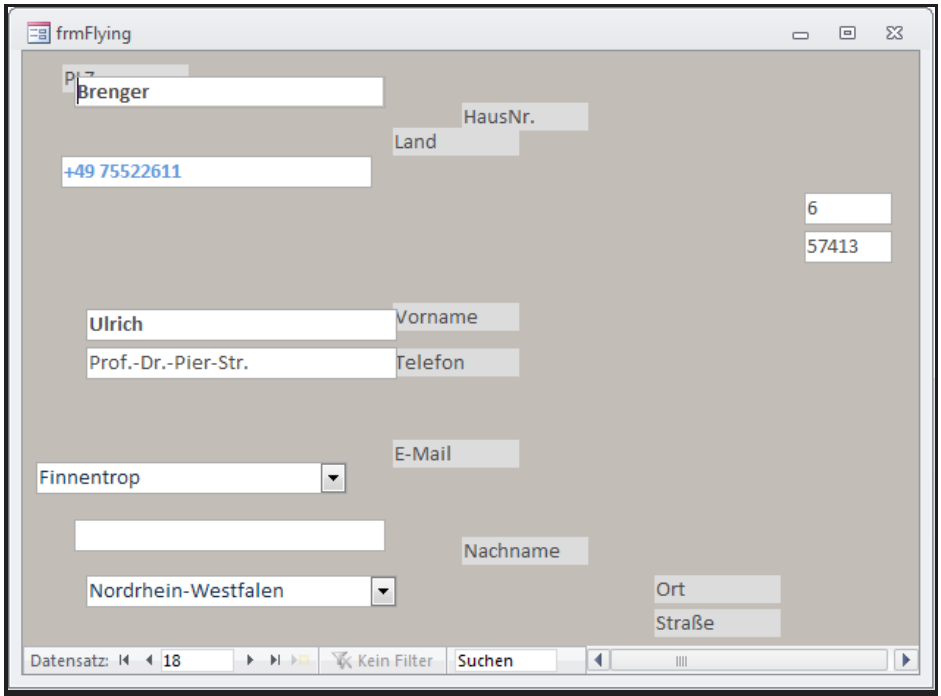


Bild 1: Das Formular frmFlying zeigt in der Gegend herumfliegende Steuerelemente

dient dazu, um später aus einem Steuerelementnamen seinen Index für **arrXY** bestimmen zu können. Außerdem wird das Array mit **X**- und **Y**-Verschiebefaktoren vorgefüllt, die sich aus Zufallszahlen (**Rnd**) ergeben. Der Ausdruck

```
1 + 5 * (Rnd - 0.5)
```

bewirkt, dass die Klammer positiv oder negativ wird, wenn **Rnd** größer oder kleiner ist, als **0.5**. Der Bereich des Klammerausdrucks beträgt damit **-2.5** bis **+2.5**.

Abschließend wird der **Timer** des Formulars aktiviert und auf 20 Millisekunden eingestellt, was zu 50 Wiederholungen pro Sekunde führen sollte. Die Prozedur des **Timer-Events** in Listing 2 nimmt nun die eigentlichen Steuerelementverschiebungen vor und wertet aus, ob sie am Rand des Formulars reflektiert werden sollen.

Auch hier wird eine Schleife auf alle Controls des Formulars gesetzt. Aus **colCtl** wird jeweils erst über den Namen der **Control**-Variable **ctl** dessen Index für das Array **arrXY** errechnet und in der Variablen **i** abgespeichert. Dann wird zu **Left** und **Top** des Controls der mit 30 multiplizierte Verschiebewert aus dem Array hinzuaddiert. Je höher dieser Multiplikator, desto schneller bewegt sich das Steuerelement zur Laufzeit über das Formular.

Die anschließenden vier **If**-Bedingungen überprüfen jeweils mit **Kleiner** oder **Größer als**, ob das Control den Formularrand erreicht hat, wobei ein Offset von 90 **Twips** (= 6 Pixel) mit einbezogen wird. Für den rechten oder unteren Rand ist das etwas komplizierter. Hier muss immer erst

von den für die Formularabmessungen verantwortlichen Eigenschaften **InsideWidth** und **InsideHeight** die Breite (**ctl.Width**) oder Höhe (**ctl.Height**) des Steuerelements abgezogen werden. Trifft die Randbedingung zu, so wird der Verschiebewert im Array negiert, also jeweils **X**- oder **Y**-Richtung umgekehrt. Außerdem bekommt das Steuerelement dann eine neue Startposition zugewiesen, damit die Reflektion an einem klaren Raster ausgerichtet erscheint.

Das abschließende **DoEvents** stellt sicher, dass die Verschiebeoperationen sich visuell auch tatsächlich auswirken und das Formular neu gezeichnet wird. Ohne diese Anweisung käme es zu Rucklern.

Diese eher der Sparte **Game-Programmierung** zuzuordnende Anwendung soll lediglich andeuten, welche Rechen- und Kodierungsvorgänge beim Verschieben von Steuerelementen anfallen können. Doch es gibt auch nützlichere Verwendungszwecke...

Akkordeon

Nein, nicht um die visuelle Nachbildung des Musikinstruments geht es hier! Sollten Sie mit Webprogrammierung vertraut sein, so kennen Sie möglicherweise aus **AJAX-Frameworks** das **Accordion**. Hierbei handelt es sich um Seitenbereiche, die sich dynamisch durch Hoch- und Herunterschieben ein- und ausblenden. Verwendet werden kann dies etwa für Navigationselemente. Das Formular **frmAccordion** der Beispieldatenbank realisiert so etwas für Kundendatensätze (siehe Bild 2).

```
Private Sub Form_Timer()
    Dim ctl As Access.Control
    Dim i As Long

    For Each ctl In Me.Controls
        i = colCtl(ctl.Name)
        ctl.Left = ctl.Left + 30 * arrXY(0, i)
        ctl.Top = ctl.Top + 30 * arrXY(1, i)

        If (ctl.Left < 91) Then
            arrXY(0, i) = -arrXY(0, i)
            ctl.Left = 90
        End If
        If ((ctl.Left - 91) > (Me.InsideWidth - ctl.Width)) Then
            arrXY(0, i) = -arrXY(0, i)
            ctl.Left = Me.InsideWidth - ctl.Width - 90
        End If
        If (ctl.Top < 91) Then
            arrXY(1, i) = -arrXY(1, i)
            ctl.Top = 90
        End If
        If ((ctl.Top + 91) > (Me.InsideHeight - ctl.Height)) Then
            arrXY(1, i) = -arrXY(1, i)
            ctl.Top = Me.InsideHeight - ctl.Height - 90
        End If
    Next ctl
    DoEvents
End Sub
```

Listing 2: Timer-Prozedur des Formulars zum Bewegen der Steuerelemente

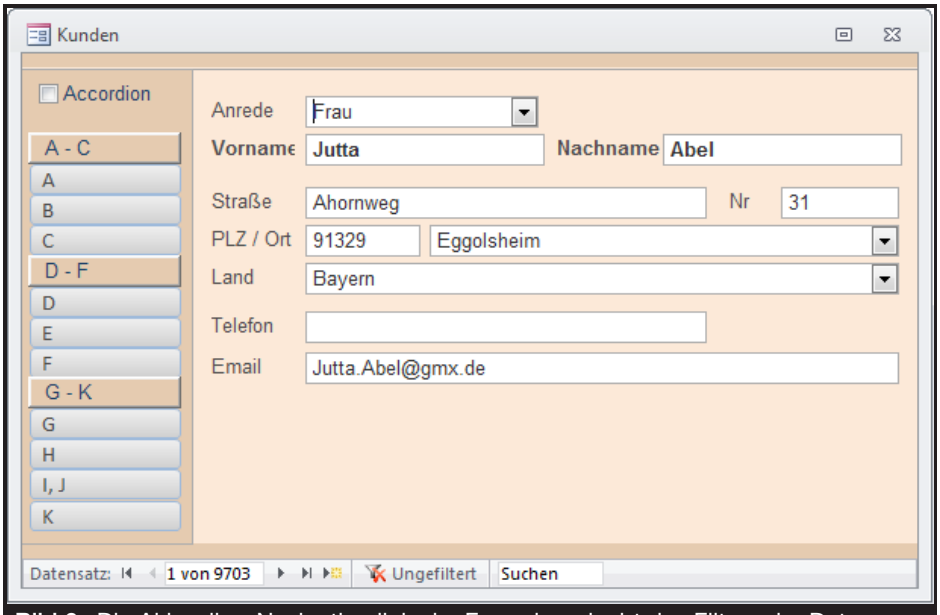


Bild 2: Die Akkordion-Navigation links im Formular erlaubt das Filtern der Daten

Auf der linken Seite finden Sie alphabetisierte Buttons, die zur Filterung der Datensätze dienen. Ein Klick auf **D** etwa zeigt im Formular nur noch jene Datensätze an, die mit einem **D** im Nachnamen beginnen.

Zugriff auf Server über ODBC

Manchmal stehen Sie vor der Aufgabe, die Daten eines SQL-Servers unter Access zu verwalten, oder sogar einen solchen als Backend zu verwenden. Grund für seinen Einsatz ist meist eine Mehrbenutzerumgebung, bei der Access-Backends schnell in die Knie gehen. Ein Buch mit sieben Siegeln? Keine Scheu, die Sache ist nicht gar so kompliziert und die ersten Schritte sind schnell gemacht!

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1607_ODBC.zip**.

SQL-Server und ODBC

Greifen Sie unter Access auf die Daten von Tabellen zu, so befinden sich diese entweder in der aktuellen Datenbankdatei, oder in einer externen **ACCDB** oder **MDB**. Handelt es sich um eine externe Datei, so müssen deren Tabellen über den Tabellenverknüpfungsdialog von Access verlinkt werden. Auch wenn dies nicht so offensichtlich ist, kommt hier in beiden Fällen eine zusätzliche Komponente in Gestalt der **Access Database Engine** als zusätzliche Schicht ins Spiel. Diese **Engine** kann sogar ohne Access selbst unter Windows installiert werden – Microsoft bietet sie als freien Download an. Diese **Engine** stellt quasi den Datenbankserver von Access dar.

Ein **SQL-Server** verrichtet ähnliche Arbeit, wie die **Access Database Engine**, verwaltet seine Tabellendateien jedoch unsichtbar im Hintergrund. Erst auf eine SQL-Anfrage über einen Kommunikationskanal zu ihm, sei es das **TCP/IP**-Protokoll oder eine sogenannte **Pipe**, liefert er die gewünschten Daten aus. Access allein ist nicht imstande, solche Anfragen zu stellen, denn es kennt ja die Schnittstelle zum Server, seine **API**, nicht. Deshalb wurde einst eine allgemeingültige Schnittstelle unter Windows geschaffen, die auf den Namen **Open Database Connectivity** (**ODBC**) hört. Diese standardisierte Schnittstelle hat einen Satz von Zugriffsmethoden, die Access anzusprechen weiß. Das **ODBC**-System vermittelt diese Zugriffe über einen speziellen Treiber, der für den jeweiligen **SQL-Server** installiert sein muss. Der Treiber allein kennt die **API** des Servers und übersetzt die Anfragen auf dessen Schnittstelle. Damit der Unterschied etwas deutlicher wird folgt hier zunächst im Groben die Darstellung des Zugriffs von Access auf eine ACCDB-Datei:

```
Access
|
Access Database Engine
|
(Netzwerk)
|
ACCDB-Datei
```

Access fragt die **Access Database Engine** an, die ihrerseits entweder lokal oder gegebenenfalls über ein Netzwerk auf die Datei zugreift, um deren Daten auszulesen oder zu schreiben. Ähnlich sieht dies bei der **ODBC**-Kommunikation aus, nur dass hier zusätzliche Schichten nach der **Access Database Engine** eingezogen sind, die ihrerseits nun natürlich andere Methoden zum Zugriff absetzen muss:

```
Access
|
Access Database Engine
|
ODBC-Administrator
|
ODBC-Treiber
|
(Netzwerk)
|
DBMS (SQL-Server)
```

Der **ODBC-Administrator** ist das Framework für alle ODBC-Vorgänge. Er lädt oder entlädt etwa einen bestimmten installierten ODBC-Treiber, der von der Engine bestimmt wird. Dieser Administrator enthält auch eine Oberfläche, die sich über die Systemsteuerung von Windows aufrufen lässt. Allerdings geht das zunächst nur mit administrativen Rechten, weil sich hier Systemeinstellungen ändern ließen.

Der schließlich geladene **ODBC-Treiber** kontaktiert den SQL-Server und verarbeitet als Mittler die Zugriffe. In der Regel geschieht dies über einen **TCP/IP**-Kanal, und das selbst dann,

wenn der **SQL-Server** lokal installiert ist. Ähnlich, wie ein HTML-**Web-Server**, nimmt der **SQL-Server** die Anfragen auf, sucht die gewünschten Daten aus seinem **DBMS** (Database Management System) heraus, und schickt sie zurück. Das Ganze schlägt nun die entgegengesetzte Richtung ein, bis die Daten schließlich an Access gelangen.

Konkrete SQL-Server

An dieser Stelle wären einige frei verfügbare Implementationen von SQL-Servern zu nennen. Der prominenteste ist wohl der aus dem Hause Microsoft selbst. Er nennt sich denn auch schlicht und ungeniert **SQL-Server** (Microsoft SQL Server). Die großen Versionen sind kostenpflichtig, und das nicht zu knapp! Die **Developer Edition**, seit der Version 2014 für jeden installierbar, der einen Account auf der **Visual Studio**-Seite anlegt, ist genauso mächtig, sein Einsatz jedoch ausschließlich für Entwicklungszwecke gedacht. Produktiver Einsatz ist untersagt. Anders bei der **Express Edition**, die beliebig einsetzbar ist, dafür aber einige Einschränkungen aufweist. Die Gegenüberstellung zeigt die wichtigsten Unterschiede:

Developer Edition
- Arbeitet mit allen CPU-Kernen
- Gesamter RAM ist ansprechbar
- Datenbankgröße beliebig
- Für den Download ist ein Visual Studio-Account erforderlich

Express 2014
- Arbeitet mit max. 4 CPU-Kernen
- Nur 1 GB RAM pro Instanz
- Datenbankgröße max. 10 GB
- Frei downloadbar

Wenn man die Limitationen von Access und ACCDBs kennt, so erscheinen die Beschränkungen der **Express**-Edition ziemlich verkraftbar. Im Gegenteil! Mit 10 GB kann eine Datenbank sogar erheblich größer sein, als eine ACCDB.

Die Tools zur Verwaltung der Server hingegen sind frei erhältlich.

Nicht weniger verbreitet ist der **MySQL-Server,** welcher einen langen Weg vom **Open Source**-Projekt über die Firmen **MySQL AB**, **Sun** und nun **Oracle** hinter sich hat. Die **MySQL Community Edition** unterliegt funktionell keinen wesentlichen Beschränkungen gegenüber der recht teureren **Enterprise Edition**. Liest man die Lizenzbedingungen des im Quellcode verfügbaren Servers allerdings genau, so wird klar, dass **Closed Source** untersagt ist. Das bedeutet für Sie als Access- Entwickler, dass Ihre Datenbank, sobald sie auf den Server zugreift, ebenfalls quelloffen sein muss. Eine **ACCDE** mit verstecktem VBA-Code wäre somit nicht erlaubt, es sei denn, ihr Quellcode wäre öffentlich dokumentiert. Die meisten Firmen wären damit nicht glücklich. Die Entscheidung, ob man das alles so genau nehmen muss, überlassen wir Ihnen...

Ein Ausweg aus dem Dilemma führt zu **MariaDB**. Der Server ist ein kompletter Ersatz für **MySQL** und wird auch vom Team um den ursprünglichen Hauptentwickler Johansson weiterprogrammiert, der Wert auf wirkliches **Open Source** legt. **MariaDB** ist kompatibel und kann sogar über den **ODBC-Treiber** für MySQL angesprochen werden.

Noch mehr **Open Source** bekommt man mit dem immer verbreiteteren **PostGreSQL**. Die Engine lässt in keiner Hinsicht zu wünschen übrig und ist inzwischen eher fortschrittlicher und funktionell umfangreicher, als **MySQL**.

Schließlich zu erwähnen wäre noch der **SQL-Server ORACLE Express 11g (XE)**, die abgespeckte Version des in Unternehmen weltweit verbreitetsten **DBMS**. Man bekommt auch hier einen sehr professionellen Server, muss dafür aber ziemliche Abstriche machen:

- Arbeitet mit nur 1 CPU-Core
- Kann nur 1 GB RAM ansprechen
- 11 GB pro Datenbank
- Nur eine Datenbank pro Rechner

All diese SQL-Server installieren Sie komfortabel über eine Setup-Datei. Anschließend unterstützen Sie in der Regel Assistenten bei der Konfiguration der Server oder dem Anlegen erster Datenbankinstanzen. Zur ausführlicheren Verwaltung der Server verwenden Sie die zumeist mitgelieferten Administrations-Tools.

SQLite

Um Ihnen den Umgang mit **ODBC** näher zu bringen, benötigen Sie zum Glück keinen der erwähnten **SQL-Server**, die ohne weiteres mehrere GB Festplattenspeicher auf die Waage bringen können. Mit **SQLite** gibt es einen **Open Source**-Server, der unter **Android** das wichtigste Daten-

banksystem darstellt. Er ähnelt stark der **Access Database Engine**, denn eine Datenbank besteht hier ebenfalls nur aus einer Datei. Krass ist der Umstand, dass die Engine selbst aus nur einer **EXE** oder **DLL** mit etwa 4 MB besteht! Der Funktionsumfang ist gegenüber ausgewachsenen **DBMS** natürlich bescheiden, aber eher vielseitiger, als der von Access.

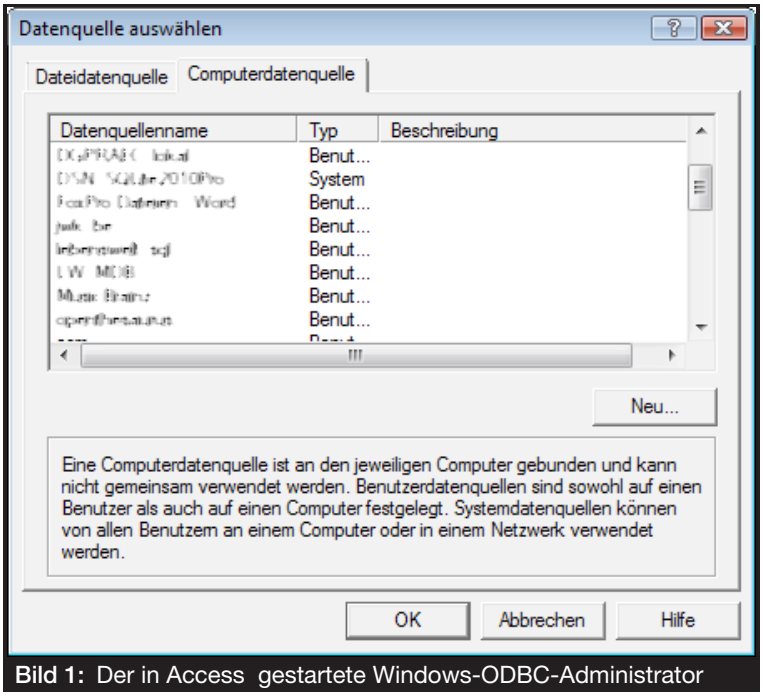


Bild 1: Der in Access gestartete Windows-ODBC-Administrator

Auch zu **SQLite** gibt es einen **ODBC-Treiber**, den **SQLite ODBC Driver**, über den Sie aus Access heraus mit der Datenbankdatei kommunizieren können. Dabei handelt es sich einen sehr speziellen Treiber: Der Server ist quasi bereits in ihn eingebaut! Sie benötigen also außer dem Treiber keine weiteren Komponenten, um auf **SQLite**-Datenbanken zuzugreifen oder solche zu erstellen. Sie finden das Treiber-Setup mit der aktuellen Version auch im Download zu diesem Beitrag.

Bild 2: Nur Benutzer-DSNs können in Access erstellt werden

Schreiten wir zur Tat!

ODBC-verknüpfte Tabellen

Der Download zum Beitrag enthält auch die **SQLite**-Datei **adressen.sqlite**, in der bereits vier Tabellen gespeichert sind. Wir machen uns nun an die Aufgabe, diese Tabellen in die Beispieldatenbank zu verknüpfen.

Voraussetzung ist die Installation des **ODBC-Treibers**, welche ganz unspektakulär verläuft, allerdings nur unter einem Administrator-Konto vollzogen werden kann. Ist dies geschehen, so klicken Sie im Ribbon unter dem Reiter **Externe Daten | Importieren und Verknüpfen** auf die Schaltfläche **ODBC-Datenbank**. Dies öffnet den Dialog zum Import oder zur Verknüpfung externer Datenbankelemente. Aktivieren Sie letztere Option. Mit Klick auf **OK** gelangen Sie zu einem Teil des Windows-**ODBC-Administrators** (siehe Bild 1), welcher bereits angelegte Datenquellen (**DSN**) auflistet, oder neue anzulegen gestattet. Eine **DSN** ist nichts weiter, als ein Set von Konfigurationsparametern, die einerseits den gefragten SQL-Treiber angeben, wie auch Einstellungen für diesen. Gespeichert werden kann dieses Set entweder als Datei (**Dateidatenquelle**), oder in der **Registry** (**Computerdatenquelle**). Entscheiden Sie sich für letzteres.

Da eine Verbindung zur **SQLite**-Datei noch nicht vorhanden sein dürfte, legen Sie über den Button **Neu...** nun eine solche an. Dies quittiert der **ODBC-Administrator** mit dem Hinweis aus Bild 2. Denn eigentlich lässt er sich, wie erwähnt, nur über administrative Rechte starten. Da Access aber nur mit normalen Benutzerrechten geöffnet wurde, ist seine Funktion eingeschränkt und erlaubt nur die Einrichtung einer **Benutzer-DSN**, welche im **CURRENTUSER**-Zweig der Registry abgelegt wird. Darüber informiert auch der zusätzliche eingeschobene Dialog in Bild 3.

Im weiteren Verlauf werden Sie nach dem Treiber und damit nach dem speziellen SQL-Server gefragt, der für die Verknüpfung zum Einsatz kommen soll (Bild 4). Bei korrekter Installation des Treiber sollten Sie hier in der Liste den **SQLite3 ODBC Driver** finden, den Sie markieren, bevor Sie mit **Weiter >** bestätigen.

Nun erscheint ein weiterer Dialog, der treiberspezifisch ist, grundsätzlich aber oben ein Eingabefeld für **Data Source Name** aufweist (siehe Bild 5). Vergeben Sie hier einen Namen für die **DSN**, wie etwa **Access Basics Demo**. Dieser Name wird dann im **ODBC-Administrator** in der Liste der **Computerdatenquellen** angezeigt werden. Sie können später auf diese **DSN** zugreifen, wenn Sie eine Verbindung zur **SQLite**-Datei wünschen.

Wir lassen die verschiedenen Parameteroptionen des **SQLite**-Treibers unbeachtet, aktivieren jedoch **Foreign Keys**, was ihm sagt, dass er **Fremdschlüssel** unterstützen soll. Dies macht bereits deutlich, dass **SQLite** ein **Relationales DBMS** ist, das Beziehungen zwischen Tabellen unterstützt, so, wie eben Access auch. Lösch- und Aktualisierungsweitergaben können damit realisiert werden.

Über den Button **Browse...** wählen Sie die Datei **adressen.sqlite** aus, welche sich aktuell im Verzeichnis der Beispieldatenbank befinden sollte, jedoch an beliebigem Ort untergebracht werden kann.

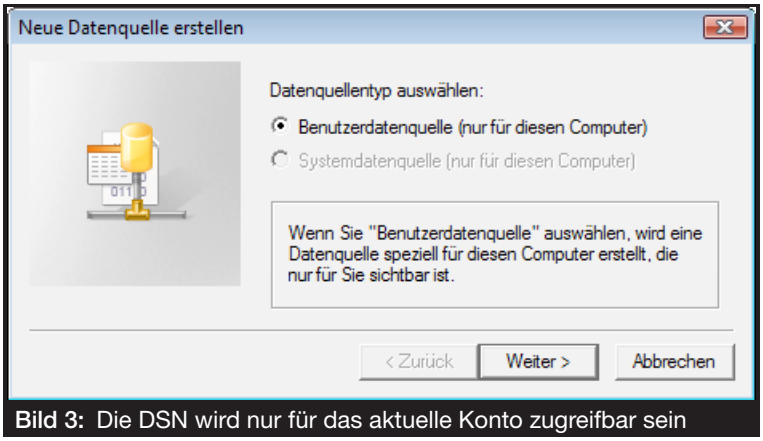


Bild 3: Die DSN wird nur für das aktuelle Konto zugreifbar sein

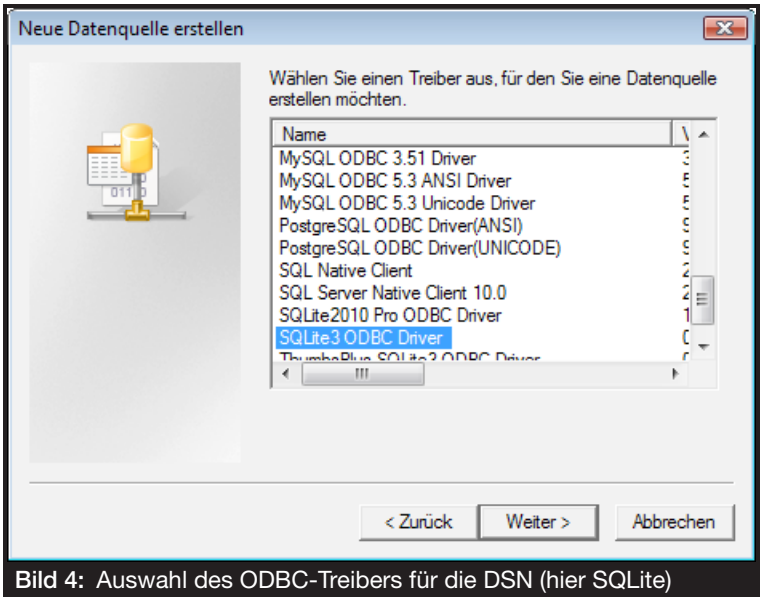


Bild 4: Auswahl des ODBC-Treibers für die DSN (hier SQLite)

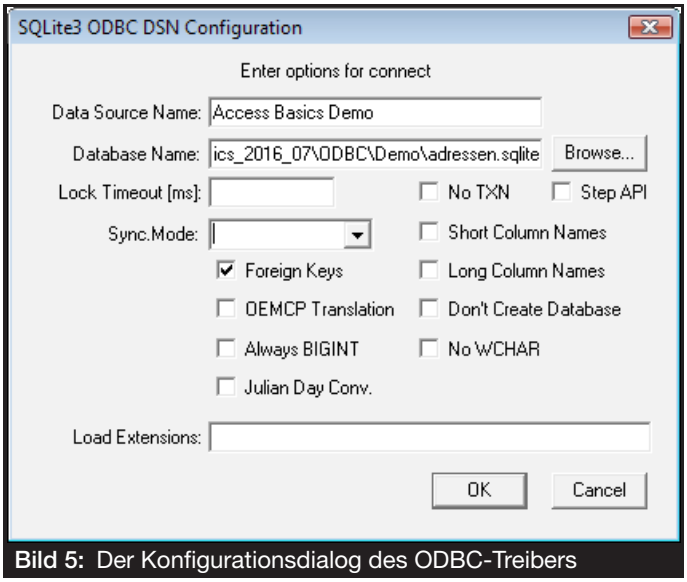


Bild 5: Der Konfigurationsdialog des ODBC-Treibers

Statt der Dateiauswahl können Sie in das Feld für **Database Name** aber auch manuell einen vollen Dateipfad einsetzen. Dann wird – das ist der Clou an diesem Treiber – beim ersten Ansprechen der **DSN** aus Access heraus automatisch die angegebene **SQLite**-Datei erzeugt! Aktivieren von **Don't Create Database** würde das verhindern. Die Endung, die Sie der Datei vergeben, spielt keine Rolle. Neben **sqlite** wird hier häufig ein **db**, **db3** oder einfach **dat** verwendet.

Wir möchten aber auf die bestehenden Tabellen zugreifen und lassen es deshalb bei der Auswahl der Datei **adressen.sqlite**. Nach dem Beenden des Treiberdialogs über **OK** würde der **ODBC-Administrator** normalerweise die Liste der **DSNs** anzeigen, in der nun die neue **DSN Access Basics Demo** verzeichnet wäre. Access geht hier aber schon einen Schritt weiter und öffnet, da **Verknüpfen** verlangt war, automatisch über den ODBC-Treiber die Datenbank, um sie genauso zu präsentieren, wie

Ribbon-Programmierung, Teil I

Mit Access 2007 sind die Menüleiste und die Symbolleisten von Access weggefallen. Die Befehle wurden stattdessen in das so genannte Ribbon übertragen. Das Ribbon bietet wesentlich mehr Möglichkeiten als seine Vorgänger, aber es ist auch aufwändiger zu programmieren. Dies gelingt nicht mehr einfach per VBA, sondern Sie müssen sich zusätzlich auch noch in XML einarbeiten – zumindest in die grundlegenden Techniken. Das ist aber kein Problem, denn wir zeigen Ihnen in dieser Artikelreihe alles, was Sie wissen müssen, um Ihren Anwendungen selbst definierte Ribbons hinzuzufügen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1607_Ribbon.accdb**.

Das Ribbon-Konzept

Das grundlegende Konzept zur Programmierung eigener Elemente für das Ribbon sieht vor, dass das Aussehen fast vollständig mit einem XML-Dokument beschrieben wird. Dieses enthält an manche Stellen sogenannte Callback-Attribute, für die Sie die Namen von VBA-Prozeduren hinterlegen können, die beim Eintreten der Ereignisse, die das jeweilige Callback-Attribut auslösen, aufgerufen werden sollen. Dies ist also etwa vergleichbar mit der Programmierung von Formularen und den durch die verschiedenen Aktionen ausgelösten Ereignisprozeduren. Mit dem kleinen Unterschied, dass Sie Formulare mithilfe einer praktischen Entwurfsansicht auf der Benutzeroberfläche zusammenklicken können, während Sie die Benutzeroberfläche im oberen Bereich des Access-Fensters, das Ribbon, komplett in XML formulieren müssen.

Es gibt zwar mittlerweile einige Tools, die einem diese Arbeit zumindest teilweise abnehmen, aber wir wollen Ihnen zunächst die Grundlagen zum selbst ausprobieren vermitteln.

Auf diese Weise können Sie später auch den von einem Tool erzeugten Code verstehen und gegebenenfalls nach Ihren Wünschen anpassen.

Die Tabelle USysRibbons

Wenn Sie eine Ribbon-Definition zur Anwendung bringen wollen, also etwa so, dass beim Start der Access-Datenbank alle eingebauten Elemente des Ribbons verschwinden und durch einen eigenen Befehl ersetzt werden, benötigen Sie eine Tabelle, in der Sie den Namen dieser Ribbon-Definition sowie die Definition selbst im XML-Format speichern. Der Inhalt dieser Tabelle wird beim nächsten Öffnen der Access-Anwendung abgefragt. Die Ribbon-Definitionen stehen dann über den Namen an verschiedenen Stellen in der Anwendung zur Auswahl:

- als Anwendungsribbon, also zur Anzeige mit dem Öffnen der Anwendung,
- als Formularribbon, das nach dem Öffnen eines Formulars angezeigt wird und mit diesem wieder geschlossen wird und
- als Berichtsribbon, das gleichzeitig mit der Anzeige eines Berichts eingeblendet wird.

Ribbons on the fly

Theoretisch besteht auch die Möglichkeit, Ribbons ad hoc per Code zusammenzustellen und diese anzuzeigen. Dies gelingt jedoch nur in Zusammenhang mit dem Öffnen eines Formulars, wie wir später in dieser Reihe zeigen werden.

Bilder im Ribbon

Wenn Sie Icons in Ihrem benutzerdefinierten Ribbon abbilden möchten, was empfehlenswert ist, um eine ergonomisch und optisch ansprechende Anwendung zu erhalten, benötigen Sie eine weitere Tabelle, die aber in den neueren Access-Versionen (ab 2010) standardmäßig in einer neuen **.accdb**-Datei vorhanden ist. Diese Tabelle heißt **MSysResources** und ist in der Regel nicht zu sehen, da diese als ausgeblendet markiert ist. Auch die Tabelle **USysRibbons** wird wegen ihres Präfix **USys...** nicht standardmäßig im Navigationsbereich angezeigt, sondern erst dann, wenn Sie

diese über die Access-Optionen einblenden. Außerdem benötigen Sie zur Anzeige von Bilddateien, die üblicherweise im .png-Format verwendet werden, noch einige Prozeduren, die wir aber in den Beispieldateien zur Verfügung stellen.

Ein erstes Ribbon

Beginnen wir jedoch mit den Basics, und zwar mit dem Anlegen einer einfachen Ribbon-Definition, die wir anschließend in der Anwendung sichtbar machen. An dieser Stelle kommt es darauf an, den grundlegenden Ablauf zu verinnerlichen. Später dann kommen wir auf fortgeschrittenere Ribbon-Techniken zu sprechen.

Tabelle USysRibbons anlegen

Als Erstes legen Sie eine neue Tabelle mit den folgenden Feldern an (siehe Bild 1):

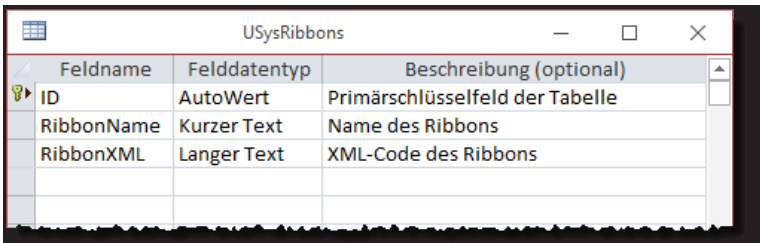


Bild 1: Entwurf der Tabelle **USysRibbons**

- **ID**: Primärschlüsselfeld der Tabelle
- **RibbonName**: Name des Ribbons, der später als Eigenschaft für die anzuzeigende Ribbon-Definition ausgewählt werden soll
- **RibbonXML**: XML-Definition des Ribbons

Speichern Sie diese Tabelle dann unter dem Namen **USysRibbons**. Wenn Sie die Tabelle nun schließen und wieder öffnen wollen, werden Sie feststellen, dass diese nicht im Navigationsbereich auftaucht. Um den Eintrag für diese Tabelle einzublenden, öffnen Sie die Navigationsoptionen, und zwar mit einem Rechtsklick auf den Titel des Navigationsbereichs. Aus dem nun erscheinenden Kontextmenü wählen Sie den Eintrag **Navigationsoptionen...** aus. Es erscheint der Dialog aus Bild 2, in dem Sie die Option **Systemobjekte anzeigen** aktivieren. Nach dem Schließen des Dialogs taucht dann auch die Tabelle **USysRibbons** im Navigationsbereich auf.

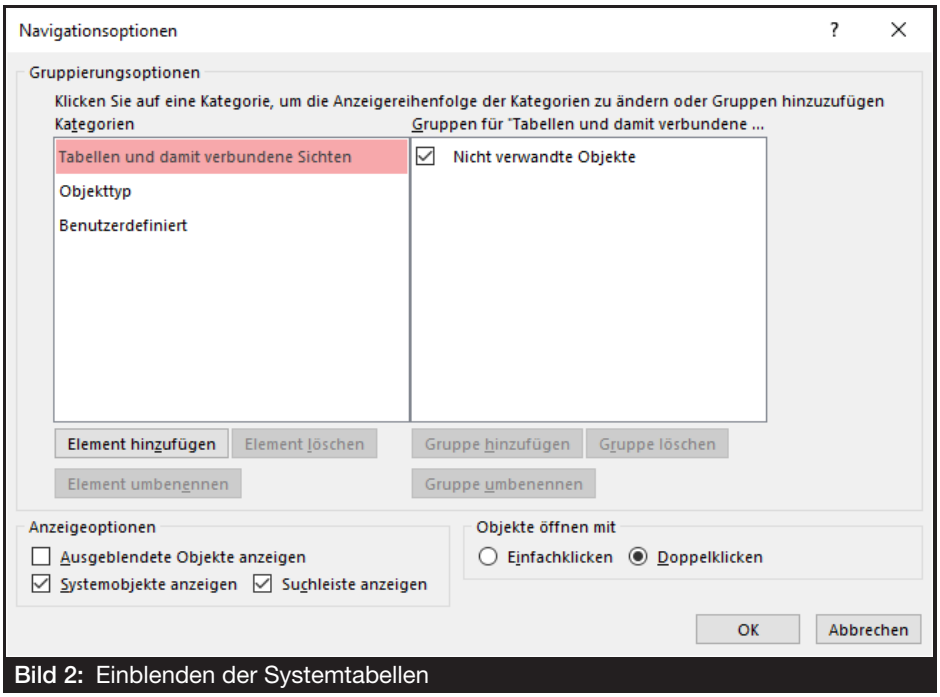


Bild 2: Einblenden der Systemtabellen

Ribbon-Definition für ein einfaches Ribbon

Eine der einfachsten Ribbon-Varianten besteht aus lediglich einem **tab**-Element, einem **group**-Element und einem **button**-Element. Das waren die sichtbaren Elemente, es gibt noch ein paar weitere, die nur im XML-Dokument auftauchen und für die Struktur verantwortlich sind. Geben Sie nun den folgenden Code in das Feld **RibbonXML** eines neuen Datensatzes der Tabelle **USysRibbons** ein und weisen Sie dem Feld **RibbonName** den Wert **Name** zu:

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.micro
    soft.com/office/2009/07/customui">
  <ribbon>
    <tabs>
      <tab id="tab1"
          label="Beispieltab">
        <group id="grp1"
            label="Beispielgruppe">
          <button
              label="Beispielbutton"
```

```
            id="btn1"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Die Tabelle sieht dann etwa wie in Bild 3 aus.

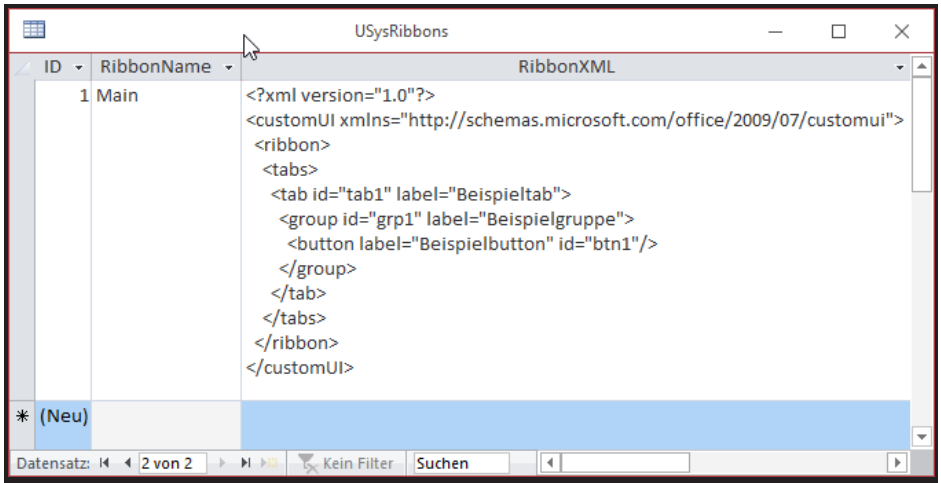


Bild 3: Die Ribbon-Definition in der Tabelle **USysRibbons**

Neustart, um USysRibbons einzulesen

Nun haben Sie die Ribbon-Definition in der Tabelle USysRibbons gespeichert. Damit Sie das so definierte Ribbon erstmals in Ihrer Datenbank sehen können, schließen Sie die Anwendung und öffnen diese neu. Dazu ist kein Neustart von Access nötig. Am schnellsten öffnen und schließen Sie die Datenbankdatei, wenn Sie diese komprimieren. Dies gelingt am einfachsten über die Schaltfläche **Datenbank komprimieren und reparieren** im Backstage-Bereich (ab Access 2010, siehe Bild 4).

Bild 4: Komprimieren der Datenbank, um diese zu schließen und wieder zu öffnen

Wenn Sie nun die Access-Optionen öffnen, finden Sie im Bereich **Aktuelle Datenbank|Menüband und Symbolleistenoptionen** die Eigenschaft **Name des Menübands**, wo nun der Name unseres Ribbons aus der Tabelle **USysRibbons** erscheint (siehe Bild 5).

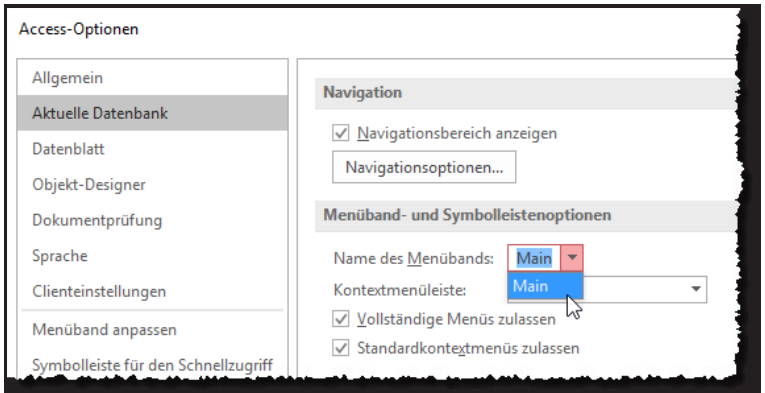


Bild 5: Auswahl des Ribbons beziehungsweise Menübands

Diesen Eintrag wählen Sie nun aus. Damit erscheinen die Elemente unserer Definition aber noch längst nicht im Ribbon, sondern wir müssen die Datenbankdatei erneut schließen und wieder öffnen. Nachdem wir dies erneut erledigt haben, findet sich rechts oben im Ribbon endlich das ersehnte tab-Element mit dem Titel **Beispieltab**. Ein Klick darauf zeigt dann auch die von uns definierte Gruppe **Beispielgruppe** sowie die Schaltfläche **Beispielbutton** an (siehe Bild 6).

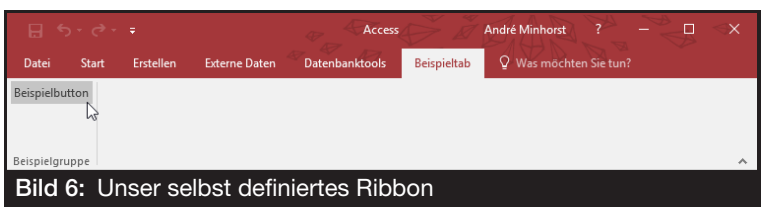


Bild 6: Unser selbst definiertes Ribbon

Shortcut

Es gibt eine Abkürzung, die einem das zweifache Schließen und Öffnen der Datenbankdatei erspart. Wenn Sie den Namen

Zugriff auf Server über ODBC, Teil II

Das manuelle Verknüpfen einer SQL-Server-Tabelle in eine Access-Datenbank über den ODBC-Administrator von Windows ist die eine Sache, das programmgesteuerte Beleben der Verbindung und das Neuverknüpfen über VBA eine andere! Zeigte die Juli-Ausgabe von ACCESS BASICS die Grundlagen zum Umgang mit ODBC-Backends, so erfahren Sie hier mehr über die Ansprache von ODBC-Verbindungen durch VBA-Programmierung.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1608_ODBC2.zip**.

ODBC-Tabellen neu verknüpfen
Die Aufteilung einer Datenbank in Logik und Daten kennen Sie bereits von reinen Access-Datenbanken: Das **Frontend** enthält nur die Oberfläche, also Formulare und Berichte, sowie den VBA-Code, und außerdem Abfragen. Das **Backend** ist eine separate Access-Datei, welche ausschließlich Tabellen beherbergt, welche in das Frontend verknüpft werden. Ebenso verhält es sich mit dem Gespann Access und **SQL-Server**, nur dass sich hier die Daten aus einem **DBMS-Server** speisen und die Tabellen über **ODBC-Verknüpfungen** in das Frontend gelangen.

Beim Verschieben eines Access-Backends etwa auf eine andere Maschine stimmen dann die Tabellenverknüpfung nicht mehr und müssen beim Start des Frontends angepasst werden, da Access in der Verknüpfung grundsätzlich absolute statische Pfade abspeichert. Ähnlich sieht es bei **ODBC-Verknüpfungen** aus. Der Ort des SQL-Servers oder der Tabellen auf ihm kann sich ändern, was eine Anpassung der Verknüpfungen im Frontend nach sich ziehen muss.

Egal, ob beim Access- oder ODBC-Backend, das Neuverknüpfen der Tabellen sollte ein Vorgang sein, der weitgehend automatisiert und möglichst ohne manuelle Interaktion vonstattengeht. Hier kommen Sie um VBA-Routinen nicht herum, die Sie etwa in Ausgabe **06/2011** **Tabellenverknüpfungen pflegen** finden.

Bei ODBC-Verknüpfungen verwenden Sie Code, der ganz ähnlich gestaltet ist. Eine Tabelle wird über ein **DAO-TableDef**-Objekt repräsentiert und dessen Eigenschaft **Connect** enthält die relevanten Informationen zum Backend-Ort. Verwenden Sie etwa diese VBA-Zeilen, um die **Connect**-Eigenschaft für die Tabelle **tblAdressen** auszulesen:

```
Set dbs = CurrentDb
? dbs.TableDefs("tblAdressen").Connect
```

Befände sich die Tabelle in einem Access-Backend, dann könnte das Ergebnis so aussehen:

```
;DATABASE=c:\users\fritz\test.accdb
```

Das erste Zeichen des **Connect**-Strings ist ein Semikolon, was darauf hinweist, dass vor ihm noch Anderes stehen könnte. Der Leer-String aber sagt Access, dass es sich hier um eine Access-Datei handeln muss, deren Pfad schließlich im Parameter **DATABASE** folgt.

Greift die Verknüpfung hingegen auf eine **ODBC-Tabelle** zu, so sieht das Ergebnis anders aus:

```
ODBC;DSN=Access Basics
Demo;Driver={SQLite3 ODBC
Driver};Database=c:\users\fritz\adres-
sen.sqlite;FKSupport=1; usw...
```

Hier steht an erster Stelle der Ausdruck **ODBC**, was Access darauf hinweist, die Tabelle über einen **ODBC-Treiber** zu finden. Die weiteren Angaben hierzu folgen im Anschluss.

Die eine Möglichkeit besteht dabei im Zugriff auf eine **DSN**, wie sie im ersten Teil des Beitrags manuell über den **ODBC-Administrator** angelegt wurde. Hinter dem Parameternamen **DSN** folgt dann der Name der angelegten **DSN** – hier: **Access Basics Demo**. Das würde an sich bereits ausreichen, denn in der **Benutzer-DSN**, welche ihrerseits alle Informationen in der Registry ablegt, stehen schon alle Angaben zu Treiber, Datenbankort und Zugriffsoptionen.

Die Alternative wäre der Verzicht auf eine zuvor angelegte **DSN**. Dann entfällt der **DSN-Parameter** im

Connect-String und alle Angaben zur ODBC-Verbindung müssen stattdessen in ihm angegeben werden. Das wäre der Teil ab **Driver**...:

```
ODBC;Driver={SQLite3 ODBC Driver};
Database=c:\users\fritz\adressen.sqlite; FKSupport=1;...usw.
```

Die aufeinanderfolgenden Parameternamen, wie **Database**, **FKSupport**, et cetera, sind keine, die Access selbst kennt! Sie unterscheiden sich je nach gewähltem **ODBC-Treiber**, der sie seinerseits kennt. Nur der **SQLite**-Treiber etwa weiß, dass **Database** der Ort der **SQLite**-Datei ist, die er laden soll, und dass **FKSupport** die Option zur Unterstützung von Fremdschlüsseln ist.

Wie kommen Sie an diese Parameternamen und deren Bedeutung? Hier bleibt Ihnen nur die Dokumentation zum Treiber auf dessen Herstellerseite oder die Recherche etwa auf **connectionstrings.com**, einer Seite, die zu allen gängigen **DBMS** die **ODBC-Connect-Strings** mehr oder weniger kommentiert auflistet.

Stimmen Parameternamen oder -werte nicht, so ignorieren ODBC-Treiber diese in der Regel. Soweit möglich, kommt dennoch eine Verbindung zustande und der Treiber setzt **Default-Werte** ein.

Fehlen relevante Angaben, so öffnet sich der jeweilige Treiber-Dialog des **ODBC-Administrators** und verlangt nach manueller Eingabe. Unter Umständen meldet Access aber auch einfach: **Kann keine ODBC-Verbindung herstellen**, oder ähnlich.

Im Falle des **SQLite-Treibers** und dem nebenstehenden **Connect**-String kommt es zur Fehlermeldung **-7778** mit dem Text: **Reservierter Fehler; es gibt keine Beschreibung für diesen Fehler**. Microsoft macht wenig erhellende Angaben zu diesem Fehler. Er kann verschiedene Ursachen haben, die in Authentifizierungsproblemen begründet scheinen. Wie auch immer, ohne Angabe einer DSN funktioniert ein SQLite-Connect-String nicht.

Glücklicherweise legt der ODBC-Treiber für **SQLite** bei Installation automatisch eine **System-DSN** an, die immer den Namen **SQLIte3 Datasource** aufweist. Diese **DSN** enthält lediglich Angaben zum Treiber, nicht aber zum Ort einer **SQLite**-Datenbankdatei. Tatsächlich kann man nun im **Connect-String** beides kombinieren: den **DSN-Namen** und die **Optionsparameter** im Anschluss! Demgemäß hat der **Connect-String** für **SQLite** immer diesen Anfang:

```
ODBC;DSN=SQLite3 Datasource;
Driver={SQLite3 ODBC Driver};...
```

Im Folgenden ergänzen Sie die weiteren Angaben, wovon der Ort der Datenbank mit **Database** die wichtigste ist.

Mit diesem Wissen im Gepäck können Sie eine ODBC-Tabellenverknüpfung nun per VBA ändern, indem Sie die Connect-Eigenschaft des zugehörigen TableDef-Objekts neu einstellen. Listing 1 zeigt eine Routine aus der Beispieldatenbank, die das demonstriert. In der String-Vari-

```
Sub ChangeConnectTable(Optional ViaConnect As Boolean = True)
    Dim sConnect As String
    Dim dbs As Database
    Dim tdf As TableDef
    Dim prp As DAO.Property

    sConnect = "ODBC;DSN=SQLite3 Datasource;" & _
               "Driver={SQLite3 ODBC Driver};" & _
               "Database=" & CurrentProject.Path & "\adressenneu.sqlite;" & _
               "LongNames=0;ShortNames=0;NoWCHAR=0;PWD=Admin;FKSupport=1;" & _
               "Timeout=500;NoTXN=0;OEMCP=1;BigInt=0;JDConv=0;StepAPI=0;" & _
               "NoCreat=1;SyncPragma=FULL;"
    Debug.Print sConnect
    If ViaConnect Then
        Set dbs = CurrentDb
        Set tdf = dbs.TableDefs("tblAdressen")
        tdf.Connect = sConnect
        tdf.RefreshLink
        dbs.TableDefs.Refresh
        Set tdf = Nothing
    Else
        DoCmd.TransferDatabase acLink, "ODBC Database", sConnect, acTable, _
            "tblAdressen", "tblAdressen"
        DoCmd.DeleteObject acTable, "tblAdressen"
        DoCmd.Rename "tblAdressen", acTable, "tblAdressen1"
    End If
End Sub
```

Listing 1: Neuverknüpfen einer ODBC-Tabelle über eine VBA-Prozedur

ablen **sConnect** wird der **Connect-String** zusammengebaut. Er besteht aus dem erwähnten Beginn für den SQLite-Treiber und allerlei Optionsparametern, auf deren Bedeutung wir hier nicht weiter eingehen. Zwischendrin wird der Pfad zur SQLite-Datei über den Parameter **Database** eingebracht, wobei sich dieser aus dem Verzeichnis des Frontends (**CurrentProject.Path**) und der SQLite-Datei **adressenneu.sqlite** bildet.

Im weiteren Verlauf – der Zweig nach **If ViaConnect** – setzt die Prozedur die Objektvariable **tdf** auf das **TableDef**-Objekt zur Tabelle **tblAdressen**. Deren **Connect**-Eigenschaft wird nun mit dem neu gebildeten String aus **sConnect** bestückt. **RefreshLink** ist eine Methode, die Access anweist, die **Connect**-Eigenschaft auszuwerten und die Verbindung zum Backend neu herzustellen. **TableDefs.Refresh** führt sicherheitshalber dazu, dass die Tabelle auch im Navigationsbereich korrekt dargestellt wird. Nach Durchlaufen des Codes ohne Fehlermeldung können Sie die Tabelle als Datenblatt öffnen und haben damit die Daten aus **adressenneu.sqlite** vor sich.

ViaConnect als optionaler Parameter ist in der Routine standardmäßig auf **True** gesetzt. Er bedeutet, dass zum Neuverknüpfen das **TableDef**-Objekt und seine **Connect**-Eigenschaft zum Einsatz kommen sollen. Es gibt jedoch noch eine alternative Methode.

TransferDatabase

Diese Methode des **DoCmd**-Objekts vollbringt das gleiche, wie die Änderung von **TableDef.Connect**.

Ist in der Routine aus Listing 1 der Parameter **ViaConnect** auf False gesetzt, so wird der zweite Teil des Bedingungszweigs angesprungen. **TransferDatabase** dient entweder zum Importieren oder zum Verknüpfen von Fremdtabellen. Die Liste der Parameter und ihrer Bedeutung entnehmen Sie der Access-Hilfe, indem Sie die Methode im Code markieren und **F1** drücken.

acLink sagt Access, dass die Tabelle verknüpft werden soll. Handelt es sich um die über ODBC angesprochene Tabelle eines SQL-Servers, so hat darauf als Typ der Ausdruck **ODBC Database** zu folgen. Anschließend übergeben Sie den gleichen **Connect-String** in **sConnect**, wie beim **TableDef**-Objekt. Zusätzlich definieren Sie mit **acTable**, dass es hier um eine Tabelle handelt, deren Namen auf dem SQL-Server (**tblAdressen**) sie als weiteren Parameter angeben. Schließlich können Sie mit dem letzten Parameter bestimmen, wie die Tabelle unter Access heißen soll. Dieser dann im Navigationsbereich angezeigte Name kann sich beliebig von dem auf dem SQL-Server unterscheiden!

Der Haken an der Geschichte: **TransferDatabase** modifiziert eine bestehende Tabellenverknüpfung nicht, sondern legt eine neue an! Da die Tabelle **tblAdressen** im Frontend bereits existiert, legt Access eigenmächtig einen anderen Namen an, nämlich **tblAdressen1**, statt die Verknüpfung zu überschreiben. Aus diesem Grund löscht die folgende Codezeile über die Methode **DeleteObject** von **DoCmd** die alte Tabellenverknüpfung und benennt nun die neue per **Rename** wieder in **tblAdressen** um.

Zum Ändern einer ODBC-Verknüpfung eignet sich also das **TableDef**-Objekt besser, zum neuen Anlegen einer Verknüpfung die Methode **TransferDatabase**.

ODBC-Tabellen

Mit den ODBC-verknüpften Tabellen können Sie im Frontend genauso arbeiten, wie mit normalen Access-Tabellen. Also etwa Formulare, Berichte und Abfragen auf ihnen basieren lassen. Das Anzeigen von Daten gelingt praktisch immer problemlos. Beim Ändern oder Hinzufügen von Datensätzen kann es hingegen manchmal zu Aktualisierungsfehlern kommen. Ob und wann dies geschieht, hängt vom verwendeten SQL-Server und dem Aufbau seiner Tabellen ab. Generell lässt sich sagen, dass diese Fehler reduziert werden, wenn jede Tabelle einen eindeutigen Primärschlüssel aufweist. Zusätzlich macht es sich gut, jeder Tabelle ein sogenanntes **Timestamp-Feld** zu spendieren. Diesen Datentyp kennen Access-Tabellen selbst nicht, jedoch fast alle anderen **DBMS**. Es handelt sich um einen speziellen Datumstyp, den der SQL-Server wie einen **Autowert** immer mit dem augenblicklichen Datum belegt, sobald ein Datensatz abgespeichert wird. Indizieren Sie das Feld auf dem SQL-Server besser auch noch eindeutig. Access interpretiert beim Verknüpfen der Tabelle diesen Typ korrekt als Datum. Verwenden Sie das Feld jedoch nicht aktiv in Formularen oder Abfragen!

PassThrough-Abfragen

Abfragen, die ODBC-verknüpfte Tabellen benutzen, lassen sich auf gewohnte Weise im **Abfrageneditor** anlegen und bearbeiten. Der manchmal geäußerte Einwand, dass damit die Performance des SQL-Server beeinträchtigt würde, weil Access erst alle Daten der Tabellen laden müsse, um sie anschließend zu filtern und zu verknüpfen, stimmt so nicht. Die **Database Engine** von Access ist intelligent genug, um intern die Abfrage so in **SQL-Statements** umzuwandeln, dass auch der SQL-Server diese versteht. Tatsächlich setzt Access bereits beim Öffnen eines Tabellendatenblatts ein **SELECT**-Statement an den SQL-Server ab, der dieses verarbeitet und das Ergebnis zurückschickt.

Nicht immer jedoch ist das Ergebnis optimal, weil andere **DBMS** vor allem mit Tabellenverknüpfungen in Abfragen (**Joins**) anders umgehen. Zudem kennen SQL-Server einen weitaus größeren Schatz an SQL-Anweisungen, von denen Access nur träumen kann. Diese können über den visuellen Abfrageneditor von Access nicht verwendet werden. Deshalb gibt es mit den sogenannten **Pass-Through-Abfragen** eine Möglichkeit, den Dialekt eines SQL-Servers direkt anzusprechen.

Um eine neue solche Abfrage anzulegen, klicken Sie im **Ribbon** unter **Erstellen** auf **Abfrageentwurf**. Fügen Sie keine Tabelle aus dem nun erscheinenden Dialog hinzu, sondern betätigen die Schaltfläche **Schließen**. Anschließend klicken Sie auf den **Ribbon-Button Pass-Through** unter dem Tab **Abfragetyp**. Automatisch schaltet sich dadurch der Editor in die **SQL-Ansicht**. Auch das Eigenschaftenblatt ändert seine Parameterliste. Der wichtigste Eintrag lautet hier ODBC-Verbindung, wobei hier bereits der String ODBC; voreingestellt ist.

Sie haben die Wahl, diesen String manuell zu ergänzen, oder über die Schaltfläche mit den Pünktchen, die erscheint, wenn das Eingabefeld den Fokus besitzt, eine **DSN** zu ermitteln. Sie öffnet den **Windows ODBC-Administrator** zur Auswahl oder zum Neuanlegen einer **DSN**. Da wir möglichst ohne eine **DSN** auskommen möchten, füllen wir das Feld mit der Angabe des gleichen **Connect**-Strings, der schon beim Neuverknüpfen der ODBC-Tabellen zum Einsatz kam.

Bild 1 zeigt exemplarisch eine **Pass-Through-Abfrage** mit ihren Eigenschaften. Das simple SQL-Statement, welches an die **SQLite**-Datenbank gesandt werden soll, lautet **VACUUM**. Das ist die Anweisung an den ODBC-Treiber, die **SQLite**-Datei zu komprimieren und aufzuräumen. Die Anweisung gleicht ziemlich genau der Access-Funktion **Komprimieren und Reparieren**.

In **ODBC-Verbindung** ist der **Connect-String** eingetragen, den wir früher bereits für die Tabellenverknüpfung ermittelten. **Liefert Datensätze** stellen Sie auf **Nein** ein, weil die Anweisung **VACUUM** ja kein Ergebnis zurückgibt. Wollten Sie eine Tabelle über ein **SELECT-Statement** abfragen, so müsste diese Position natürlich auf **Ja** stehen.

Bei den weiteren Parametern können Sie es getrost bei den voreingestellten Werten belassen. **Meldungen protokollieren** hat nur bei der Entwicklung zur Fehleranalyse eine Bedeutung. Die **ODBC-Wartezeit** ist, vor allem bei **SQLite**-Datenbanken, ohne Belang. Liefert der SQL-Server nach der angegebenen Zeit kein Ergebnis, so bricht Access die Ausführung der Abfrage ab. **Max Datensätze** begrenzt

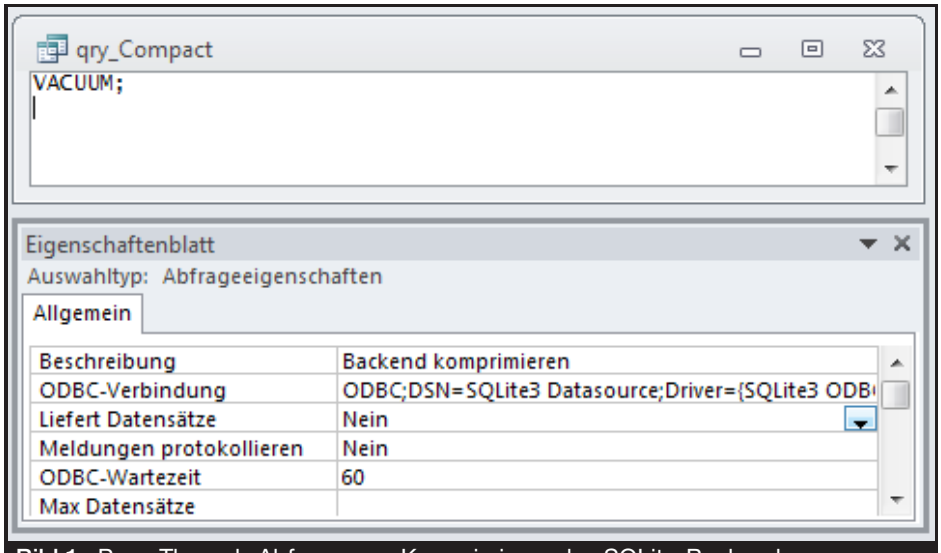


Bild 1: Pass-Through-Abfrage zum Komprimieren des SQLite-Backends

Lokaler Webshop

Nur wenige Web-Applikationen kommen ohne eine zugrundeliegende Datenbank aus. Ob CMS-Systeme oder eCommerce-Anwendungen, alle speichern unter der Haube ihre Daten in DBMS-Servern, wie etwa MySQL. Wie stellen in dieser Ausgabe einen unter Access nachgebildeten Shop vor, der ähnliche Features aufweist, wie seine Vorbilder im Internet. Zwar nützt eine solche Anwendung reell wohl für nur wenige Einsatzbereiche, dafür aber lässt sich an ihr eine Menge demonstrieren.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1608_Webshop.accdb**

Demo-Webshop

Vorstellbar wäre etwa der Einsatz auf Messe- oder Verkaufsstandrechnern, wo Kunden nicht vorrätige Artikel über den lokalen Shop beziehen könnten. Weniger auf dem Praxiseinsatz liegt unser Augenmerk hier jedoch, als vielmehr auf der Darstellung des Datenmodells und dem, was sich aus ihm herausholen lässt.

Unsere einfache alte Kundendatenbank hat zunächst ausgesorgt. Dennoch bildet sie die Basis für die Kundenverwaltung des Shops. Angedockt ist an sie eine Bestellverwaltung, wie sie ebenfalls schon mehrfach in unseren Beiträgen verwendet wurde und in der ehrwürdigen Nordwind-Datenbank ihr Vorbild hat. Neuer hingegen ist ein Teil zur Verwaltung von Artikeln und Produktkategorien. Wie auch immer: Das Hauptformular der Anwendung zeigt sich, wie in Bild 1, in dem Hardware-

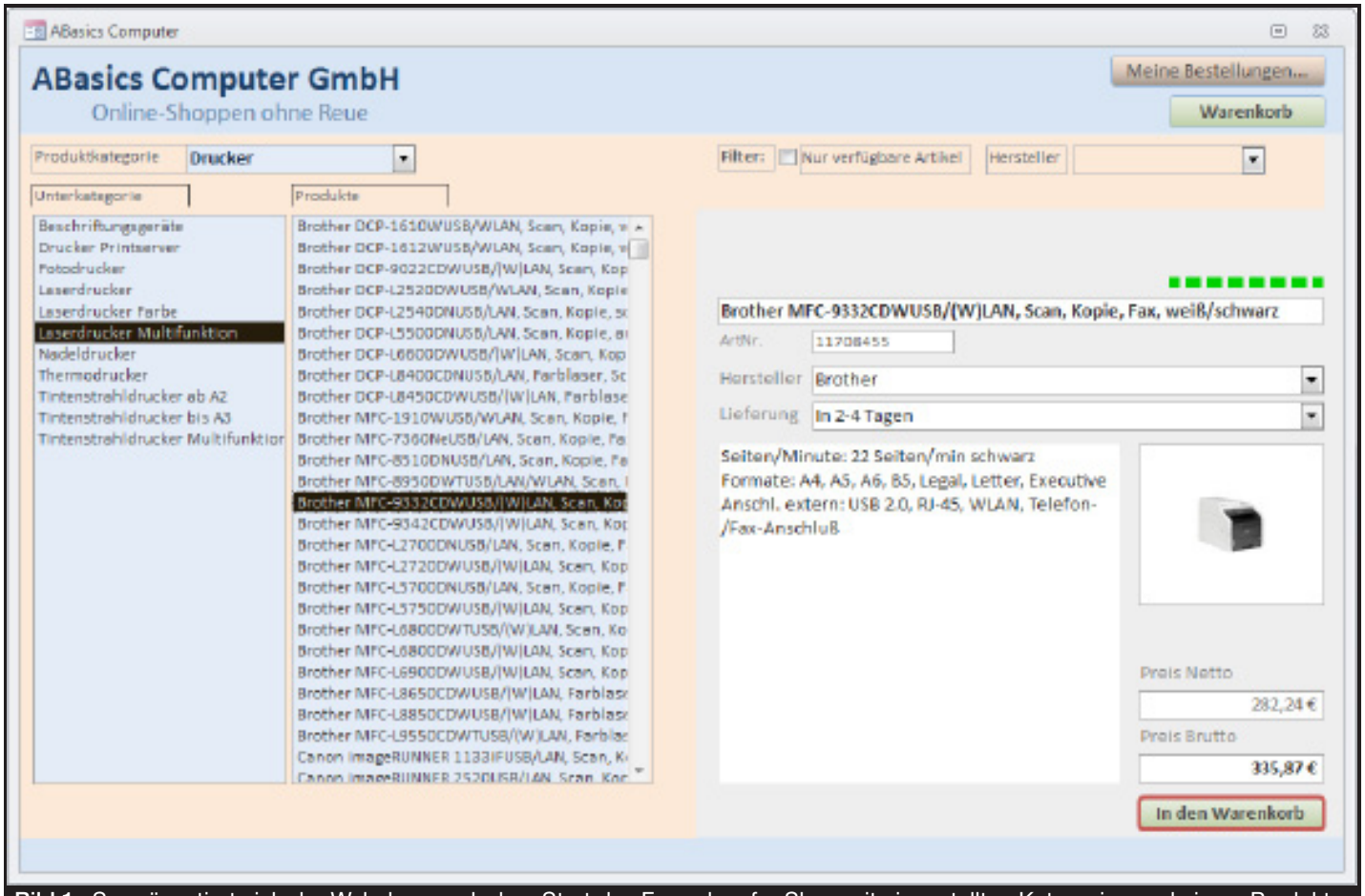


Bild 1: So präsentiert sich der Webshop nach dem Start des Formulars frmShop mit eingestellten Kategorien und einem Produkt

Produkte über die virtuelle Firma **ABasics Computer GmbH** bezogen werden können. Hier wählen Sie Artikel über das Kombinationsfeld mit den Hauptkategorien oben links und weiter mit dem Listenfeld links und dessen Produktunterkategorien aus. Die Spezifikation des Produkts inklusive eines Vorschaubilds wird dann rechts im Formular eingeblendet. Der Artikel kann schließlich über den grünen Button rechts unten in den Warenkorb gelegt werden. Auf den weiteren Bestellvorgang kommen wir später noch zu sprechen.

Selbstverständlich ist dies nur ein stark vereinfachtes Muster, denn echte Webshop-Systeme weisen meist über hundert verknüpfte Tabellen auf und stellen in der Regel ausgewachsene Warenwirtschaftsanwendungen dar. Dennoch zeigt sich hier das Prinzip. Die Demodatenbank enthält übrigens über 10.000 existierende Artikel, die zumeist mit Vorschaubildern abgespeichert sind. Auf größere Produktbilder musste verzichtet werden, da dies den Download auf einige Gigabyte vergrößert hätte.

Datenmodell des Shops

In Bild 2 sind die Beziehungen zwischen sämtlichen Tabellen der Datenbank dargestellt. Ganz links finden sich die bekannten Kundentabellen im hellblauen Bereich. Die daran angeflanschten Bestelldaten sind gelb hinterlegt. Rechts finden Sie die zu Artikeln gehörigen Tabellen.

Die Kundentabelle **tblKunden** hat nun ein zusätzliches Feld **Login** bekommen. In diesem Textfeld wird für einen im Shop registrierten Kunden dessen Passwort gespeichert. Die Verbindung von Kunden zu Bestellungen passiert über das Schlüsselfeld **KundeID** der Tabelle **tblBestellungen** über eine 1:n-Beziehung mit Referenzieller Integrität. Das Löschen von Kunden zieht damit automatisch das Entfernen der zugehörigen Bestelldatensätze nach sich.

Bestellungen

Eine Bestellung wird über die vier Felder der Tabelle **tblBestellungen** identifiziert. Ihr Primärschlüssel **ID** (**Autowert**) dient gleichzeitig als **Bestellnummer**. Das **Bestelldatum** gibt den Zeitpunkt der Bestellaufgabe wieder. **IDStatus** bezieht sich auf ihren Bearbeitungsstatus, dessen Text aus der Nachschlagetabelle **tblBearbeitungsstati** kommt.

Die möglichen Werte sind hier:

```
0 - Bestellung noch nicht abgeschickt
1 - Bestellung aufgegeben
2 - Bestellung versandt
3 - Bestellung abgeschlossen
```

Der Vorgabewert **0** bedeutet dabei, dass eine Bestellung angelegt wurde, indem Artikel temporär in den Wa-

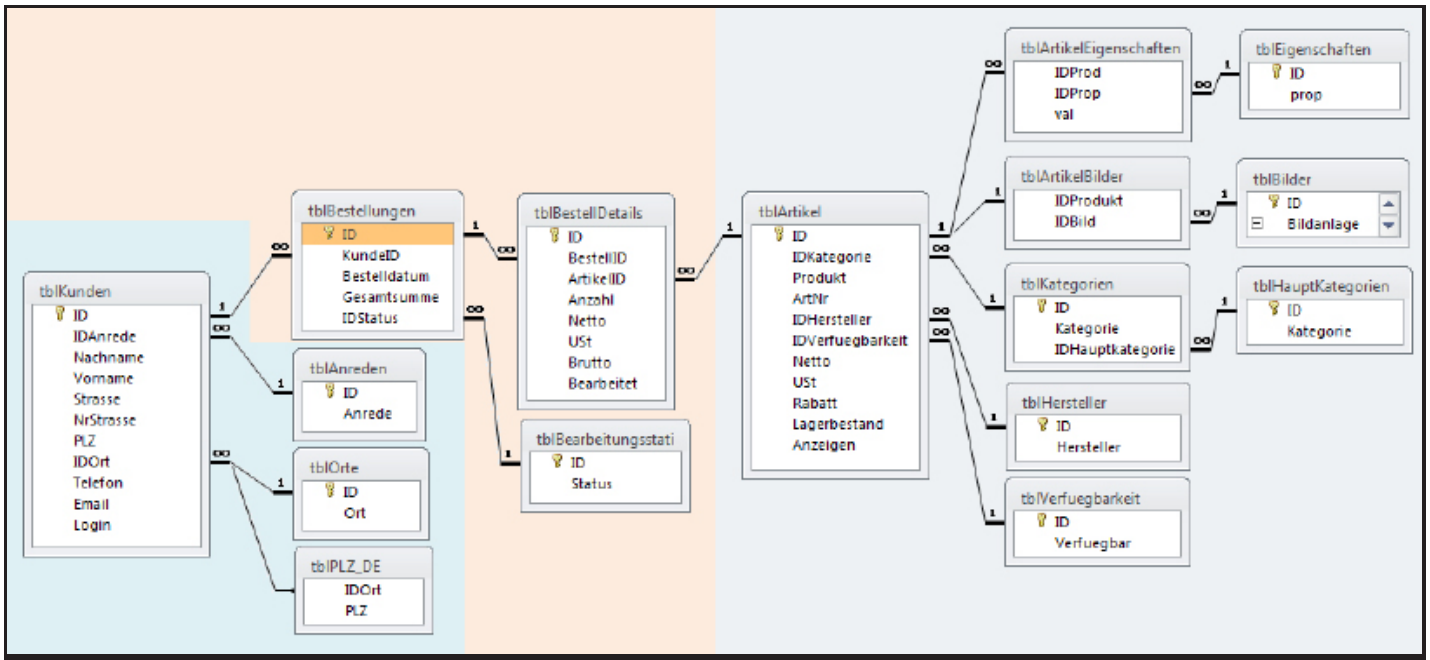


Bild 2: Layout des von einer Bestelldatenverwaltung abgeleiteten neuen Datenmodells des Webshops im Beziehungsfenster

renkorb gelegt wurden, dieser jedoch noch nicht zur Kasse gelangte.

Nicht ganz einleuchtend mag das Feld **Gesamtsumme** sein, welches den Bruttogesamtbetrag aller Artikel enthält. Tatsächlich ergibt sich dieser ja aus den Artikelpreisen, deren Anzahl und Steuersätzen. Das wiederspricht scheinbar den Regeln der **Normalisierung** von Datenbankmodellen. Tut es auch, doch ganz so akribisch muss man nicht vorgehen. Denn bei der Berechnung etwa von Umsatzstatistiken fallen die benötigten Abfragen einfacher aus, weil dafür nun die eine Tabelle **tblBestellungen** ausreicht. Im Interesse erhöhter Performance ist so eine Denormalisierung deshalb durchaus statthaft.

Die einzelnen Artikel einer Bestellung listet die **1:n-Tabelle tblBestelldetails** auf. Die **BestellID** jedes Datensatzes bezieht sich auf die Bestellnummer **ID** in **tblBestellungen**. Auch hier ist Referenzielle Integrität mit Lösch- und Aktualisierungsweitergabe eingestellt. Das Löschen einer Bestellung löscht damit auch alle bestellten Positionen. Und damit führt das Löschen eines Kunden automatisch sowohl zum Entfernen all seiner Bestellungen und zusätzlich aller zugehörigen Bestellpositionen. Um verwaiste Datensätze müssen Sie sich deshalb nicht sorgen.

Die **ArtikelID** verweist auf einen der Shop-Artikel in der Tabelle **tblArtikel**. Die Beziehung zwischen beiden ist ebenfalls referenziell, jedoch ist hier keine Löschweitergabe festgelegt. Schließlich soll das Entfernen eines Artikels aus dem Produktsortiment ja nicht die auf diesen bezogenen Bestellungen durcheinanderbringen. Die Referenzielle Integrität sorgt hier im Gegenteil dafür, dass ein Artikel gar nicht gelöscht werden kann, wenn mit ihm Bestellungen verbunden sind. Die Datenbank-Engine verhindert dies nun automatisch und gibt gegebenenfalls einen entsprechenden Hinweis aus.

Zu einer Bestellposition gehört weiterhin die gewünschte **Anzahl** des Artikels. Sein Preis wird im Währungsfeld **Netto** verewigt, die angesetzte Umsatzsteuer im **Double**-Feld **Ust**, das als Prozentzahl formatiert wird. Auch hier scheint mangelhafte Normalisierung vorzuliegen, denn die gleichen Felder finden sich auch beim Produkt in der Tabelle **tblArtikel** wieder. Dem ist jedoch nicht so! Denn die Produktpreise werden ja fortwährend dem Markt angepasst. Entnähme man die Preise lediglich der Tabelle **tblArtikel**, so wiese die Rechnung nach Warenversand eventuell falsche Preisangaben aus. Die Preise müssen daher an dieser Stelle fest abgespeichert werden, damit sie die gleichen Werte haben, wie zum Bestelldatum, was dann sowohl in der Rechnung zum Ausdruck kommt, wie auch in der Bestellübersicht eines Kunden.

Das Feld **Brutto** ist eigentlich überflüssig, vereinfacht jedoch Abfragen auf die Tabelle. Es handelt sich bei ihm nicht um ein Eingabefeld, sondern um das in **Access 2010** eingeführte **Berechnete Feld**. Hier ist eine Formel hinterlegt, die sich in Bild 3 aus der Feldeigenschaft **Ausdruck** ergibt. Als Ergebnistyp ist **Double** eingetragen. Für absolut korrekte Summenbildung in Abfragen wäre dieser Typ allerdings auf Währung zu ändern.

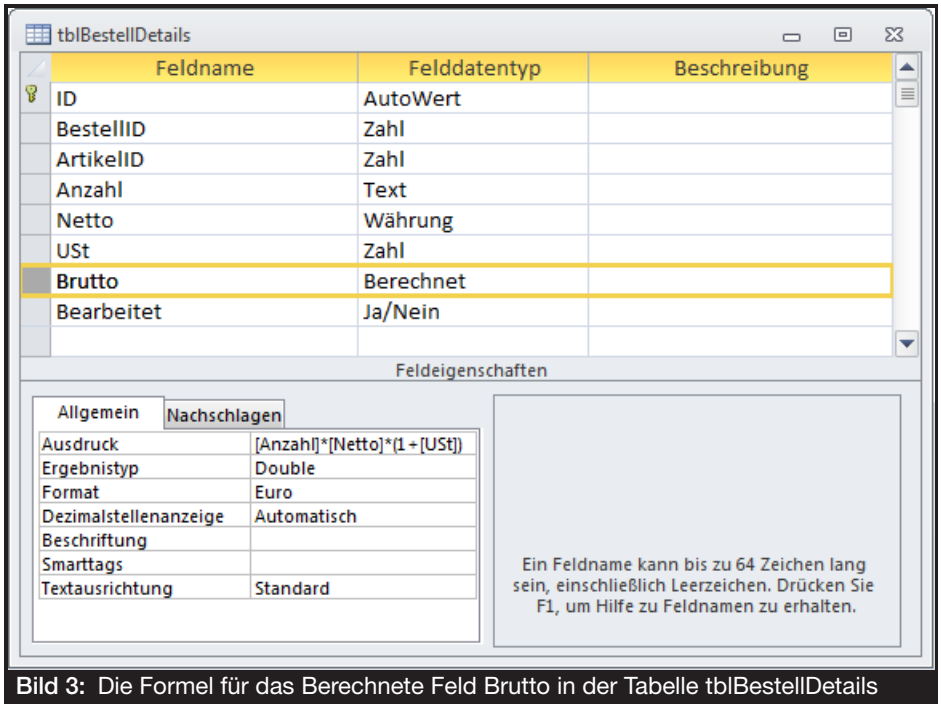


Bild 3: Die Formel für das Berechnete Feld Brutto in der Tabelle tblBestellDetails

Bliebe noch das Ja/Nein-Feld Bearbeitet. Dieses bekommt der Kunde im Shop nicht zu Gesicht, denn hier handelt es sich um ein Hilfsfeld für den Vertrieb. Beim Packen einer Bestellung setzt der Bearbeiter für die jeweilige Bestellposition den Wert in einem noch zu besprechenden Bearbeitungsformular auf Ja, um die Übersicht zu behalten.

Produktkategorien

Bevor wir zu Artikeln und deren Detailtabellen kommen, seien noch die Kategorien des Shops erläutert. Sie teilen sich in die zwei Tabellen **tblKategorien** und **tblHauptkategorien** auf. Eine Hauptkategorie wären zum Beispiel **Drucker**. Deren Unterkategorien wäre **Laserdrucker**, **Multifunktionsdrucker**, **Tintenstrahldrucker**. Letztere verzeichnet die **tblKategorien**, wobei hier zu jedem Datensatz die **ID** der Hauptkategorie im Feld **IDHauptkategorie** abgespeichert ist. In der Artikeltabelle wiederum verweist der Long-Wert **IDKategorie** auf einen Datensatz in **tblKategorien**.

Im wahren Leben käme diese Kombination eher nicht zum Einsatz. Stattdessen verwendete man eine **Rekursive Tabelle**, die sowohl über- wie untergeordnete Kategorien enthielte. Eine **Eltern-ID** eines Datensatzes bezöge sich dann auf einen weiteren Datensatz der Tabelle. Derlei wurde in Ausgabe **09/2105** ausführlicher dargestellt. Der Vorteil wäre eine beliebige Schachtelungstiefe von Kategorien, wie diese auch tatsächlich in vielen Webshops anzutreffen ist. Der Nachteil ist die wesentlich komplexere Auswertung solcher Tabellen und deren Darstellung in der Benutzeroberfläche. Deshalb wurde für unsere Demo davon Abstand genommen.

Artikel

Der Primärschlüssel **ID** der Tabelle **tblArtikel** ist ein **Autowert**, auf den sich eine Bestellposition in **tblBestelldetails** bezieht. Zu den weiteren Features eines Artikels in **tblArtikel** zählen seine Bezeichnung im Feld **Produkt**, die Artikelnummer in Textfeld **ArtNr**, sein Nettopreis im Währungsfeld **Netto** und die vorgeschriebene Mehrwertsteuer im Prozentfeld **Ust**. Daneben kann für ein Produkt auch ein **Rabatt** im gleichnamigen Prozentfeld angesetzt werden. Der Standardwert beträgt hier **0**. Welche Menge des Artikel sich im Lager befindet, verzeichnet das **Long**-Feld **Lagerbestand**. **Anzeigen** ist ein **Ja/Nein-Feld**, welches nur zu Verwaltung des Shops gedacht ist. Der Standardwert **True** führt dazu, dass der Artikel im Shop aufgelistet und angezeigt wird. Ein Filter in den dafür vorgesehenen Datenquellen sorgt hierfür.

Einige Nachschlagefelder und deren Tabellen vervollständigen den Entwurf. Die **ID** zu einer Produktkategorie in **tblKategorien** hatten wir ja schon. IDHersteller verweist auf einen Datensatz der Tabelle **tblHersteller**. Und **IDVerfuegbarkeit** steht in Beziehung zur Nachschlagetabelle **tblVerfuegbarkeit**, welche folgende Datensätze enthält:

```
-3      (leer)
-2      Aktuell nicht lieferbar
-1      Sofort (Auf Lager)
0       In 2-4 Tagen
1       In 5-7 Tagen
2       In 8-14 Tagen
3       Kann bestellt werden
```

Der Standardwert steht in tblArtikel zunächst auf **-1**. Jeder neu eingefügte Artikel bekommt damit die Eigenschaft **Sofort (Auf Lager)**.

Nun soll jeder Artikel auch ein Vorschaubild zeigen. Dieses könnte man als **Anlagefeld** direkt in die Tabelle **tblArtikel** aufnehmen. Tatsächlich verwenden Webshops häufig aber ein und dasselbe **Thumbnail** mehrfach für unterschiedliche Produkte. Das machen auch wir.

Die Bilder sind in die Tabelle **tblBilder** ausgelagert, die neben dem Autowert-Primärschlüsselfeld **ID** noch das Anlagefeld **Bildanlage** aufweist. Die Verbindung zwischen Artikel und Bild kommt nun jedoch nicht direkt zustande – man könnte dies durchaus tun! –, sondern über die **n:m-Tabelle tblArtikelBilder**. Damit können einem Artikel optional auch mehrere Bilder zugeordnet werden. **IDProdukt** verweist dabei auf den Artikel, **IDBild** auf die Anlage.

Ähnlich verhält es sich mit den Produkteigenschaften. Die Liste der Eigenschaften befindet sich in der Tabelle **tblEigenschaften** (Auszug in Bild 4).

Die Verknüpfungstabelle **tblArtikelEigenschaften** verweist mit **IDProd** einerseits auf die **ID** eines Artikels in **tblArtikel**, andererseits mit **IDProp** auf eine Produkteigenschaft. Im Unterschied zum Entwurf mit den Vorschaubildern findet sich hier jedoch

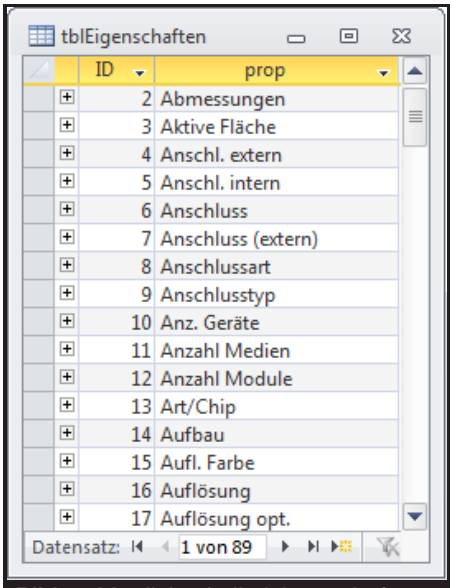


Bild 4: Mögliche Artikeleigenschaften

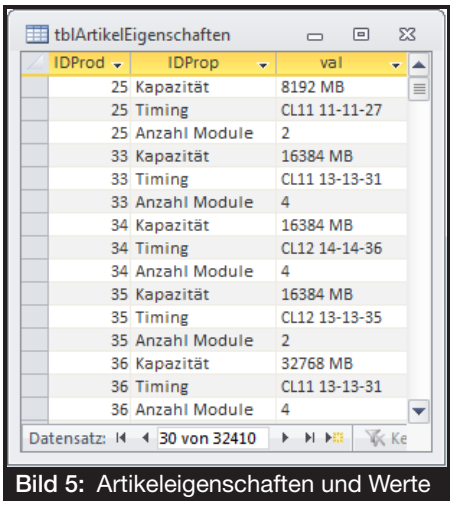


Bild 5: Artikeleigenschaften und Werte

noch das Textfeld **val**, welches den eigentlichen Wert der Eigenschaft aufnimmt. Bild 5 demonstriert, wie diese Kombination funktioniert.

Obwohl das Feld **IDProp** vom Typ Zahl ist, wird hier der Textwert angezeigt, weil für das Feld als Datenblattanzeigetyp ein Kombinationsfeld vorgesehen ist, das die Nachschlagetabelle **tblEigenschaften** verwendet.

Ein Artikel kann somit eine beliebige Anzahl von Eigenschaften und Werten für sie aufweisen. Aus deren Gesamtheit soll sich später dann das Produktdatenblatt ergeben.

Für den Artikel mit der **ID 25**, ein RAM-Speichermodul, gibt es etwa die drei Eigenschaften, welche in der Abbildung ganz oben stehen.

Soweit das Wesentliche zum Datenmodell des Webshops.

Shop-Funktionen

An dieser Stelle eine Übersicht der aktuell implementierten Features des Shops, die nicht nur in diesem Beitrag, sondern auch in weiteren Ausgaben näher beleuchtet werden sollen:

- Anzeigen von Kategorien
- Anzeigen der Artikel mit Produktdatenblatt, Vorschaubild, Bestand und Verfügbarkeit
- Filtern der Artikel nach Verfügbarkeit und Hersteller
- Hinzufügen von Artikel zum Warenkorb
- Anzeige des Warenkorbs mit der Möglichkeit zum Ändern der Artikelanzahl und/oder Löschen von Positionen
- Eingabe und Änderung der Kundenadresse
- Login registrierter Kunden
- Abschluss der Bestellung über Kasse mit E-Mail-Versand einer Bestellbestätigung über Outlook
- Übersicht über getätigte Bestellungen und deren Stati für eingeloggte Kunden
- Bearbeitung einer Bestellung vom Vertrieb mit anschließendem E-Mail-Versand nach Warenausgang
- Hilfsformular zum Anlegen der Shop-Artikel

Im Folgenden besprechen wir nach und nach die in dieser Anwendung vorhandenen Formulare, Berichte, Abfragen und Module, sowie die benötigten Techniken. Wir gehen dabei weitgehend chronologisch vor und orientieren uns an den einzelnen Vorgängen, die beim Öffnen der Datenbank vonstattengehen.

Webshop-Formular laden

Die Beispieldatenbank sieht das Formular **frmIntro** als Startformular vor, wie Sie das von allen unseren Demos kennen. Es ist in den Optionen der Aktuellen Datenbank eingetragen. Durch Klick auf dessen Detailbereich oder das Titel-Label wird die Prozedur **CloseAndDisplay** aufgerufen, die ihrerseits das Shop-Formular **frmShop** lädt und sich selbst beendet:

```
DoCmd.Close acForm, Me.Name
DoCmd.OpenForm "frmShop"
```

Haben Sie sich schon einmal gefragt, wieso überhaupt die zweite Zeile ausgeführt wird, wo das Formular doch zuvor bereits geschlossen wird? Es verhält sich so: Die **Close**-Methode setzt intern lediglich eine Meldung ab, dass das Formular geschlossen werden soll. Zu diesem

Zeitpunkt ist es auch noch sichtbar. Entladen ist es ebenso wenig und der VBA-Code wird weiterhin abgearbeitet. Hätten Sie eine Ereignisprozedur **Beim Entladen** (**Form_Unload**) im Formularcode, so würde diese unmittelbar nach Aufruf von **Close** angesprungen. Erst nach komplettem Beenden der Prozedur sieht Access keinen Grund mehr, das Formular im Speicher zu halten und schließt es aufgrund der internen Meldung endgültig. Diese Meldung übrigens ließe sich entfernen, wenn in der **Unload**-Prozedur der Parameter **Cancel** auf **True** eingestellt würde. Das Formular schließt sich dann eben nicht! Stattdessen bekämen Sie eine Fehlermeldung zu Gesicht, die besagt, dass die Schließen-Aktion abgebrochen wurde.

Die angegebenen zwei Zeilen Code sind aber noch nicht alles, was in der Prozedur **CloseAndDisplay** abläuft. Sie finden dort zunächst noch die Anweisung

```
CurDB.Execute "DELETE FROM" & _
"tblBestellungen WHERE KundeID=999999"
```

Hier wird eine SQL-Aufforderung abgesetzt, die die Datenbank anweist, alle Bestelldatensätze zu löschen, welche die spezielle **Kunden-ID 999.999** enthalten. Es wird sich gleich aufklären, was es mit dieser auf sich hat. Die Methode **CurDB** werden Sie im Befehlssatz von Access und VBA vergeblich suchen. Es ist eine Funktion im Modul **mdlShop** der Datenbank, die eine Objektinstanz von **Database** zurückgibt und in Listing 1 abgebildet ist. Die modulweit gültige Objektvariable thisdb steht im Kopf des Moduls. Die Prozedur ermittelt zunächst über Vergleich auf **Nothing**, ob die Variable bereits belegt ist. Falls nicht, so setzt sie sie deren Wert auf **CurrentDb**. Der Rückgabewert der Funktion selbst wiederum ist dann der Inhalt von **thisdb**.

```
Private thisdb As Database

Function CurDB() As Database
    If thisdb Is Nothing Then Set thisdb = CurrentDb
    Set CurDB = thisdb
End Function
```

Listing 1: Die Funktion CurDB gibt eine Instanz der aktuellen Datenbank, also CurrentDb , zurück

Wozu gibt es diese Funktion, und warum wird **CurrentDb** nicht direkt für die **Execute**-Methode zum Ausführen der SQL-Anweisung verwendet? In umfangreicheren Datenbanken wird auf das aktuelle **Database**-Objekt häufig Bezug genommen. Der häufigste Fall ist sicherlich das Öffnen von **Recordsets** über **OpenRecordset**. **CurrentDb** aber ist eine Funktion, die quasi eine Kopie des **Database**-Objekts von Access zum Zeitpunkt des Aufrufs erstellt. Das ist ein Vorgang, der etwas Zeit benötigt. Aus Performancegründen macht es daher Sinn, diese Kopie nicht jeweils neu zu erstellen, sondern in einer Objektvariablen ständig bereitzuhalten.

Noch eine weitere VBA-Zeile finden Sie in der Prozedur **CloseAndDisplay**:

```
LoadTempVar "KundeID"
```

Die Prozedur **LoadTempVar** steht ebenfalls im Modul **mdlShop** (Listing 2). Sie simuliert das Laden eines **Cookies**. Besuchen Sie einen Webshop, so hinterlässt dieser in der Regel auf Ihrem Rechner als Textdatei ein **Cookie**. Was der Betreiber in dieses hineinschreibt, bleibt ihm überlassen. Sie kennen aber sicher das Feature **Automatisches Einloggen**. Haben Sie sich auf einer Seite registriert, so schreibt sie unter Umständen Ihre Login-Daten in ein **Cookie**. Bei erneutem Besuch lädt sie dieses und stellt fest, dass Sie ein bereits registrierter Kunde sind und beim Registrieren ausdrücklich das automatische Einloggen wünschten. Die Kundendaten werden nun aus dem **Cookie** entnommen.

```
Function LoadTempVar(ByVal sName As String) As Variant
    Dim val As Variant
    Dim F As Integer

    F = FreeFile
    Open CurrentProject.Path & _
         "\cookie.txt" For Binary As F
    Get F, , val
    Close F
    TempVars(sName) = val
    LoadTempVar = TempVars(sName)
End Function
```

Listing 2: Laden eines Variablenwerts aus der Datei cookie.txt im Datenbankverzeichnis

Dasselbe Feature soll unser lokaler Webshop bieten. In der Datei **cookie.txt** im Datenbankverzeichnis steht die **ID** des Kunden, der den Shop zuletzt geladen hatte. Die Aufga-

Steuerelemente im Eigenbau: Range Control

Der Fundus an Access-Steuerelementen fällt aus heutiger Sicht, gerade im Vergleich zu anderen visuellen Entwicklungsumgebungen, etwas bescheiden aus. Zwar lassen sich allerlei ActiveX-Steuerelemente in Formulare integrieren, doch beliebt ist dies bei vielen Entwicklern wegen der externen, unter Admin-Rechten zu registrierenden, Komponenten nicht besonders. Abhilfe schaffen können aber Steuerelemente, die Sie mit Bordmitteln einfach selbst erstellen. Wir stellen hier ein Beispiel vor: das Range Control.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1609_RangeCtl.accdb**.

Was ist denn ein Range Control?

Eine Übersetzung des Begriffs wäre **Bereichssteuerelement**. Sie kennen solche Regler etwa von Webshops oder Shopping-Portalen, wo Ihnen angeboten wird, die angezeigten Produkte über eine Preisspanne zu filtern. Dabei können Unter- und Obergrenzen mit der Maus und den Ziehpunkten des Elements eingestellt werden. Bild 1 zeigt unsere Umsetzung in einem Testformular. Mit vier **Range Controls** stellen sie dort visuell Wertebereiche ein.

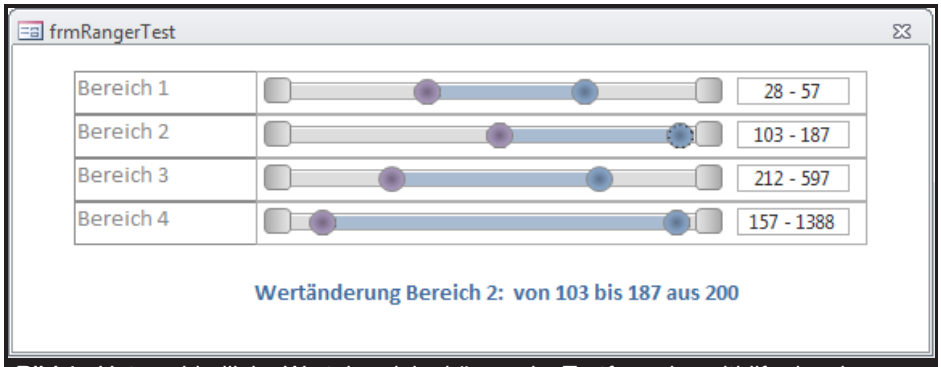


Bild 1: Unterschiedliche Wertebereiche können im Testformular mithilfe der vier Range Controls mit der Maus über Ziehpunkte eingestellt werden

Ein Anwendungsfall für derartige Steuerelemente wäre etwa die Filterung von Datensätzen. Im Datenblatt von Access können Sie bereits, je nach Felddatentyp, Wertebereiche zur Filterung der Anzeige bestimmen. Klicken Sie etwa, wie im Datenblatt zur Tabelle **tblArtikel** in Bild 2, auf den Dropdown-Pfeil des Spaltenkopfs zu **Einzelpreisen**. Es öffnet sich das Filtermenü mit diversen Möglichkeiten. Um einen Zahlenbereich festzulegen navigieren Sie über **Zahlenfilter** auf den Untermenüeintrag **Zwischen...**. Es erscheint ein einfaches Dialogfenster zur manuellen Eingabe der Bereichsgrenzen für die Preise, nach dessen Bestätigung sich die Datensätze entsprechend filtern. Während dieses Feature im Datenblatt noch ganz praktisch ist, verhielte es sich in Formularen wegen der Menüverschachtelung etwas undurchsichtig. Hier käme man mit einem **Range Control** schneller zum Ziel.

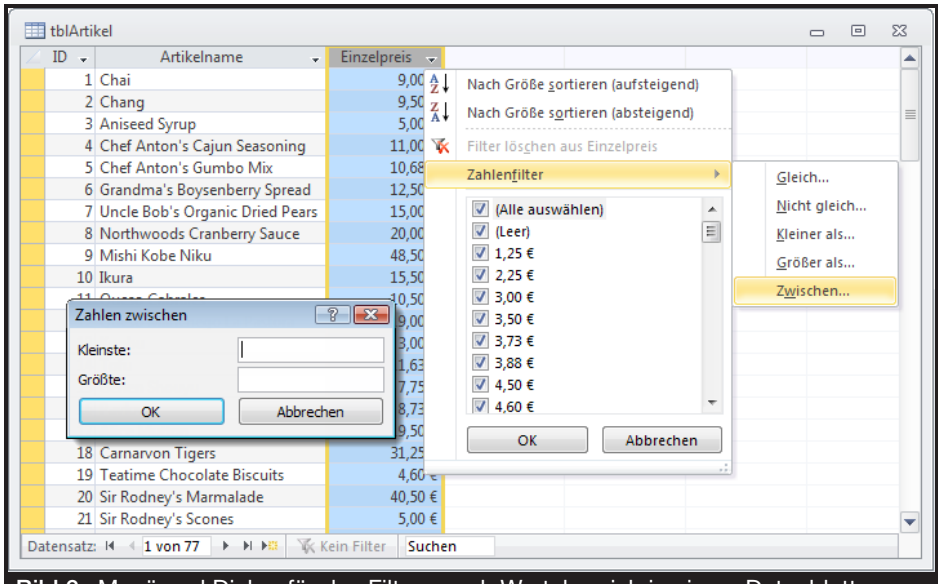


Bild 2: Menü und Dialog für das Filtern nach Wertebereich in einem Datenblatt

Formular als Steuerelement

Wenn Sie ein neues virtuelles Steuerelement entwickeln, werden Sie dieses in der Regel aus mehreren verfügbaren Access-Steuerelementen zusammensetzen wollen. Das könnten Sie direkt in jenem Formular tun, in dem dieses zum Einsatz kommen soll. Sobald dies jedoch in mehreren Formularen geschehen soll oder Sie das neue Element in andere Datenbanken importieren möchten, wird der Aufwand zu groß.

Die beteiligten Steuerelemente müssten alle kopiert werden und die zugehörigen Code-Abschnitte ebenso. Bequemer kommen Sie weg, wenn Sie die Steuerelemente und den Code in einem eigenen Formular anlegen, welches Sie dann schließlich nur als Unterformular in andere einsetzen. Zudem lässt sich dieses dann als Objekt auch in andere Datenbanken exportieren.

Eigenes Range Control

In der Beispieldatenbank finden Sie das neue Steuerelement in Gestalt des Formulars **sfrmRanger**. Sein Entwurf zeigt sich in Bild 3. Sie können es auch unmittelbar öffnen und bedienen, wie in Bild 4 dargestellt. Die Breite des virtuellen Steuerelements passt sich dabei der Größe des Formulars an, dessen Rahmen auf **Veränderbar** eingestellt ist, wodurch es mit der Maus dimensioniert werden kann. Dies wirkt sich später auch aus, wenn Sie es als Unterformular dimensionieren.

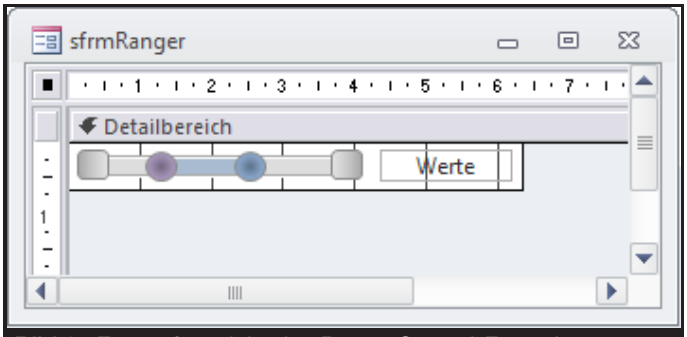


Bild 3: Entwurfsansicht des Range Control-Formulars

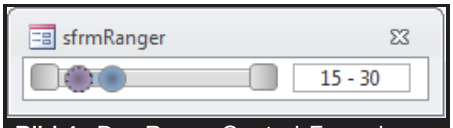


Bild 4: Das Range Control-Formular zur Laufzeit nach dem Öffnen

Die weiteren relevanten Einstellungen des Formulars: Der **Datenbankmarkierer** ist ausgeschaltet und ebenso die **Navigationsschaltflächen**. Die **Standardansicht** ist **Einzelnes Formular**. Gleichzeitig ist dies die einzige zugelassene Ansicht. Es gibt auch keine **Trennlinien** oder **Bildlaufleisten**.

Sieben Steuerelemente ergeben zusammen das **Range Control**. Die beiden Ziehpunkte sind Schaltflächen, die über deren Design-Eigenschaften auf blaue und violette Hintergrundfarbe mit Verlauf eingestellt sind (Bild 5). Außerdem ist ihr Rahmen rund (Bild 6). Sie erreichen diese Layout-Steuerung bei markiertem Button über den Ribbon **Format | Steuerelementformatierung** und das Menü zu **Fülleffekt**, sowie die Schaltfläche **Form ändern**.

Die beiden Schaltflächen heißen **cmd1** und **cmd2**. Für den linken und rechten Rand des Elements kommen ebenfalls Schaltflächen zum Einsatz, die sich **cmdL** und **cmdR** nennen.

Sie sind auf ähnliche Weise formatiert, wie die Ziehpunkte.

Unter diesen vier Schaltflächen liegen die zwei **Rechteck**-Steuerelemente **r0** und **r1**. **r0** ist grau gefüllt und stellt den Hintergrundbalken dar, **r1** ist blau und symbolisiert den gewählten Bereich. Damit diese sechs Elemente korrekt angezeigt werden, muss eine bestimmte **Z-Order** eingehalten werden, weil sie ja übereinander liegen. Die **Z-Order** ist jene Reihenfolge, in der Steuerelemente übereinander zu liegen kommen, also deren Tiefeneigenschaft. Unter Access lässt

Bild 5: Farbverlauf für die Buttons

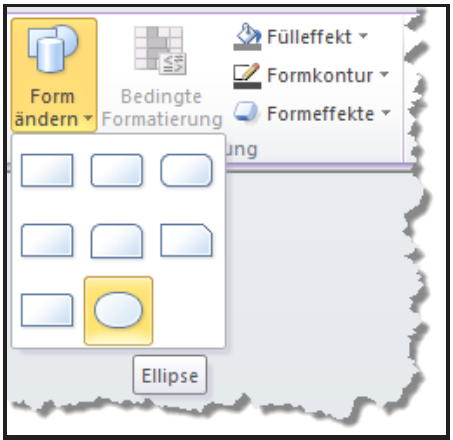


Bild 6: Form für die Buttons

sich auf diese **Z-Order** direkt leider weder über die Oberfläche beim Entwurf, noch per VBA zugreifen. Stattdessen bemühen Sie die beiden Ribbon-Schaltflächen **In den Vordergrund** und **In den Hintergrund** im Ribbon-Tab **Anordnen** ganz rechts. Für unser Beispiel gehen Sie dabei so vor:

Markieren Sie **r0** und klicken **In den Hintergrund**. Markieren Sie nun **r1** und klicken **In den Vordergrund**. Schließlich markieren Sie **cmdL**, **cmd1**, **cmd2** und **cmdR** und klicken abermals **In den Vordergrund**. Damit ist gewährleistet, dass die Buttons nie hinter die beiden **Rechteck**-Steuerelemente geraten.

Ein Bezeichnungsfeld **LblVal** am rechten Rand des Formulars dient zur Anzeige des eingestellten Wertebereichs.

Da das **Range Control** in der Breite variabel ausfallen soll, müssen sich die enthaltenen Steuerelemente entsprechend der Größenänderung des Formulars in Position und Breite anpassen (s. Bild 7). Der Versuch, dies über das **Verankern**-Feature der Elemente zu bewerkstelligen lief indessen ins Leere. Zwar lassen sich die Anpassungen damit visuell korrekt steuern, doch die aus den Button-Positionen abgeleiteten Werte können damit leider nicht mehr ermittelt werden. Stelen Sie etwa den Hintergrundbalken **r0** auf **Quer nach oben dehnen** ein, so macht er die Größenänderungen des

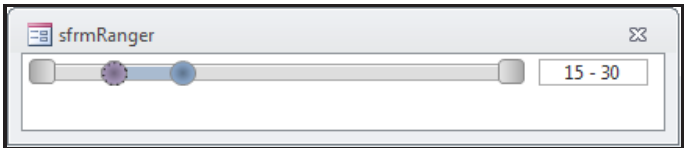


Bild 7: Bei Größenänderung des Formulars positionieren und dimensionieren sich die integrierten Elemente passend

Formular anstandslos mit. Möchten Sie nun dessen Breite ermitteln, die ja die Basis für den Wertebereich darstellt, so würden Sie dafür die VBA-Eigenschaft **Width** des Steuerelements verwenden. Die jedoch bleibt auf dem Wert stehen, der im Entwurf angelegt wurde. Das Dehnen ändert also **Width** nicht! Gleiches gilt etwa für **Left** bei Steuerelementen, die auf **Oben rechts** eingestellt sind. Die linke Position wandert dann zwar im Formular bei Größenänderung, doch **Left** bleibt konstant auf dem im Entwurf festgelegten Wert.

Deshalb kommt man nicht umhin, die Positionierung stattdessen per VBA im Ereignis **Bei Größenänderung** (**Form_Resize**) des Formulars vorzunehmen. Listing 1 zeigt den überschaubaren Code hierfür. Der linke Rand der rechten Bereichsschaltfläche **cmdR** ergibt sich aus der Breite des Formulars zur Laufzeit (**InsideWidth**, nicht **Width**!) minus der Breite des Labels **LblVal** minus seiner eigenen Breite minus 300 **Twips**. Die Position des Labels **LblVal** wiederum errechnet sich ähnlich. Die Breite des Hintergrundbalkens **r0** ermittelt

```
Private Sub Form_Resize()
    On Error Resume Next
    cmdR.Left = Me.InsideWidth - LblVal.Width - cmdR.Width - 300
    LblVal.Left = Me.InsideWidth - LblVal.Width - 150
    r0.Width = cmdR.Left - cmdL.Left
End Sub
```

Listing 1: Positionierung der Steuerelemente des Formulars bei Größenänderung in der Resize-Ereignisprozedur

sich aus der Formularbreite minus Label **LblVal** minus 150 **Twips**. Die linke Bereichsschaltfläche **cmdL** hingegen bleibt an Ort und Stelle. Die beiden Ziehpunkt-Buttons **cmd1** und **cmd2**, sowie der Bereichsbalken **r1**, aber werden ja nicht in dieser Prozedur eingestellt, sondern beim Ziehen mit der Maus; und wie das geht erfahren Sie im Folgenden.

Bewegen von Steuerelementen mit der Maus

Mit wenig Code kann die Mausbewegung auf einem Steuerelement dafür verwendet werden, dieses im Formular zu verschieben. Die dafür geeigneten Ereignisprozeduren sind **Bei Maustaste Ab** (**MouseDown**), **Bei Maustaste Auf** (**MouseUp**) und **Bei Mausbewegung** (**MouseMove**). Exemplarisch zeigt Listing 2, wie dies für den linken Ziehpunkt **cmd1** realisiert ist.

Wird eine Maustaste gedrückt, während sich der Cursor über dieser Schaltfläche befindet, so löst dies das Ereignis **MouseDown** aus. Die zugehörige Prozedur übergibt einige Parameter. **Button** ist eine Zahl, die sich auf die Taste bezieht. Bei **1** handelt es sich um die linke, bei **2** um die rechte Maustaste. **Shift** ist ein Wert, der die Kombination aus **STRG**- **SHIFT**- und **ALT**-Taste wiedergibt. Wir lassen diesen hier unberücksichtigt. **x** und **y** sind die Koordinaten, bei denen der Cursor quasi auf das Element traf. Uns interessiert natürlich nur die horizon-

Rezepte verwalten

Die wichtigste Information eines Kochrezepts ist seine Zutatenliste mit ihren Mengenangaben. Diese basiert auf verschiedenen Lebensmitteln, die bereits in der Nährwertdatenbank dieser Ausgabe vorliegen. Es liegt also nahe, diese Datenbank um einen Rezepteteil zu erweitern. Dabei kümmern wir uns hier lediglich um diese Liste und lassen die Zubereitungsanleitungen außen vor.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1609_Rezepte.accdb**

Rezepttabellen

Das Datenmodell der Nährwertdatenbank hatten wir schon vorgestellt. Es enthielt bereits die Tabellen zur Rezepteverwaltung. Werfen Sie nochmals einen Blick darauf (Bild 1). Die Tabelle **tblRezepte** besteht aus einem Primärschlüssel **ID** als **Autowert** zur eindeutigen Identifizierung und aus der Bezeichnung des Rezepts im Feld **Rezept**. **Portionen** ist ein **Long**-Wert, der darüber Auskunft gibt, für wie viele Personen die Mengenangaben des Gerichts gedacht sind.

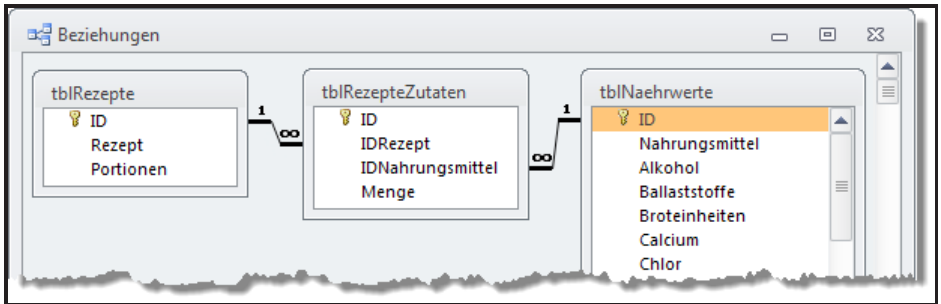
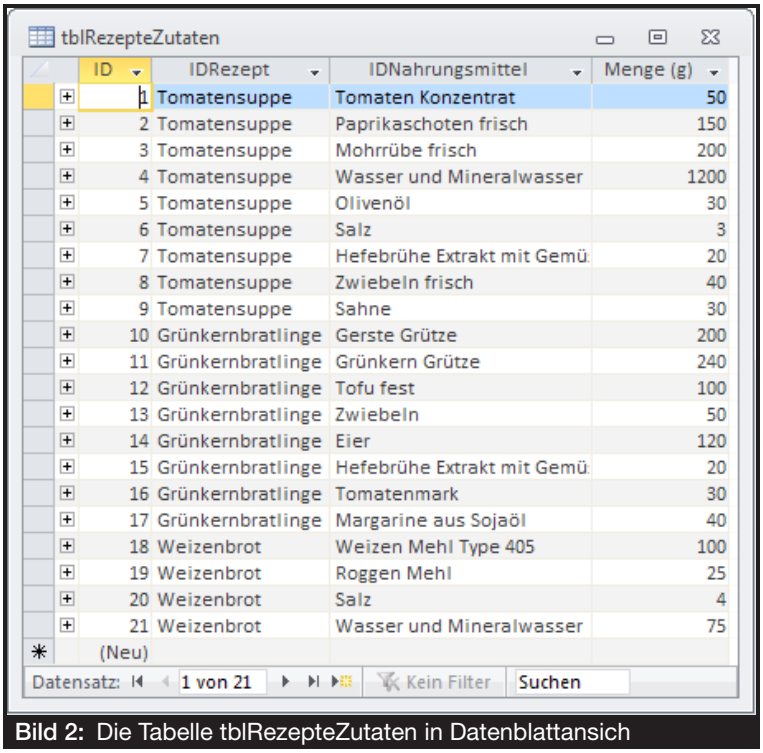


Bild 1: Das Modell der Rezepteverwaltung ist gleich wie das der Nährwertdatenbank

Die Tabelle **tblRezepteZutaten** stellt die Verbindung zur Nährwertetabelle her und verknüpft sie **n:m**. Also bezieht sich das Feld **IDRezept** jeweils auf einen Datensatz der Rezepttabelle und **IDNahrungsmittel** auf einen in **tblNaehrwerte**. Ist diese Verbindung für eine Zutat hergestellt, braucht es zusätzlich die zu verwendende **Menge** des Lebensmittels, die im gleichnamigen Feld festgehalten wird. Rufen Sie die Tabelle in der Datenblattansicht auf, so zeigt sie bereits eine sprechende Übersicht über die Rezepte und deren Zutaten an (Bild 2). Obwohl es sich bei den Feldern **IDRezept** und **IDNahrungsmittel** um indizierte Zahlenwerte handelt, finden sich der Ansicht hier die zugehörigen Bezeichnungen der verwiesenen Tabellen. Das wird durch Ändern der Feldeigenschaften erreicht.


tblRezepteZutaten

ID	IDRezept	IDNahrungsmittel	Menge (g)
1	Tomatensuppe	Tomaten Konzentrat	50
2	Tomatensuppe	Paprikaschoten frisch	150
3	Tomatensuppe	Mohrrübe frisch	200
4	Tomatensuppe	Wasser und Mineralwasser	1200
5	Tomatensuppe	Olivenöl	30
6	Tomatensuppe	Salz	3
7	Tomatensuppe	Hefebrühe Extrakt mit Gemü	20
8	Tomatensuppe	Zwiebeln frisch	40
9	Tomatensuppe	Sahne	30
10	Grünkernbratlinge	Gerste Grütze	200
11	Grünkernbratlinge	Grünkern Grütze	240
12	Grünkernbratlinge	Tofu fest	100
13	Grünkernbratlinge	Zwiebeln	50
14	Grünkernbratlinge	Eier	120
15	Grünkernbratlinge	Hefebrühe Extrakt mit Gemü	20
16	Grünkernbratlinge	Tomatenmark	30
17	Grünkernbratlinge	Margarine aus Sojaöl	40
18	Weizenbrot	Weizen Mehl Type 405	100
19	Weizenbrot	Roggen Mehl	25
20	Weizenbrot	Salz	4
21	Weizenbrot	Wasser und Mineralwasser	75
	(Neu)		

Datensatz: 1 von 21 | Kein Filter | Suchen

Bild 2: Die Tabelle tblRezepteZutaten in Datenblattansich

Im Entwurf der Tabelle ist etwa für das Feld **IDRezept** der Reiter **Nachschlagen** aktiviert und dort unter **Steuerelement anzeigen** der Typ von **Textfeld** auf **Kombinationsfeld** geändert worden (siehe Bild 3). Für dessen Datensatzherkunft ist eine Abfrage als SQL-String eingetragen, die aus der Rezeptetabelle nur die beiden Felder **ID** und **Rezept** herausholt. **Spaltenanzahl** steht folglich auf **2**, während die **Gebundene Spalte** die erste, also die **ID**, ist. Diese wird in der

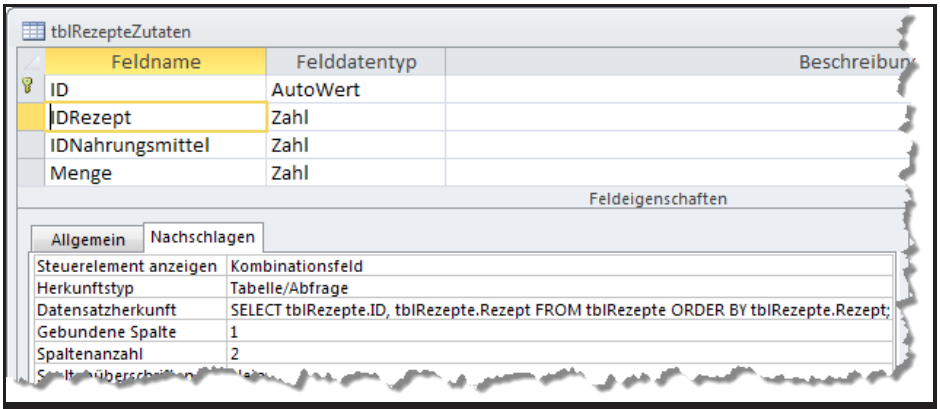


Bild 3: Details zum Entwurf der Tabelle tblRezepteZutaten für das Feld IDRezept

Datenblattansicht dann automatisch ausgeblendet.

Ähnlich wird mit dem Feld **IDNahrungsmittel** verfahren, nur dass die Abfrage sich dann hier auf die Nährwertetabelle bezieht.

Für die Tabelle **tblRezepte** ist in ihren Eigenschaften als Unterdatenblatt eben die Tabelle **tblRezepteZutaten** eingetragen. Das wäre dezidiert eigentlich nicht nötig, weil die Voreinstellung **(Automatisch)** dies selbständig vornimmt. Es ergibt sich aus der angelegten Verknüpfung im Beziehungsfenster. Öffnen Sie die Tabelle und klappen Sie ein Unterdatenblatt aus, so zeigt sich eine Ansicht, wie in Bild 4. Der Datensatz **Grünkernbratlinge** listet alle Zutaten auf. Eine Zutat können Sie ihrerseits ausklappen, weil für die Tabelle **tblRezepteZutaten** als Unterdatenblatt die Tabelle **tblNaehrwerte** spezifiziert ist. Damit können Sie auch gleich die Kennzahlen dieser Zutat ablesen.

Nun möchten Sie die Rezepte sicher nicht in Datenblättern verwalten. Anschaulicher wird das erst mit einem Formular.

Rezepteformular

Folgende Anforderungen werden an das Formular gestellt: Es soll pro Datenseite nur ein Rezept anzeigen. Zu einem Rezept soll sich über ein Auswahlkombinationsfeld navigieren lassen. Die Zutaten listen sich in einem Unterformular auf. Und alles soll editierbar sein.

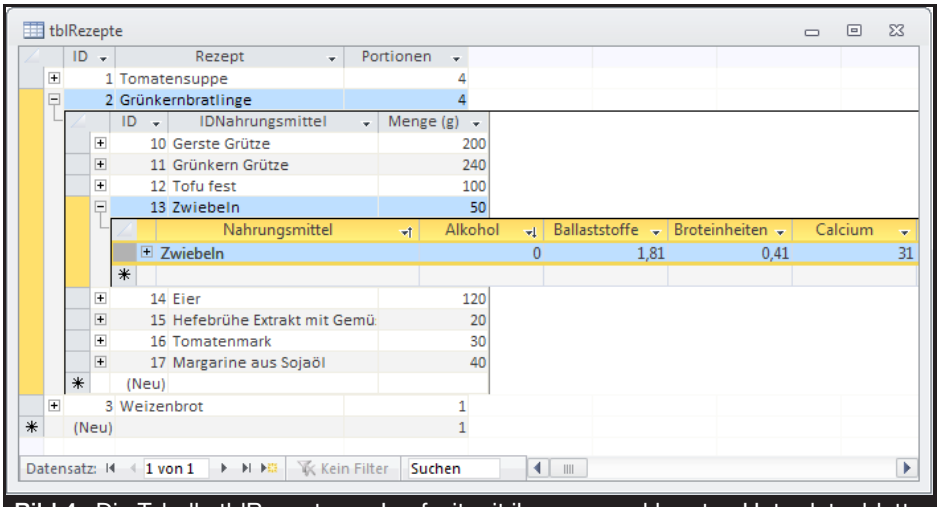


Bild 4: Die Tabelle tblRezepte zur Laufzeit mit ihrem ausgeklappten Unterdatenblatt

Bild 5 stellt das fertige Formular zur Laufzeit dar. Hier zeigen sich noch einige Zusatzfunktionen. Der Button **Mengen auf 1 Portion umrechnen** ermöglicht es, alle Zutatenmengen so zu ändern, dass die Angaben für genau eine Portion stimmen. Der Button ist nur dann aktiv, wenn im Rezept für **Portionen** eine Zahl abweichend von **1** eingetragen ist.

Im Formularfuß sind zwei Schaltflächen untergebracht, die zum einen das bekannte **Nährwerteformular** aufrufen lassen, zum anderen den **Kalorienrechner** zur Ermittlung des täglichen Energiebedarfs. Beide werden im Beitrag Nährwertdatenbank dieser Ausgabe erläutert.

Ist der Energiebedarf ermittelt worden, so zeigt ein Label unter der Zutatenliste zusätzlich an, welchen Anteil das aktuelle Rezept an Ihrem Tagesbedarf haben kann.

Und schließlich erwirkt ein Klick auf die Schaltfläche **Nährwertdaten des Rezepts** das Öffnen einer Abfrage, die weitere Kennzahlen zu den Inhaltsstoffen des Gesamtrezepts ausgibt. Dazu später mehr. Schauen wir uns zunächst Schritt für Schritt den Aufbau und die Code-Teile des Formulars frmRezepte an.

Das Unterformular für die Zutaten kommt aus dem Formular **sfrmRe-**

zepte (Bild 6), welches an eine Abfrage auf Basis der Tabelle **tblRezeptezutaten** gebunden ist. Es enthält ein Kombinationsfeld für das Nahrungsmittel, gebunden an das Feld **IDNahrungsmittel**, und eine Textbox für die Mengenangabe. Als Standardansicht ist **Datenblatt** eingestellt. Die Abfrage **qry_sfrmRezepte** als Datenherkunft dieses Unterformulars gibt allerdings noch mehr Felder zurück, als aus diesen beiden Steuerelementen ersichtlich wird (siehe Bild 7).

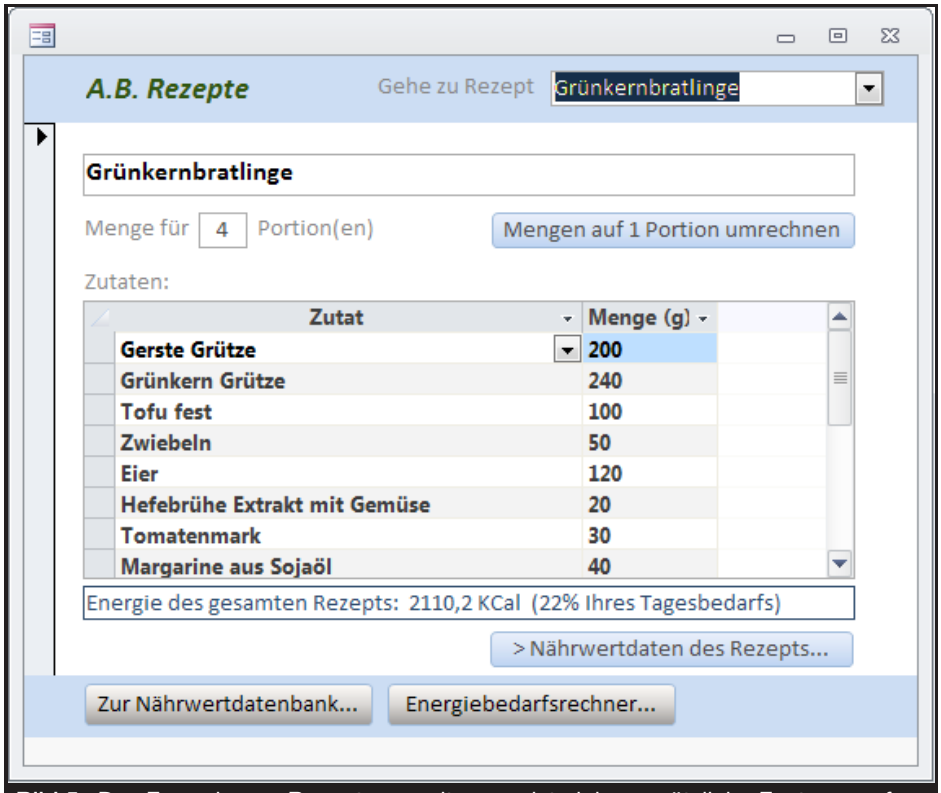


Bild 5: Das Formular zur Rezepteverwaltung weist einige zusätzliche Features auf

Wichtig ist vor allem das Feld **IDRezept**, weil es zur Verknüpfung des Unterformulars mit der **ID** des Hauptformulars dient. Das Feld **Energie** berechnet die Kalorienzahl für den Zutatendatensatz gemäß seiner Menge nach dieser Formel:

```
[Kcal]*[Menge]/100 AS Energie
```

Dieser Wert wird später dafür benötigt, um die Summe der Kalorien des Rezepts zu bestimmen und damit auch seinen Anteil am Tagesbedarf.

Bild 6: Für Unterformular der Zutatenliste ist das Formular sfrmRezepte in Datenblattansicht zuständig

Zu einem Rezept navigieren Sie über das Kombinationsfeld rechts oben im Hauptformular. Seine Daten bezieht es aus der Rezeptetabelle.

Eine Auswahl in ihm zeitigt das Ereignis **AfterUpdate** (**Nach Aktualisierung**). Hier ist der folgende Code hinterlegt:

```
Private Sub cbRezept_AfterUpdate()
  Me.Filter="ID=" & Me!cbRezept.Value
  Me.FilterOn=True
End Sub
```

Die Datenherkunft des Formulars (**tblRezepte**) wird damit nach der gewählten **ID** gefiltert. Nicht nur im Hauptformular wirkt sich dies aus, sondern auch im über die **ID** verknüpften Unterformular, welches nun nur jene Zutaten auflistet, die dem Rezept zugeigen sind.

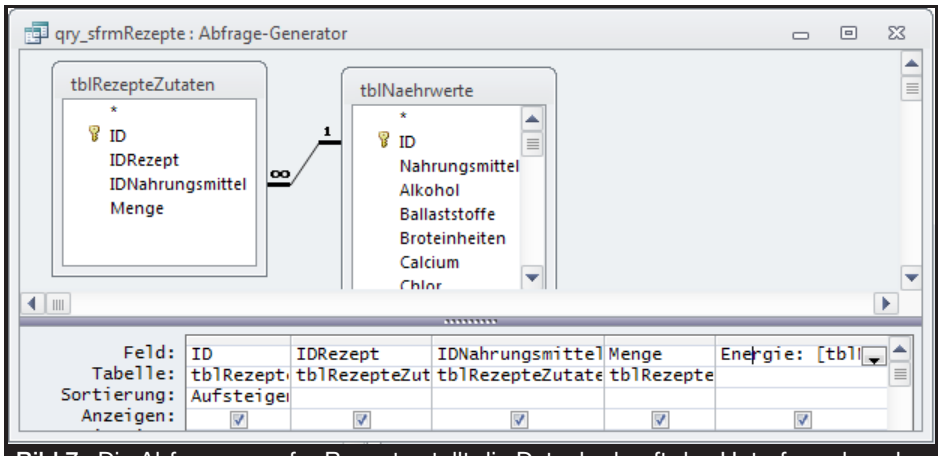


Bild 7: Die Abfrage qry_sfrmRezepte stellt die Datenherkunft des Unterformulars dar

Nährwertdatenbank

Es gibt eine ganze Menge Objekte auf der Welt, die sich abseits der üblichen Adress- und Kundendaten in einer Datenbank verwalten lassen. Auf der Suche nach weniger trivialen Inhalten stießen wir auf die Daten von Lebensmitteln und deren Nährwerten – in Zeiten zunehmender Verunsicherung in Sachen Ernährung möglicherweise ein interessantes Thema. Herausgekommen ist eine Anwendung mit einer zusätzlichen Rezepteverwaltung und einem Kalorienrechner.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1609_Naehrwerte.accdb**

Lebensmittel und Nährwerte

Im Netz lassen sich zahlreiche Datensammlungen zu Lebensmitteln und deren Nährwerten finden. Die Qualität dieser Daten ist allerdings sehr inhomogen. Im einen Fall ist der Umfang zu gering, im anderen stimmen die Inhalte teilweise nicht. Größer ist allerdings das Problem mit den Datenformaten. Man findet die Tabellen im Text- oder **CSV**-Format, als Excel-Files oder gar als **SQL-Postgres-Dump**. Derlei zieht eine Konvertierung zu Access nach sich, die meist mit Nachbearbeitungen verbunden ist. Doch selbst dann, wenn Sie eine originäre Access-Datenbank zum Download finden, stehen Sie häufig vor Problemen.

Ein gutes Beispiel ist die regelmäßig aktualisierte Datenbank des US-Landwirtschaftsministeriums (**Link**), die unter anderem auch im ACCDB-Format vorliegt. An Vollständigkeit lässt diese Datenbank mit ihren etwa 9.000 Datensätzen nichts zu wünschen übrig. Indessen ist ihre Nomenklatur komplett englisch und weiter das Datenmodell mit den 13 Tabellen so verzweigt, dass sich ihre Verwendung für eine simple Nährwertdatenbank praktisch ausschließt.

Lange Rede, kurzer Sinn: Unsere Nährwerttabelle leitet sich aus mehreren Quellen ab, die schließlich über Konvertierungen und Anfüge- wie Aktualisierungsabfragen erhalten wurde. Wir sehen davon ab, hier das erfolgte Procedere zu beschreiben. Ergebnis ist ein Datenpool mit circa 12.000 Datensätzen, die jeweils 48 Kennzahlen enthalten. Die hohe Zahl ist dem Umstand geschuldet, dass nicht nur elementare Lebensmittel verzeichnet sind, sondern auch zusammengesetzte Produkte, wie etwa der **Allgäuer Käsesuppentopf** oder die **Schweinezunge mit Weißweinsoße**, also Imgrunde fertige Rezepturen.

Datenmodell

Einfacher geht's nicht: Die Nährwertdaten befinden sich alle in nur einer Tabelle **tblNaehrwerte**, die sich rechts in Bild 1 findet. Die **ID** als **Autowert** stellt den Primärschlüssel der Tabelle, das Feld **Nahrungsmittel** seine Bezeichnung. Es ist ebenfalls mit einem eindeutigen Index versehen, um Doppelungen auszuschließen. Darauf folgen 48 Felder zu allen möglichen Kennzahlen, wie der Gehalt an **Calcium**, **Fett**, oder **Aminosäuren**. Diese Zahlenfelder sind

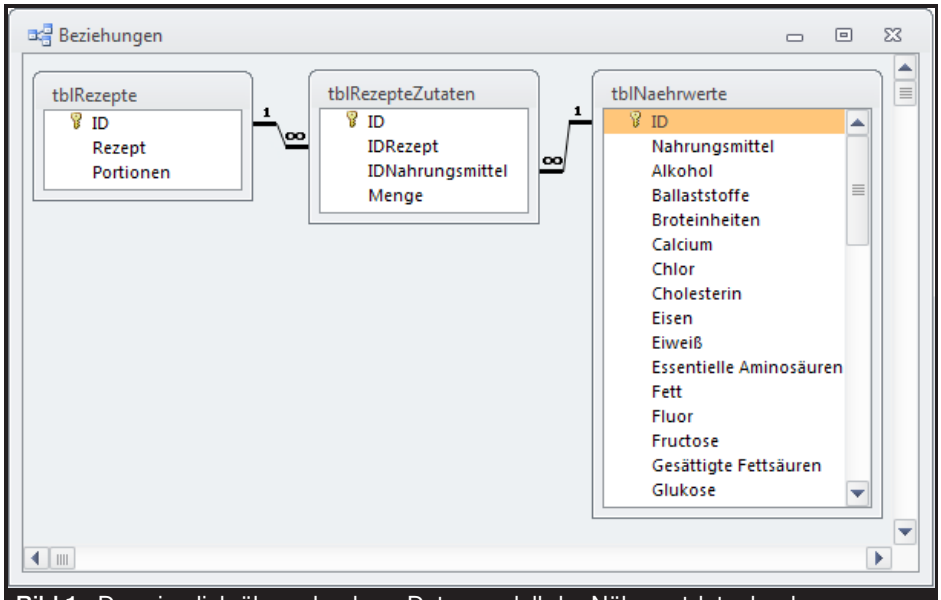


Bild 1: Das ziemlich überschaubare Datenmodell der Nährwertdatenbank

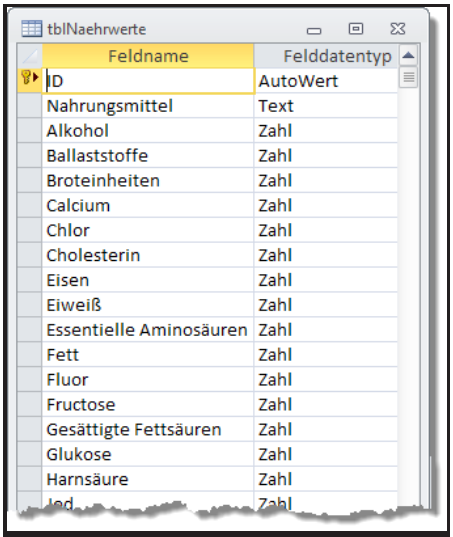


Bild 2: tblNaehrwerte im Entwurf

allesamt als **Double** definiert. Den Entwurf der Tabelle zeigt Bild 2. Verbunden mit der Nährwerttabelle sind **tblRezepte** und **tblRezepteZutaten**. In der ersten werden die angelegten Rezepte namentlich verzeichnet, wobei im Rezept zusätzlich die gedachte **Portionen**-Menge spezifiziert ist. Die zweite verknüpft Rezepte und Nahrungsmittel **n:m**. Ein Rezept kann demnach mehrere Nahrungsmittel enthalten, deren **Menge** in der Verknüpfungstabelle als **Double**-Zahl für das Gewicht in Gramm aufgeführt ist.

Zur Laufzeit sieht die Nährwerttabelle als Datenblatt aus, wie in Bild 3.

Sie können sie schon hier über die Filterpfeile der Spaltenköpfe gemäß Ihrer Kriterien filtern und sortieren. Sortieren Sie etwa absteigend nach **Eiweiß**, um die Lebensmittel mit dem höchsten Eiweißgehalt zu ermitteln.

Sehr unpraktisch sind indessen die vielen Felder, die ein horizontales Scrollen erfordern, sodass die Daten eines Lebensmittels auch auf einem großen Monitor nie auf einen Blick zu sehen sind. Hier muss ein Formular mit Detailfeldern her!

Nährwerteformular

Bei 12.000 Datensätzen ist das Auffinden eines Lebensmittels eigentlich nur über eine Suchfunktion praktikabel. Deshalb statten wir

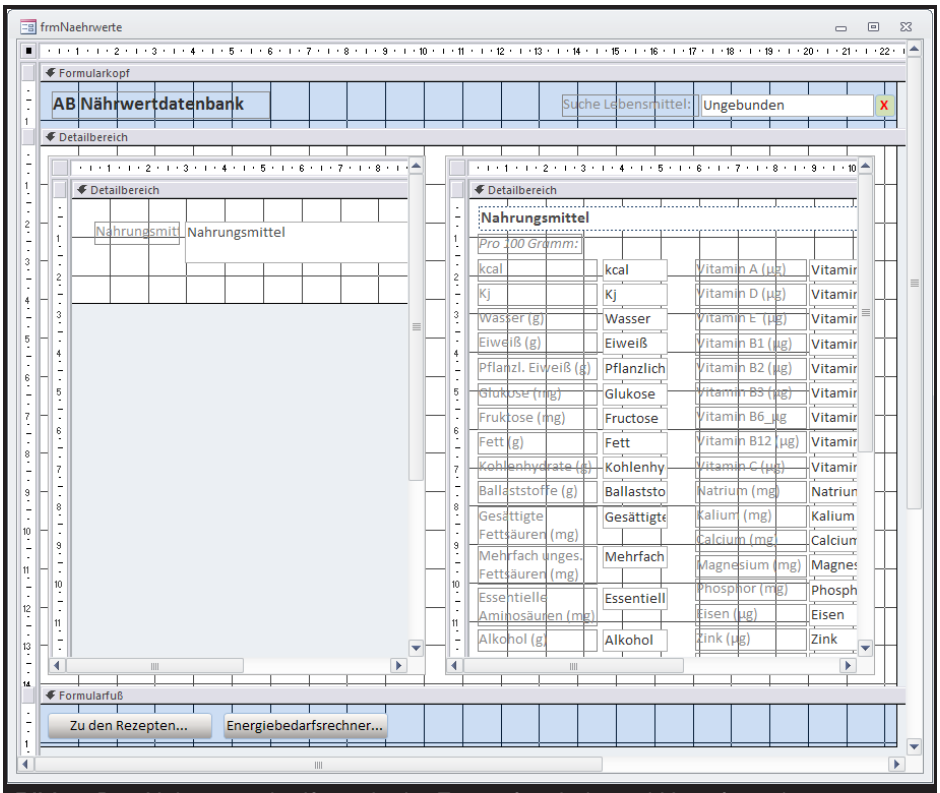


Bild 4: Das Nahrungsmittelformular im Entwurf enthält zwei Unterformulare

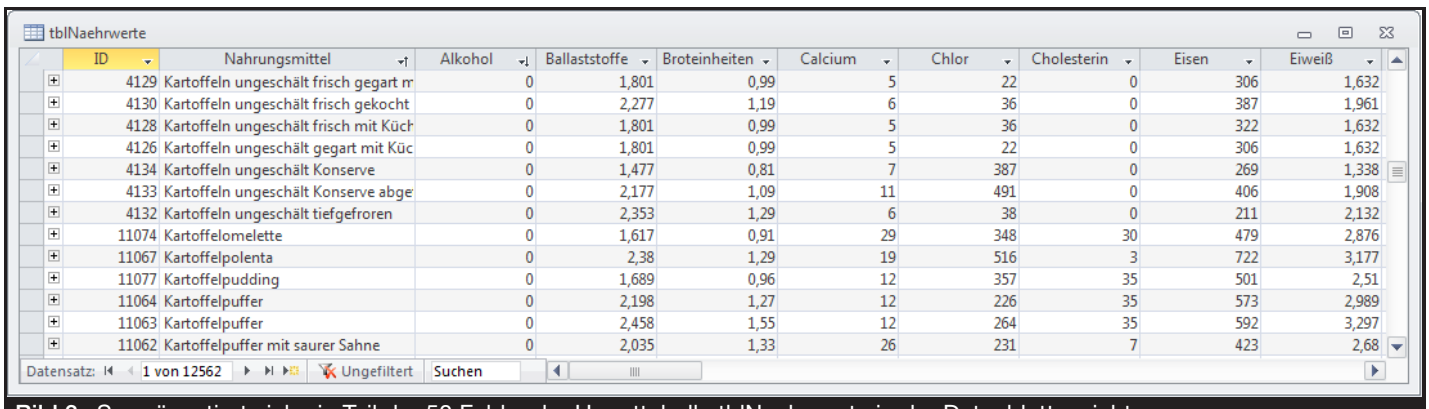


Bild 3: So präsentiert sich ein Teil der 50 Felder der Haupttabelle tblNaehrwerte in der Datenblattansicht

unser Formular auch mit einer solchen aus. Bild 4 zeigt den Aufbau. Rechts oben befindet sich das Textfeld zur Eingabe eines Suchbegriffs. Unmittelbar nach jedem Tastendruck soll sich dabei die Liste der Nahrungsmittel links im Unterformular **sfrmList** kontinuierlich aktualisieren. Bei diesem Unterformular auf Basis des Formulars **sfrmNahrungsmittel** handelt es sich um ein schlichtes Formular in Datenblattansicht, welches aus der Tabelle **tblNaehrwerte** nur die Lebensmittelbezeichnungen über diese Abfrage auflistet:

```
SELECT ID, Nahrungsmittel
FROM tblNaehrwerte
ORDER BY Nahrungsmittel;
```

Wird in dieser Liste ein Lebensmittel angeklickt, so sollen im Unterformular **sfrmDetail** rechts dessen Daten erscheinen. Das hier verwendete Formular **sfrmNahrungsmittel_E** ist direkt an die Tabelle **tblNaehrwerte** gebunden und sieht im Entwurf aus, wie in Bild 5. Das Hauptformular selbst ist, wohlgemerkt, an gar keine Datenherkunft gebunden! Damit entfallen auch etwaige Feldverknüpfungen zwischen Haupt- und Unterformularen.

Aufgabe der VBA-Routinen des Formulars also ist es, einerseits die Filterung des Datenbestands über das Suchfeld zu erreichen, andererseits eine aktive Verbindung zwischen Listenunterformular links und Detailunterformular rechts herzustellen. Im Betrieb präsentiert es sich etwa wie in Bild 6. Als Suchbegriff wurde hier **Weizenbr** eingegeben. Wie an der Liste links zu erkennen, filtert die Datenbank nicht nach dem Anfang des Begriffs, sondern nach dem Inhalt, also **LIKE**. Folglich tauchen auch die **Knäckebrot-Weizenbrote** im Ergebnis auf. Markiert aber wurde das **Weißbrot-Weizenbrot**, dessen Detaildaten darum rechts im Formular erscheinen.

Bild 5: Das Detailunterformular sfrmNahrungsmittel_E im Entwurf

Dort sind von den 48 Feldern tatsächlich nur 32 untergebracht. Es handelt sich dabei um die relevantesten Kennwerte. Die restlichen sind eher akademischer Natur und aus Platzgründen außen vor gelassen. Tragen Sie diese selbst nach, wenn Sie möchten.

Datentechnisch sind die drei verbundenen Formulare also alles andere als komplex. Legen wir unser Au-

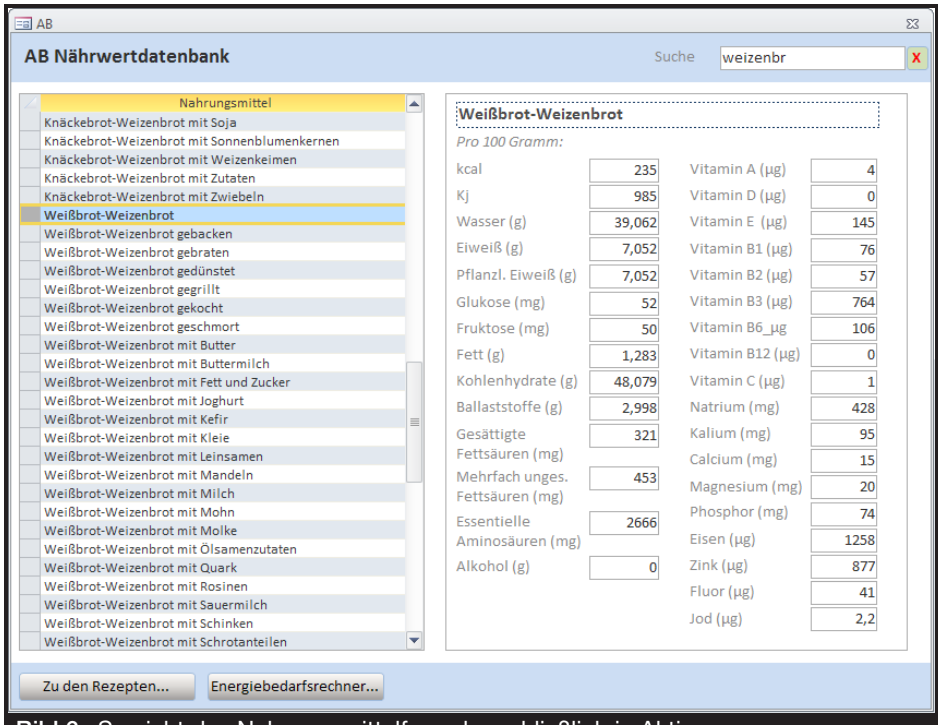


Bild 6: So sieht das Nahrungsmittelformular schließlich in Aktion aus

genmerk deshalb auf deren VBA-Steuerung. Beim Laden des Hauptformulars tritt bereits eine Filterung ein, die dafür sorgen soll, dass beide Unterformulare leer erscheinen:

```
Private Sub Form_Load()
  Me!sfrmList.Form.Filter = _
      "Nahrungsmittel='_Lebensmittel'"
  Me!sfrmList.Form.FilterOn = True
  Me!txtSuche.SetFocus
End Sub
```

Hier wird für das Listenunterformular **sfrmList** ein Filter gesetzt, der die Datensätze auf genau einen begrenzt, nämlich auf jenen mit der Bezeichnung **Lebensmittel**. Das ist ein spezieller Dummy-Datensatz in **tblNaehrwerte**, dessen Feldinhalte bis auf die Bezeichnung alle leer gelassen wurden. Ergo findet sich im Listenunterformular lediglich der Eintrag **Lebensmittel**. Das Detailunterformular sfrmDetail zeigt ebenfalls diesen Datensatz an. Wie das, wo dieses doch in der Load-Prozedur des Hauptformulars gar keiner Behandlung unterzogen wird?

Der Trick liegt in einer Prozedur des Listenformulars. Im Ereignis **Beim Anzeigen** läuft die Routine in Listing 1 ab. Hier wird über **Me.Parent** auf sein Elternformular Bezug genommen. Das enthält wiederum das Unterformular **sfrmDetail** als Steuerelement.

Seine Filtereigenschaft wird auf dieselbe Datensatz-**ID** gesetzt, die im Listenunterformular aktuell markiert ist – und das wäre, gesteuert über das Hauptformular und dessen **Load**-Prozedur, nun mal der Datensatz **Lebensmittel**. Ist die Filterung von Erfolg gekrönt, was die Fehlerbehandlung der Routine durch die Fehlernummer **0** quittiert, so wird das Detailformular auf **Sichtbar** (**Visible**) geschaltet und sein Filter aktiviert.

```
Private Sub Form_Current()
    On Error Resume Next
    Me.Parent!sfrmDetail.Form.Filter = "ID=" & Me!ID.Value
    If Err.Number = 0 Then
        Me.Parent!sfrmDetail.Visible = True
        Me.Parent!sfrmDetail.Form.FilterOn = True
    End If
End Sub
```

Listing 1: In der Current-Prozedur des Listenformulars wird das Detailformular sfrmDetail des Haupt- und Elternformulars angesprochen und gefiltert

Das Anzeigen des korrekten Datensatzes im Detailunterformular übernimmt also grundsätzlich nicht sein Hauptformular, sondern das Listenunterformular. Dieses Beispiel demonstriert anschaulich den Bezug zwischen Haupt- und Unterformularen über einerseits die **Parent**-Eigenschaft, und andererseits das Unterformularsteuerelement plus dessen **Form**-Eigenschaft über VBA.

Formulare filtern

Für das Suchfeld des Hauptformulars, eine Textbox namens **txtSuche**, ist in der Ereigniseigenschaft **Bei Taste Auf** (**KeyUp**) eine Routine angegeben, die das Listenunterformular bei jedem Tastenanschlag entsprechend filtert. Listing 2 bildet sie ab.

Als Parameter übergibt das Ereignis nicht einen String für die Taste, sondern einen **Keycode**. Dieser entspricht dem Ordinalwert des ASCII-Zeichens. Berücksichtigt werden sollen nur alphanumerische Tasten. Deshalb wertet der **Select-Case**-Block zunächst den **Keycode** aus und verwendet nur die Zeichen von **65** (**A**) bis **123** (**z**). Die String-Va-

```
Private Sub txtSuche_KeyUp(KeyCode As Integer, Shift As Integer)
    Dim S As String

    Select Case KeyCode
    Case Is > 64, Is < 123
        S = Nz(Me!txtSuche.Text)
    End Select

    Select Case Len(S)
    Case 0
        Me!sfrmList.Form.Filter = "ID=0"
        Me!sfrmDetail.Visible = False
    Case Is < 4
        Me!sfrmList.Form.Filter = "Nahrungsmittel LIKE '" & S & "*'"
    Case Else
        Me!sfrmList.Form.Filter = "Nahrungsmittel LIKE '*" & S & "*'"
    End Select
    Me!sfrmList.Form.FilterOn = True
End Sub
```

Listing 2: Filtern des Listenunterformulars per VBA nach Suchfeldeingaben

Lokaler Webshop, Teil II

Nach dem Anzeigen der Artikel unseres in Access nachgebildeten Webshops der vorletzten Ausgabe geht es nun an das Bestellen der Produkte bei der **ABasics Computer GmbH**. Für den hier benötigten Warenkorb, die Kasse und die Bestellvorgänge kommen ganz neue Formulare ins Spiel. Deren Design, Aufbau und Programmierung widmet sich dieser Beitrag, wobei allerlei Tricks und Kniffe nicht zu kurz kommen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1610_WebshopII.zip**.

Bestellen im Webshop

Sie möchten im Hauptformular **frmShop** der Anwendung einen Drucker bestellen. Nach Auswahl der Produktkategorie **Drucker** im Kombinationsfeld links oben, und nach Markierung der Unterkategorie **Tintenstrahler ab A3** im Listenfeld links, füllt sich die Produktliste **lstArtikel** in der Mitte mit den verfügbaren Produkten. Da Ihnen der Hersteller **Hewlett-Packard** am vertrauenswürdigsten erscheint, filtern Sie diese Liste über das Kombinationsfeld rechts oben (siehe Bild 1). In dieser Combobox finden sich ausschließlich jene Hersteller, die auch in der Produktliste aufgeführt sind. Ihre Datenherkunft verhält sich also dynamisch in Abhängigkeit von der gewählten Unterkategorie. Listing 1 zeigt, wie sie bei Auslösen des Ereignisses **Nach Aktualisierung** des Listenfelds **lstKategorien** über einen neuen **SQL-Select-Ausdruck** festgelegt

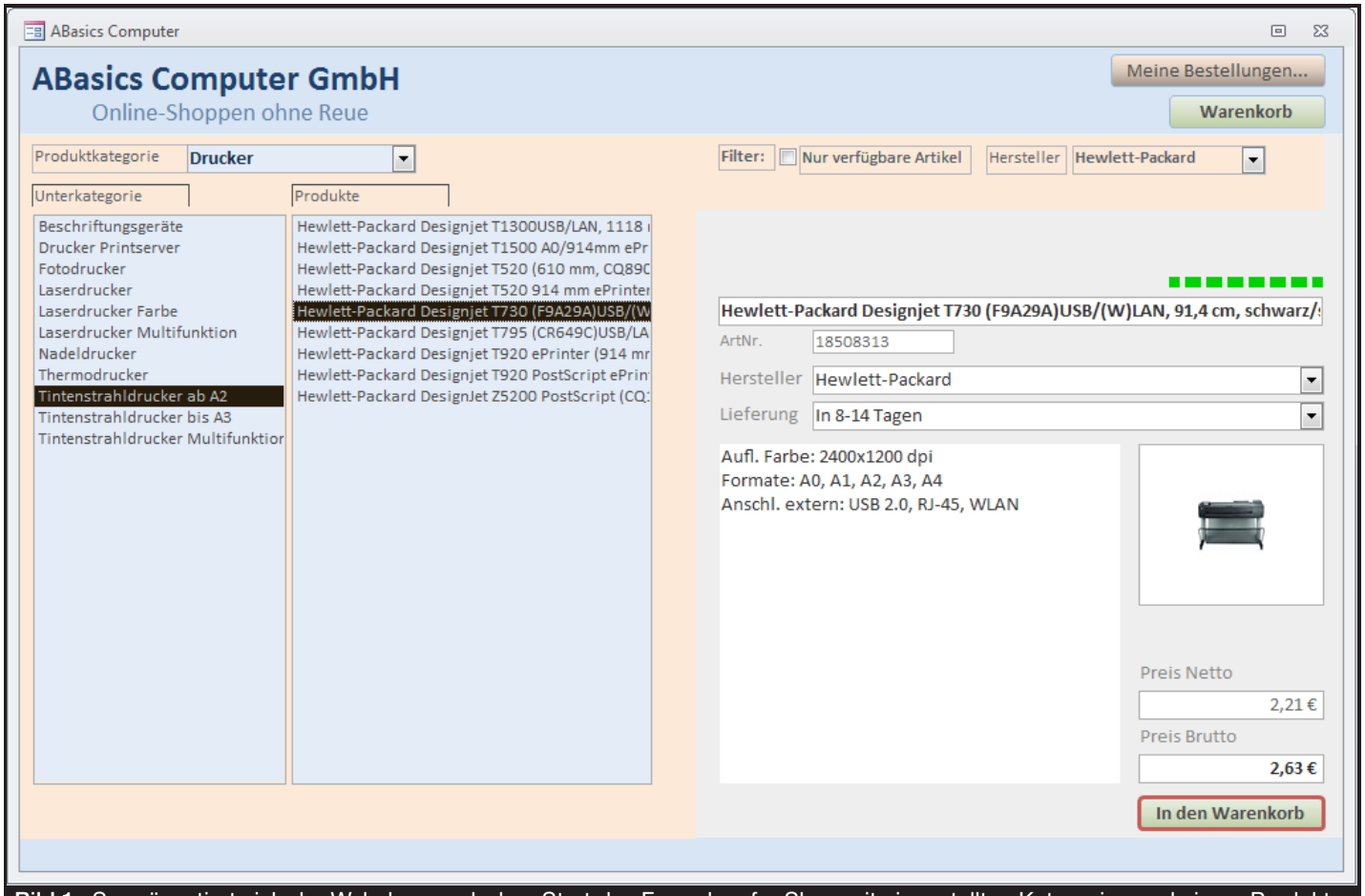


Bild 1: So präsentiert sich der Webshop nach dem Start des Formulars frmShop mit eingestellten Kategorien und einem Produkt

```
Private Sub lstKategorien_AfterUpdate()
    ...
    Me!cbHerstellerFilter.RowSource = "SELECT * FROM tblHersteller " & _
        "WHERE ID IN(SELECT IDHersteller FROM tblArtikel WHERE IDKategorie=" & _
        Me!lstKategorien.Value & ")"
    ...
End Sub
```

Listing 1: Setzen der Datenherkunft der Hersteller-Combobox cbHersteller

wird. Die **RowSource** des Steuerelements ist im Prinzip an die Tabelle **tblHersteller** gebunden. Damit aus dieser aber nur jene Datensätze ausgewählt werden, die in ihrem Feld **ID** den gleichen Wert aufweisen, der auch in der Kategorienliste markiert ist (**lstKategorien.Value**), benötigen wir eine Zwischenabfrage, denn die Herstellertabelle kennt selbst keine Kategorien. Das ist ein schönes Beispiel für den Einsatz der **IN**-Klausel. Aus **tblArtikel** ermittelt die **Select**-Abfrage innerhalb der Klammer alle Datensätze, deren Kategorien (**IDKategorie**) mit dem markierten Eintrag des Listenfelds übereinstimmen und gibt aus ihren nur jeweils die **IDHersteller** zurück. Befindet sich wiederum eine **ID** der Herstellertabelle **IN** diesem Pool, so ist die Bedingung positiv und der Datensatz mitsamt der Herstellerbezeichnung wird ausgegeben, so dass er im Kombinationsfeld erscheinen kann.

Sie interessieren sich für den **Deskjet T730** und haben ihn in der Produktliste markiert. Seine Details gibt nun das Unterformular rechts im Bild wieder. Den Preis finden Sie OK und Sie klicken auf den Button **In den Warenkorb**. Damit erscheint sogleich in einer **Msgbox** die Info aus Bild 2.

```
Private Sub cmdAdd_Click()
    Dim n As Long
    Dim ID As Long

    n = Nz(DLookup("Anzahl", "tblBestellDetails", "ArtikelID=" & Me!ID.Value & _
           " AND BestellID=" & BestellID))
    If n = 0 Then
        CurDB.Execute "INSERT INTO tblBestellDetails" & _
                      " (BestellID,ArtikelID,Anzahl,Netto,Ust)" & _
                      " VALUES (" & BestellID & "," & Me!ID.Value & ",1," & _
                      Str(Me!Netto.Value) & "," & Str(Me!USt.Value) & ")"
        MsgBox "Sie haben '" & Me!Produkt & "' in den Warenkorb gelegt.", vbInformation
    Else
        ID = DLookup("ID", "tblBestellDetails", "ArtikelID=" & Me!ID.Value & _
             " AND BestellID=" & BestellID)
        CurDB.Execute "UPDATE tblBestellDetails SET Anzahl=" & (n + 1) & _
                      " WHERE ID=" & ID
        MsgBox "Sie haben die Anzahl von '" & Me!Produkt & "' im Warenkorb erhöht."
    End If
    Me!cbWarenkorb.Requery
End Sub
```

Listing 2: Das Legen eines Produkts in den Warenkorb über den Button cmdAdd

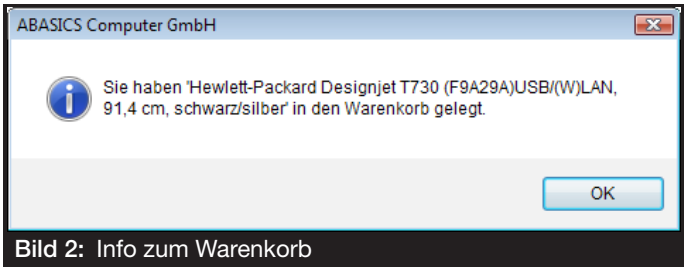


Bild 2: Info zum Warenkorb

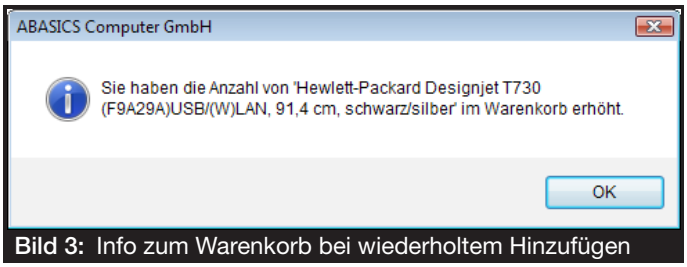


Bild 3: Info zum Warenkorb bei wiederholtem Hinzufügen

Meldung (Bild 3). Die beim Klicken auf den Button **cmdAdd** ausgelöste Prozedur finden Sie in Listing 2.

In der ersten Zeile ermittelt eine **DLookup**-Funktion zunächst die Anzahl jener Artikel, die sich bereits

Klicken Sie erneut auf den Button, so zeigt sich eine abweichende

in der Bestellung befinden, wobei nach dem aktuell ausgewählten Produkt (**ArtikelID = Me!ID**) und nach der momentanen Bestellung (**BestellID**) gefiltert wird. Im ersten Betrag wurde erläutert, dass schon beim Öffnen des Shop-Formulars ein temporärer Bestelldatensatz in der Tabelle **tblBestellDetails** angelegt und ein Verweis auf dessen **ID** in der globalen Variablen **BestellID** abgespeichert wird. Hier nun kommt diese **ID** zum Tragen. Ist der Artikel bereits in der Bestellung

vorhanden, so gibt **DLookup** einen von **0** abweichenden Wert zurück und weist diesen der Variablen **n** zu. Die Prozedur verzweigt jetzt in Abhängigkeit von diesem Wert.

Ist der Artikel neu, was den einfacheren Fall darstellt so findet das Einfügen eines Datensatzes über die **INSERT INTO**-Anweisung statt. Ihr werden als Feldparameter die aktuelle **BestellID**, der Preis (**Me!Netto**) und der Umsatzsteuersatz (**Me!Ust**) verabreicht. Diese letzten beiden Werte können nicht aus der Artikeltabelle entnommen werden! Denn bald könnte sich der Preis ja ändern, was bei Bearbeitung der Bestellung zu einer erhöhten Summe führen würde.

Hier muss der aktuelle Preis gelten, weshalb er fest in den Bestelldatensatz gespeichert werden muss. Die anschließende **MsgBox** zeigt nun lediglich das Produkt (**Me!Produkt**) in einem entsprechenden Text an.

Ist der Artikel in der Bestellung bereits vorhanden, also der Wert von **n** ungleich **0**, so führt sich der nächste Zweig der Prozedur aus. Hier muss erst ermittelt werden, welche **ID** der betreffende Bestelldatensatz hat, wobei abermals eine **DLookup**-Funktion verwendet wird. Anschließend erhöht eine **SQL-UPDATE**-Anweisung den Wert des Felds Anzahl im gefundenen Datensatz. Die Meldung sieht nun geringfügig anders aus.

Auf die Bedeutung der letzten Zeile, die ein Requery auf das Kombinationsfeld **cbWarenkorb** auslöst, kommen wir gleich zu sprechen.

```
Private thisdb As Database

Function CurDB() As Database
    If thisdb Is Nothing Then Set thisdb = CurrentDb
    Set CurDB = thisdb
End Function
```

Listing 3: CurDB gibt die aus CurrentDb erhaltene Objektvariable thisdb zurück

Die SQL-Anweisungen geschehen übrigens über die Objektvariable **CurDB**. Das ist tatsächlich eine Funktion, die nur die Access-Eigenschaft **CurrentDb** ersetzt. Sie befindet sich im Modul **mdlShop** der Datenbank (Listing 3). Die im Kopf des Moduls deklarierte Objektvariable **thisdb** bekommt beim ersten Aufruf der Funktion den Inhalt von **CurrentDb** verpasst. Forthin gibt sie deren Wert zurück. Der Vorteil der Funktion ist, neben kürzerer Schreibweise, eine bessere Performance.

Artikel im Warenkorb

Für den Warenkorb ist dieselbe Tabelle verantwortlich, wie für die Bestellungen: **tblBestelldetails**. Ohne das Formular für den Warenkorb zu öffnen können bereits im Shop-Formular die hinzugefügten Artikel inspiziert werden. Dazu klicken Sie mit der Maus auf den Button Warenkorb rechts. Sogleich klappt eine Combobox mit der Liste der bestellten Produkte auf (Bild 4). Der Vorgang mag ungewöhnlich sein. In Webshops passiert dies häufig beim Überfahren der Maus auf ein Warenkorbsymbol. Dies nachzustellen wäre hier des Aufwands etwas zu viel gewesen. Die Combobox soll genügen. Ihre Datenherkunft ist an eine Abfrage gebunden, die folgenden Text hat:

```
SELECT
  "- " & [tblArtikel].[Produkt] AS P,
  tblArtikel.ID,
  tblArtikel.IDKategorie,
  tblBestellDetails.BestellID
FROM tblArtikel
INNER JOIN
tblBestellDetails ON tblArtikel.ID =
   tblBestellDetails.ArtikelID
WHERE tblBestellDetails.BestellID =
   [Forms]![frmShop]![txtBestellID]
```

Die Datensätze bilden sich über den **JOIN** aus der Verknüpfung zwischen **tblBestellDetails** mit **tblArtikel**. Da nur die aktuelle Bestellung berücksichtigt werden soll, filtert die **WHERE**-Bedingung nach der **BestellID**. Und die kommt direkt aus dem Formular **frmShop**, in dem das unsichtbare Textfeld **txtBestellID** vorhanden ist. Superelegant ist derlei nicht, aber als Datenherkunft für ein Kombinations-

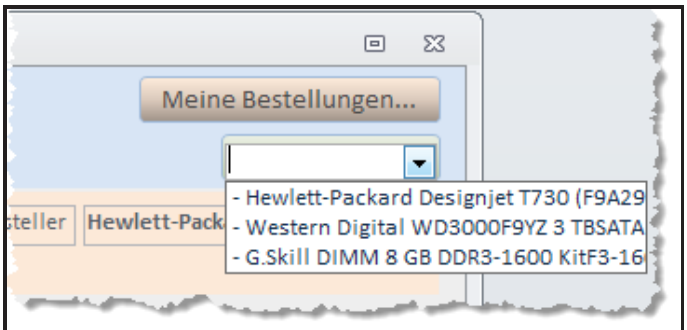


Bild 4: Der Inhalt des Warenkorbs kann über eine Combobox eingesehen werden, die sich hinter dem Button befindet

feld innerhalb des Formulars ist so ein fester Bezug durchaus akzeptabel. Die **Requery**-Anweisung auf **cbWarenkorb** führt zur Neuauswertung der SQL-Abfrage, so dass ihr Datenbestand immer aktuell ist.

Das Ausklappen der Combobox erledigt die Routine in Listing 4. Bei Rechtsklick auf die Schaltfläche **Warenkorb** (**cmdWarenkorb**) ereignet sich das **MouseDown**-Event. Steht der Parameter **Button** auf **2**, so handelt es sich um die rechte Maustaste. Dann stellt die Prozedur den Fokus auf die Combobox **cbWarenkorb** und bewirkt über die Methode **Dropdown** deren Ausklappen. Sie gerät dadurch in den Vordergrund, obwohl sie im Formularentwurf in den Hintergrund verfrachtet wurde. Bei Verlassen des Steuerelements wird automatisch wieder der Ausgangszustand hergestellt.

Beim normalen Linksklick auf den Button **Warenkorb** ruft dieser die **Click**-Prozedur auf und öffnet das Formular **frmWarenkorb** (Bild 5). Allerdings auch nur dann, wenn sich im Warenkorb Artikel befinden. Ansonsten erfolgt eine Meldung, dass der Warenkorb leer sei. Die erste Zeile in Listing 5 wertet über **COUNT(*)** aus, wie viele Datensätze die Tabelle **tblBestellDetails** zur aktuellen **BestellID** aufweist. Ist die Anzahl **0**, so kommt es zur Meldung und andernfalls zum Öffnen des Formulars.

Warenkorbformular

Das gestartete Warenkorbformular **frmWarenkorb** (Bild 5) weist die folgenden Grundeinstellungen auf. Die Eigenschaft **Popup** steht auf **Ja** und die Eigenschaft **Gebunden** ebenfalls. Dadurch steht es im Vordergrund vor dem Shop-Formular, auf welches nun kein Zugriff mehr möglich ist. Als Ansichtsmodus ist **Endlosformular** vorgesehen. Im Detailbereich finden sich die Felder für die Bestelldetails, also die Produkte mit deren Artikelnummer, der Bezeichnung, der gewünschten Anzahl, dem Preis und der Verfügbarkeit. Bis auf das Steuerelement für die **Anzahl** sind alle weiteren gesperrt (Eigenschaftenblatt für das Steuerelement | **Daten** | **Gesperrt** = **Ja**). Die Anzahl kann der Kunde im Warenkorb noch ändern.

```
Private Sub cmdWarenkorb_MouseDown(Button As Integer, Shift As Integer, _
                                   X As Single, Y As Single)
    If Button = 2 Then
        Me!cbWarenkorb.SetFocus
        Me!cbWarenkorb.Dropdown
    End If
End SUb
```

Listing 4: Ausklappen der Combobox beim Maustaste Ab-Ereignis des Buttons

```
Private Sub cmdWarenkorb_Click()
    Dim n As Long

    n = CurDB.OpenRecordset("SELECT COUNT(*) FROM tblBestellDetails" & _
                            " WHERE BestellID=" & BestellID)(0)
    If n = 0 Then
        MsgBox "Ihr Warenkorb ist leer!", vbExclamation, sLogo
    Else
        DoCmd.OpenForm "frmWarenkorb", , , , , , BestellID
    End If
End SUb
```

Listing 5: Das Starten des Warenkorbformulars hängt von der Anzahl der aktuell bestellten Artikel in tblBestellDetails ab

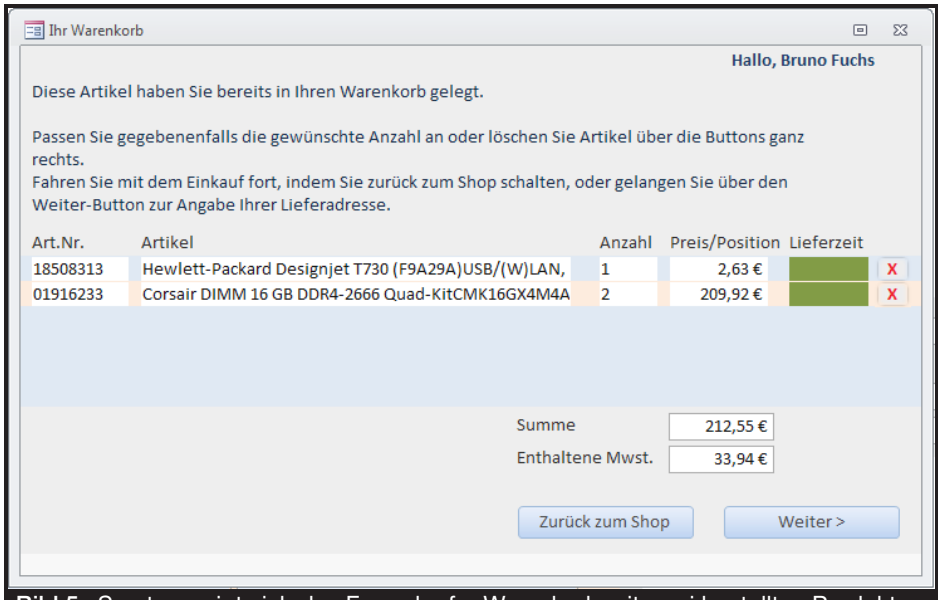


Bild 5: So etwa zeigt sich das Formular frmWarenkorb mit zwei bestellten Produkten

Die Felder für Rechnungssumme und enthaltene Mehrwertsteuer sind im Fußbereich des Formulars platziert. Ganz oben gibt es noch ein Label, das Sie, Herrn Fuchs, persönlich anspricht, falls Sie sich bereits registriert und eingeloggt hatten. Und zuunterst geht es über den Button **Weiter >** zum nächsten Bestellschritt, oder über **Zurück zum Shop** schließt sich das Formular wieder.

Im Entwurf des Formulars (Bild 6) gibt es nicht viele Besonderheiten zu entdecken. Seine Datenherkunft ist über einen SQL-Ausdruck an die Tabelle **tblBestelldetails** geknüpft, wobei hier noch die Artikeltabelle für die Produktbezeichnung angeflanscht ist.

Interessant allerdings ist das Textfeld **txtLieferzeit** für die **Verfügbarkeit**. Es ist an kein Datenfeld gebunden, zeigt also auch keinen Text an. Seine **Hintergrundfarbe** aber ist über **Bedingte Formatierung** mit dem Wert von **IDVerfuegbarkeit** des Datensatzes verknüpft. Der Dialog mit den Ausdrücken zur **Bedingten Formatierung** des Felds wird aus Bild 7 ersichtlich. Sie öffnen ihn über das Kontextmenü des Steuerelements und dessen Eintrag **Bedingte Formatierung...**. Hier wurden drei Regeln festgelegt, die jeweils die Hintergrundfarbe des Textfelds steuern. Je nach dem Wert von **IDVerfuegbarkeit** kommt eine andere Farbe zum Zug. Diese Werte finden Sie wiederum in der Tabelle **tblVerfuegbarkeit**.

Die Programmierung des Formulars verlangt einige Erläuterungen. Da wäre zunächst das Ereignis **Beim Öffnen** (**Form_Open**), welches die Routine in Listing 6 anwirft. Sie weist der globalen Variablen BestID, deren Wert im weiteren Verlauf auch noch für das Kassenformular benötigt wird, den dem Formular beim Starten (Listing 5) verabreichten Parameter **OpenArgs** zu. Ist dieser Wert nicht

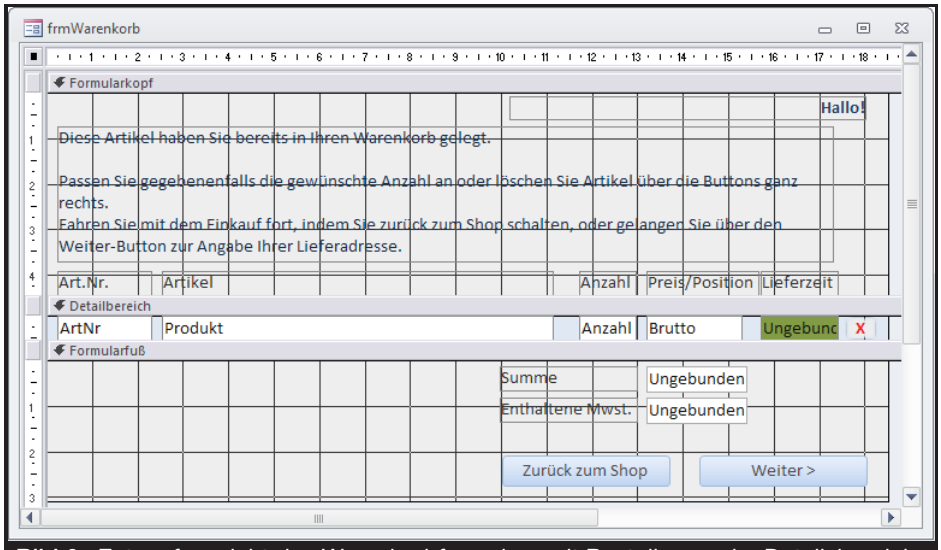


Bild 6: Entwurfsansicht des Warenkorbformulars mit Bestellungen im Detailsbereich

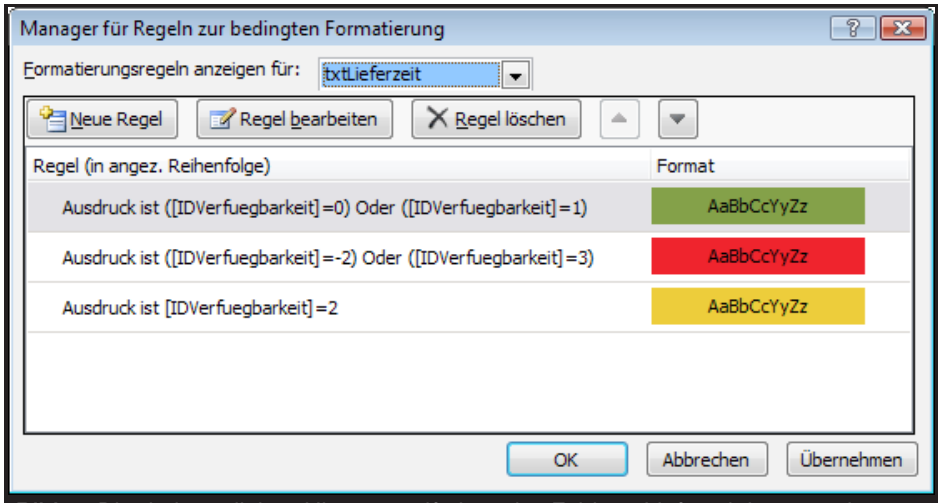


Bild 7: Die drei möglichen Hintergrundfarben des Felds txtLieferzeit hängen über Bedingte Formatierung vom Wert des Datenfelds IDVerfuegbarkeit ab

```
Private Sub Form_Open(Cancel As Integer)
    BestID = Nz(Me.OpenArgs, 0)
    If BestID = 0 Then
        Cancel = True
    Else
        Me.Filter = "BestellID=" & BestID
        Me.FilterOn = True
    End If
End SUb
```

Listing 6: Die Ereignisprozedur Open des Formulars wertet die OpenArgs aus

Datei- und Verzeichnisoperationen mit der Shell

Zu den häufigsten Aufgaben außerhalb einer Access-Datenbankanwendung gehört sicher der Zugriff auf Dateien und Verzeichnisse. Ob Sie nun externe Daten exportieren oder importieren, Dokumente fernsteuern, oder Eigenschaften von Dateien ermitteln möchten, all dies erfordert Navigation in Verzeichnissen und das Ansprechen von Dateien. Und das möglicherweise auch noch innerhalb einer Netzwerkstruktur. Die VBA-Grundfunktionen machen es Ihnen da nicht immer leicht. Über die Shell von Windows kommen Sie jedoch spielend zum Ziel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1610_ShellObjekte.accdb**

VBA-Funktionen für Dateien und Verzeichnisse

Die Klasse **FileSystem** der VBA-Bibliothek gibt im Objektkatalog eine nur dürftige Liste von Methoden aus, über die verschiedene Datei- und Verzeichnisoperationen ausgeführt werden können. Dieses altehrwürdige Ensemble stammt noch aus Zeiten, in denen **Visual Basic** eine gewichtige Rolle spielte, und das war Mitte der 90er-Jahre, als noch erheblich geringere Anforderungen gestellt wurden. Seitdem hat sich am Befehlsbestand nichts geändert und die Funktionen weisen leider immer noch Mängel auf.

Die Liste ist schnell beschrieben. Zu den Verzeichnisfunktionen gehören diese:

CurDir ermittelt das aktuelle Verzeichnis. **ChDir** wechselt zu einem anderen, **ChDrive** zu einem anderen Laufwerk. Derlei braucht heute niemand mehr – es spielte eher unter **DOS** eine Rolle. **MkDir** erzeugt ein neues Verzeichnis mit dem im Parameter angegebenen. Leider funktioniert das nur dann, wenn das oberste Verzeichnis bereits existiert. Das neue kann nur eine Ebene tiefer angelegt werden. Ähnliches gilt für **RmDir**, was ein Verzeichnis löscht. Die Anweisung klappt nicht, wenn ein Verzeichnis nicht leer ist.

Und das war's auch schon! Die Dateimethoden beschränken sich nun auf folgende:

Kill löscht die angegebene Datei. **FileCopy** kopiert eine Datei in den im zweiten Parameter angegebenen Ordner. Verschieben geht nicht. Es ist nur durch die Kombination aus **FileCopy** und anschließendem **Kill** zu realisieren. **FileLen** gibt die Dateigröße zurück, **FileDateTime** das Erstelldatum, **GetAttr** die Zugriffsattribute.

Lesend und schreibend greifen Sie über die **Open**-Anweisung auf eine Datei zu. Das ist wohl von allen der wichtigste Befehl. Und schließlich lassen sich über **Dir** sowohl Dateien, wie auch Verzeichnisse, enumerieren oder nach ihrer Existenz befragen.

Mindestens zwei Kritikpunkte sind für sämtliche dieser Dateisystemfunktionen hervorzubringen. Zum einen scheitern sie häufig an Objekten, deren Namen in Unicode formatiert sind und Sonderzeichen enthalten. Hier ruft VBA schlicht einen Fehler aus. Zum anderen – und das ist viel schwerwiegender! – kommen sie nicht mit Netzwerkpfaden klar. Auf die Dateien eines Server in der Syntax **\\serverxy\c\...** besteht kein Zugriff. Hier wird dann ein nicht existierendes Verzeichnis bemängelt oder eine falsche Dateinummer gemeldet. Behelfen können Sie sich da dann nur mit zusätzlich eingerichteten Netzlaufwerken.

Einst führten diese Beschränkungen zu einer Menge Programmierlösungen unter Verwendung von **Windows-API**-Funktionen, die im Internet fröhliche Urstände feierten. Das aber ist inzwischen obsolet.

Die Windows-Shell-Bibliothek

Es lohnt sich uneingeschränkt, auf diese **COM-Bibliothek**, die grundsätzlich jedes Windows mit sich bringt, ein Auge zu werfen, denn sie ermöglicht einfache Lösungen, die auf andere Weise viel umständlicher zu erreichen wären. Sie hängen sie in ihr VBA-Projekt ein, indem Sie zunächst einen Verweis auf **Microsoft Shell Controls And Automation** setzen. Verantwortlich für die Bibliothek ist die System-Dll **shell32**. So lautet auch der Name der Bibliothek, wenn

Sie sie dann unter VBA ansprechend möchten. Es gibt genau ein Objekt in ihr, das die Zentrale für alle weiteren Vorgänge darstellt, nämlich das **Shell**-Objekt.

Sie erzeugen das Objekt über den **New**-Operator:

```
Dim oShell As Shell32.Shell
Set oShell = New Shell32.Shell
```

Nun lassen sich über dieses Objekt allerlei Dinge anstellen, wie ein Blick in den Objektkatalog zeigt, nachdem Sie im Kombinationsfeld links oben **Shell32** auswählten und in der Klassenliste **Shell** markierten. Wir lassen hier jedoch alle Methoden unbeachtet, bis auf eine sehr mächtige, die sich **NameSpace** nennt.

NameSpace

Die Funktion gibt zu dem im Parameter übergebenen Ausdruck ein Verzeichnisobjekt zurück. Dieses ist vom in der Verweisbibliothek definierten Typ **Folder**, mit dem sich anschließend weitere Operationen ausführen lassen. Als Parameter für NameSpace kommen sowohl Strings, wie auch Zahlen infrage, wie wir gleich sehen werden.

Ein kleines Bespiel macht alles deutlicher:

```
Dim oShell As New Shell32.Shell
Dim oFld As Shell32.Folder
Set oFld = _
        oShell.NameSpace("c:\windows")
Debug.Print oFld.Title
```

Im VBA-Direktfenster wird bei Ausführung dieses Schnipsels der Text **Windows** ausgegeben. Das Folder-Objekt **oFld** hat diesen Titel, der nicht mit dem Pfad zu verwechseln ist. Den erhält man erst über einen Umweg:

```
Debug.Print oFld.Self.Path
> c:\windows
```

Das ist soweit noch nicht sonderlich aufregend. Die Besonderheit der **Windows-Shell** liegt jedoch darin begründet, dass sie all ihre Objekte wie Verzeichnisse und Dateien behandeln kann, also auch Objekte außerhalb des Dateisystems. Der Explorer zeigt das deutlich. Deshalb kann man **NameSpace** auch unterschiedliche Werte für den Parameter zuweisen, eben jene, die auch der Explorer verarbeiten kann. Hier einige Beispiele für mögliche Parameter:

```
c:\windows\
\\nas\public\images
3
http://www.access-im-unternehmen.de
ftp://ftp.rz.uni-wuerzburg.de/pub
e:\dokumente\alles.zip
::{21EC2020-3AEA-1069-A2DD- _
                        08002B30309D}
```

Da kann die **Dir**-Funktion von VBA wohl nicht mithalten! Die erste Zeile bezieht sich auf den lokalen Pfad von Windows. Die zweite ist eine ein Serverpad im Netzwerk – für die Shell kein Problem! Die dritte Zeile übergäbe einfach eine Zahl. NameSpace nimmt dann an, dass es sich um einen der im System verankerten sogenannten **Special Folders** handelt, die jeweils eine eindeutige **ID** besitzen. Das sind vorgegebene Windows-Spezialordner, wie etwa der **Desktop**, welcher über den Parameterwert **0** anzusprechen wäre. Mit der **3** hingegen erhalten Sie als **Folder.Title** die Bezeichnung **Systemsteuerung**, denn auch die ist ein virtueller Ordner. Und was wäre deren Pfad? **Folder.Self.Path** gibt dies aus:

```
::{21EC2020-3AEA-1069-A2DD- _
                        08002B30309D}
```

Es handelt sich hier um einen **GUID**-String, da es ja, im Gegensatz zum Desktop, keinen physischen Ordner für die Systemsteuerung gibt. Geben Sie diesen String einmal spaßeshalber in die Adresszeile des Explorers ein. Sie gelangen damit tatsächlich zur Systemsteuerung!

Selbst Internet-**URLs** kann **NameSpace** verarbeiten, wie die beiden folgenden Zeilen der Beispielparameter zeigen. Sowohl das **HTTP**-Protokoll, wie auch das **FTP**-Protokoll, werden klaglos angenommen. Hier kommen ebenfalls gültige **Folder**-Objekte zustande!

Noch krasser ist die Übergabe der **ZIP**-Datei **alles.zip**. Denn auch der Explorer behandelt **ZIP**-Archive wie Ordner. Deshalb ist ein ZIP-Archiv automatisch auch bei **NameSpace** ein gültiges Ordner-Objekt.

Hier wird schon deutlich, dass die Möglichkeiten der Shell-Objekte weit über das hinausgehen, was VBA an Bordmitteln zu bieten hat.

FolderExists

Aus dem bisher Erläuterten geht die einfachste Anwendung hervor, nämlich der Test, ob ein Verzeichnis denn

existiert. Übergeben Sie **NameSpace** nämlich einen ungültigen String oder eine ungültige Zahl, so ereignet sich kein Fehler. Stattdessen ist das zurückgegeben **Folder**-Objekt einfach leer, hat also quasi den Wert **Nothing**. Das lässt sich nun leicht überprüfen, wie die Routine **FolderExists** in Listing 1 zeigt. Ihr übergeben Sie das fragliche Verzeichnis als Parameter.

```
Function FolderExists(sFolder As String) As Boolean
    Dim oFld As Shell32.Folder

    Set oFld = oShell.NameSpace(sFolder)
    If Not oFld Is Nothing Then FolderExists = True
End Function
```

Listing 1: Überprüfen der Existenz eines Verzeichnisses

```
Public m_Shell As Shell32.Shell

Public Function oShell() As Shell32.Shell
    If m_Shell Is Nothing Then Set m_Shell = New Shell32.Shell
    Set oShell = m_Shell
End Function
```

Listing 2: Funktion zum Zurückgeben des Shell-Objekts

In der Objektvariablen **oFld** erhalten Sie aus **NameSpace** das **Folder**-Objekt. Ist dieses nicht **Nothing**, so existiert das Verzeichnis, womit der Rückgabewert der Funktion auf **True** gesetzt wird. Wohlgemerkt: Nicht nur lokale Verzeichnisse können mit dieser Funktion befragt werden, sondern ebenso Netzwerkpfade und Internetpfade!

Nutzt man die **Shell**-Funktionen in mehreren Prozeduren, so ist es übrigens hilfreich, statt dem dauernden Neuerzeugen des **Shell**-Objekts für dieses eine gesonderte Funktion zu schreiben. Auch **FolderExists** verwendet eine solche (**oShell**), wie sie in Listing 2 abgebildet ist. Hier ist im Modulkopf die globale Variable **m_Shell** deklariert. In der Prozedur wird erst getestet, ob deren Inhalt leer ist. In diesem Fall wird per **New** eine neue **Shell**-Instanz erzeugt und ihr zugewiesen. Die Funktion gibt schließlich den Wert der Objektvariablen **m_Shell** zurück.

Special Folders

Es gibt keine genaue Aufstellung von Microsoft, welche Zahlenwerte für **NameSpace** welchem Spezialverzeichnis zuzuordnen sind. Das variiert je nach Betriebssystem und Sprache. Bei den gebräuchlichsten stimmen sie allerdings überein.

Den Desktop erreichen Sie immer über die **0**, das Startmenü über **22**, oder das Systemverzeichnis über **37**. Diese Werte sind im **Windows-SDK** in den **Header**-Dateien deklariert. Doch machen Sie für Ihr System doch einfach selbst die Aufstellung und verwenden die Routine **StoreSpecialFolders** im Modul **mdlSpcFolders** der Beispieldatenbank!

Die in Listing 3 dargestellte Prozedur speichert alle möglichen Werte in der Tabelle **tblShellFolders**, die zuerst über die **DELETE**-Anweisung geleert wird. Dann durchläuft eine Schleife auf den Zähler **i** die Werte von **0** bis **255** und testet **NameSpace** damit.

Die Variable **sPath** nimmt den Pfad des so ermittelten Ordners **fld** entgegen. Statt des Überprüfens auf **Nothing** für das **Folder**-Objekt wird hier ein **On Error Resume Next** eingesetzt, denn bei nicht existierendem Ordner erzeugt **Self.Path** einen

```
Sub StoreSpecialFolders()
    Dim fld As Folder
    Dim i As Long
    Dim sPath As String
    Dim rs As DAO.Recordset

    CurrentDb.Execute "DELETE FROM tblShellFolders"
    Set rs = CurrentDb.OpenRecordset("SELECT * FROM tblShellFolders", dbOpenDynaset)
    On Error Resume Next
    For i = 0 To 255
        Set fld = oShell.NameSpace(i)
        Err.Clear
        sPath = fld.Self.Path
        If Err.Number = 0 Then
            rs.AddNew
            rs!SFID = i
            rs![Name] = fld.Title
            rs!Path = sPath
            rs.Update
        End If
    Next i
    rs.Close
End Function
```

Listing 3: Alle Windows-Spezialordner kommen in die Tabelle tblShellFolders

Fehler. Ist die Fehlernummer aber **0**, so wird dem Recordset **rs** ein Datensatz hinzugefügt. Das Feld **SFID** nimmt dabei den Zahlenwert für **NameSpace** auf, **Name** seinen Titel, und **Path** den vollen Verzeichnispfad.

Nach dem Durchlaufen der Prozedur können Sie die Tabelle in der Datenblattansicht begutachten. Bild 1 gibt einen Ausschnitt der 56 Spezialordner wieder. In Bild 2 sind diese über den Spaltenfilter auf den Textbeginn '**::**' reduziert. Die Ansicht gibt damit alle virtuellen Ordner von Windows aus.

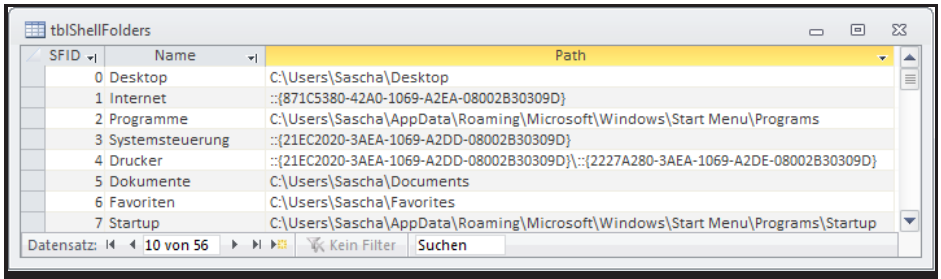


Bild 1: Die Windows-Spezialordner sind in der Tabelle tblShellFolders abgespeichert

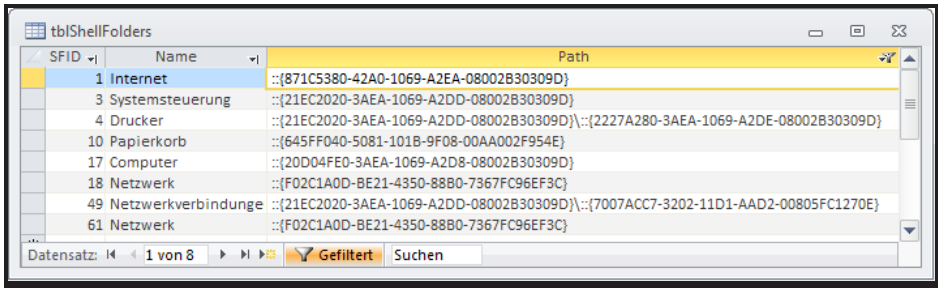


Bild 2: Hier ist die Tabelle nach allen Ordnerpfaden gefiltert, die mit :: beginnen

Sie haben nun die Auswahl, wie Sie zum Ordner für Windows gelangen:

```
? oShell.NameSpace(36).Self.Path
```

oder

```
? Environ("Windows")
```

Beides gibt dasselbe aus. **Environ** ist eine Funktion, die seltsamerweise nicht in der Klasse **FileSystem** von VBA untergebracht ist. Sie gibt aber keine Spezialordner zurück, sondern die Umgebungsvariablen von Windows. Die enthalten aber erheblich weniger vorgegebene Orte, als **NameSpace** auszugeben vermag.

FolderItems

Bisher beschäftigten wir uns nur mit Verzeichnisobjekten. Ein **Folder**-Objekt ist aber nur der Ausgangspunkt für weitere Elemente. Seine wichtigste Eigenschaft ist die Aufzählung **Items** vom Typ **FolderItems**. Das sind die Elemente, die der Ordner enthält.

Man kann diese Elemente in einer **For-Each**-Schleife enumerieren:

```
Dim oFld As Shell32.Folder
Dim oItm As  Shell32.FolderItem
Set oFld = NameSpace("c:\windows")
For Each oItm In oFld.Items
    Debug.Print oItm.Name
Next oItm
```

In diesem Beispiel werden alle Elemente des Windows-Verzeichnisses ausgegeben. Die **Items**-Auflistung des **Folders** wird durchlaufen und zu jedem **FolderItem**-Objekt **oItm** darin dessen Name erhalten.

Ähnlich, wie dies auch bei der Funktion **Dir** der Fall ist, können diese Elemente vom Typ Datei oder Verzeichnis sein. Hier kommen noch virtuelle Elemente hinzu, denn die Systemsteuerung etwa enthält weder Dateien, noch Ordner. Zum Glück ist die Unterscheidung einfach:

```
? oItm.IsFileSystem
```

Hier kommt ein Wahr heraus, wenn es sich um ein Element des Dateisystems handelt. Ob das dann eine Datei oder ein Verzeichnis ist, gibt eine weitere Funktion wieder:

```
? oItm. IsFolder
```

Bei Dateien wäre die Rückgabe False. Und für die gibt es noch eine zusätzliche Spezialität:

```
? oItm.IsLink
```

Das gibt immer dann **True** zurück, wenn es sich um eine Verknüpfung handelt. Der Desktop etwa ist voll davon.

Ein **FolderItem** kennt daneben noch einige weitere Methoden. Die Dateigröße etwa ermitteln Sie über **oItm.Size**, das Datum der letzten Änderung mit **oItm.ModifyDate** und den Typ des Elements über **oItm.Type**. Letzteres gibt einen Text aus, wie etwa **Dateiordner**. Wie weitere

Eigenschaften des Elements erhalten werden können, folgt später.

Natürlich können Sie ein Element eines Ordners auch gezielt ansprechen. Dafür kommen zwei Methoden in Betracht. Die eine benutzt wieder die **Items**-Auflistung:

```
Set oFld = NameSpace("c:\windows")
Set oItm = oFld.Items.Item( _
                    "notepad.exe")
Debug.Print oItm.Name
```

Sicherer jedoch ist eine spezielle Methode des **Folder**-Objekts selbst, weil sie fehlerrobuster ist:

```
Set oFld = NameSpace("c:\windows")
Set oItm = oFld.ParseName( _
                    "notepad.exe")
Debug.Print oItm.Name
```

ParseName nämlich kann nicht nur Elemente des aktuellen Ordners wiedergeben, sondern auch untergeordnete:

```
Set oFld = NameSpace("c:\windows")
Set oItm = oFld.ParseName( _
           "system32\drivers\etc")
Debug.Print oItm.Name
> 'etc'
```

Hier werden also mehrfache Unterordner ermittelt.

Mit diesem Wissen im Gepäck können Sie sich auch an die Behandlung von Dateien machen.

FileExists

Ob eine Datei existiert, kann die Funktion **FileExists** in Listing 4 herausfinden. Wieder wird hier ein **Folder**-Objekt **oFld** erhalten, aus dem es sich, so es nicht **Nothing** ist, über **ParseName** ein **FolderItem oItm** holt. Ist dieses Objekt gültig, abermals durch Vergleich mit **Nothing** ermittelt, so existiert die Datei:

```
? FileExists("c:\windows\notepad.exe")
> Wahr
```

Die Sache ist allerdings nicht ganz so einfach, wie es beim ersten Blick auf die Prozedur den Anschein hat. Denn übergeben wird ihr ja ein voller Dateipfad. Aus diesem muss erst der Verzeichnispfad extrahiert werden, um ihn an die **NameSpace**-Methode zu geben. Das übernimmt eine zusätzlich Hilfsfunktion **GetFileFolder** (Listing 5). In **sExpression** kommt der Dateipfad hinein. Andererseits werden die String-Variablen für den zurückgegebenen Ordner und den Dateinamen in **sFolder** und **sFile** spezifiziert. Als Charakteristikum für die Trennung zwischen Pfad- und Dateiname hält das letzte **Backslash** her, welches über **InstrRev** ermittelt wird. Das macht man in der Regel so, doch hier sind ja auch noch **URLs** für den Internetzugriff zu berücksichtigen, die stattdessen **Slashs** enthalten. Das wird in der eingeschobenen Bedingung ausgewertet. Sind gar keine **Slashs** oder **Backslashs** vorhanden, so bleibt die Rückgabe leer.

Sie können mit **FileExists** also sogar auswerten, ob sich eine bestimmte Datei einem **FTP**-Verzeichnis im Internet befindet. Oder auch, ob ein **ZIP**-Archiv eine solche beheimatet. Dabei sollten Sie allerdings auf die Syntax für den Parameter achten:

```
Function FileExists(sFile As String) As Boolean
    Dim sFld As String
    Dim sFil As String
    Dim oFld As Shell32.Folder
    Dim oItm As Shell32.FolderItem

    GetFileFolder sFile, sFld, sFil
    Set oFld = oShell.NameSpace(sFld)
    If Not oFld Is Nothing Then
        Set oItm = oFld.ParseName(sFil)
        If Not oItm Is Nothing Then FileExists = True
    End If
End Function
```

Listing 4: Funktion zum Zurückgeben des Shell-Objekts

```
Sub GetFileFolder(sExpression As String, sFolder As String, sFile As String)
    Dim n As Long

    n = InStrRev(sExpression, "\")
    If n = 0 Then
        n = InStrRev(sExpression, "/")
        If n = 0 Then Exit Sub
    End If
    sFolder = Left(sExpression, n)
    sFile = Mid(sExpression, n + 1)
End Sub
```

Listing 5: Funktion zum Zurückgeben des Shell-Objekts

Erfahrungen mit LibreOffice Base

Hin und wieder lohnt ein Blick über den Tellerrand: Gibt es Alternativen zu Microsoft Access und wenn ja, dann welche? Bei Recherchen stoßen Sie möglicherweise auf die Anwendung Base als Teil der Open-Source-Pakete OpenOffice und LibreOffice. Wir haben uns angesehen, wie dieses Datenbanksystem zu handhaben ist und mit welchen Features es aufwarten kann. Hier finden Sie einen kurzen Überblick über die grundlegendsten Eigenschaften.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1611_Suedsturm.mdb**

Open Source im Unternehmenseinsatz

Nicht immer muss es **Microsoft Office** sein. In größeren Institutionen schlagen die Lizenzkosten für die unvermeidlichen Office-Pakete deutlich zu Buche. Es existieren **Open-Source**-Lösungen, die im Funktionsumfang an die professionellen Anwendungen heranreichen. Warum da nicht Geld sparen?

OpenOffice und **LibreOffice** sind beide **Open-Source**-Ableger des einstigen **StarOffice** und werden parallel seit bald 20 Jahren kontinuierlich weiterentwickelt. Dabei ist **LibreOffice** der jüngere Zweig, welcher, was die Aktivitäten angeht, **OpenOffice** eindeutig überholt hat.

Das Augenmerk liegt aber bei beiden auf möglichst hoher Analogie zu **Microsoft Office** und zur Austauschbarkeit der Dateiformate, um den Umstieg so reibungslos wie möglich gestalten zu können und die Lernkurven flach zu halten.

Tatsächlich hat **LibreOffice** in Deutschland in zahlreichen Institutionen der Öffentlichen Verwaltung, in Unternehmen und Schulen Einzug gehalten, um die Kosten zu reduzieren. Was die Fähigkeiten zumindest von **Write** (entspricht **Word**) und **Calc** (entspricht **Excel**) angeht, brauchen sich die Anwendungen keinesfalls vor dem Original zu verstecken, wenn die Ansprüche nicht gerade von Spezialisten zu hoch gesteckt werden. Gilt dies denn auch für die Datenbankanwendung **Base**, die sich ebenfalls an Access orientiert?

Dazu eine kurze Geschichte aus dem Umfeld. Der Sohn hatte in der Oberstufe des Gymnasiums einen Kurs in Informatik zu absolvieren. Im Abitursjahrgang ging es um das Erlernen datenbanktechnischer Themen.

Hier wurden auf etwas ungelenke und altbackene Weise die theoretischen Grundlagen erörtert. Abschließend ging es um die praktische Umsetzung, wobei die Aufgabe in der Anlage eines eigenen dokumentierten Datenmodells bestand, welches um einige Auswertungen in Form von Abfragen und SQL-Anweisungen zu bereichern war. Zum Einsatz sollte dazu das in der Schule verfügbare **LibreOffice Base** kommen.

Der Umgang damit war allerdings nicht Gegenstand des Unterrichts gewesen. Benotet wurden schließlich die Datenbankdatei und die Dokumentation derselben. Tja, und ohne die Unterstützung des Autors wären dem Schüler keine 15 Punkte beschert gewesen, denn die Mitschüler berichteten von unzähligen Ungereimtheiten, Fehlermeldungen und Abstürzen in **Base**.

Und ohne Informationen aus diversen Foren im Netz wäre auch der Autor selbst leider nicht weitergekommen und hätte wenig Hilfe leisten können ... Aber alles ist ja einfach, wenn man weiß, wie's geht. Berücksichtigen Sie deshalb besser gleich die folgenden Ausführungen.

Installation von LibreOffice

An der Installation des etwa 200 MB großen Pakets gibt es nichts zu meckern. Es besteht aus einer ausführbaren Setup-Datei, die, ähnlich wie **Microsoft**

Office, eine benutzerdefinierte Auswahl der Office-Bestandteile zulässt, wobei die Möglichkeiten hier vergleichsweise bescheiden ausfallen. **Base** sollten Sie aktiviert haben, **Calc** ebenfalls, während Sie für Testzwecke die anderen Anwendungen auch abwählen können.

Sie können sich für die 32-Bit-Variante entscheiden, oder für die 64-Bit-Variante. Sie werden mit dem 64-Bit-**Base** jedoch Probleme bekommen, sobald Sie auf auf eine Access-Datenbank zugreifen möchten, wenn **Microsoft Office** in der 32-Bit-Version installiert ist.

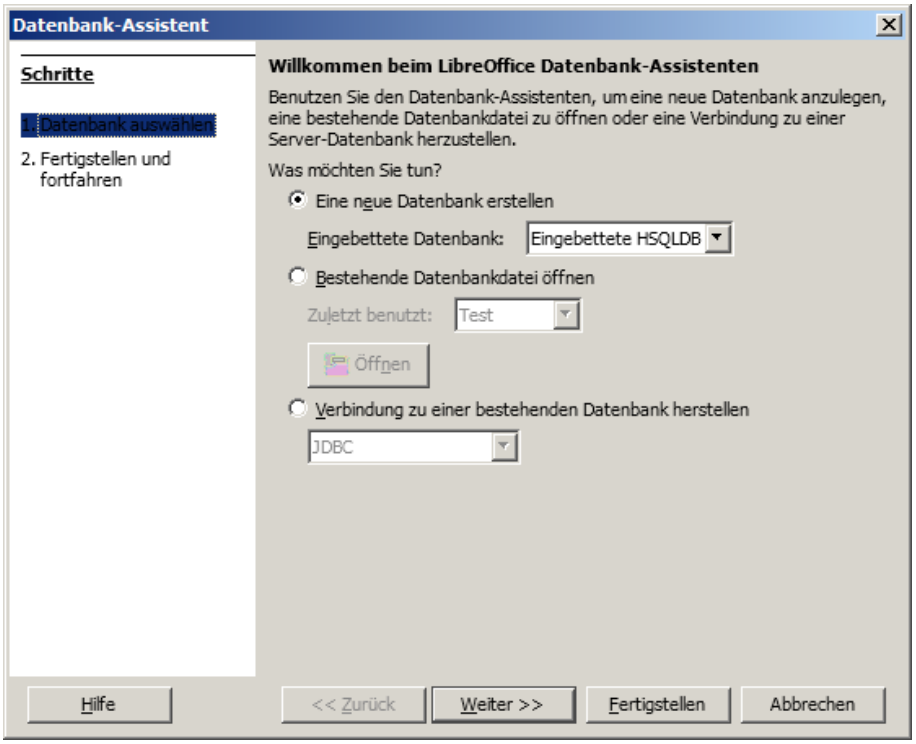


Bild 1: Eine neue Datenbank wird in Base immer über den Assistenten angelegt

Denn Base verwendet für den Zugriff entweder das **ADO** oder **ODBC** von Windows und MS Office. Das 64-Bit-**Base** findet dann keine passenden Treiber, da diese nur in 32-Bit installiert sind. Verwenden Sie deshalb besser gleich das 32-Bit-Setup von **LibreOffice**. Worüber sich das Setup indessen ausschweigt ist die Tatsache, dass **Base** für verschiedenen Zwecke eine **Java-Runtime**-Umgebung benötigt. Sollte diese auf Ihrem System noch nicht vorhanden sein, so holen Sie die Installation nun nach, indem Sie eine der Runtimes von **java.org** oder **Oracle** downloaden.

Es spielt keine Rolle für welches Produkt Sie sich entscheiden, wohl aber, welche Version Sie nehmen. Denn die 32-Bit-Version von **LibreOffice** kann nichts mit einer 64-Bit-Java-Runtime anfangen. Also müssen Sie auch hier darauf achten das **32-Bit-Java** zu installieren.

Weiter gibt es keine Systemanforderungen. Sie können übrigens problemlos sowohl 32-Bit-, wie auch 64-Bit-Versionen parallel installieren. Das gilt gleichfalls für die Java-Runtimes.

Das Anlegen einer Datenbank in Base

Es gibt in **LibreOffice** keine Einzelanwendungen, wie bei Microsoft Office. Sie starten es vielmehr als Gesamtanwendung und entscheiden dann im erscheinenden Navigationsfenster, welche Anwendung Sie benötigen, oder welche Datei Sie öffnen möchten. Unter der Rubrik **Erstellen** links im Anwendungsfenster klicken Sie auf **Base Datenbank**.

Das öffnet den Datenbank-Assistenten (Bild 1), der Ihnen drei Auswahloptionen anbietet. Die zweite (**Bestehende Base-Datei öffnen**) fällt für uns zunächst weg, die dritte (**Verbindung herstellen**) lassen wir an dieser Stelle ebenfalls unberücksichtigt. Denn hier ginge es um den Zugriff auf Fremdsysteme über **ODBC**, **JDBC**, oder **ADO**.

Mit der ersten Option kommen Sie zu einer neuen Eingebetteten **HSQL**-Datenbank, dem **Base** eigenen **DBMS**. So, wie Access die **Office Database Engine** zur Grundlage hat, verwendet Base das Open-Source-DBMS **H2SQL**. Leider! Denn aus unserer Sicht ist diese Engine nicht so vielseitig, fortschrittlich, perfor-

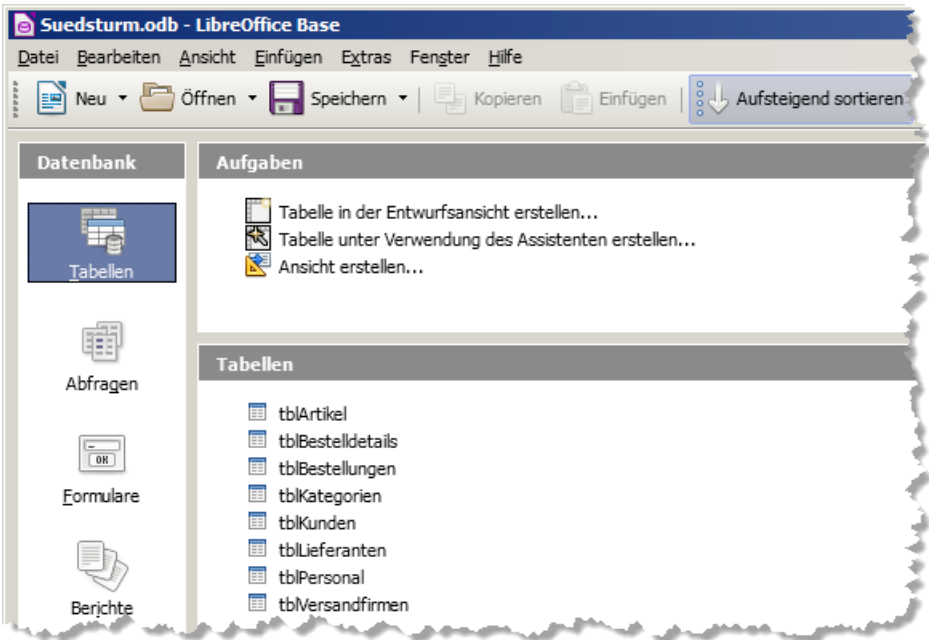


Bild 2: So sieht die Oberfläche von LibreOffice Base mit geladener Datenbank aus

befindet sich der **Aufgaben**-Bereich, der etwas überflüssig wirkt und gut hätte durch einfache Menü- oder Symbolleisten-Schaltflächen ersetzt werden können, da hier grundsätzlich die gleichen drei Befehle zu finden sind. Im Hauptbereich listen sich dann die jeweiligen Elemente auf. In der Abbildung sind bereits einige Tabellen der **Südsturm**-Datenbank aus **Access-im-Unternehmen** zu sehen, die importiert wurden. Wir kommen gleich noch darauf zu sprechen, wie das realisiert wurde.

mant und ausgereift, wie etwa das ähnliche **SQLite**, obwohl dieses genauso schlank daherkommt.

Mit einem Klick auf **Weiter** gelangen Sie zur optionalen Registrierung der Datenbank in **LibreOffice**. Dieser Schritt ist hier überflüssig. Geben Sie abschließend mit **Fertigstellen** den Ort an, an dem Sie die auf **odb** endende Datenbankdatei abspeichern. Daraufhin öffnet sich die GUI, die Oberfläche zur Verwaltung, der Datenbank, wie in Bild 2.

Gemessen am Ribbon von Access mutet diese Oberfläche ziemlich spartanisch an. Die Steuerbereiche, welche den Navigationsbereich von Access ersetzen, teilen sich in drei auf. Links kann man die Abteilungen für Tabellen, Abfragen, Formulare und Berichte ansteuern. Oben

Sie könnten nun Tabellen von Grund auf neu anfertigen. Wir geben hier aber keine umfassende Anleitung, wie Base zu handhaben ist. Einfacher ist es ja wahrscheinlich bestehende Access-Datenbanken zu

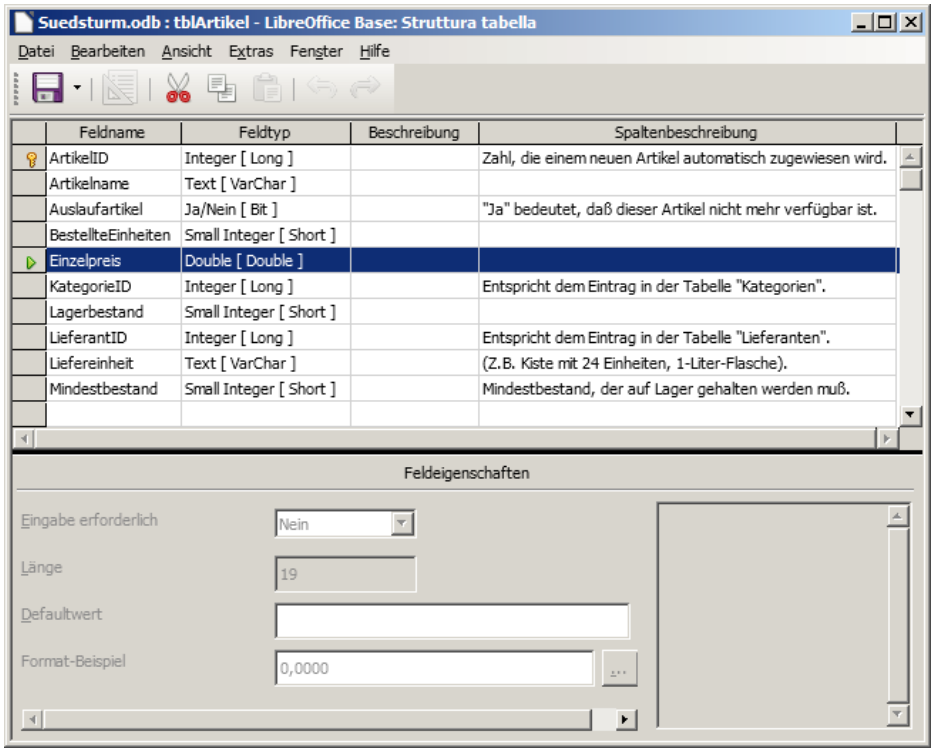


Bild 3: Die Definition der Tabelle tblArtikel im Entwurfsfenster von Base

importieren? Doch eine solche Importmöglichkeit werden Sie nirgendwo finden. Zwar ist Base über ODBC und ADO durchaus imstande, auf eine Access-Datenbank zuzugreifen und sie auszulesen, doch ein direkter Import ist nicht vorgesehen.

Datenbank von Access nach Base importieren

Der Weg ist leider lang und steinig. Sie erzeugen eine neue Datenbank in **Base** und wählen diesmal aus den drei Optionen **Verbindung zu einer bestehenden Datenbank herstellen**. Das Kombinationsfeld darunter sollten Sie auf den Eintrag **Microsoft Access** einstellen. Im nächsten Schritt werden Sie dann zur Auswahl der Datenbankdatei aufgefordert. Bei Access ist dabei nur das **mdb**-Format erlaubt, bei **Access 2007** das **accdb**-Format. Anschließend vergeben Sie einen Namen für die **Base-odb-Datei**. Die Datenbank öffnet sich nun nach etwas Bedenkzeit und zeigt die scheinbar importierten Tabellen alle korrekt an. Wir absolvierten dies für die **Suedsturm.mdb**. Doch was genau haben wir nun vor uns?

Imgrunde stellt **Base** hier ein Frontend für die Access-Datenbank dar. Sie können die Tabellen öffnen und die Daten bearbeiten, so, wie bei

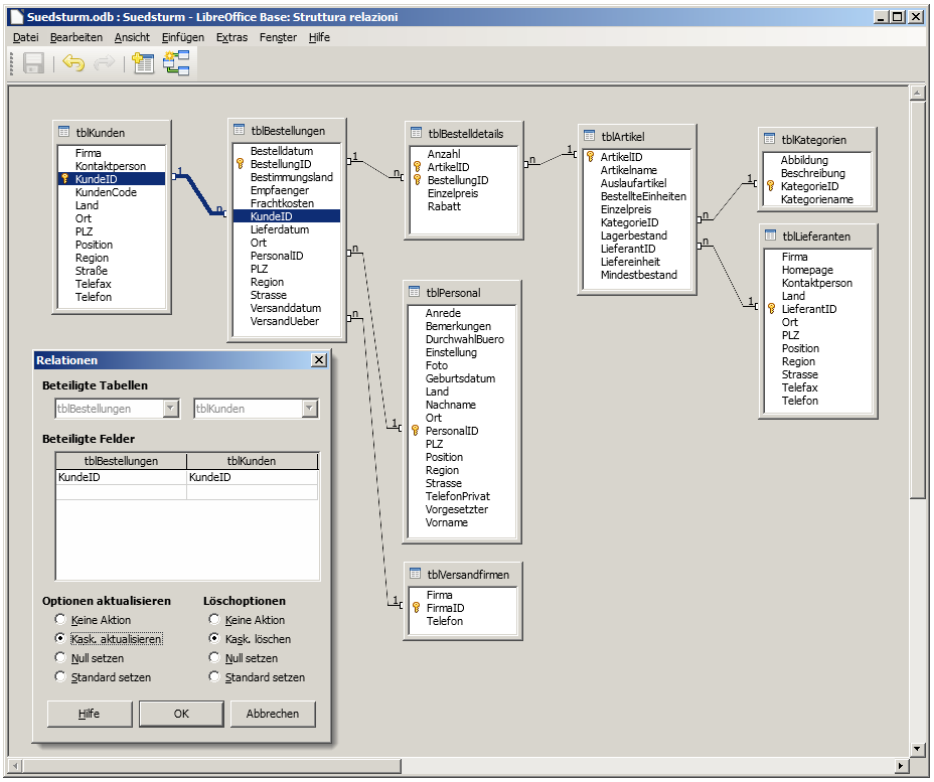


Bild 4: Das Beziehungsfenster von Base ist nach dem Access-Vorbild gestaltet

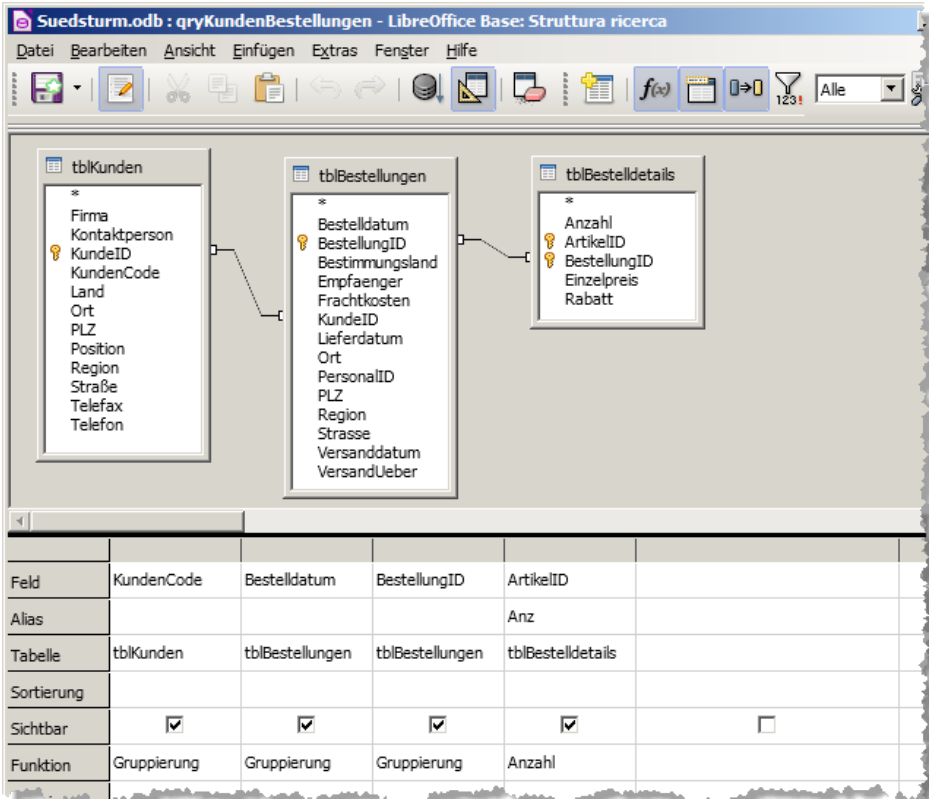


Bild 5: Auch der visuelle Abfrageneditor gleicht dem von Access weitgehend

Outlook-E-Mails importieren

Über die Automationsschnittstelle von Outlook erhalten Sie Zugriff erhalten Sie Zugriff auf fast alle Aspekte der Anwendung. Das Versenden von E-Mails über das Outlook-Objektmodell aus Access heraus etwa wurde bereits in früheren Beiträgen dargestellt. Hier geht es nun um das Auslesen von E-Mails und deren Import in die Tabellen einer Datenbank. Ein einfaches Formular, welches die Mails zur Anzeige bringt, fehlt ebenfalls nicht.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1611_OutlookMails.accdb**

Die Schnittstelle zu Outlook

Für den Zugriff auf Outlook benötigen Sie einen zusätzlichen Verweis im VBA-Projekt Ihrer Datenbank. Sie markieren im **Verweise**-Dialog des Menüs **Extras** den folgenden Eintrag: **Microsoft Outlook xx.0 Object Library**. Das **xx** ist die Versionsnummer Ihres Office-Pakets, welche bei **Office 2010** etwa **14** lautet. Im Objektkatalog finden Sie anschließend das Objektmodell unter der Rubrik **Outlook**, in dem die Klasse **Application** den Ausgangspunkt für alle weiteren Aktionen darstellt.

Vor die Automation, also die Fernsteuerung, von Outlook hat Microsoft allerdings einen Riegel geschoben, da es hier um sicherheitsrelevante Daten gehen kann. In früheren Office-Versionen erschien beim automatisierten Zugriff grundsätzlich der sogenannte **Security Guard**. Das ist ein kleiner Dialog, der anzeigt, wenn offenbar Zugriff auf die Elemente von Outlook erhalten werden soll. Dies ist zu bestätigen, weil sonst die Schnittstelle erst gar nicht freigeschaltet wird. Außerdem ist dieser Zugriff zeitlich eingeschränkt.

In den neueren Versionen wurde dieses Verhalten geändert. Der Dialog erscheint nur dann, wenn kein funktionstüchtiges Antiviren-Programm installiert ist. Die Existenz eines solchen scheint ausreichend zu sein.

Wie die Seite für den **Programmgesteuerten Zugriff** im **Sicherheitscenter** der **Optionen** von Outlook zeigt (Bild 1), sind sämtliche Auswahlmöglichkeiten deaktiviert, wenn Windows ein akzeptiertes Antiviren-Programm vorfindet. In diesem Fall, von dem wir auch ausgehen, brauchen Sie sich um weiter nicht zu kümmern.

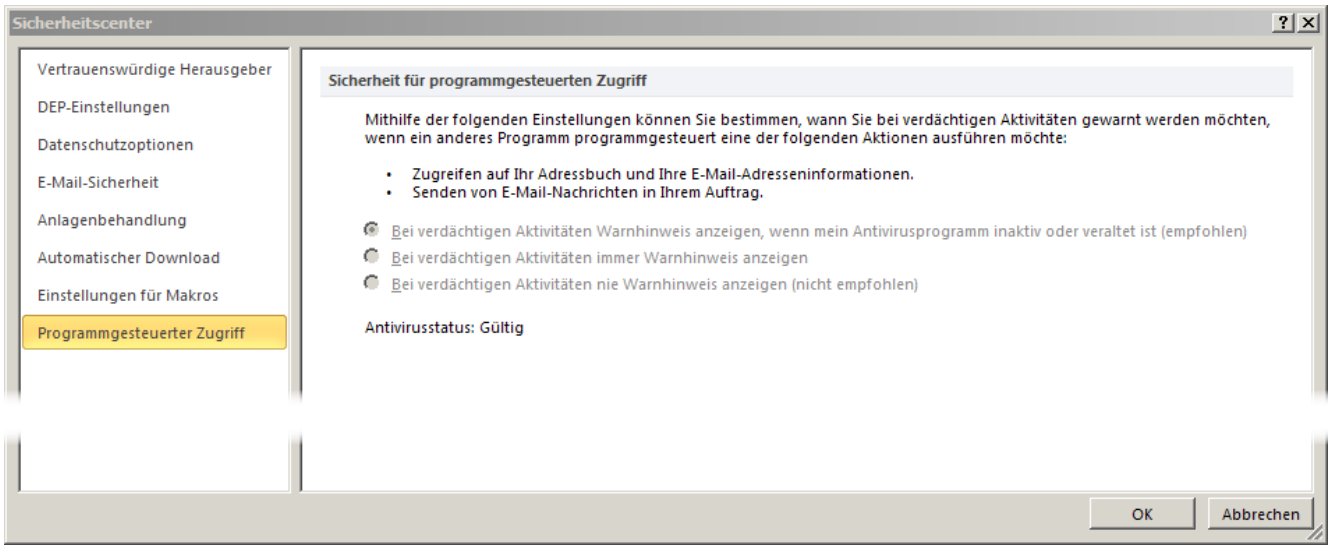


Bild 1: Die Sicherheitseinstellungen von Outlook beim Zugriff über Automation

Outlook-Application

Alle Elemente von Outlook erreichen Sie über dessen **Application**-Objekt. Sie erzeugen eine neue Instanz des Objekts, das die Outlook-Anwendung wiederspiegelt, am besten über diese Funktion:

```
Dim objOL As Outlook.Application
Set objOL = CreateObject( _
              "Outlook.Application")
```

Dabei spielt es keine Rolle, ob Outlook bereits geöffnet ist. Denn Outlook lässt ja auch manuell keine mehrfachen Anwendungsinstanzen zu. **CreateObject** führt damit bei bereits geöffnetem Outlook zum gleichen Resultat, wie **GetObject**.

Über die Variable **objOL** können Sie sich nun daran machen, alle Aspekte der Anwendung zu inspizieren. Möglicherweise benötigen SIe die Variable auch in anderen Prozeduren, und dann macht es Sinn, für sie eine Hilfsfunktion zu erstellen, wie in Listing 1. Auf erneute Instanziierungen der Objektvariablen in Ihren Routinen können Sie nun verzichten, indem Sie einfach die Funktion **OLApp** abrufen, die ihrerseits auf die im Modulkopf definierte Variable **m_OL** zugreift.

War Outlook noch nicht offen, so startet die Routine eine neue Instanz der Anwendung. Davon bekommen Sie zunächst allerdings nichts mit, denn die **Visible**-Eigenschaft von **Application** steht auf **False**. Lediglich das Outlook-Symbol im Infobereich der Taskleiste weist Sie auf den Start hin.

Nun möchten Sie im Prinzip nicht die Anwendung fernsteuern, sondern E-Mails auslesen. Dazu begeben Sie sich eine Ebene tiefer auf die **MAPI**-Umgebung, welche für den E-Mail-Verkehr zuständig ist:

```
Dim objNS As Outlook.NameSpace
Set objNS = OLApp.GetNamespace("MAPI")
```

```
Public m_OL As Outlook.Application

Function OLApp() As Outlook.Application
    If m_OL Is Nothing Then Set m_OL = CreateObject("Outlook.Application")
    Set OLApp = m_OL
End Function
```

Listing 1: Genereller Zugriff auf eine Outlook-Instanz über die Hilfsfunktion OLApp

Dieser **Namespace** erlaubt nun die Navigation durch die Ordner und deren Inhalte von Outlook.

Ordner und Unterordner

Wie der Windows-Explorer auch, gestattet Outlook die Einrichtung von beliebig verschachtelten Ordnern und Unterordnern. Um die Analogie weiter zu treiben: Gibt es unter Windows verschiedene Laufwerke, so kann es unter Outlook auch mehrere **PST**-Dateien geben, die in die Anwendung eingeklinkt wurden und die dann jeweils die oberste Ebene der Ordnerstruktur einnehmen. Unterhalb dieser finden die eigentlichen Verzeichnisse ihren Platz.

Der Zugriff auf die Ordner geschieht über die **Folders**-Auflistung des **Namespace MAPI**. Sie kann komfortabel in einer **For-Each**-Schleife enumeriert werden:

```
Dim objFolder As Outlook.Folder
For Each objFolder in objNS.Folders
    Debug.Print objFolder.Name
Next objFolder
```

Als Ergebnis erhalten Sie hier einen Liste der verwendeten **PST**-Dateien über deren Ordnernamen.

Jeder Folder enthält seinerseits eine Folders-Auflistung für die möglichen Unterordner:

```
For Each objFolder in objNS.Folders
   Debug.Print objFolder.Name, _
               objFolder.Folders.Count
Next objFolder
```

Hier wir zusätzlich zum Namen auch die Anzahl an Unterordnern ausgegeben, die die PST-Dateien aufweisen.

Datenmodell

Bevor es an den weiteren programmatischen Zugriff auf die Outlook-Objekte geht, machen wir uns Gedanken über die Tabellenstruktur, in welche die Outlook-Ordner und die E-Mails importiert werden sollen. Am einfachsten ist das anhand des Beziehungsfensters der Beispieldatenbank (siehe Bild 2) erklärt.

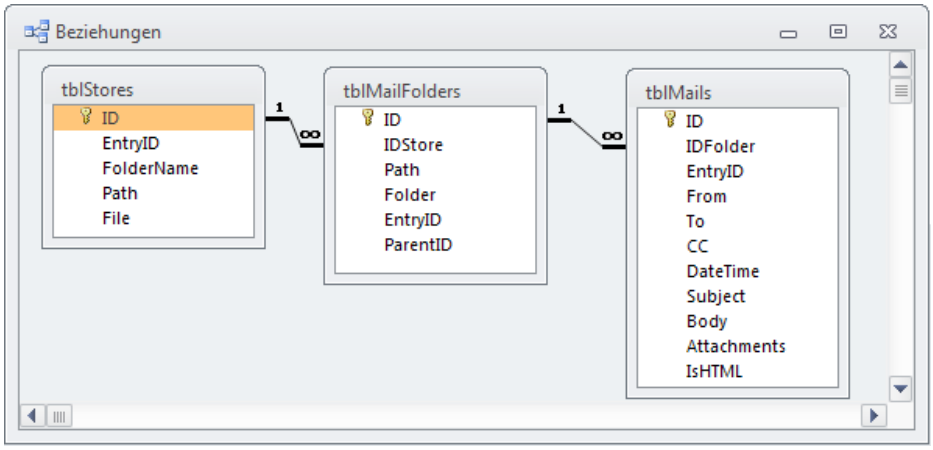


Bild 2: Das Datenmodell der Beispieldatenbank enthält nur drei verknüpfte Tabellen

Die Tabelle **tblStores** nimmt Informationen zu den **PST**-Dateien auf. Es macht Sinn, die Hauptordner dieser Dateien in eine eigene Tabelle zu speichern, um diese von den eigentlichen Verzeichnisstrukturen abzugrenzen. Neben dem Primärschlüssel **ID** (Autowert) steht in **FolderName** der Name der **PST**-Datei, der häufig etwa **Standard** lautet. **File** speichert den physischen Dateipfad zur **PST**-Datei, während **Path** den logischen Verzeichnispfad innerhalb Outlook wiederspiegelt. Bild 3 zeigt einige Beispiele. **EntryID** ist ein Zeichenausdruck, der den **Store**, ähnlich wie ein Schlüssel, eindeutig in Outlook kennzeichnet. Auf den ersten Blick sieht diese **ID** ähnlich aus, wie eine **GUID**, ist jedoch 48 Zeichen lang. Anhand dieser **EntryIDs** kann ein bestimmtes Element in Outlook später immer wieder gefunden werden, auch wenn es verschoben wird. Die überschaubare Routine **GetStorageFolders** in Listing 2 zeigt, wie die Tabelle gefüllt wird.

Zunächst werden über die **Execute**-Anweisung alle Datensätze aus der Tabelle gelöscht. Da im Datenmodell für die verknüpfte Tabelle **tblMailFolders**, die die zu den **PST**-Dateien gehörigen Ordner enthält, referenziell **Löschweitergabe** eingestellt ist, werden automatisch auch aus dieser alle Daten entfernt. Dasselbe gilt übrigens auch für den Inhalt der weiter verknüpften Tabelle **tblMails**, die die E-Mail-Informationen speichert.

Dann werden alle Hauptordner das Namespace **MAPI** in einer **For-Each**-Schleife durchlaufen. Der Schleifenzähler **oFld** ist ein Objekt des Typs **Outlook.Folder**.

Da er im Folgenden mehrmals benötigt wird, ist auf ihn ein **With**-Block gesetzt. Wichtig ist die **If**-Bedingung, welche abfragt, ob es sich beim Ordnertyp um einen E-Mail-Container handelt.

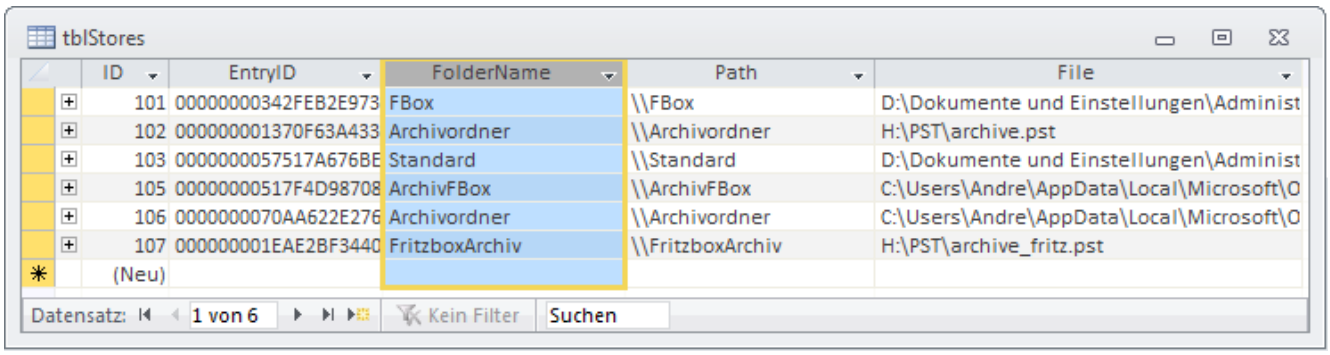


Bild 3: Die Tabelle **tblStores** speichert diverse Informationen zu den PST-Dateien, die Outlook verwendet

```
Sub GetStorageFolders()
    Dim oFld As Outlook.Folder
    Dim rs As DAO.Recordset
    CurrentDb.Execute "DELETE FROM tblStores"
    Set rs = CurrentDb.OpenRecordset("SELECT * FROM tblStores", dbOpenDynaset)
    For Each oFld In OLApp.GetNamespace("MAPI").Folders
        With oFld
            If .DefaultItemType = olMailItem Then
                rs.AddNew
                rs!FolderName = .Name
                rs!Path = .FolderPath
                rs!File = .Store.FilePath
                rs!EntryID = .EntryID
                rs.Update
            End If
        End With
    Next oFld
    rs.Close
End Sub
```

Listing 2: Prozedur zum Auslesen der Informationen zu verwendeten PST-Dateien

Das ist in der Regel dann der Fall, wenn seine Eigenschaft **DefaultItemType** den Wert **olMailItem** aufweist. Allerdings garantiert das noch nicht, dass nicht auch andere Elemente in ihm landen könnten – die Ordner sind im Prinzip recht unspezifisch. Trifft die Bedingung zu, so wird ein neuer Datensatz über das **Recordset rs** in der Tabelle **tblStores** angelegt.

Der Outlook-Pfad ergibt sich aus der Eigenschaft **FolderPath**. Auf den **Dateipfad** kann indessen nur indirekt zugegriffen werden. Das Eigenschaftsobjekt **Store** des **Folders** hat die Methode **FilePath**, die erst die **PST**-Datei hergibt. Die Tabelle speichert also im Wesentlichen direkt Eigenschaften des **Folder**-Objekts. Die zweite Tabelle im Modell, **tblMailFolders**, verweist über das Fremdschlüsselfeld **IDStore** auf einen Datensatz in der **PST**-Tabelle **tblStores**. In der Tabelle werden in flacher Hierarchie sämtliche vorkommenden Outlook-Ordner aufgelistet. Bild 4 zeigt, die das funktioniert.

Im Feld **Folder** steht jeweils der Name des Ordners, in **Path** dessen Outlook-Pfad. Darin gleichen diese Informationen denen der Tabelle **tblStores**. Auch die **EntryID** zur eindeutigen Identifizierung des Ordners fehlt hier nicht. Zusätzlich gibt es nun aber eine Spalte **ParentID**, die die Datensätze rekursive voneinander abhängig macht.

Im Beispiel hat der Ordner **Minhorst** der **PST**-Datei **Standard** die Schlüssel-**ID 1865**. Der Pfad lautet **\\Standard\Minhorst**. Dieser Ordner enthält mehrere Unterordner, so etwa das Verzeichnis **\Bestell**. Damit deutlich wird, dass dies ein Unterordner zu einem anderen Datensatz ist, enthält die **ParentID** eben den Wert **1865**. Die Baumstruktur wird somit über die abhängigen Datensätze mit ihren **ID-ParentID**-Paaren realisiert. Hat **ParentID** aber den Wert **0**, so handelt es sich um ein Root-Verzeichnis.

tblMailFolders

ID	IDStore	Folder	Path	EntryID	ParentID
1865	103	Minhorst	\\Standard\Minhorst	0000000057517A676BE7D24B8D2B35681FCC1	0
1866	103	Forumuser	\\Standard\Minhorst\Forumuser	0000000057517A676BE7D24B8D2B35681FCC1	1865
1867	103	Bestell	\\Standard\Minhorst\Bestell	0000000057517A676BE7D24B8D2B35681FCC1	1865
1868	103	Kommentare AiU	\\Standard\Minhorst\Kommentare AiU	0000000057517A676BE7D24B8D2B35681FCC1	1865
1869	103	Aiu & Buchforum	\\Standard\Minhorst\Aiu & Buchforum	0000000057517A676BE7D24B8D2B35681FCC1	1865
1870	103	Orgs	\\Standard\Orgs	0000000057517A676BE7D24B8D2B35681FCC1	0

Datensatz: 1 von 157 | Kein Filter | Suchen

Bild 4: Die Tabelle tblMailFolders speichert alle Outlook-Ordner aller verwendeten PST-Dateien in rekursiver Anordnung

```
Sub GetFolders()
    Dim itm As Variant
    Dim oFlds As Outlook.Folders
    Dim oFld As Outlook.Folder
    Dim rsFld As DAO.Recordset
    Dim rsStore As DAO.Recordset

    Set rsFld = CurrentDb.OpenRecordset("SELECT * FROM tblMailFolders", dbOpenDynaset)
    Set rsStore = CurrentDb.OpenRecordset("SELECT * FROM tblStores", dbOpenDynaset)
    Set oFlds = OLApp.GetNamespace("MAPI").Folders
    Do While Not rsStore.EOF
        Set oFld = oFlds.Item(rsStore!FolderName.Value)
        If oFld.Folders.Count > 0 Then GetSubFolders oFld.Folders, rsFld, rsStore!ID
        rsStore.MoveNext
    Loop
    rsFld.Close
    rsStore.Close
End Sub

Sub GetSubFolders(oFolders As Outlook.Folders, rs As DAO.Recordset, ID As Long, Optional lParent As Long)
    Dim oFld As Outlook.Folder
    For Each oFld In oFolders
        rs.AddNew
        rs!Folder = oFld.Name
        rs!IDStore = ID
        rs!EntryID = oFld.EntryID
        rs!ParentId = lParent
        rs!Path = Left(oFld.FolderPath, 255)
        rs.Update
        rs.MoveLast
        If oFld.Folders.Count > 0 Then GetSubFolders oFld.Folders, rs, ID, rs!ID.Value
    Next oFld
End Sub
```

Listing 3: Die beiden Prozeduren ermitteln sämtliche Outlook-EMail-Verzeichnisse

Die beiden Routinen in Listing 3 demonstrieren, wie der Verzeichnisbaum in die Tabelle kommt. Dazu öffnet **GetFolders** gleich zwei Recordsets auf die Tabellen **tblStores** und **tblMailFolders**. Alle zuvor in **tblStores** gespeicherten Datensätze werden nacheinander in einer Schleife abgearbeitet und über die Hauptordnernamen (**FolderName**) jeweils **Folder**-Objekte in der Variablen **oFld** erhalten. Über die Methode **Items** der **Folders**-Auflistung (**oFlds**) kann man nämlich entweder über einen numerischen Index oder über den Ordnernamen als Parameter an ein einzelnes Element gelangen. Anschließend ermittelt **Count**, ob das Verzeichnis Unterordner aufweist.

Ist dem so, dann kommt es zum Aufruf der zweiten Routine **GetSubFolders**, der nun drei Parameter übergeben werden. Das sind die **Folders**-Eigenschaft dieses Verzeichnisses, das Recordset für die Ordner (**rsFld**) und außerdem die **ID** des Datensatzes in **tblStore**, damit also die **ID** des Root-Verzeichnisses.

Ribbons, Teil II: Bilder im Ribbon

Mit Access 2007 sind die Menüleiste und die Symbolleisten von Access weggefallen. Die Befehle wurden stattdessen in das so genannte Ribbon übertragen. Das Ribbon bietet wesentlich mehr Möglichkeiten als seine Vorgänger, aber es ist auch aufwändiger zu programmieren. Dies gelingt nicht mehr einfach per VBA, sondern Sie müssen sich zusätzlich auch noch in XML einarbeiten – zumindest in die grundlegenden Techniken. Das ist aber kein Problem, denn wir zeigen Ihnen in dieser Artikelreihe alles, was Sie wissen müssen, um Ihren Anwendungen selbst definierte Ribbons hinzuzufügen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1611_Ribbon.accdb**.

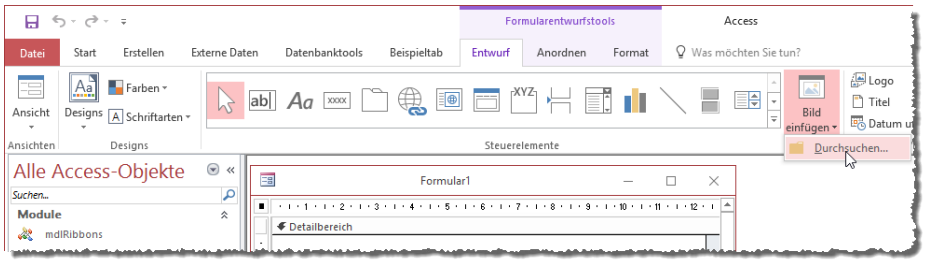


Bild 1: Hinzufügen einer Bilddatei per Formular-Entwurf

Schaltfläche, Aktion – fertig!

In der ersten Ausgabe dieser Artikelreihe haben wir Ihnen gezeigt, wie Sie **tab**-, **group**- und **button**-Elemente zu einem Ribbon hinzufügen und eine Schaltfläche mit einem Attribut namens onAction ausstatten, für das wir eine VBA-Routine hinterlegen, die beim Anklicken der Schaltfläche ausgelöst wird. Damit haben wir schon die notwendigsten Funktionen des Ribbons abgedeckt. Aber es gibt natürlich noch viele Feinheiten und Erweiterungsmöglichkeiten sowie einige Steuerelemente, die wir Ihnen noch vorstellen werden. Bevor wir damit fortfahren, schauen wir uns allerdings noch an, wie Sie eine Schaltfläche eines Ribbons (und später auch andere Steuerelemente) mit Bilder ausstatten. So können Sie eine ansprechende und gleichzeitig ergonomische Benutzeroberfläche erstellen.

Bilder im Ribbon

Um Bilder auf den Schaltflächen des Ribbon anzuzeigen, benötigen Sie folgende Dinge:

- die anzuzeigende Bilddatei,
- die Angabe eines auf die Bilddatei verweisenden Eintrags für das Attribut **image** des jeweiligen Steuerelements,
- eine für das Callback-Attribut **loadImage** des Elements **customUI** hinterlegte VBA-Funktion, welche das Bild in die Schaltfläche lädt.

Außerdem benötigen wir einige Hilfsfunktionen, die wir in einem eigenen Modul in der Datenbank hinterlegen wollen.

Bilddatei hinzufügen

Um Bilder im Ribbon anzuzeigen, benötigen Sie zunächst eine entsprechende Bilddatei. Diese fügen Sie dann in die seit Access 2010 standardmäßig in einer Anwendung enthaltene Tabelle **MSysRecourses** ein. Das hört sich aufwendiger an, als es ist. Alles, was Sie dazu tun müssen, ist das Öffnen eines Formulars in der Entwurfsansicht, ein Mausklick auf den Ribbon-Eintrag **Entwurf|Bild einfügen|Bild einfügen|Durchsuchen ...** (siehe Bild 1) und die Auswahl der gewünschten Bilddatei im **Grafik einfügen**-Dialog.

Hier können Sie optimalerweise auf eine Bibliothek von Icons zugreifen wie im Beispiel aus Bild 2. Wir verwenden übrigens die Icon-Sammlung von **www.iconexperience.com**.

Nachdem wir ein Bild zur Datenbank hinzugefügt haben, erscheint dieses in der Liste der Bilder im soeben zum Hinzufügen genutzten Ribbon-Dialog (siehe Bild 3).

Außerdem, und das ist für uns entscheidend, erscheint die Bilddatei auch in der Tabelle **MSys-Resources**. Diese können Sie erst sehen, wenn Sie die Optionen **Systemobjekte anzeigen** und **Ausblendete Objekte anzeigen** der Optionen des Navigationsbereichs aktiviert haben. Dann erscheint in dieser Tabelle ein neuer Eintrag mit der hinzugefügten Bilddatei (siehe Bild 4). Dies ist der gängige Speicherplatz für Bilddateien, die in Schaltflächen in Formularen oder auch im Bild-Steuerelement angezeigt werden sollen. Wichtig ist, dass wir uns den Namen ohne Dateiendung merken, in diesem Fall also schlicht **apple**.

Hilfsroutinen hinzufügen

Nun fügen Sie das Modul **mdlRibbonImages** aus der Beispieldatenbank zu der Zieldatenbank hinzu, in der Sie das Ribbon anlegen und mit Bildern versehen möchten. Dieses enthält als wichtigstes Element für uns die Funktion **PicFromSharedResource_Ribbon**, die als einzigen Parameter den Wert des Bildes in der Spalte **Name** der Tabelle **MSys-Resources** erwartet. Sie liefert ein Objekt des Typs **StdPicture** zurück, welches wir in der nachfolgend definierten Ribbon-Callback-Routine direkt dem Ribbon-Steuerelement zuweisen können.

Routine loadImage hinzufügen

Zu dem Modul namens **mdlRibbons**, dem Sie schon im ersten Teil der Artikelreihe die **onAction**-Routine hinzufügt haben, fügen Sie nun noch die Routine aus Listing 1 hinzu. Diese hat zwei Parameter, von denen der erste vom aufrufende Steuerelement geliefert (**control**) und der zweite an dieses Steuerelement zurückgegeben wird (**image**). Innerhalb dieser Prozedur müssen wir nun einige Zeilen Code hinterlegen, welche dafür sorgen, dass die Bilddatei aus der Tabelle **MSysResources** an die Ribbon-Schaltfläche zurückgegeben wird, deren Name mit dem ersten Parameter übergeben wurde. Dazu lesen wir zunächst mit der

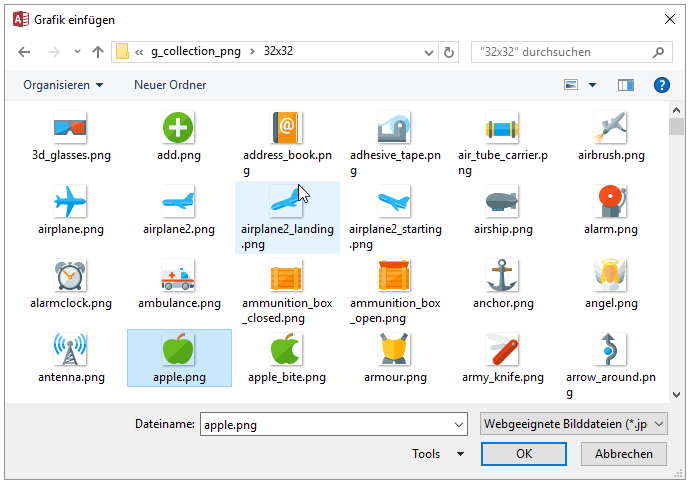


Bild 2: Auswahl der Bilddatei

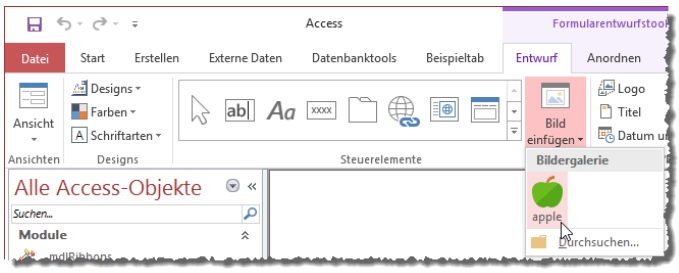


Bild 3: Hinzugefügte Bilddatei im Ribbon

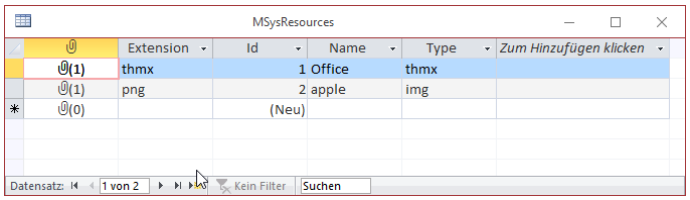


Bild 4: Neue Bilddatei in der Tabelle **MSysResources**

DLookup-Funktion den Wert des Feldes **ID** für den Datensatz mit dem übergebenen Wert im Feld **Name** aus. Konnte dieser nicht gefunden werden, wird die Variable **lngID** nicht mit dem Wert aus **ID**, sondern mit dem Wert **0** gefüllt. Sollte bei diesem Aufruf der Fehler **3078** auftreten, ist die Tabelle **MSysResources** offensichtlich nicht vorhanden und muss dann nachträglich hinzugefügt werden, worauf eine entsprechende Meldung hinweist. Hat **lngID** den Wert **0**, ist das angeforderte Bild nicht vorhanden, was ebenfalls zu einer Meldung führt.

Anderenfalls ruft die Routine die oben schon erwähnte, im Modul **mdlRibbonImages** befindliche Funktion

Ereignisse anderer Objekte nutzen

Normalerweise implementieren Sie Ereignisse immer in dem Klassenmodul des Objekts, durch das sie ausgelöst werden. Beispiele sind »Beim Laden« oder »Beim Anzeigen« eines Formulars oder auch »Beim Klicken« einer Schaltfläche. Aber wussten Sie, dass Sie auch von anderen Klassenmodulen aus die Ereignisse von externen Objekten wie Formularen oder Steuerelementen implementieren können? Dieser Artikel zeigt, wie das gelingt und welche Vorteile Sie dadurch erlangen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1611_EreignisseAndererObjekteNutzen.accdb**.

Fremde Ereignisse implementieren

Die grundlegende Technik zeigen wir an einem einfachen Beispiel. Dabei wollen wir ein Formular öffnen, die Eingabe des Benutzers abwarten und dann vor demSchließen die eingegebenen Daten im aufrufenden Formular einlesen. Üblicherweise würden Sie von einem Formular aus dazu die folgende Anweisung nutzen:

```
DoCmd.OpenForm "frmQuelle", WindowMode:=acDialog
```

Dies hält den Code der aufrufenden Prozedur an. Sie würden das aufgerufene Formular dann beim Klick auf die Schaltfläche **OK** mit der folgenden Anweisung unsichtbar machen:

```
Me.Visible = False
```

Die aufrufende Prozedur würde dies dann als Beenden des Dialog interpretieren, prüfen, ob das Formular zwar unsichtbar, aber noch geöffnet ist, und die gewünschten Daten auslesen, bevor sie das Formular endgültig schließt:

```
DoCmd.Close acForm, "frmQuelle"
```

Zu dieser Vorgehensweise gibt es eine elegante Alternative, die es nicht erfordert, das aufgerufene Formular erst noch unsichtbar zu schalten und dann im aufrufenden Formular zu prüfen, ob dieses überhaupt noch vorhanden ist.

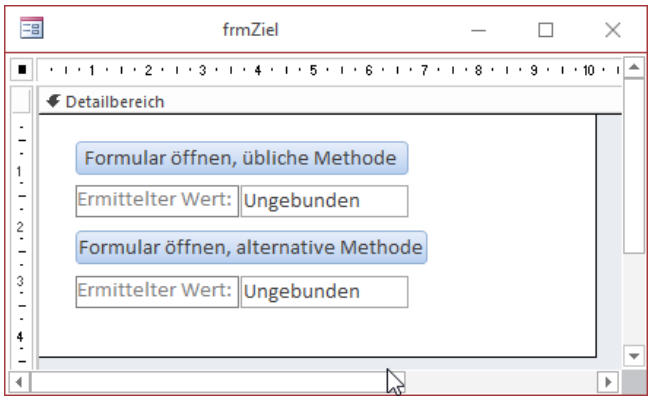


Bild 1: Formular, das andere Formular aufruft und vor dem Schließen deren Werte einliest

Beispielformular

Wir wollen uns diese beiden Vorgehensweisen im Vergleich anschauen. Dazu legen wir ein Formular namens **frmZiel** an, welches zwei Schaltflächen zum Öffnen zweier Formulare mit jeweils einer der beiden Methoden enthält. Außerdem besitzt dieses Formular zwei Textfelder, in welche die in die geöffneten Formulare eingegebenen Texte beim Schließen übertragen werden sollen (siehe Bild 1).

Methode 1: Öffnen mit DoCmd.OpenForm

Die erste Schaltfläche namens **cmdOpenForm** verwendet die folgende Prozedur, um das Formular **frmQuelle1** mit der **DoCmd.OpenForm**-Methode als modalen Dialog zu öffnen:

```
Private Sub cmdOpenForm_Click()
    DoCmd.OpenForm "frmQuelle1", WindowMode:=acDialog
```

```
    If IstFormularGeoeffnet("frmQuelle1") Then
        Me!txtZiel1 = Forms!frmQuelle1!txtEingabe
        DoCmd.Close acForm, "frmQuelle1"
    End If
End Sub
```

Nach dem Öffnen gibt der Benutzer einen Text in das einzige Textfeld dieses Formulars ein und klickt auf die Schaltfläche **OK**, um das Formular wieder zu schließen. Dies lässt die aufrufende Prozedur weiterlaufen, die dann mit der Hilfsfunktion **IstFormularGeoeffnet** aus dem Modul **mdlTools** prüft, ob das geöffnete Formular überhaupt noch geöffnet ist. Falls ja, wurde der modale Modus durch Setzen der Eigenschaft **Visible** auf den Wert **True** aufgehoben.

In diesem Fall trägt die erste Zeile innerhalb der **If... Then**-Bedingung den Wert des Textfeldes **txtEingabe** des Formulars **frmQuelle1** in das Textfeld **txtZiel1** des Formular **frmZiel** ein. Die zweite Anweisung schließt das noch geöffnete, aber unsichtbare Formular **frmQuelle** nun endgültig.

Der Ablauf sieht so wie in Bild 2 aus. Die Schaltfläche **cmdOK** von **frmQuelle1** löst die folgende Ereignisprozedur aus:

```
Private Sub cmdOK_Click()
    Me.Visible = False
End Sub
```

Methode 2: Öffnen per New und Reaktion auf das Schließen per Implementierung von OnClose

Die zweite Methode ist ähnlich aufgebaut, aber verwendet einen etwas anderen Code. Voraussetzung für das Funktionieren ist die Deklaration einer Variablen, mit dem wir das zu öffnende Formular referenzieren können. Die Deklaration muss im Kopf des Klassenmoduls **Form_frmZiel** erfolgen, damit die Variable nach dem Initialisieren immer noch verfügbar ist. Zusätzlich benötigen wir ein spezielles Schlüsselwort, nämlich **WithEvents** – mehr dazu gleich:

```
Dim WithEvents frm As Form
```

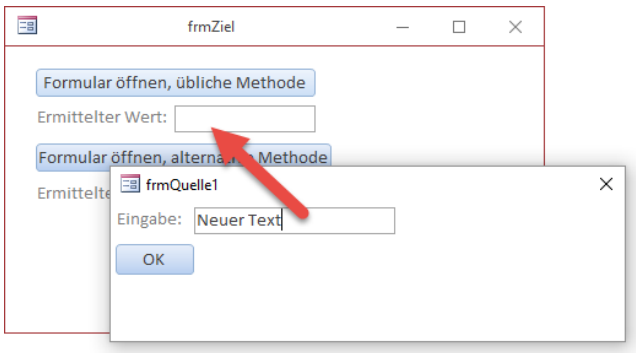


Bild 2: Aufruf und Übergabe des Textes von Formular zu Formular

Die zweite Schaltfläche namens **cmdNewInstance** löst nun beim Anklicken die folgende Ereignisprozedur aus:

```
Private Sub cmdNewInstance_Click()
    Set frm = New Form_frmQuelle2
    With frm
        .Modal = True
        .Visible = True
        .OnClose = "[Event Procedure]"
    End With
End Sub
```

Diese erstellt eine neue Instanz des Formulars **frmQuelle2**, wozu es das **New**-Schlüsselwort mit dem Klassenmodul dieses Formulars verwendet. Das Ergebnis landet in der Variablen **frm**. Für diese legen wir nun die Eigenschaft **Modal** auf den Wert **True** fest, damit das Formular als modaler Dialog geöffnet wird. Mit **Visible = True** wird das noch unsichtbare Formular eingeblendet. Danach folgt eine Anweisung, die mit dem zuvor verwendeten Schlüsselwort **WithEvents** in Zusammenhang steht:

```
frm.OnClose = "[Event Procedure]"
```

Dies sorgt dafür, dass wir in dem Klassenmodul, in dem auch die Variable **frm** deklariert ist, eine Implementierung des Ereignisses **OnClose** hinterlegen können, die dann ausgelöst wird, wenn das Ereignis **Beim Schließen** dieses Formulars ausgelöst wird. Im Prinzipiell entspricht dies dem Setzen der Eigenschaft

Formularpositionen lesen und setzen

Die Position der Formulare Ihrer Datenbank wird normalerweise nur über die Eigenschaft Automatisch zentrieren eingestellt, was dazu führt, dass ein Formular direkt in der Mitte des Access-Fensters geöffnet wird. Wenn Sie weitergehende Ansprüche haben, können Sie die genaue Position von Formularen nicht nur ermitteln, sondern diese auch setzen. Dieser Artikel zeigt die Funktionen, mit denen Sie dies realisieren können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1611_FormularePositionieren.accdb**.

Formularpositionen lesen

Wenn Sie die Position eines Formulars lesen wollen, können damit verschiedene Positionen gemeint sein: Die absolute Position auf dem Desktop oder die Position innerhalb des Access-Fenster oder auch die Position innerhalb der Arbeitsfläche im Access-Fenster, also des Bereichs ohne Rahmen, Ribbon und Navigationsbereich.

Um diese Werte zu ermitteln und damit zu experimentieren, legen wir ein neues Formular an, das wie in Bild 1 aussieht und acht Textfelder für die Ausgabe der verschiedenen Werte bereitstellt.

Nun fehle nur noch einige Zeilen VBA-Code, gewürzt mit einigen API-Funktionen, um diese Textfelder zu füllen – optimalerweise im regelmäßigen Abstand, damit Sie gleich beim Ziehen oder Anpassen der Größe von Access-Fenster und dem Formular live die aktuellen Werte angezeigt bekommen.

Access-Fenster und Formular

Wenn wir die Position eines Formulars ermitteln wollen, können wir diese direkt relativ zum Bildschirmrand ermitteln oder relativ zum Rand des Access-Fensters.

Wenn Sie den Abstand zum Rand des Access-Fensters ermitteln wollen, geht dies nicht direkt, sondern wir müssen dann den Abstand des Formulars zum Bildschirmrand ermitteln und dann den Abstand des Access-Fensters vom Bildschirmrand abziehen.

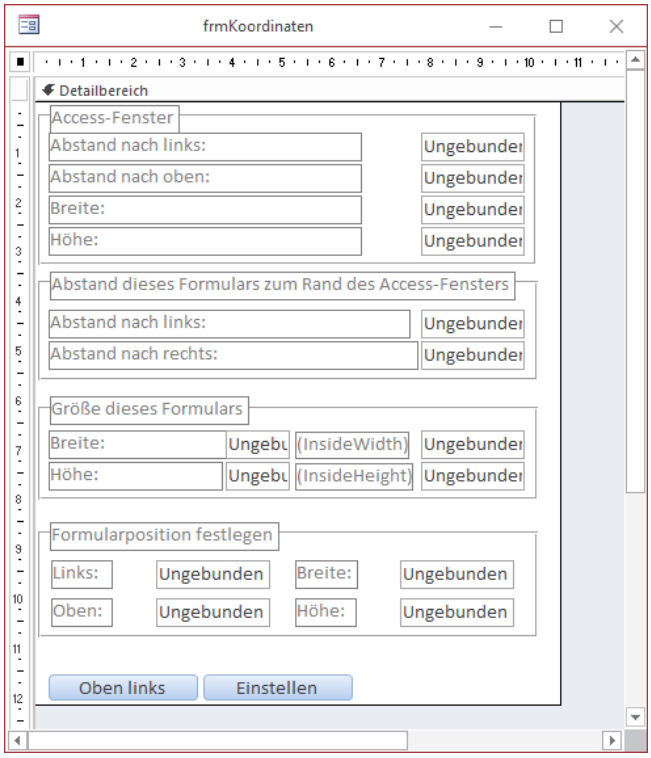


Bild 1: Formular zur Anzeige der aktuellen Abmessungen des Formulars und des Access-Fensters

Funktion zum Ermitteln Daten des Access-Fensters

Also programmieren wir zunächst eine Funktion, welche die Koordinaten des Access-Fensters ermittelt. Diese sieht wie in Listing 1 aus. Diese Funktion namens **GetMainWindowPosition** enthält vier Parameter namens **sngLeft, sngTop**, **sngWidth** und **sngHeight**.

Diesen müssen Sie beim Aufruf vier entsprechende Parameter übergeben, die dann gefüllt werden. Dies sieht in einer kleinen Beispielprozedur etwa wie in Listing 2 aus.

```
Public Function GetMainWindowPosition(sngLeft As Single, sngTop As Single, sngWidth As Single, sngHeight As Single)
    Dim rectMainWindow As Rect
    GetWindowRect Application.hWndAccessApp, rectMainWindow
    sngLeft = rectMainWindow.Left
    sngTop = rectMainWindow.Top
    sngWidth = rectMainWindow.Right - rectMainWindow.Left
    sngHeight = rectMainWindow.Bottom - rectMainWindow.Top
End Function
```

Listing 1: Diese Funktion ermittelt die Koordinaten der Position des Access-Hauptfensters.

```
Public Sub Test_GetMainWindowPosition()
    Dim sngLeft As Single
    Dim sngTop As Single
    Dim sngWidth As Single
    Dim sngHeight As Single
    GetMainWindowPosition sngLeft, sngTop, sngWidth, sngHeight
    MsgBox "Aktuelle Position und Größe des Access-Fensters:" & vbCrLf & "Links: " & sngLeft & vbCrLf & "Oben: " & sngTop _
        & vbCrLf & "Breite: " & sngWidth & vbCrLf & "Höhe: " & sngHeight
End Sub
```

Listing 2: Prozedur mit einem Beispielaufruf der Funktion **GetMainWindowPosition**

Das Ergebnis kommt wie in Bild 2 in Form eines Meldungsfensters. Die Funktion **GetMainWindowPosition**, die wir übrigens im Standardmodul **mdlKoordinaten** hinterlegt haben, deklariert zunächst eine Variable des Typs **Rect**.

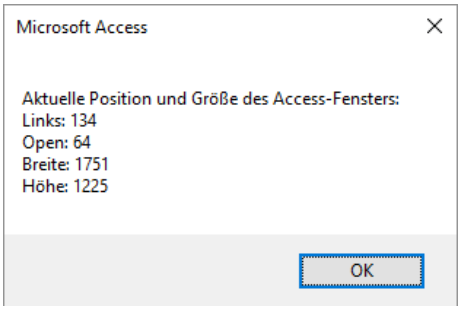


Bild 2: Test der Funktion zum Ermitteln von Position und Größe des Access-Fensters

Diesen Typ müssen Sie zunächst noch deklarieren, was wie folgt aussieht:

```
Public Type Rect
    Left As Long
    Top As Long
    Right As Long
    Bottom As Long
End Type
```

Diesem benutzerdefinierten Typ können Sie Werte über die Punkt-Syntax zuweisen, wie Sie gleich sehen werden. Einen solchen benutzerdefinierten Typ benötigen wir, um Daten mit API-Funktionen auszutauschen. Eine solche verwenden wir nämlich gleich in der ersten Anweisung der Funktion, nämlich **GetWindowRect**. Diese muss ebenfalls zuerst noch per Deklaration verfügbar gemacht werden:

```
Public Declare Function GetWindowRect Lib "user32" (ByVal hWnd As Long, lpRect As Rect) As Long
```

Nun kann die Funktion die API-Funktion nutzen und dieser mit **Application.hWndAccessApp** ein Handle auf die aktuelle Access-Anwendung sowie die noch leere Variable **rectMainWindow** des Typs **Rect** übergeben. Diese wird dann gefüllt. Die vier Werte des Typs **rectMainWindow** übergeben wir dann an die Rückgabeparameter der Funktion – zum Beispiel so:

```
sngLeft = rectMainWindow.Left
```

```
Public Function GetFormPosition(frm As Form, sngLeft As Single, sngTop As Single, sngWidth As Single, sngHeight As Single)
    Dim rectForm As Rect
    Dim rectParent As Rect
    GetWindowRect frm.hWnd, rectForm
    GetWindowRect Application.hWndAccessApp, rectParent
    sngLeft = rectForm.Left - rectParent.Left
    sngTop = rectForm.Top - rectParent.Top
    sngWidth = rectForm.Right - rectForm.Left
    sngHeight = rectForm.Bottom - rectForm.Top
End Function
```

Listing 3: Diese Funktion ermittelt die Koordinaten der Position eines Formulars relativ zum Access-Hauptfenster.

Funktion zum Ermitteln der Daten eines Formulars

Damit kommen wir zur zweiten Funktion namens **GetFormPosition** (siehe Listing 3). Diese erwartet einen Parameter mehr als die vorherige Funktion. Dabei handelt es sich um einen Verweis auf das Formular, das untersucht werden soll. Die Funktion soll die Position des Formulars innerhalb des Access-Fensters zurückgeben. Dazu verwenden wir wieder die bereits weiter oben vorgestellte API-Funktion **GetWindowRect**, und zwar direkt zwei Mal. Beim ersten Mal ermitteln wir die Koordinaten des Formulars, das wir mit der Eigenschaft **hWnd** des Formulars als Handle referenzieren, und speichern diese im Typ **rectForm**. Der zweite Einsatz liefert mit dem Typ **rectParent** die Koordinaten des Hauptfensters.

Die folgenden vier Anweisungen ermitteln rechnerisch die Position des Formulars innerhalb des Access-Fensters, indem jeweils der Abstand des Hauptfensters zum entsprechenden Bildschirmrand vom Abstand des Access-Formulars zum Bildschirmrand subtrahiert wird – etwa wie folgt:

```
sngLeft = rectForm.Left - rectParent.Left
```

Eine kleine Prozedur zum Testen dieser Funktion sieht wie folgt aus – hier öffnen wir noch das zu untersuchende Formular und referenzieren es mit der Variablen **frm**:

```
Public Sub Test_GetFormPosition()
    Dim sngLeft As Single
    Dim sngTop As Single
    Dim sngWidth As Single
    Dim sngHeight As Single
    Dim frm As Form
    DoCmd.OpenForm "frmKoordinaten"
    Set frm = Forms!frmKoordinaten
    GetFormPosition frm, sngLeft, sngTop, _
        sngWidth, sngHeight
    MsgBox "Aktuelle Position und Größe ...
End Sub
```

Formular dynamisch mit Daten füttern

Damit haben wir schon die wichtigsten Voraussetzungen zum Füllen der Textfelder des Formulars wie in Bild 3 erfüllt. Dem Formular **frmKoordinaten** fügen wir nun eine Ereignisprozedur für das Ereignis **Bei Zeitgeber** hinzu und stellen die Eigenschaft **Zeitgeberintervall** auf den Wert **100** ein. Dadurch wird das Ereignis alle 100 Millisekunden ausgelöst, was eine zeitnahe Aktualisierung der Textfelder bewirkt.

Die Prozedur **Form_Timer** finden Sie in Listing 4. Die Prozedur deklariert die Variablen, die an die Funktionen **GetMainWindowPosition** und **GetFormPosition** übergeben werden. Außerdem benötigen wir im weiteren Verlauf noch die beiden Variablen **lngX** und **lngY**.

Die Ereignisprozedur ruft zunächst die Funktion **GetMainWindowPosition** auf und erhält die gefüllten Variablen **sngLeft**, **sngTop**, **sngWidth** und **sngHeight** zurück. Die folgenden vier Anweisungen tragen diese Werte in die Textfelder **txtAccLeft**, **txtAccTop**,

Mehrere Instanzen eines Formulars

Die meisten Anwendungen zeigen, wie Sie aus einer Liste von Datensätzen einen per Doppelklick öffnen und in einem Detailformular anzeigen. Nach dem Bearbeiten schließen Sie diesen und wenden sich dem nächsten Datensatz zu. Was aber, wenn Sie einmal mehrere Datensätze parallel anzeigen möchten, um diese beispielsweise zu vergleichen – oder weil Sie einfach den einen Datensatz später noch einmal weiterbearbeiten möchten. Dieser Artikel zeigt Ihnen, wie Sie dieses Verhalten mit Access-Formularen abbilden können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1611_Formularinstanzen.accdb**.

Formulare öffnen auf herkömmliche Art

Wenn Sie auf die herkömmliche Art ein Formular öffnen möchten, verwenden Sie dazu beispielsweise den folgenden Befehl:

```
DoCmd.OpenForm "frmKunde"
```

Damit erhalten Sie dann eine Instanz des gewünschten Formulars. Mit weiteren Parametern können Sie etwa noch angeben, dass Sie gleich beim Öffnen einen bestimmten Kunden anzeigen wollen:

```
DoCmd.OpenForm "frmKunde", WhereCondition:="KundeID = 1"
```

Wenn Sie die gleiche Anweisung jedoch noch ein weiteres Mal ausführen, um damit beispielsweise einen anderen Kunden im gleichen Formular anzuzeigen, gelingt dies nicht wie gewünscht.

Wir schauen uns dies am Beispiel des Formulars **frmKunde** an, dass im Entwurf wie in Bild 1 aussieht und die Tabelle **tblKunden** mit einigen Beispieldatensätzen als Datenherkunft verwendet.

Öffnen wir das Formular mit der obigen **DoCmd.OpenForm**-Anweisung, die Sie der Einfachheit halber einfach im Direktbereich eingeben (zu öffnen mit **Strg + G**) und mit der Eingabetaste ausführen, erhalten wir das Formular aus Bild 2. Führen Sie diesen Befehl jedoch direkt danach erneut aus, wobei Sie

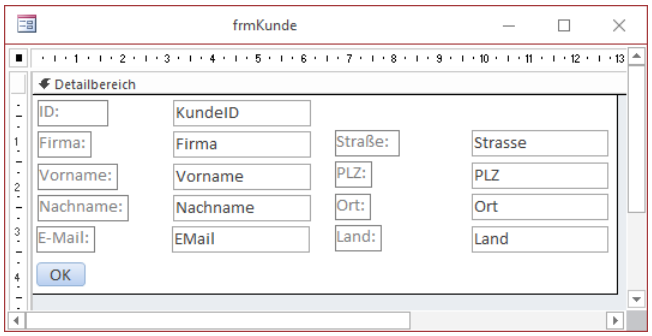


Bild 1: Formular-Entwurf unseres Beispielformulars

Bild 2: Erster Datensatz im Beispielformular

lediglich den **WhereCondition**-Parameter wie folgt ändern, wird zwar der zweite Kunde im Formular angezeigt, der erste verschwindet allerdings:

```
DoCmd.OpenForm "frmKunde", WhereCondition:="KundeID = 2"
```

Das bedeutet, dass wir uns eine alternative Möglichkeit einfallen lassen müssen.

Formular-Instanz öffnen

Zum Glück gibt es noch eine zweite Methode, um ein Formular zu öffnen. Im Grunde ist **DoCmd.OpenForm** eine vereinfachte Variante, die Sie so etwa in anderen Sprachen wie Visual Basic oder C# nicht finden wer-

den. Dort wird immer eine Objektvariable auf Basis der entsprechenden Klasse, hier eben das betroffene Formular, deklariert und dann initialisiert. Gegebenenfalls ist das initialisierte Objekt dann noch sichtbar zu machen, was bei Access-Formularen der Fall ist. Wie also sieht diese alternative Methode unter Access aus und können wir auch hier den gewünschten Datensatz an das Formular übermitteln?

Dazu wollen wir uns erst einmal ein Standardmodul anlegen, in dem wir eine Prozedur erstellen, um das Verhalten nachzuprogrammieren.

Das Standardmodul heißt **mdlFormularinstanzen** und wird im Visual Basic Editor (zu öffnen mit **Alt + F11**) mit dem Menübefehl **Einfügen|Modul** eingefügt.

Keine Formularinstanz ohne Formularklasse

Ein Schritt ist noch nötig, bevor wir per VBA eine neue Instanz unseres Formulars **frmKunde** erstellen können. Genau genommen erstellen wir nämlich keine Instanz des Formulars, sondern der Code behind-Klasse, also des Klassenmoduls, dass beispielsweise dann angelegt wird, wenn Sie ein Ereignis des Formulars implementieren oder die Eigenschaft **Enthält Modul** des Formulars auf **Ja** einstellen.

Da das Formular eine Schaltfläche mit der Beschriftung **OK** enthält, welche das Formular nach erfolgter Bearbeitung oder Ansicht der Daten schließen soll, legen wir gleich den Wert **[Ereignisprozedur]** für die Ereigniseigenschaft **Beim Klicken** dieses Steuerelements an und klicken auf die Schaltfläche mit den drei Punkten, um die entsprechende Ereignisprozedur anzulegen – und somit auch das benötigte Klassenmodul:

```
Private Sub cmdOK_Click()
    DoCmd.Close acForm, Me.Name
End Sub
```

Danach wechseln wir wieder zum Modul **mdlFormularinstanzen**, wo wir eine neue Prozedur namens **Formularinstanz** hinzufügen und für diese eine Variable namens **frm** deklarieren. Diese soll den Typ des

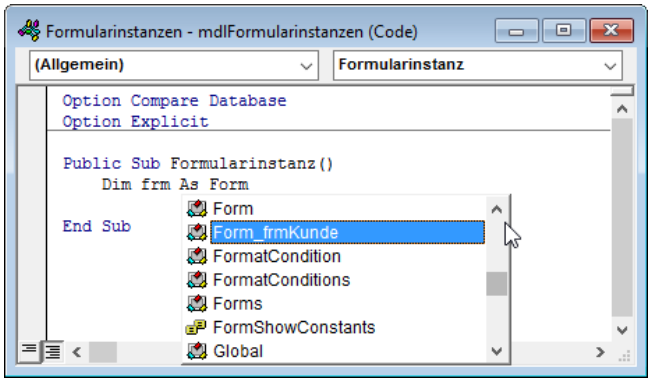


Bild 3: Formularklasse per IntelliSense

Klassenmoduls des Formulars **frmKunde** erhalten, der dann auch prompt per IntelliSense angeboten wird (siehe Bild 3). Also führen wir nun die geplanten Schritte durch, die wie folgt aussehen – deklarieren, initialisieren und sichtbar machen:

```
Public Sub Formularinstanz()
    Dim frm As Form_frmKunde
    Set frm = New Form_frmKunde
    frm.Visible = True
End Sub
```

Der Plan war nicht gut genug: Das Formular erscheint für Bruchteile einer Sekunde und verschwindet dann wieder. Der Grund ist eindeutig: Die Variable **frm**, mit der wir es referenzieren, ist innerhalb der Prozedur Formularinstanz deklariert.

Das heißt, dass ihre Lebensdauer mit dem Ablauf des Formulars endet. Die Folgerung: Die Variable muss außerhalb der Prozedur deklariert werden. Mit der folgenden Variante klappt es besser – das Formular erscheint wie gewünscht:

```
Dim frm As Form_frmKunde

Public Sub Formularinstanz()
    Set frm = New Form_frmKunde
    frm.Visible = True
End Sub
```

Mit der Schaltfläche **cmdOK** lässt es sich auch problemlos wieder schließen.

Gewünschten Datensatz anzeigen

Es fehlt allerdings noch die Möglichkeit, gleich beim Anzeigen einen bestimmten Datensatz anzuzeigen. Normalerweise haben wir diesen mit dem Parameter **WhereCondition** übergeben, der uns hier aber nicht zur Verfügung steht.

Allerdings haben wir ja nun eine Objektvariable, mit der wir auf die Instanz des Formulars verweisen. Warum also nicht gleich auch die Eigenschaften des Formulars nutzen – wie beispielsweise die **Filter**-Eigenschaft? Erweitern wir also unsere Prozedur und fügen einen Filterausdruck hinzu, der den Kunden mit dem Wert **2** für das Feld **KundeID** liefert:

```
Public Sub Formularinstanz()
    Set frm = New Form_frmKunde
    frm.Visible = True
    frm.Filter = "KundeID = 2"
    frm.FilterOn = True
End Sub
```

Das erneute Ausführen der Prozedur zeigt nun den gewünschten Kunden an (siehe Bild 4). Nun ändern wir die Zeile mit dem Filter wie folgt:

```
frm.Filter = "KundeID = 3"
```

Rufen wir die Prozedur nun erneut auf, zeigt das Formular zwar den gewünschten Kunden an, allerdings ist das Formular mit dem vorherigen Kunden nicht mehr vorhanden. Kein Wunder: Wir haben ja nur eine Instanzvariable namens **frm**, deren Inhalt beim Überschreiben gelöscht wird – und mit ihm das referenzierte Formular.

Mehrere Instanzen öffnen

Wie öffnen wir nun mehrere Instanzen gleichzeitig? Die einfachste Methode: Wir deklarieren einfach für jede Instanz eine eigene Variable. Dies dient zunächst nur Beispielzwecken, um zu veranschaulichen, wie es funktioniert. Jede der beiden Instanzen aus dem folgenden Beispiel wir durch

Bild 4: Formularinstanz mit Wunschkunde

die neue Prozedur Formularinstanzen initialisiert, eingeblendet und mit dem gewünschten Datensatz versehen. Das Ergebnis sieht wie in Bild 5 aus:

```
Dim frm1 As Form_frmKunde
Dim frm2 As Form_frmKunde

Public Sub Formularinstanzen()
    Set frm1 = New Form_frmKunde
    With frm1
        .Visible = True
        .Filter = "KundeID = 1"
        .FilterOn = True
    End With
    Set frm2 = New Form_frmKunde
    With frm2
        .Visible = True
        .Filter = "KundeID = 2"
        .FilterOn = True
    End With
End Sub
```

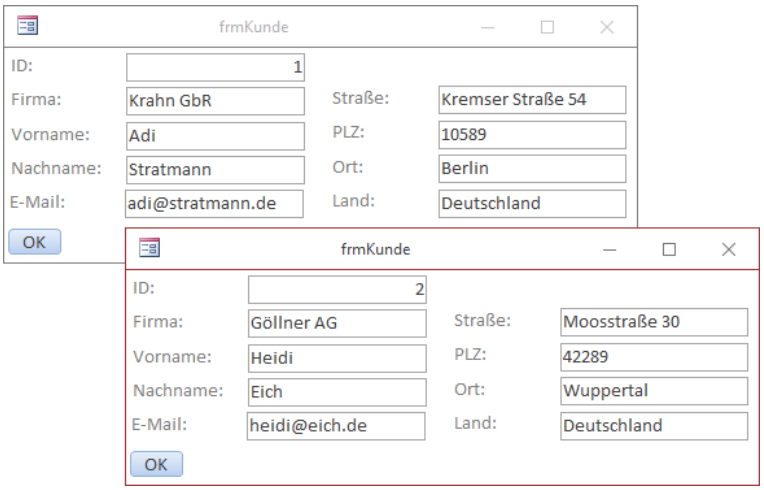


Bild 5: Zwei Formularinstanzen

Damit haben wir schon einmal gezeigt, wie es gelingen kann. Allerdings ist dies natürlich keine tragfähige Lösung, denn wir wollen ja nicht für jeden zu öffnenden Kunden eine eigene Objektvariable deklarieren und die Prozedur Formularinstanzen entsprechend erweitern – zudem wir ja nicht absehen können, wie viele Kunden der Benutzer anzeigen möchte.

Und es gibt noch ein weiteres Problem: Die beiden Formular werden genau übereinander geöffnet, wodurch der Benutzer gar nicht erkennen kann, dass mehrere Formular existieren.

Viele Objekte verwalten

Wir kümmern uns zunächst darum, mehrere Formularinstanzen zu öffnen, ohne dass wir für jede eine eigene Variable deklarieren müssen. Aber wie erledigen wir das? Voraussetzung ist, dass wir irgendwie eine Referenz auf die Formularinstanz halten, ohne dafür jeweils eine eigene Variable bereitstellen zu müssen. Dafür gibt es eine einfache Möglichkeit: Wir verwenden nur eine Objektevariable, die wir diesmal wieder innerhalb der Prozedur deklarieren und füllen. Diesmal speichern wir diese allerdings vor dem Prozedurende in einer Collection! Diese deklarieren wir wie folgt im Kopf des Standardmoduls:

```
Public colForms As Collection
```

Dann verwenden wir die folgende Prozedur, um zwei Formulare mit der gleichen Objektvariable zu initialisieren und in der Collection zu speichern. Diese müssen wir natürlich zuvor mit dem **New**-Schlüsselwort initialisieren. Dann erstellen wir wie gewohnt die **Form_frmKunde**-Objekte, blenden das Formular ein und lassen den gewünschten Kunden anzeigen. Der wichtige Schritt ist das Hinzufügen der Objektvariablen zur Collection **colForms** mit den **Add**-Befehl:

```
Public Sub FormularCollection()
    Dim frm As Form_frmKunde
    Set colForms = New Collection
    Set frm = New Form_frmKunde
    With frm
        .Visible = True
        .Filter = "KundeID = 1"
        .FilterOn = True
    End With
    colForms.Add frm
    Set frm = New Form_frmKunde
    With frm
        .Visible = True
        .Filter = "KundeID = 2"
        .FilterOn = True
    End With
    colForms.Add frm
End Sub
```

Das Ergebnis ist das Gleiche wie zuvor – die Prozedur öffnet zwei Formulare und zeigt diese an. Die Collection **colForms** haben wir mit dem **Public**-Schlüsselwort deklariert. So können wir nun betrachten, warum die Lösung in dieser Form noch nicht perfekt ist. Schließen Sie dazu einmal die beiden soeben geöffneten Formulare und geben Sie dann den folgenden Befehl im Direktbereich des VBA-Editors ein:

```
? colForms.Count
 2
```

Wie Sie sehen, enthält die Collection noch zwei Objekte, die allerdings, da die Formulare bereits geschlossen wurden, nicht mehr nützlich sind. Also sollten wir uns auch darum kümmern, die Collection nach dem Anlegen der Formulare auch wieder aufzuräumen, wenn die Formulare geschlossen wurden.

Dazu wollen wir aber zunächst ein kleines Übersichtsformular erstellen, mit dem wir die Formulare komfortabel öffnen und schließen können.

Übersichtsformular

Dieses Formular soll alle Kunden in einem Listenfeld anzeigen und das Öffnen per Doppelklick ermöglichen. Das Formular **frmKundeuebersicht** enthält neben dem Listenfeld **lstKunden** noch zwei Schaltflächen namens **cmdOK** und **cmdAlleSchliessen**. Die Funktion der ersten Schaltfläche ist klar, die zweite Schaltfläche dient dazu, alle vom Übersichtsformular aus geöffneten Kunden-Formulare zu schließen (siehe

Bild 6). Einzelne Formulare soll der Benutzer mit der **OK**-Schaltfläche der Formulare selbst schließen. Schließlich sollen auch alle Kunden-Formulare automatisch geschlossen werden, wenn der Benutzer das Formular **frmKundenuebersicht** schließt.

Das Listenfeld des Formulars erhält die folgende Datensatzherkunft:

```
SELECT KundeID, Firma & " - " &  Nachname & ", " & Vorname FROM tblKunden;
```

Die Eigenschaft **Spaltenanzahl** stellen wir auf **2**, die Eigenschaft **Spaltenbreiten** auf **0cm** ein. So wird der Primärschlüsselwert ausgeblendet und nur der gewünschte Inhalt angezeigt. In der Formularansicht sieht das Übersichtsformular nun wie in Bild 7 aus.

Vorüberlegung

Wir wollen mit dem Übersichtsformular ja nicht nur wild Formularinstanzen öffnen, sondern auch dafür sorgen, dass diese kontrolliert geschlossen werden – und vor allem sollen Kunden doppelt in den Formularen angezeigt werden geschweige denn Fehler auftreten, wenn wir einen Kunden anzeigen, schließen und erneut anzeigen.

Um zu tracken, welche Kunden bereits in Formularen angezeigt werden und welche nicht, können wir die Collection wunderbar nutzen. Die Collection bietet nämlich für jeden Eintrag die Möglichkeit, einen eigenen Schlüsselwert (**Key**) anzugeben. Einträge der Collection können wir dann referenzieren, indem wir entweder über den **Index** oder den **Key** auf diese zugreifen. Dabei gibt man beide einfach in Klammern hinter der Collection an.

Der Unterschied ist nur, dass der Index ein **Long**-Wert ist und der Key ein **String**. **colForms(1)** liefert also den Eintrag mit dem Index **1**. **colForms("1")** würde den Eintrag liefern, der den Key **1** aufweist. Um zielgerichtet auf die **Collection**-Einträge zugreifen zu können, die mit dem jeweiligen Kunden-Formular assoziiert sind, müssen wir also einen **Key** hinzufügen, der den Primärschlüsselwert des im jeweiligen

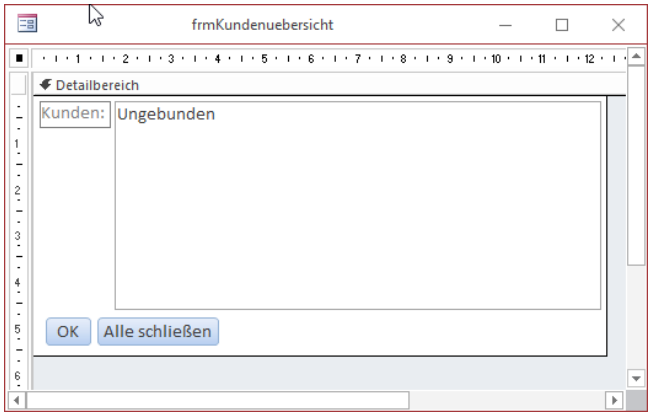


Bild 6: Übersichtsformular in der Entwurfsansicht

Bild 7: Übersichtsformular in der Formularansicht

Formular angezeigten Kunden enthält. Um hier Verwirrungen zu vermeiden, haben wir uns entschlossen, statt des reinen Primärschlüsselwertes eine aus dem Ausdruck **Kunde_** und dem Primärschlüsselwert zusammengesetzten Wert als **Key** zu übergeben, also etwa **Kunde_1**.

Funktionen des Übersichtsformulars

Die Variable für die Collection, welche die Objektvariablen der geöffneten Detailformulare aufnehmen soll, deklarieren wir wie folgt im Klassenmodul des Formulars **frmKundenuebersicht**:

```
Dim colForms As Collection
```

Diese Variable können wir nun nicht mehr in der Prozedur initialisieren, die auch die Formularinstanzen öffnet, denn die Initialisierung soll ja nur einmal stattfinden. Also fügen wir den entsprechenden Befehl zur Ereignisprozedur hinzu, die durch das Ereignis **Beim Laden** des Formulars ausgelöst wird:

Anwendungs-Design mit positionierten Formularen

Wie Sie die Positionen und Abmessungen von Formularen ermitteln oder per VBA setzen können, oder auch die des Access-Rahmenfensters, das erfuhren Sie schon in der letzten Ausgabe von Access Basics. Einige Beispiele, wie Sie solche Routinen gewinnbringend in Ihrer Anwendung einsetzen können, folgen in diesem Beitrag.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1701_AppDesign.accdb**.

Access und Monitorauflösungen

Die Auflösung von PC-Monitoren hat eine etwas seltsame Geschichte hinter sich. Es ist noch nicht so lange her, da kamen die ersten Flat-Screen-Monitore in der Größe von 15 bis 19 Zoll auf den Markt. Ihre Auflösung betrug in der Regel 1280 x 1024 Pixel. Dann wurden Multimedia-Anwendungen immer wichtiger, die ein Breitformat erforderten. Inzwischen sind 27-Zoll Monitore fast schon der Standard. Doch an der vertikalen Auflösung hat sich indessen fast nichts geändert. Immer noch sind 1080 Pixel vertikal der weitaus verbreitetste Wert. Besonders gilt das für Laptops, die jedoch immer mehr auch im Büro Einzug halten. Eignet sich diese Auflösung tatsächlich für Datenbankanwendungen?

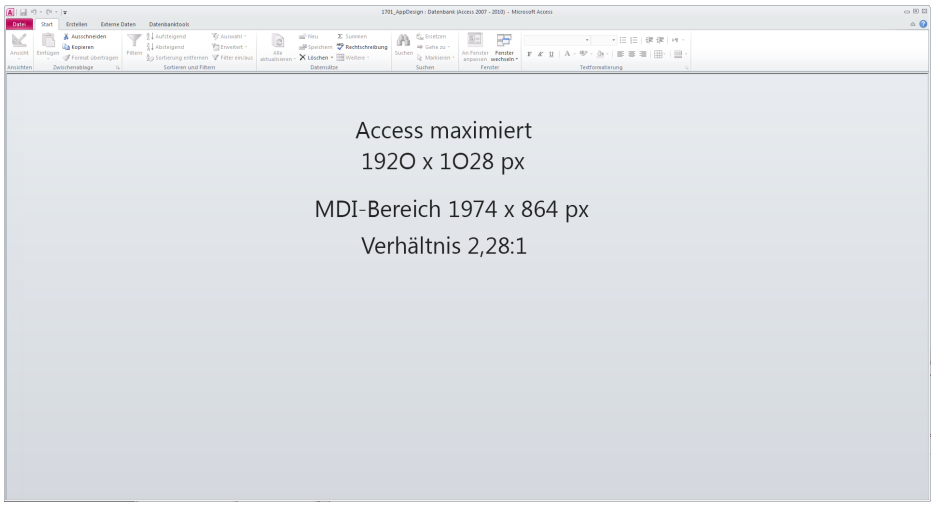


Bild 1: Access maximiert auf einem Laptop-Monitor ohne Navigationsbereich

Bild 1 demonstriert, wie **Access 2010** sich unter **Windows 7** auf so einem Monitor präsentiert, wenn der Navigationsbereich ausgeblendet ist. Das dürfte nicht selten der Fall sein, da man dem Anwender diesen Bereich eher nicht zur Ansicht bringen will und andere Navigationsmethoden, etwa benutzerdefinierte Ribbons, vorzieht. Da die Taskleiste von Windows die Höhe schmälert, bleiben einer Anwendung in maximiertem Zustand noch 1028 Pixel. Für die Titelleiste von Access gehen davon weitere 24 Pixel ab. Und schließlich nimmt der Ribbon ganze 110 Pixel ein.

Somit kann ein maximiertes Formular oder ein Bericht nach Abzug des Rahmens des MDI-Bereichs, also der Arbeitsfläche, höchstens noch die Ausdehnung von 1974 x 864 Pixeln besitzen. Daraus ergibt sich ein Seitenverhältnis von etwa 2,28:1.

Unter Excel mag dieses Verhältnis noch einigermaßen sinnvoll sein, da man es hier häufig mit zahlreichen Spalten zu tun hat. Aber Access? Formulare nennen sich nicht zufällig so. Eingabeformulare haben ihr papierenes Vorbild im DIN-A4-Hochformat.

Die Felder sind in der Regel mehr oder weniger untereinander angeordnet. Aber auch Datenblätter lassen sich besser lesen, wenn sie eher vertikal ausgedehnt und mehr Datensätze im Zugriff sind.

Das scheint ein Rückschritt zu sein. Einerseits verbreiterten sich Monitore zunehmend, anderseits leistet der Ribbon gegenüber den früheren Menüleisten zusätzlich seinen Beitrag zu einem unpassenden Design.

Damit taucht die Frage auf, ob solchen Umständen abzuhelfen ist? Das Menüband kann man ausblenden, doch wie realisiert man dann eine Navigation durch die Anwendung?

Bild 2: Ohne Ribbon aber mit Switchboard ändert sich die verfügbare Arbeitsfläche

Eine Möglichkeit wäre ein **Switchboard** am linken Rand, das den eingebauten Navigationsbereich von Access ersetzte. Dafür käme ein Navigationsformular in Betracht. In der Beispieldatenbank haben wir so etwas entwickelt, ohne den Aufwand zu hoch zu treiben.

Switchboard-Formular

Aussehen soll das Ganze, wie in Bild 2. Der Ribbon ist ausgeblendet und am linken Rand befindet sich das Menü, über das die Elemente der Anwendung aufgerufen werden.

Die Arbeitsfläche vergrößert sich damit vertikal um immerhin 135 Pixel. Dass die horizontale Ausdehnung sich durch das Switchboard verringert, ist zu verschmerzen, denn noch immer haben wir es hier mit einem Breitformat zu tun.

Diese Lösung sieht einfacher aus, als sie es ist. Denn wir haben es mit folgenden Anforderungen zu tun:

- Das Switchboard-Formular muss links angenagelt bleiben
- Es sollte sich vertikal den Abmessungen des Access-Rahmenfensters anpassen können
- Formular und Berichte sollten das Switchboard nicht überdecken können. Das soll auch für maximierte Fenster gelten. Der linke Rand der Anwendungsformulare muss rechts vom Switchboard liegen.

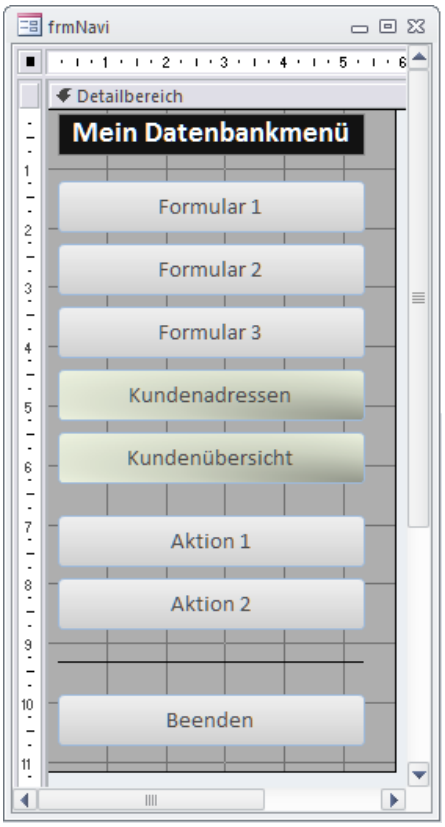


Bild 3: Das Switchboard-Formular frmNavi in der Entwurfsansicht

Sowohl das Switchboard-Formular, wie auch die Anwendungsformulare, müssen dazu automatisierten Positionierungsoperationen unterzogen werden. Die Basics dazu erfuhren Sie in der letzten Ausgabe.

Ribbon ausblenden

Das geht mit einer Zeile VBA:

```
DoCmd.ShowToolbar "Ribbon", _
                  acToolbarNo
```

Lassen Sie diese Zeile beim Start der Anwendung ausführen. Ob dies über ein Startformular geschieht, oder über ein **AutoExec**-Makro und eine öffentliche Funktion, das bleibt Ihnen überlassen. Einblenden können Sie den Ribbon im Zweifel wieder über den Aufruf der Methode **ShowToolbar**:

```
DoCmd.ShowToolbar "Ribbon", acToolbarYes
```

Navigationselemente

In der Beispieldatenbank nennt sich das Switchboard-Formular **frmNavi**, im Entwurf in Bild 3 dargestellt. Hier sind ein Label für den Titel und einige Schaltflächen untergebracht, die unterschiedliche Aktionen auslösen. Die Gestaltung ist willkürlich. Passen Sie sie nach ihren Vorstellungen an.

Wichtig jedoch ist die Einstellung **Ohne** für die Eigenschaft **Rahmenart** des Formulars, damit das Fenster dann nicht über die Titelzeile verschoben werden kann. Zur Sicherheit ist die Eigenschaft **Verschiebbar** zusätzlich auf **Nein** eingestellt und **MinMaxSchaltflächen** auf **Keine**. Selbstverständlich sind auch **Datensatzmarkierer** und **Navigationsschaltflächen** ausgeblendet.

Öffnen Sie dieses Formular direkt, so gibt es keine Möglichkeit, es mit der Maus zu schließen. Nur die Tastenkombination **STRG-F4** lässt es wieder verschwinden. Im Prinzip ließe sich auch das verhindern, indem Sie die Eigenschaft **Tastenvorschau** des Formulars (Tab **Ereignis** des Eigenschaftenblatts) auf **Ja** stellten und in die Ereignisprozedur für **Bei Taste Ab** diese Zeilen schrieben:

```
Private Sub Form_KeyDown( _
                  KeyCode As Integer, Shift As Inte-
ger)
    If KeyCode = 115 And Shift = 2 Then
        KeyCode = 0
    End If
End Sub
```

Dann hilft nur ein Ausdruck im VBA-Direktfenster weiter:

```
DoCmd.Close acForm, "frmNavi"
```

Damit Sie derlei während der Entwicklung nicht bewerkstelligen müssen, ist diese Zeile in das **Click**-Ereignis des Detailbereichs eingebaut. Klicken Sie also auf den grauen Hintergrund des Formulars, so schließt es sich und außerdem wird der möglicherweise fehlende Ribbon wieder eingeblendet.

```
Public Enum eAppFunction
    eAppOpenForm1
    eAppOpenForm2
    eAppOpenForm3
    eAppOpenFormKunden
    eAppOpenDSKunden
    eAppMsg1
    eAppMsg2
    eAppClose
End Enum

Public Function NaviFunction(fu As eAppFunction)
    Select Case fu
    Case eAppOpenForm1
        DoCmd.OpenForm "frmTest1"
    Case eAppOpenForm2
        DoCmd.OpenForm "frmTest2"
    Case eAppOpenForm3
        DoCmd.OpenForm "frmTest3"
    Case eAppOpenFormKunden
        DoCmd.OpenForm "frmAdressen"
    Case eAppOpenDSKunden
        DoCmd.OpenForm "frmAdressenDS", acFormDS
    Case eAppMsg1
        MsgBox "Das ist eine beliebige Meldung", vbInformation
    Case eAppMsg2
        MsgBox "Das ist eine weitere Meldung", vbExclamation
    Case eAppClose
        If MsgBox("Anwendung wirklich schließen?", _
                  vbQuestion Or vbYesNo,  _
                 "Bestätigen:") = vbYes Then
            Application.Quit acQuitSaveAll
        End If
    End Select
End Function
```

Listing 1: Navigationsfunktion

In die **Click**-Prozeduren der Schaltflächen könnten Sie direkt die gewünschten Aktionen programmieren, etwa den Aufruf eines Formulars:

```
DoCmd.OpenForm "frmAdressen"
```

Wir wollten jedoch die Navigationsfunktionen an zentraler Stelle verwalten, weshalb im Modul **mdlNavigation** die Funktion in Listing 1 zu finden ist.

Im Kopf des Moduls stehen **Enumerationskonstanten**, die die Aktionen versinnbildlichen. **eAppOpenFormKunden** etwa soll das Kundenformular öffnen, **eAppOpenDSKunden** das Datenblatt mit Kundenadressen. Diese Konstanten können der Funktion **NaviFunction** als Parameter übergeben werden. Der Vorteil besteht darin, dass nun **IntelliSense** bei Schreiben des Funktionsnamens an beliebiger Stelle gleich die Konstanten auflistet. Das Kundenformular rufen Sie also so auf:

```
NaviFunction eAppOpenFormKunden
```

Genau diese Zeile steht auch im **Click**-Ereignis der entsprechenden Schaltfläche des Switchboard-Formulars:

```
Private Sub cmd4_Click()
    NaviFunction eAppOpenFormKunden
End Sub
```

Die Funktion ermittelt nun über ein **Select-Case**-Statement die gewünschte Aktion und übernimmt sie dann im Programmzweig. In jeden **Case**-Abschnitt können Sie beliebigen VBA-Code unterbringen. Im Beispiel sind das neben dem Öffnen von Formularen etwa noch Meldungen. **eAppClose** ist eine Aktion, die die Anwendung schließen lassen soll. Das geschieht erst nach Bestätigung eines Nachfragedialogs. Bei Bejahung erfolgt das **Application.Quit**, welches die Anwendung und Access beendet.

Formulare an den Access-Rahmen anpassen

Wenn Sie das Access-Fenster vergrößern oder verkleinern, dann ändert sich auch die Arbeitsfläche. Da das Switchboard-Formular grundsätzlich den linken Rand einnehmen soll, muss es solchen Größenänderungen folgen. Doch wie kriegt es die mit?

```
Public Sub AdjustSize()
    Dim hwndMDI As Long
    Dim RCT As RECT
    hwndMDI = GetParent(Me.hwnd)
    Call GetWindowRect(hwndMDI, RCT)
    MDIHeight = 15 * (RCT.Bottom - RCT.Top - 4)
    MDIWidth = 15 * (RCT.Right - RCT.Left - 6) - Me.InsideWidth
    If Me.InsideHeight <> MDIHeight Then
        Me.SetFocus
        DoCmd.MoveSize 0, 0, , MDIHeight
    End If
End Sub
```

Listing 2: Anpassen des Switchboards an den Access-MDI-Bereich

In der letzten Ausgabe wurde erläutert, wie Sie mithilfe einer **API**-Funktion (**GetWindowRect**) die Ausdehnung des Access-Fensters zurückbekommen. Diese muss regelmäßig abgefragt werden, da es leider kein Ereignis gibt, welches durch Größenänderung des Access-Fensters hervorgerufen würde. Dafür kommt, wenn man auf komplizierte **API**-Routinen verzichten möchte, eigentlich nur ein Formular-**Timer** infrage. Also ist die Eigenschaft **Zeitgeberintervall** des Formulars **frmNavi** auf den Wert **300** eingestellt. Etwa drei Mal pro Sekunde ermittelt die **Timer**-Prozedur dann die Abmessungen des Rahmenfensters, um darauf gegebenenfalls reagieren zu können:

```
Private Sub Form_Timer()
    AdjustSize
End Sub
```

Die eigentliche Routine ist in die Prozedur **AdjustSize** (Listing 2) ausgelagert. Grund dafür ist, dass diese Prozedur als **Public** deklariert ist, wodurch Sie auch von außen aufgerufen werden könnte:

```
Forms!frmNavi.AdjustSize
```

Nun ist es so, dass das Switchboard ja nicht direkt dem Access-Rahmen folgen soll, sondern der Arbeitsfläche. Auch ohne Größenänderung desselben ändert etwa auch das Ein- und Ausschalten des Ribbon den **MDI**-Bereich. Deshalb benötigen wird

das Fenster-Handle der Arbeitsfläche und nicht das von Access (**hwndAccessApp**). Von einem Formular heraus ist es zum Glück nicht schwierig, an dieses zu gelangen. Da der Arbeitsbereich immer das Elternfenster eines Formulars darstellt, außer, wenn es im **Popup**-Modus betrieben wird, muss nur das Elternfenster mit der API-Funktion **GetParent** ermittelt werden:

```
hwndMDI = GetParent (Me.hwnd)
```

Die Deklaration der **API**-Funktion steht im Modul **mdlNavigation**:

```
Public Declare Function GetParent _
    Lib "user32.dll" ( _
    ByVal hwnd As Long) As Long
```

Nun können über **GetWindowRect** auf die Variable **RCT** vom API-Typ **RECT** die Abmessungen und Position des **MDI**-Bereichs erhalten werden. Breite und Höhe werden in den globalen Variablen **MDIWidth** und **MDIHeight** zwischengespeichert. Dabei müssen die Pixel-Angaben erst in **Twips** umgerechnet werden, was durch Multiplikation mit **15** geschieht. Außerdem ist ein Offset von 4 oder 6 abzuziehen, der dem Rahmen des **MDI**-Bereichs geschuldet ist.

Nun kommt der Vergleich: Ist der **MDI**-Bereich (**MDIHeight**) ungleich der Höhe des Formulars (**InsideHeight**), so kommt es zur eigentlichen Anpassung über die Methode **DoCmd.MoveSize**. Hier wird keine API-Funktion benötigt. Die Methode wirkt auf das aktuelle Access-Objekt, weshalb der Fokus zunächst über SetFocus auf das Switchboard-Formular gelegt wird. Die ersten beiden Parameter mit den Werten **0** positionieren das Formular in die linke obere Ecke. Darauf folgt ein leerer Parameter, weil die Breite des Formulars unangetastet bleiben soll. Und schließlich wird seine Höhe über das zwischengespeicherte **MDIHeight** festgelegt.

Diese Überprüfung findet, wie erwähnt, alle 300 ms statt. Ändern Sie die Größe des Access-Fensters, so verstreicht unter Umständen genau diese Zeit, bis sich das Formular den neuen Bedingungen anpasst. Verringern Sie gegebenenfalls das Zeitgeberintervall, wenn die Anpassung weniger augenscheinlich ausfallen soll.

Warum überhaupt der Vergleich mit der aktuellen Größe des Formulars? Könnte man es nicht einfach regelmäßig dem **MDI**-Bereich anpassen? Das wäre möglich, hätte aber fatale Folgen: Wäre ein weiteres Formular geöffnet, etwa das Kundenformular, so

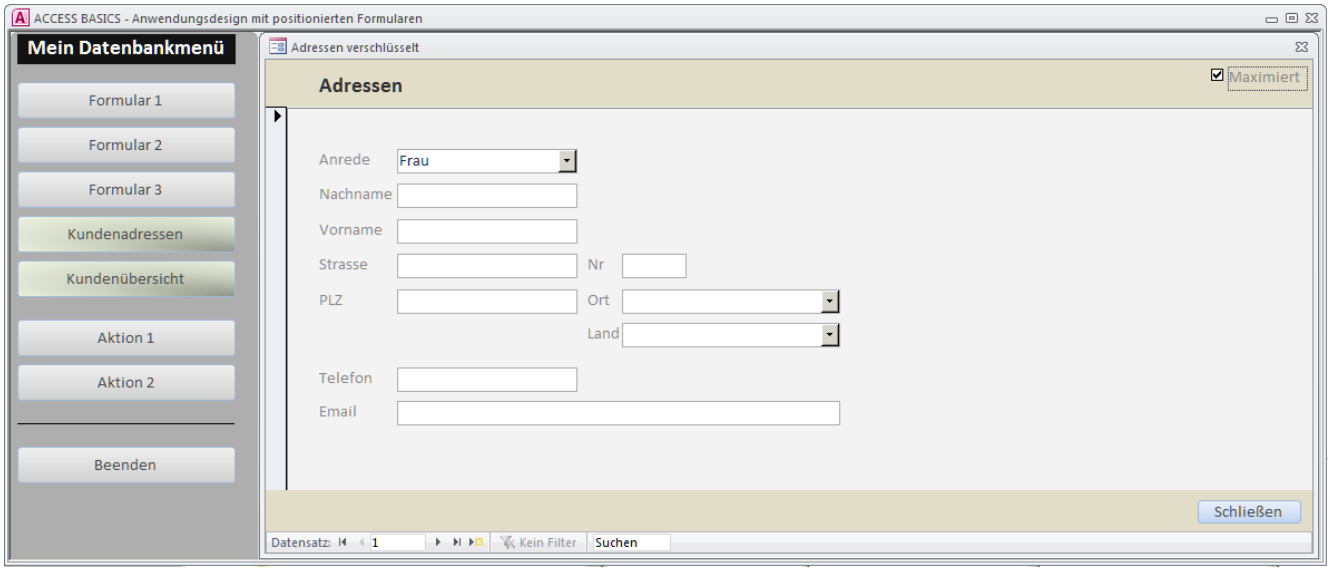


Bild 4: Das Kundenformular dimensioniert sich automatisch gemäß der verbleibenden Arbeitsfläche und folgt Größenänderungen

Berichte zur Laufzeit filtern

Die Datenquelle eines Berichts verhält sich gemeinhin statisch. Die Auswahl der Daten geschieht entweder im Vorfeld über die Filterung einer Abfrage oder über Gruppierungsebenen des Berichts. Mit der in Access 2007 eingeführten Berichtsansicht jedoch kommt mehr Leben ins Spiel. Diese erlaubt zur Laufzeit eine Modifikation der zugrundeliegenden Daten. Der folgende Beitrag zeigt zudem, wie Sie das Layout über Bedingte Formatierung von Textfeldern aufpeppen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1701_BerichtsFilter.accdb**.

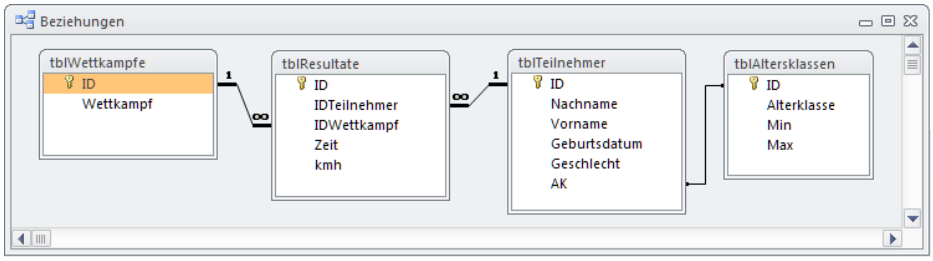


Bild 1: Das Datenmodell der Beispieldatenbank zu fingierten Wettkampfdaten

Die Berichtsansicht

Ein Bericht kommt üblicherweise für den gestalteten Ausdruck von Daten zur Anwendung. Wollten Sie dabei mehrere Auszüge der gleichen Daten mit unterschiedlichen Bedingungen ausdrucken, so war dies einst ein umständlicheres Unterfangen. Die Auswahl der Daten hatte etwa über eine Abfrage stattzufinden, in der Sie die Bedingungen über Parameter von außen steuerten. Hier könnten Sie VBA-Funktionen verwenden, die statt fest eingebauter Parameter variable Vergleichswerte liefern.

Bei der **Berichtsansicht** benötigen Sie derlei Kniffe nicht. Sie ermöglicht die dynamische Filterung der Datenquelle zur Laufzeit, ähnlich, wie bei einem Formular. Sie schalten die Berichtsansicht entweder über die entsprechende Ribbon-Schaltfläche ein, oder Sie setzen sie gleich im Entwurf des Berichts, indem Sie die Eigenschaft **Standardansicht** auf **Berichtsansicht** einstellen. Diese Ansicht ist, im Gegensatz zur **Layoutansicht,** immer verfügbar und muss in den Optionen der Datenbank nicht ausdrücklich erlaubt werden. Wohl aber existiert eine Eigenschaft **Berichtsansicht zulassen** unter dem **Format**-Reiter des Eigenschaftenblatts, die auf **Ja** stehen muss, damit der Bericht zur Laufzeit in diese umgeschaltet werden kann. Sonst stehen nur die **Seitenansicht** und gegebenenfalls die **Layoutansicht** zur Verfügung.

Tatsächlich verändert der Bericht in dieser Ansicht komplett sein Wesen und wird imgrunde zu einem Formular. Ein Fenster-Tool etwa zeigt, dass sich die verwendete Fensterklasse nun nicht mehr **OReport** nennt, sondern **OForm**, also identisch zur für Formularfenster verwendeten.

Für das Beispiel zu diesem Beitrag kommt eine Datenbank zum Einsatz, die bereits in der Ausgabe **5/2015**, **Daten ins Web exportieren**, zur Demonstration von Web-Exporten diente. Das Datenmodell für jene Wettkampfdatenbank wird hier nicht nochmals erläutert. Das Beziehungsmodell mit seinen vier Tabellen ist in Bild 1 gezeigt. Es gibt Radwettkämpfe, Teilnehmer in verschiedenen Altersklassen und deren Wettkampfresultate.

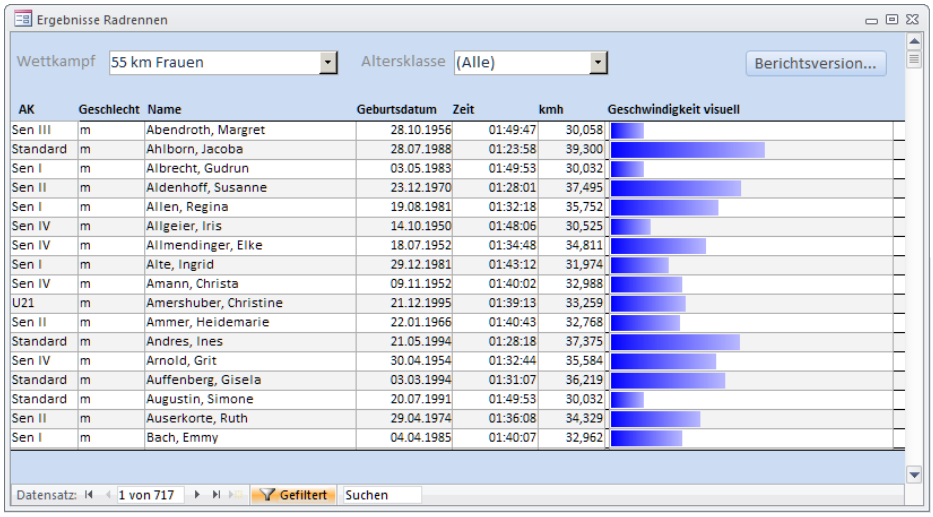
Ziel der aktuellen Beispieldatenbank ist nun nicht ein Web-Export, sondern der Ausdruck der Resultate über einen Bericht. Diese sollen wahlweise nach Wettkampfkategorien und Altersklassen filterbar sein. Das damals dazu verwendete Formular ist in der Datenbank ebenfalls enthalten (**frmErgebnisse**), aber um eine Spalte erweitert, die die Durchschnittgeschwindigkeit visuell darstellt (siehe Bild 2). Dabei kommt für dieses Textfeld kein Grafiksteuerelement zum

Einsatz, sondern ein Textfeld mit **Bedingter Formatierung**, dessen Einstellungen wir gleich noch eingehender beleuchten.

Die beiden Kombinationsfelder oben dienen der Filterung der Resultate, wobei wir hierfür kleine VBA-Ereignisprozeduren benutzen. Das Formular baut direkt auf der Abfrage **qry_Resultate** auf und kommt im Endlosmodus daher.

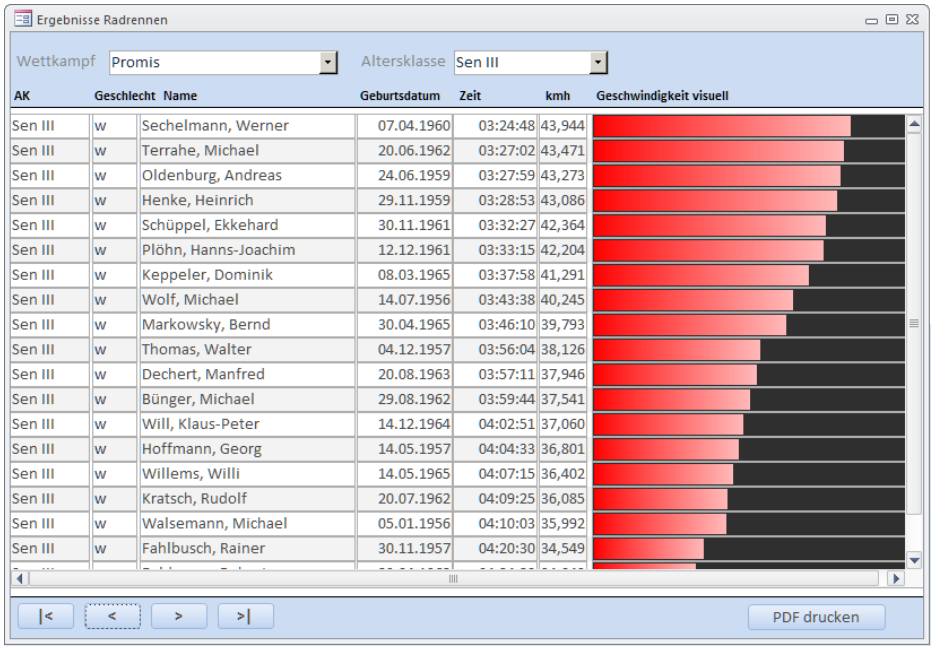
Mit Klick auf die Schaltfläche Berichtsversion... gelangen Sie zu einem anderen Formular, das jedoch einen Unterbericht einbaut hat, der die Ergebnisse im Prinzip analog darstellt, auch wenn sich die Farbgebung hier etwas unterscheidet. Dieses Formular (siehe Bild 3) und seinen Unterbericht sehen wir uns genauer an.

Das umrahmende Formular ist selbst an keine Datenquelle gebunden und steuert lediglich den Unterbericht. Eben dies ist eine Neuerung seit **Access 2007**. Früher konnte man einen Unterbericht nicht als Steuerelement in ein Formular integrieren, sondern nur wiederum in Berichte. Diese Möglichkeit ist allerdings nur auf Unterberichte beschränkt, die die **Berichtsansicht** als Standard eingestellt haben. Die **Seitenansicht** kann nicht in ein Formular eingebettet werden.



AK	Geschlecht	Name	Geburtsdatum	Zeit	kmh	Geschwindigkeit visuell
Sen III	m	Abendroth, Margret	28.10.1956	01:49:47	30,058	
Standard	m	Ahlborn, Jacoba	28.07.1988	01:23:58	39,300	
Sen I	m	Albrecht, Gudrun	03.05.1983	01:49:53	30,032	
Sen II	m	Aldenhoff, Susanne	23.12.1970	01:28:01	37,495	
Sen I	m	Allen, Regina	19.08.1981	01:32:18	35,752	
Sen IV	m	Allgeier, Iris	14.10.1950	01:48:06	30,525	
Sen IV	m	Allmendinger, Elke	18.07.1952	01:34:48	34,811	
Sen I	m	Alte, Ingrid	29.12.1981	01:43:12	31,974	
Sen IV	m	Amann, Christa	09.11.1952	01:40:02	32,988	
U21	m	Amershuber, Christine	21.12.1995	01:39:13	33,259	
Sen II	m	Ammer, Heidemarie	22.01.1966	01:40:43	32,768	
Standard	m	Andres, Ines	21.05.1994	01:28:18	37,375	
Sen IV	m	Arnold, Grit	30.04.1954	01:32:44	35,584	
Standard	m	Auffenberg, Gisela	03.03.1994	01:31:07	36,219	
Standard	m	Augustin, Simone	20.07.1991	01:49:53	30,032	
Sen II	m	Auserkorte, Ruth	29.04.1974	01:36:08	34,329	
Sen I	m	Bach, Emmy	04.04.1985	01:40:07	32,962	

Bild 2: Das Formular **frmErgebnisse** zur Auswertung der Wettkampfresultate



AK	Geschlecht	Name	Geburtsdatum	Zeit	kmh	Geschwindigkeit visuell
Sen III	w	Sechelmann, Werner	07.04.1960	03:24:48	43,944	
Sen III	w	Terrahe, Michael	20.06.1962	03:27:02	43,471	
Sen III	w	Oldenburg, Andreas	24.06.1959	03:27:59	43,273	
Sen III	w	Henke, Heinrich	29.11.1959	03:28:53	43,086	
Sen III	w	Schüppel, Ekkehard	30.11.1961	03:32:27	42,364	
Sen III	w	Plöhn, Hanns-Joachim	12.12.1961	03:33:15	42,204	
Sen III	w	Keppeler, Dominik	08.03.1965	03:37:58	41,291	
Sen III	w	Wolf, Michael	14.07.1956	03:43:38	40,245	
Sen III	w	Markowsky, Bernd	30.04.1965	03:46:10	39,793	
Sen III	w	Thomas, Walter	04.12.1957	03:56:04	38,126	
Sen III	w	Dechert, Manfred	20.08.1963	03:57:11	37,946	
Sen III	w	Bünger, Michael	29.08.1962	03:59:44	37,541	
Sen III	w	Will, Klaus-Peter	14.12.1964	04:02:51	37,060	
Sen III	w	Hoffmann, Georg	14.05.1957	04:04:33	36,801	
Sen III	w	Willems, Willi	14.05.1965	04:07:15	36,402	
Sen III	w	Kratsch, Rudolf	20.07.1962	04:09:25	36,085	
Sen III	w	Walsemann, Michael	05.01.1956	04:10:03	35,992	
Sen III	w	Fahlbusch, Rainer	30.11.1957	04:20:30	34,549	

Bild 3: Hier haben wir es nicht mehr mit einem Formular in Endlosansicht zu tun, sondern mit einem Unterbericht im Modus **Berichtsansicht** (**frmErgebnisseReport**)

Bild 4: Der Entwurf des Unterberichts für das Formular **frmErgebnisseReport**

Den recht einfachen Entwurf des Berichts **rptErgebnisse**, der nur einen Detailbereich enthält, finden Sie in Bild 4 dargestellt.

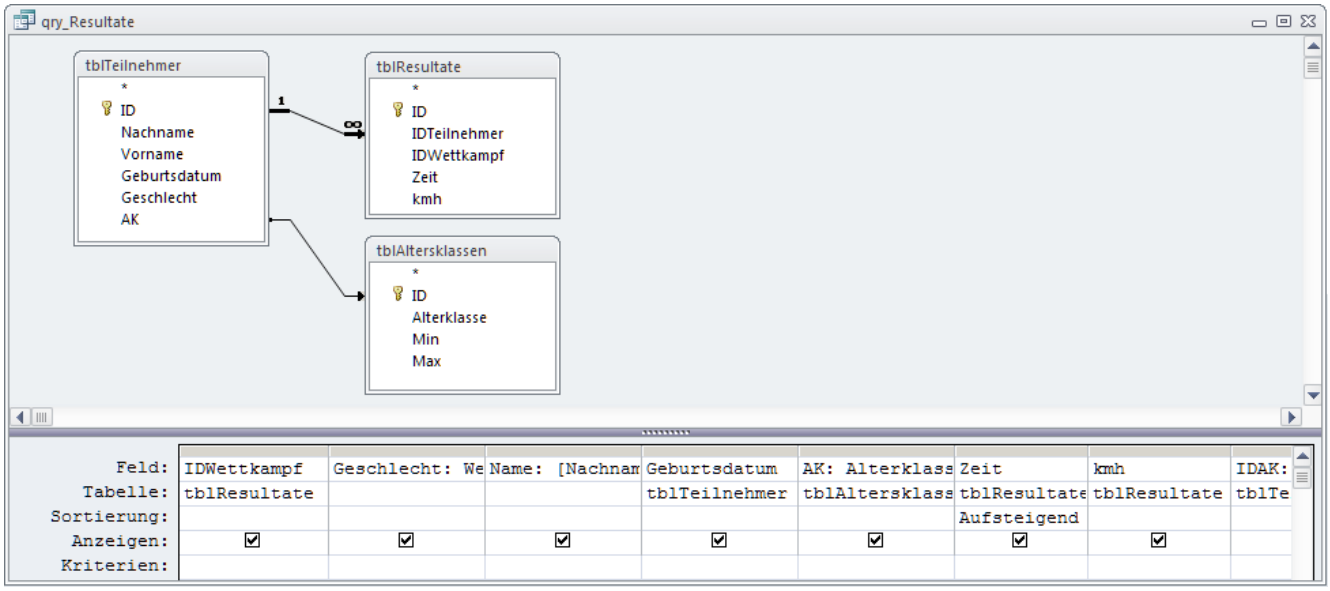


Bild 5: Diese Abfrage stellt die Datenherkunft des Unterberichts dar und kombiniert drei Tabellen zur Darstellung der Resultate

Die Felder der Datenherkunft **qry_Resultate** (siehe Bild 5) liegen nebeneinander im Detailbereich, so dass sich letztendlich eine Tabellendarstellung ergibt. Das Feld für die Geschwindigkeit (**kmh**) ist zwei Textboxen zugewiesen, wobei die linke den Wert anzeigt, die rechte einen korrelierenden Farbbalken. Für das Namensfeld ist die Eigenschaft **Vergrößerbar** auf **Ja** eingestellt, damit längere Teilnehmernamen nicht abgeschnitten werden, sondern das Textfeld nach unten ausdehnen. Hier zeigt sich bereits ein Vorteil des Unterberichts gegenüber einem Unterformular, denn dieses kann Steuerelemente ja nicht dynamisch skalieren.

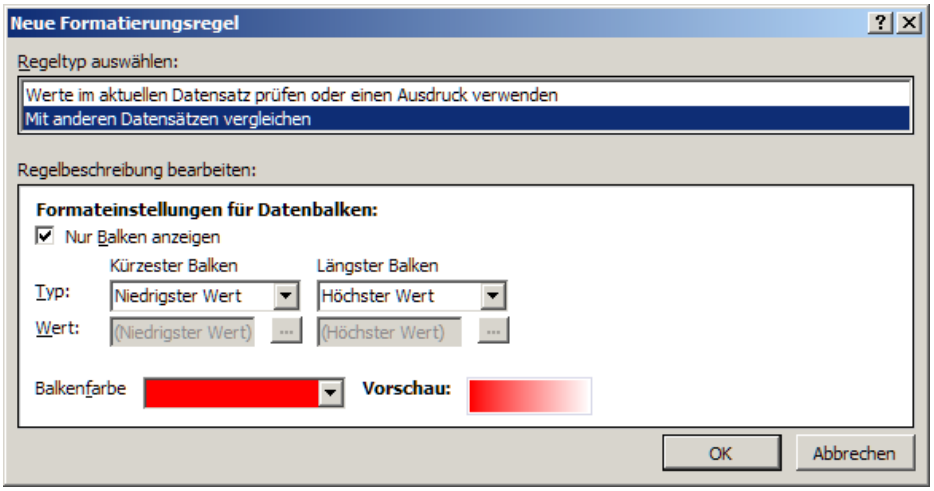


Bild 6: Der zweite Dialog zur Einstellung des Farbbalkens für die Feldformatierung

Farbbalken über Bedingte Formatierung

Klicken Sie im Berichtsentwurf auf das rechte Textfeld und wählen aus dem Kontextmenü den Eintrag **Bedingte Formatierung...** Alternativ können Sie dafür die gleichnamige Schaltfläche im **Ribbon-Tab Format** verwenden. Damit öffnet sich ein Dialogfenster zur Eingabe der Einstellungen für die Formatbedingungen, der zunächst leer ist. Sie legen deshalb eine neue Formatregel über den linken Button **Neue Regel** an, was einen untergeordneten Detaildialog hervorbringt (siehe Bild 6).

Entscheidend ist hier die Auswahl des **Regeltyps** oben. Mit **Werte im aktuellen Datensatz prüfen** haben Sie dieselben Einstellmöglichkeiten, wie auch früher schon in anderen Access-Versionen, und kön-

nen etwa Schriftfarbe oder Hintergrund des Textfelds in Abhängigkeit vom Wert oder von einem Vergleichsausdruck bestimmen. Mit dem Typ **Mit anderen Datensätzen vergleichen** aber ändert sich die Gestalt des Dialogs komplett. Man hätte diesen Typ auch gleich **Farbbalken** nennen können.

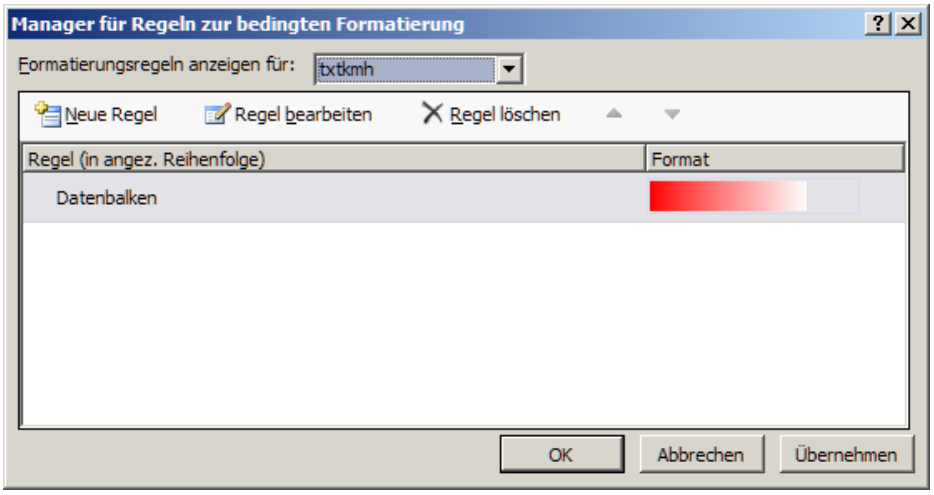


Bild 7: Der erste Dialog zur Einstellung der Bedingten Formatierung eines Textfelds

Aktivieren Sie **Nur Balken anzeigen**, so fehlt im Feld nachher die textuelle Wertangabe. Die Farbe des Balkens bestimmen Sie im entsprechenden Kombinationsfeld unten, die Vorschau verdeutlicht, wie das dann ungefähr aussehen wird. Grundsätzlich können Sie keinen Farbverlauf zwischen zwei Farben einstellen, denn links ist immer die gewählte Farbe, rechts der Verlauf nach Weiß. Denn die Vorschau lässt hier den Hintergrund des Textfelds außer Acht und rendert diesen nicht.

Tatsächlich handelt es sich rechts aber nicht um Weiß, sondern um den Alpha-Kanal des Balkens. Das bedeutet, dass hier der Hintergrund des Textfelds zunehmend durchscheint. Im Entwurf weiter oben ist zu sehen, dass der Hintergrund auf ein Schwarz eingestellt ist. Somit entsteht schließlich ein Verlauf von Rot nach Schwarz.

Ebenso könnten Sie auch andere Hintergründe verwenden und den Verlauf damit modifizieren. Sogar mehrere Balkenregeln parallel sind möglich, wodurch im Verein mit den Bedingungswerten weitere Gestaltungen entstehen können.

Wir haben für den Bedingungstyp im Beispiel die Angaben **Niedrigster Wert** und **Höchster Wert** gemacht. Damit ermittelt die Bedingung den kleinsten aller Geschwindigkeitswerte der Abfrage, welcher rein Rot wird, und den größten, der vollständig transparent wird. Zwischen diesen Polen bewegt dann sich der Verlauf.

Komplizierter wird es bei der Auswahl von **Zahl** für den Typ. Hier könnten Sie in den dann aktivierten **Wert**-Feldern selbst Werte eintragen, oder sogar über Ausdrücke errechnen lassen, die den Bereich bestimmen. Wir belassen es jedoch bei der einfachsten Variante.

Nach dem Bestätigen mit **OK** schließt sich der Detaildialog und im Hauptdialog findet sich nun die neue Regel (Bild 7), die Sie übrigens über Doppelklick auch später noch verändern können. Außerdem können Sie nun auch weitere Regeln anlegen, wobei das nicht zwingend Farbbalken sein müssen, sondern ebenso die üblichen Bedingten Formatierungen für Schriftart und Hintergrund des Felds. Experimentieren Sie einfach etwas mit einigen Kombinationen herum, um die Gestaltungsmöglichkeiten auszuloten.

Bericht im Formular

Den nun fertiggestellten Bericht integrieren Sie in den Detailbereich des Formulars **frmErgebnisse-Report**, wie in Bild 8, indem Sie ein Steuerelement **Unterformular/-bericht** (Ribbon-Tab **Entwurf**) in ihm anlegen. Stellen Sie für die Eigenschaft **Verankern** das **Nach unten und quer dehnen** ein (Ribbon-Tab **Anordnen**), damit der Unterbericht möglichen Größenänderungen des Detailbereichs folgt. Als **Herkunftsobjekt** legen Sie **Bericht.rptErgebnisse** fest. Da das Formular selbst keine Datenherkunft aufweist, entfallen hier entsprechende Feldverknüpfungen.

Tabelleninhalte verschlüsseln

Der Schutz sensibler Daten in Access-Datenbanken ist keine triviale Angelegenheit. Die Tabellen liegen offen in nur einer Datei, die problemlos auf einen USB-Stick kopiert werden kann, um deren Inhalte anderenorts in aller Ruhe zu inspizieren. Frühere Access-Versionen bis Office 2003 erlaubten die Absicherung über ein Berechtigungsmodell, das später leider ersatzlos gestrichen wurde. Wer auf Nummer Sicher gehen will, der sollte sich Gedanken über eine Datenverschlüsselung machen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1701_Cryto.accdb**.

Strings verschlüsseln per VBA

Zu den Daten, die in einer Tabelle nicht unbedingt auf den ersten Blick einsehbar sein müssen, gehören vor allem Texte. In einer Adressenliste etwa beträfe dies die Namensfelder, die Telefon- und E-Mail-Daten, sowie andere Adressfelder. Verschlüsselt man diese, so kann ein Unbefugter mit ihnen nicht allzu viel anfangen.

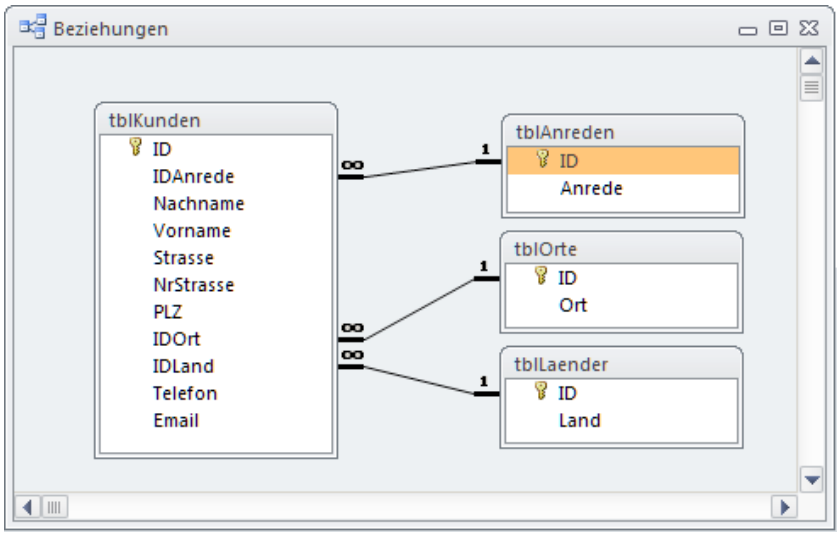


Bild 1: Das Datenmodell der Beispieldatenbank zu Kundenadressen

In der Beispieldatenbank bekommen wir es wieder einmal mit der bekannten Kundentabelle zu tun, deren Datenmodell Bild 1 wiedergibt. Neben den Nachschlagetabellen zu Anreden, Orten und Ländern enthält die Haupttabelle **tblKunden** die üblichen Adressfelder. Wir machen uns zur Aufgabe, die folgenden Felder zu verschlüsseln: **Nachname**, **Vorname**, **Strasse**, **Telefon** und **Email**. Das sollte ausreichen, um die Datensätze so zu verunstalten, dass aus ihnen keine Rückschlüsse auf den ursprünglichen Inhalt mehr möglich sind.

Der Ausgangszustand zeigt sich in Bild 2. Hier sind alle Adressdaten in konventioneller Weise abgespeichert. Jeder kann sie einsehen und gegebenenfalls stehlen. Dazu benötigt man noch nicht einmal eine **Office**-Installation, denn die Datenzugriffsbibliothe-

tblKunden

ID	IDAnrede	Nachname	Vorname	Strasse	NrStrasse	PLZ	IDOrt	IDLand	Telefon	Email
23	Herr	Fuchs	Bruno	Hölderlinstr.	62	41236	Mönchengladbach	Nordrhein-Westfalen	+49 747413384	Bruno@Fuchs.de
25	Frau	Schwarz	Roswitha	Bahner Straße	6	41238	Mönchengladbach	Nordrhein-Westfalen		
29	Frau	Hoffmann	Marlene	Forstweg	14	42119	Wuppertal	Nordrhein-Westfalen	+49 55755985	
31	Herr Dr.	Köhlhoff	Rebekka						+49 310101032	Rebekka.Köhlhoff@yahoo.de
37	Frau	Schwarzer-Queise	Kurt	Mintarder Weg	49	40885	Ratingen	Nordrhein-Westfalen	+49 64263112	Kurt@Schwarzer-Queiser.de
41	Frau	Levermann	Monika						+49 17214534	
43	Herr	Liersch	Hans	Wachtelgarten	3	46499	Hamminkeln	Nordrhein-Westfalen	+49 216195	
47	Frau	Szamlewski	Thomas	Wulfshäger Str.	23	50169	Kerpen	Nordrhein-Westfalen	+49 9533410	
49	Herr	Ricken	Brigitte	Am Sonnenbrink	1a	50670	Köln	Nordrhein-Westfalen	+49 47894977	

Datensatz: 8 von 9706 | Kein Filter | Suchen

Bild 2: Einige Datensätze der normalen Kundentabelle in der Datenblattansicht

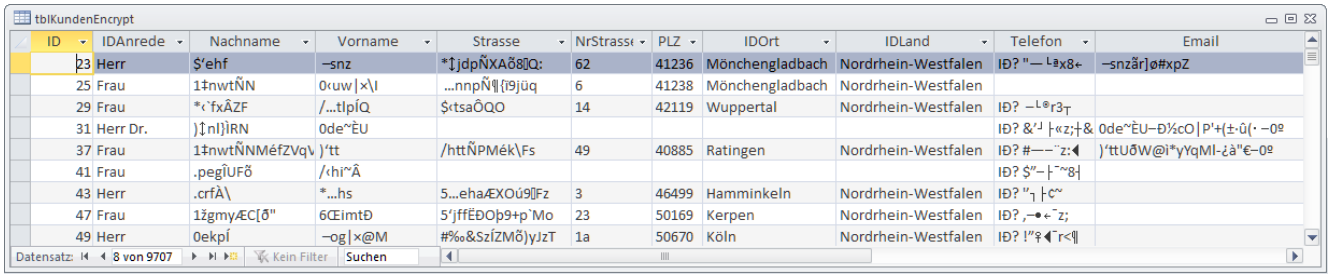


Bild 3: Dieselben Datensätze der verschlüsselten Kundentabelle in der Datenblattansicht

ken **DAO** und **ADO** bringt **Windows** bereits von Haus aus mit. Selbst unter **Linux/Android** und etwa das Zugriffssystem **JDBC** käme man an die Tabelle heran. Das wäre zwar auch nach Verschlüsselung noch der Fall, doch dann nützt der Einblick wenig, wie Bild 3 beweist. Hier sind die erwähnten Textfelder über eine Aktualisierungsabfrage einer **RC4**-Verschlüsselung unterzogen worden, auf die wir noch zu sprechen kommen.

Verschlüsselungsalgorithmen

Access und VBA selbst bringen direkt keine Verschlüsselungsmethoden mit. Sie sind deshalb auf eigene VBA-Implementierungen in Form von Funktionen angewiesen.

Listing 1 zeigt eine extrem einfache Methode, die sich die VBA-Funktion **StrReverse** zunutze macht. Sie kehrt einen übergebenen String schlicht um:

```
CryptReverse "Access Basics"
->  scisaB sseccA /  Access Basics
```

Das Ergebnis stellt einen Hacker vor keinerlei Herausforderungen. Der Algorithmus ist ganz offensichtlich – von Verschlüsselung kann man hier wohl kaum sprechen. Anders sieht es bei der Routine zum zweiten Beispiel aus (siehe Listing 2), das alle Zeichen des übergebenen Strings einer binären **XOR**-Operation unterzieht:

```
CryptXOR "Access Basics"
->  ¾œœšŒŒƁ½ŽŒ-œŒ  / Access Basics
```

In der ersten **For-Next**-Schleife werden die einzelnen Zeichen des Strings zunächst über die **Mid**-Funktion ermittelt und deren **ASCII**-Ordinalzahl in der Variablen **nOrd** (**Long**) gespeichert. Diese wird binär mit **Xor 255** behandelt und anschließend per **Chr**-Funktion wieder in ein Zeichen verwandelt. Die Stringvariable **sCrypt** nimmt diese Zeichen nacheinander auf. Die zweite Schleife macht die Probe und wiederholt den

```
Sub CryptReverse(S As String)
    Dim sCrypt As String

    sCrypt = StrReverse(S)
    S = StrReverse(sCrypt)
    Debug.Print sCrypt, "/", S
End Sub
```

Listing 1: Textverschlüsselung über eine String-Umkehr

```
Sub CryptXOR(S As String)
    Dim sCrypt As String
    Dim i As Long, n As Long
    Dim nOrd As Long
    n = Len(S)
    sCrypt = String(n, 0)
    For i = 1 To n
        nOrd = Asc(Mid(S, i, 1))
        Mid(sCrypt, i) = Chr((nOrd Xor 255))
    Next i
    S = String(n, 0)
    For i = 1 To n
        nOrd = Asc(Mid(sCrypt, i, 1))
        Mid(S, i) = Chr((nOrd Xor 255))
    Next i
    Debug.Print sCrypt, "/", S
End Sub
```

Listing 2: Textverschlüsselung über ein Byte-weises XOR

Vorgang auf genau die gleiche Weise. Verschlüsseln und Entschlüsseln geschehen also über den identischen Algorithmus, so, wie dies auch bei **StrReverse** der Fall ist. Tatsächlich handelt es sich bei Anwendung von **Xor 255** um eine ähnliche Angelegenheit, da hier die Bits eines Bytes schlicht umgekehrt werden:

```
00110101 -> 11001010
```

Der Ergebnis-String ist nun ziemlich kryptisch, doch auch hier hat ein Hacker sicher schnell die Lösung gefunden. Also versuchen wir es ihm noch schwerer zu machen.

Die Routine **CryptXORReverse** (Listing 3) kombiniert die beiden beschriebenen Prozeduren. Erst greift wieder der **XOR**-Algorithmus, doch anschließend kommt noch **StrReverse** zum Einsatz. Ergebnis:

```
CryptXORReverse "Access Basics"
->  Œœ-ŒŽ½ßŒŒŠœœ¾  /  Access Basics
```

```
Sub CryptXORReverse(S As String)
    Dim sCrypt As String
    Dim i As Long, n As Long
    Dim nOrd As Long
    n = Len(S)
    sCrypt = String(n, 0)
    For i = 1 To n
        nOrd = Asc(Mid(S, i, 1))
        Mid(sCrypt, i) = Chr((nOrd Xor 255))
    Next i
    sCrypt = StrReverse(sCrypt)
    S = String(n, 0)
    For i = 1 To n
        nOrd = Asc(Mid(sCrypt, i, 1))
        Mid(S, i) = Chr((nOrd Xor 255))
    Next i
    S = StrReverse(S)
    Debug.Print sCrypt, "/", S
End Sub
```

Listing 3: Textverschlüsselung über Byte-weises XOR plus String-Umkehr

Deutlich schwerer nachvollziehbar sind Algorithmen, die sich etwa die **Mod**-Funktion von VBA zunutze machen, welche die Ordinalzahl eines Zeichens quasi rotieren lässt. Listing 4 demonstriert so eine Lösung.

Statt **Xor** wird der Ordinalzahl in **nOrd** hier die willkürliche Zahl **144** hinzuaddiert und über **Mod** sichergestellt, dass dabei keine Zahl **>255** herauskommt.

Ansonsten gleicht der Algorithmus dem aus Listing 2. Zum Entschlüsseln können Sie allerdings nicht die identische Funktion verwenden. Der zweite Teil der Routine zeigt, dass hier zunächst vom Byte-Wert die Zahl **144** subtrahiert werden muss, um daraus wieder das korrekte Zeichen zu machen. Das Ergebnis hier:

```
CryptAddMod "Access Basics"
->  Ñóóõ°Òñùó  /  Access Basics
```

Die bisherigen Algorithmen haben den Nachteil, dass der Ergebnis-String die gleiche Länge hat, wie der Eingabe-String.

Die Routine **CryptDist** in Listing 5 ändert das. Hier bekommt man einen String variabler Länge heraus, in

```
Sub CryptAddMod(S As String)
    Dim sCrypt As String
    Dim i As Long, n As Long
    Dim nOrd As Long
    n = Len(S)
    sCrypt = String(n, 0)
    For i = 1 To n
        nOrd = Asc(Mid(S, i, 1))
        Mid(sCrypt, i) = Chr(((nOrd + 144) Mod 256))
    Next i
    S = String(n, 0)
    For i = 1 To n
        nOrd = Asc(Mid(sCrypt, i, 1)) - 144
        Mid(S, i) = Chr((nOrd + 256) Mod 256)
    Next i
    Debug.Print sCrypt, "/", S
End Sub
```

Listing 4: Textverschlüsselung über die Ordinalzahl der Zeichen und Mod

```
Sub CryptDist(S As String)
    Dim sCrypt As String
    Dim i As Long, n As Long
    Dim nOrd As Long
    Dim arr() As String
    For i = 1 To Len(S)
        nOrd = Asc(Mid(S, i, 1))
        sCrypt = sCrypt & StrReverse(CStr(nOrd)) & Chr$(224)
    Next i
    S = String(Len(S), 0)
    arr = Split(sCrypt, Chr$(224))
    For i = 0 To UBound(arr) - 1
        nOrd = Val(StrReverse(arr(i)))
        Mid(S, i + 1) = Chr(nOrd)
    Next i
    Debug.Print sCrypt, "/", S
End Sub
```

Listing 5: Textverschlüsselung über mehrere kompinierte Methoden

```
Sub CryptHex(S As String)
    Dim sCrypt As String
    Dim i As Long, n As Long
    Dim sOrd As String, nOrd As Long
    For i = 1 To Len(S)
        sOrd = "00"
        Mid(sOrd, 1) = StrReverse(Hex(Asc(Mid(S, i, 1))))
        sCrypt = sCrypt & sOrd
    Next i
    S = ""
    For i = 1 To Len(sCrypt) Step 2
        sOrd = Mid(sCrypt, i, 2)
        nOrd = Val("&H" & StrReverse(sOrd))
        S = S & Chr(nOrd)
    Next i
    Debug.Print sCrypt, "/", S
End Sub
```

Listing 6: Textverschlüsselung unter Einsatz der Hex-Funktion

dem nur noch sehr schwer das Original auszumachen ist:

```
CryptDist "Access Basics"
->  56à99à99à101à511à511à23à66à79à511à501à99à511à  /
Access Basics
```

Eine Beschreibung der Funktionsweise lassen wir an dieser Stelle außen vor. Trotz der Kürze der Prozedur fällt sie doch recht komplex aus.

Und schließlich ein letztes Beispiel (siehe Listing 6), das die **Hex**-Funktion von VBA heranzieht:

```
CryptHex "Access Basics"
-> 14363656373702241637963637 / Access Basics
```

Auch hier lässt das Ergebnis sicher kaum Rückschlüsse auf den Ausgangs-String zu.

Sie finden die beschriebenen Prozeduren alle im Modul **mdlCrypt** der Beispieldatenbank. Sie demonstrieren im Prinzip lediglich die Vorgehensweise bei der Verschlüsselung von Texten. Sie müssen sich jedoch keine Gedanken um eine eigene wasserdichte Implementierung machen, da derartige Routinen zuhauf im Netz zu finden sind.

Als Standardverfahren hat sich etwa der **RC4**-Algorithmus bewährt. Die Kodierung benötigt neben dem Eingabe-String noch einen Schlüssel, der dann bei der Dekodierung ebenso verwendet wird. Die Funktion **RC4** im Modul **mdlRC4**, welche wir hier nicht abbilden, erzeugt über dieses Paar einen Ergebnis-String:

```
globkey = "akrakadabra"
GenerateKey
S = RC4("Access Basics")
Debug.Print S, "/", RC4(S)
->  üÁ6-´ÑÚ_¥Ó  / Access Basics
```

Die globale Variable **globKey** nimmt den Schlüssel auf. Dieser kann von der Prozedur **RC4** allerdings nicht direkt verwendet werden, sondern muss erst über einen Aufruf der Hilfsroutine **GenerateKey** in ein Byte-Array überführt werden.

Einmal erzeugt, arbeitet die Routine forthin mit diesem Schlüssel-Array. Das Schöne an der Prozedur

Ribbons, Teil III: Interaktion im Ribbon

Das Ribbon lässt sich nicht so einfach programmieren wie es bei Menüleisten, Symbolleisten oder Kontextmenüs der Fall war. Diese Elemente ließen sich nach Wunsch mit Objektvariablen referenzieren und zum Beispiel aktivieren oder deaktivieren. Das Ein- oder Ausblenden von Menüs war auch recht einfach. Beim Ribbon gibt es einen alternativen Programmieransatz, bei dem der XML-Code zur Beschreibung des Aussehens des Ribbons und der VBA-Code für seine Steuerung interagieren sollen. Welche Vorbereitungen dazu notwendig sind und wie die Programmierung im Detail aussieht, zeigt dieser Artikel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1701_Ribbon.accdb**.

Voraussetzungen

Beachten Sie, wie im ersten Teil dieser Artikelreihe beschrieben, dass Sie einen Verweis auf die Bibliothek **Microsoft Office x.0 Object Library** benötigen, um das Ribbon zu programmieren.

Eigenschaften dynamisch gestalten

Die Ribbon-Elemente haben eine ganze Reihe von Attributen, die Sie gleich in der XML-Definition des Ribbons festlegen – zum Beispiel den Namen, die Beschriftung, die Größe bei Schaltflächen, den Namen eventuell ausgelöster Callback-Prozeduren und viele mehr. Attribute, die Sie per XML definieren, können Sie später nicht mehr ändern. In vielen Fällen ist das auch sinnvoll. Wer möchte schon den nur intern verwendeten Namen eines Ribbon-Elements ändern?

Bisher haben wir allerdings noch nicht darüber gesprochen, dass sich einige Eigenschaften von Ribbon-Elementen auch zur Laufzeit ändern lassen können beziehungsweise dass Sie deren Werte nicht gleich bei der XML-Definition des Ribbons anlegen, sondern dass die Werte von Attributen auch erst beim Erstellen des Ribbons beziehungsweise beim ersten Anzeigen mithilfe von VBA-Funktionen ermittelt werden können.

Die Attribute, die das dynamische Abfragen von Werten zur Laufzeit ermöglichen, beginnen in der Regel mit dem Präfix **get...** gefolgt vom Namen des statischen Attributs. Schauen wir uns dies am Beispiel des **button**-Elements an, das wohl das häufigste Ziel von dynamischen Einstellungen sein dürfte – allein, um dieses zu aktivieren oder deaktivieren oder ein- und auszublenden. Hier gibt es etwa die Attribute **label**, **visible** oder **enabled**. Diese können Sie bereits in der XML-Definition festlegen. Wenn Sie dies allerdings erst beim Anzeigen des jeweiligen Elements erledigen wollen, verwenden Sie die Attribute **getLabel**, **getVisible** oder **getEnabled** und hinterlegen für diese den Namen der Callback-Funktion, welche den Wert für das entsprechende Attribut liefert.

Schauen wir uns dies an einem Beispiel an. Die folgende Ribbon-Definition enthält in einem **tab**-Element ein **group**-Element mit zwei untergeordneten **button**-Elementen. Das erste hat eine statische Bezeichnung, die wir mit dem **label**-Attribut eingestellt haben. Die zweite verwendet das Attribut **getLabel** und gibt dafür den Callback-Namen **getLabel** an:

```
<customUI ...>
  <ribbon>
    <tabs>
      <tab id="tabInteraktion" label="Interaktion">
        <group id="grpInteraktion" label="Beispiele">
          <button label="Statisches Label"
            id="btnLabelStatisch"/>
          <button getLabel="getLabel"
            id="btnLabelDynamisch"/>
        </group>
      </tab>
```

```
    </tabs>
  </ribbon>
</customUI>
```

Die VBA-Prozedur, die beim Auslösen des Callback-Ereignisses **getLabel** aufgerufen werden soll, sieht so aus:

```
Sub getLabel(control As IRibbonControl, ByRef label)
    label = "Dynamisches Label"
End Sub
```

Sie erwartet zwei Parameter, von denen der zweite namens **label** in diesem einfachen Beispiel der wichtigere ist: Er nimmt schlicht die Zeichenkette entgegen, die als Bezeichnung angezeigt werden soll. Dementsprechend füllen wir die Variable **label** mit dem Wert **Dynamisches Label**. Das Ergebnis finden Sie in Bild 1.

Wann erfolgt der Aufruf?

Wie schon weiter oben erwähnt, werden die Callback-Ereignisse, deren Name mit **get...** beginnt, bei der Anzeige der jeweiligen Elemente ausgelöst. Um dies zu belegen, haben wir ein zweites **tab**-Element hinzugefügt, dass ein **button**-Element mit dem Wert **getLabel** für das Attribut **getLabel** aufweist:

```
<tab id="tabMehrInteraktion"
    label="Mehr Interaktion ...">
  <group id="grpBeispiele" label="Beispiele">
    <button getLabel="getLabel" id="btnDynamisch2"/>
  </group>
</tab>
```

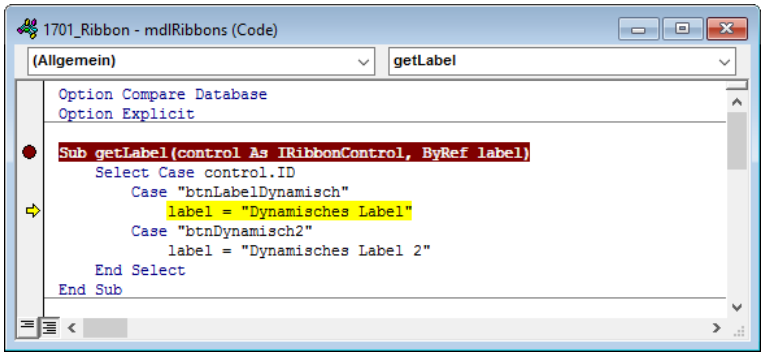


Bild 2: Ansteuern eines Haltepunktes in einer Callback-Prozedur

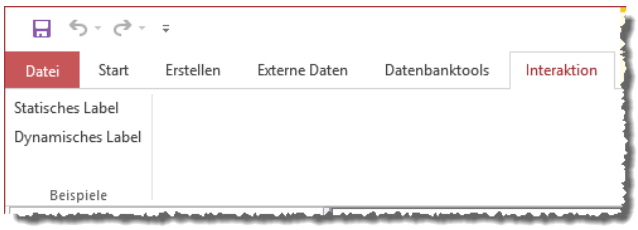


Bild 1: Zwei Label mit verschiedenen Beschriftungen

Außerdem müssen wir nun der Anforderung gerecht werden, dass wir nicht mehr nur eine einzige Schaltfläche haben, von der aus das Callback-Ereignis **getLabel** ausgelöst wird, sondern zwei.

In diesem Fall kommt der erste Parameter der Callback-Prozedur zum Zuge, der mit dem Parameter **control** einen Verweis auf das auslösende Steuerelement liefert. **control** enthält eine Eigenschaft namens **ID**, mit der wir auf den Wert des Attributs **id** der **button**-elemente zugreifen können. Diese prüfen wir in einer **Select Case**-Anweisung und weisen verschiedene Bezeichnungen über den Rückgabeparameter label **zu**:

```
Sub getLabel(control As IRibbonControl, ByRef label)
    Select Case control.ID
        Case "btnLabelDynamisch"
            label = "Dynamisches Label"
        Case "btnDynamisch2"
            label = "Dynamisches Label 2"
    End Select
End Sub
```

Wenn Sie nun die Datenbank komprimieren und reparieren (schnellster Weg, die Datenbank zu schließen und wieder zu öffnen) und einen Haltepunkt in die erste Zeile der Prozedur setzen, landen Sie beim Anzeigen des Ribbon-Tabs **Interaktion** in dieser Prozedur und dort im **Case**-Zweig für den Wert **btnLabelDynamisch** (siehe Bild 2).

Wechseln Sie danach zum zweiten **tab**-Element des Beispiels, wird **getLabel** wieder ausgelöst – diesmal allerdings für das Element **btnDynamisch2**.

Label während der Laufzeit aktualisieren

Damit haben wir allerdings nur gezeigt, wie wir eine Callback-Eigenschaft beim erstmaligen Anzeigen aufrufen, um ein Attribut zur Laufzeit mit einem Wert zu versehen. Wenn Sie anschließend weitere Male zwischen den beiden **tab**-Elementen hin- und herwechseln, wird die Callback-Prozedur **getLabel** nicht mehr aufgerufen.

Wenn Sie dies erreichen wollen, um beispielsweise einen regelmäßig aktualisierten Wert wie die Uhrzeit anzuzeigen, müssen Sie den Wert der per **get...**-Prozedur zugewiesenen Attribute ungültig machen. Welche Voraussetzungen dafür erforderlich sind und wie dies gelingt, zeigt der folgende Abschnitt.

Referenz auf das Ribbon speichern

Der wichtigste Schritt, um ein Attribut ungültig zu machen, ist das Speichern einer Objektreferenz auf die aktuell angezeigte Ribbon-Definition. In diesem Fall wollen wir die Uhrzeit in einem **label**-Element in einem weiteren Tab anzeigen, das wir wie folgt definieren:

```
<customUI ... onLoad="OnLoad_Interaktiv">
  <ribbon>
    <tabs>
      ...
      <tab id="tabDynamisch" label="Dynamische Werte">
        <group id="grpUhrzeit" label="Uhrzeit">
          <labelControl getLabel="getLabel"
            id="lblUhrzeit"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Entscheidend ist hier, dass wir für das **customUI**-Element das Attribut **onLoad** mit dem Wert **onLoad_Interaktiv** füllen (**Interaktiv** ist der Name, unter dem wir das Ribbon in der Tabelle **USysRibbons** gespeichert haben):

```
<customUI ... onLoad="OnLoad_Interaktiv">
```

Wir erweitern nun noch die Prozedur **getLabel**, um dem Label **lblUhrzeit** beim Anzeigen die aktuelle Uhrzeit zuzuweisen (mit der **Time**-Funktion):

```
Sub getLabel(control As IRibbonControl, ByRef label)
    Select Case control.ID
        Case "btnLabelDynamisch"
            label = "Dynamisches Label"
        Case "btnDynamisch2"
            label = "Dynamisches Label 2"
        Case "lblUhrzeit"
            label = Time
    End Select
End Sub
```

Außerdem müssen wir noch die Prozedur hinzufügen, die durch das Attribut **onLoad** ausgelöst wird und die wie folgt aussieht:

```
Sub onLoad_Interaktiv(ribbon As IRibbonUI)
    Set objRibbon_Interaktiv = ribbon
End Sub
```

Die Prozedur liefert mit dem Parameter **ribbon** einen Verweis auf das zuvor definierte **IRibbonUI**-Objekt, was alle mit der XML-Definition benutzerdefiniert angepassten Elemente des Ribbons umfasst. Diesen Verweis speichern wir in einer öffentlich deklarierten Variablen im gleichen Modul, und zwar unter dem Namen **objRibbon_Interaktiv**:

```
Public objRibbon_Interaktiv As IRibbonUI
```

Viele andere Quellen verwenden hier einfach den allgemeinen Bezeichner **objRibbon** und füllen diesen von jeder neu angewendeten Ribbon-Definition aus. Das kann jedoch problematisch werden, wenn Sie tatsächlich mehrere Ribbon-Definitionen gleichzeitig laden – etwa eine als Hauptribbon und eine als Formularribbon. Dann sollten Sie sauber zwischen den beiden Definitionen trennen und für jede eine eigene Objektvariable zum Speichern der Ribbon-Definition vorhalten. Wir verwenden dazu die Schreibweise auf **objRibbon**, einem Unterstrich und dem Namen der Ribbon-Definition, also etwa **objRibbon_Interaktiv**.

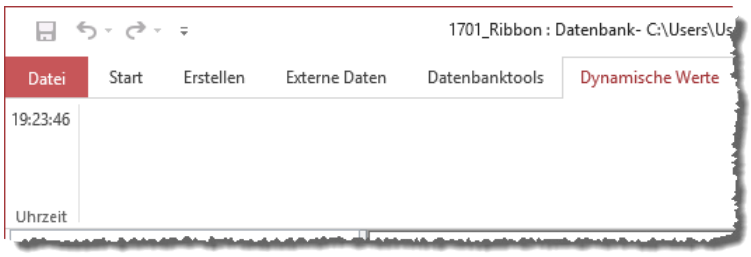


Bild 3: Uhrzeit in einem **label**-Element

Wenn Sie das Ribbon nun anzeigen, liefert dieses die aktuelle Uhrzeit (siehe Bild 3). Allerdings ändert sich diese nicht, wenn Sie zu einem anderen Tab und wieder zurück zum Tab mit der Uhrzeit wechseln. Was soll das nun wieder?

Nun, wir erwähnten ja bereits, dass man das betroffene Element für das erneute Auslösen der Callback-Prozedur erst »ungültig« machen muss. Und hier kommt die Objektvariable **objRibbon_Interaktiv** ins Spiel: Diese bietet nämlich unter anderem drei Methoden an, welche die aktuell mit dem **get...**-Attributen dynamisch ermittelten Werte ungültig macht (siehe Bild 4).

Die Methode **Invalidate** macht dabei alle Elemente des mit **objRibbon_Interaktiv** referenzierten Ribbons ungültig. Das heißt, dass alle **get...**-Attribute nach dem Aufruf dieser Methode neu ausgeführt werden müssen, sobald diese aus- und wieder eingeblendet werden.

Setzen Sie einmal den folgenden Befehl im Direktbereich des VBA-Editors ab:

```
objRibbon_Interaktiv.Invalidate
```

Dadurch, dass Sie **objRibbon_Interaktiv** als öffentliche Variable in einem Standardmodul deklariert haben, können Sie von überall, also auch vom Direktbereich aus, auf seine Methoden zugreifen. Wenn Sie während des Aufrufs das **tab**-Element mit dem **label**-Element zur Anzeige der Uhrzeit anzeigen, wird die Anzeige nun auf die aktuelle Uhrzeit geändert.

Wenn Sie gezielt nur ein einzelnes Element ungültig machen wollen, damit seine **get...**-Attribute erneut eingelesen werden, dann verwenden Sie die Methode **InvalidateControl** und übergeben den Namen des Elements, also den Inhalt des Attributs **id**, mit dem Aufruf als Parameter. Das sieht beispielsweise wie folgt aus:

```
objRibbon_Interaktiv.InvalidateControl "lblUhrzeit"
```

Die dritte Methode namens **InvalidateControlMso** macht das als Parameter benannte Steuerelement ungültig. Hiermit können Sie auf die eingebauten Steuerelemente zugreifen. In der Regel werden deren Inhalte aber automatisch aktualisiert.

Wenn Sie wie im aktuellen Beispiel die Uhrzeit im Ribbon anzeigen möchten, benötigen Sie einen regelmäßigen Aufruf der **Invalidate**- oder **InvalidateControl**-Methode. Diesen können Sie beispielsweise mit dem **Timer**-Ereignis eines unsichtbaren Formulars realisieren. Mehr über das **Timer**-Ereignis lesen Sie im Artikel **Ereignisse in Formularen: Zeitgeber**.

Steuerelemente aktivieren und deaktivieren

Neben dem Ändern von Inhalten etwa des Labels können Sie auch noch weitere Eigenschaften der Ribbon-Elemente ändern.

Ein oft von Änderungen betroffenes Attribut ist zum Beispiel das Attribut enabled. Hier verwenden wir also das Attribut **getEnabled**, um den Wert der Eigenschaft **enabled** zur Laufzeit einzustellen.

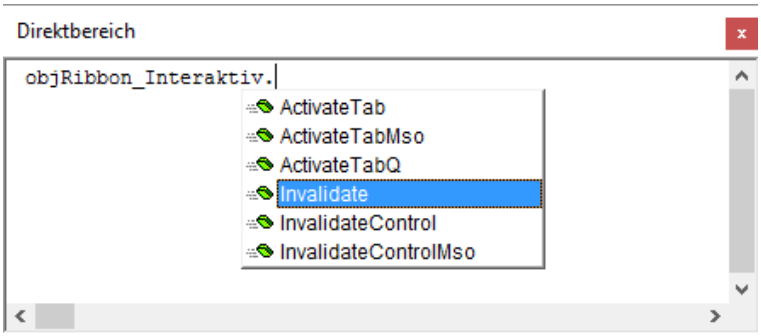


Bild 4: Verschiedene **Invalidate**-Methoden

Platzhalter in Textdateien füllen

Texte mit Platzhaltern zu füllen und die Platzhalter dann durch in der Datenbank gespeicherte Daten zu ersetzen ist ein oft benötigter Anwendungsfall. Im Prinzip bildet Word das mit der Seriendokument-Funktion ab: Hier werden im Dokument Verknüpfungen zur Datenquelle hergestellt und dann gefüllt. Wir wollen es eine Nummer kleiner angehen und einfache Text mit entsprechenden formatierten Platzhaltern wie [Vorname] versehen und diese dann nach Bedarf mit den Daten aus der Datenbank füllen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1701_PlatzhalterInTextenFuellen.accdb**.

Textdatei mit Platzhaltern erstellen

Der erste Schritt besteht daraus, eine geeignete Textdatei zu erstellen, die wir mit Platzhaltern versehen. Hier kommt bereits die erste Hürde: Wie sollen die Platzhalter aussehen, damit wir diese später auch zuverlässig finden können? Die erste Voraussetzung ist, dass die Namen der Platzhalter mit den Datenfeldern korrelieren. Wenn wir aus einer Tabelle Felder namens **Vorname**, **Nachname** und **Strasse** in die Textdateien füllen wollen, sollten die Platzhalter auch entsprechend benannt werden. Das hat allein praktische Gründe: Wir können so später einfach die Felder der Datenherkunft in einer Schleife durchlaufen und die Werte direkt in die Platzhalter einsetzen.

Die zweite Voraussetzung ist, dass die Platzhalter vom übrigen Text unterschieden werden können. Wenn Sie beispielsweise nur den Namen des betroffenen Feldes als Platzhalter verwenden, ist die Wahrscheinlichkeit gegeben, dass dieser Name auch im eigentlichen Text zum Einsatz kommt.

Also müssen wir die Platzhalter noch mit einer Start- und einer Endmarkierung versehen, die in dieser Art sonst nicht im Text vorkommt. Dazu eignen sich beispielsweise eckige Klammern. **[Vorname]** wäre also ein geeigneter Platzhalter. Auf Nummer Sicher gehen Sie, wenn Sie die Start- und die Endmarkierung noch ein wenig eindeutiger gestalten, beispielsweise mit **@[Vorname@]**. Auf diese Weise sollte nichts anbrennen.

Die Vorlage sieht dann beispielsweise wie in Bild 1 aus. Sie enthält einige Platzhalter, die mit @[beginnen und mit]@ enden. Nun benötigen wir eine Tabelle beziehungsweise Abfrage, welche die angegebenen Platzhalterbezeichnungen innerhalb der Begrenzungszeichen aufweist. Dazu legen wir eine Tabelle namens **tblKunden** an, deren Entwurf Sie in Bild 2 finden. Das Feld **AnredeID** ist ein Fremdschlüsselfeld, das noch mit der Tabelle **tblAnreden** verknüpft ist.

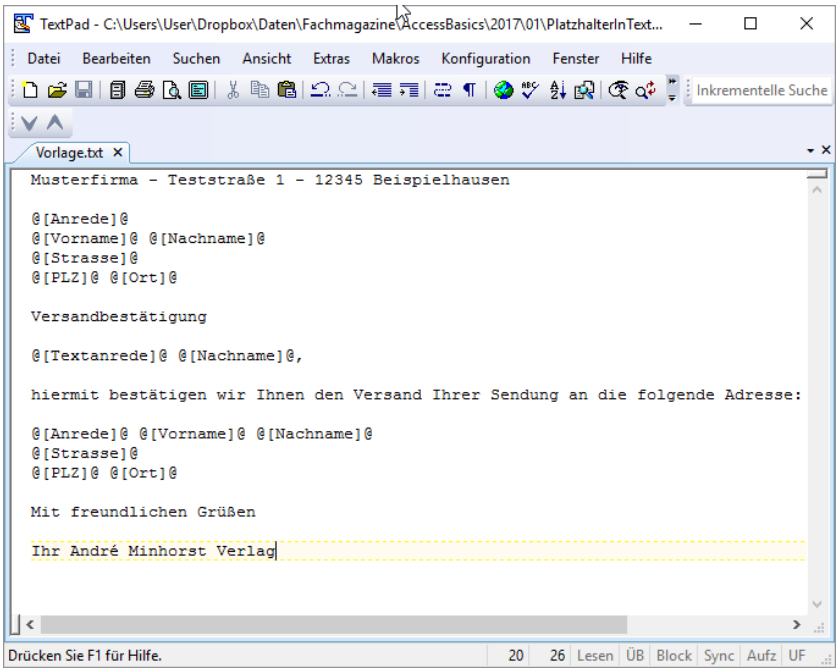


Bild 1: Beispielvorlage als Textdatei

tblAnreden enthält die Felder **ID** (Primärschlüsselfeld) und **Bezeichnung** sowie zwei Felder mit speziellen Formen der Anrede: **AnredeAdresse** nimmt die Form der Anrede für den Adressblock auf (**Herrn/Frau**), **AnredeBrief** die Anrede für die erste Zeile des Brieftextes (**Sehr geehrter Herr/Sehr geehrte Frau**) – siehe Bild 3.

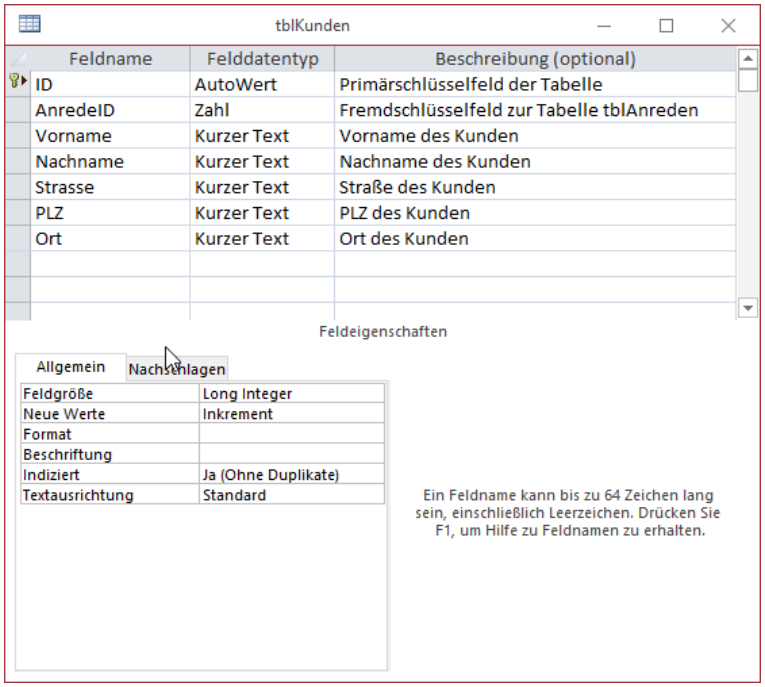


Bild 2: Entwurf der Tabelle **tblKunden**

Abfrage als Quelle für die Platzhalter

Nun benötigen wir noch eine Abfrage, welche die Daten der beiden Tabellen so zusammenführt, dass wir diese in der anschließend zu erstellenden VBA-Prozedur zum Ersetzen der Platzhalter nutzen können. Diese Abfrage soll die beiden Tabellen **tblKunden** und **tblAnreden** als Datenherkunft nutzen und alle Felder dieser beiden Tabellen mit Ausnahme des Feldes **AnredeID** der Tabelle **tblKunden** und des Feldes **ID** der Tabelle **tblAnreden** enthalten. Der Entwurf dieser Abfrage, die wir unter dem Namen **qryKundenMitAnrede** speichern, sieht wie in Bild 4 aus.

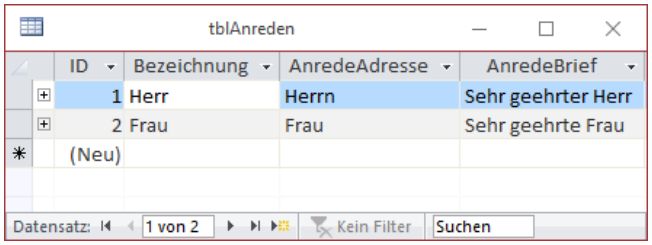


Bild 3: Die Tabelle **tblAnreden** mit einigen Werten

Das Ergebnis der Abfrage aus Bild 5 liefert alle Daten der Tabelle **tblKunden** plus die dazugehörigen Anreden aus der Tabelle **tblAnreden**.

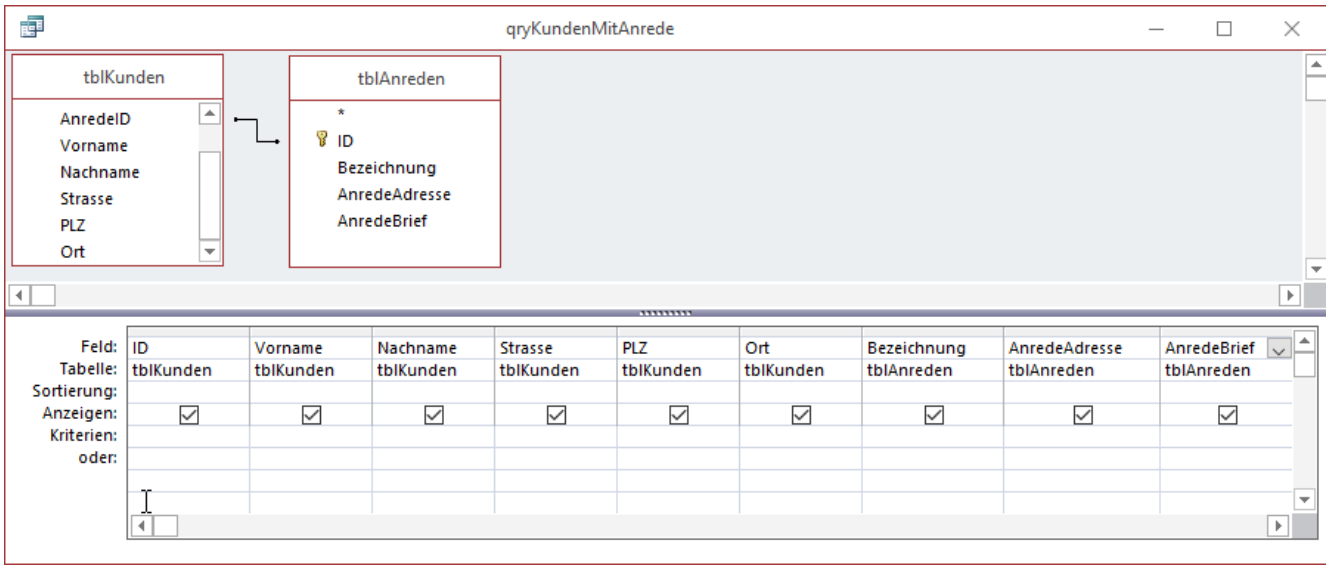


Bild 4: Entwurf der Abfrage **qryKundenMitAnrede**

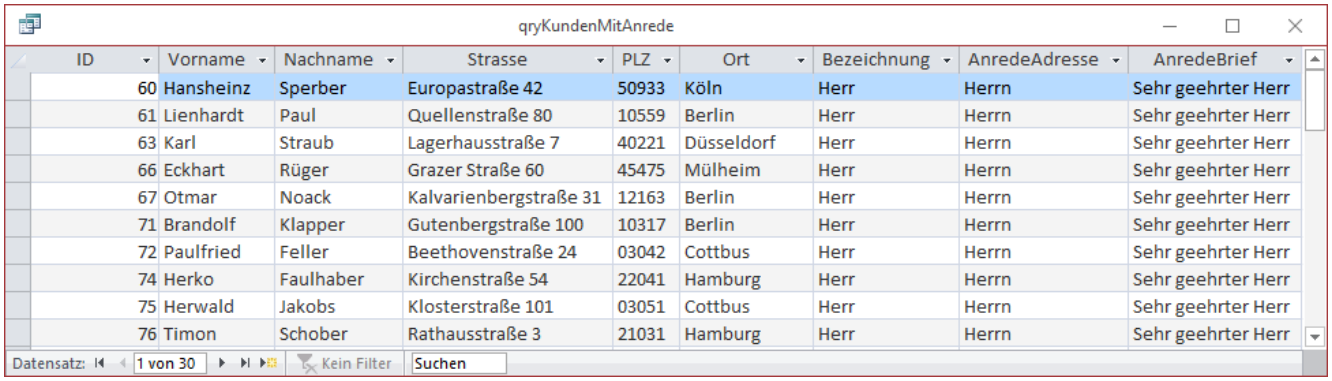


Bild 5: Die Abfrage **qryKundenMitAnrede** in der Datenblattansicht

Formular zum Füllen der Platzhalter in der Textdatei

Die Funktionen zum Auswählen der Vorlage mit den Platzhaltern, zum Auswählen der Zieldatei und zum Festlegen des Kunden, für den die Vorlage gefüllt werden soll, wollen wir in eine Benutzeroberfläche integrieren – sprich: in ein Formular. Dieses sieht in der Entwurfsansicht wie in Bild 6 aus. Es enthält im oberen Bereich zwei Textfelder namens **txtVorlage** und **txtZiel**. In beiden soll der Benutzer die Namen der Dateien eintragen, die verwendet werden sollen. Beide Dateien sollen der Einfachheit halber im Verzeichnis der aktuellen Datenbankdatei liegen.

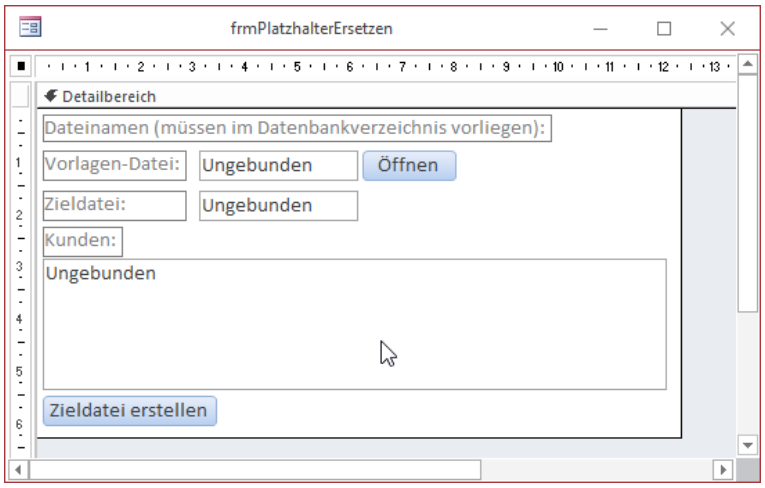


Bild 6: Das Formular **frmPlatzhalterErsetzen** in der Entwurfsansicht

Vorlagendatei öffnen

Die Schaltfläche rechts vom Textfeld für die Vorlagendatei soll die angegebene Datei zum Betrachten und Bearbeiten öffnen. Dazu löst diese die Prozedur **cmdVorlageOeffnen** aus (siehe Listing 1). Diese fügt den mit **CurrentProject.Path** ermittelten Datenbankpfad und die im Textfeld **txtVorlage** angegebene Datei in einer Variablen namens **strVorlage** zusammen und prüft mithilfe der **Dir**-Funktion, ob die Datei überhaupt vorhanden ist. Diese Funktion würde in diesem Fall den reinen Dateinamen zurückliefern. Hat **Dir(strVorlage)** also eine Länge größer **0**, dann ist die Datei vorhanden.

```
Private Sub cmdVorlageOeffnen_Click()
    Dim strVorlage As String
    strVorlage = CurrentProject.Path & "\" & Me!txtVorlage
    If Len(Dir(strVorlage)) > 0 Then
        DateiOeffnen strVorlage
    Else
        MsgBox "Die Datei '" & Me!txtVorlage & "' ist nicht im aktuellen Ordner der Datenbank vorhanden."
    End If
End Sub
```

Listing 1: Anzeige der Vorlagendatei im Text-Editor

```
Public Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" (ByVal hWnd As Long, ByVal lpOperation As String,
ByVal lpFile As String, ByVal lpParameters As String, ByVal lpDirectory As String, ByVal nshowcmd As Long) As Long
Public Const SW_NORMAL = 1

Public Function DateiOeffnen(strPfad As String) As Boolean
    Call ShellExecute(0, "open", strPfad, "", "", SW_NORMAL)
End Function
```

Listing 2: Diese Prozedur öffnet ein neues Formular oder zeigt dieses an, wenn es bereits geöffnet ist.

In diesem Fall ruft die Prozedur eine weitere Routine namens **DateiOeffnen** auf, die im Modul **mdlTools** angelegt ist. Diese verwendet eine API-Funktion namens **ShellExecute**, um die mit dem Parameter **strPfad** übergebene Datei zu öffnen (siehe Listing 2).

Auswahl des Kunden, mit dem die Platzhalter gefüllt werden sollen

Das Listenfeld **lstKunden** dient der Anzeige aller Kunden der Abfrage **qryKundenMitAnrede** und der Auswahl des Kunden, für den die Platzhalter in der Vorlage ersetzt werden sollen. Dazu stellen Sie die Eigenschaft **Datensatzherkunft** auf den Namen der Abfrage **qryKundenMitAnrede** ein. Damit das Primärschlüsselfeld der Abfrage nicht angezeigt wird, sondern nur die ersten beiden Felder **Vorname** und **Nachname**, stellen Sie die Eigenschaft **Spaltenanzahl** auf den Wert **3** und die Eigenschaft **Spaltenbreiten** auf den Wert **0cm;4cm;4cm** ein.

Das gefüllte Listenfeld sieht wie in Bild 7 aus.

Voreinstellungen

Damit das Formular beim Erstellen und Ausprobieren immer gleich ein paar sinnvolle Dateinamen anzeigt und einen Eintrag der Kundenliste auswählt, hinterlegen Sie für die Ereigniseigenschaft **Beim Laden**, dem Sie den Wert **[Ereignisprozedur]** zuweisen, die folgende Prozedur:

```
Private Sub Form_Load()
    Me!txtVorlage = "Vorlage.txt"
    Me!txtZiel = "Ziel.txt"
    Me!lstKunden = Me!lstKunden.ItemData(0)
End Sub
```

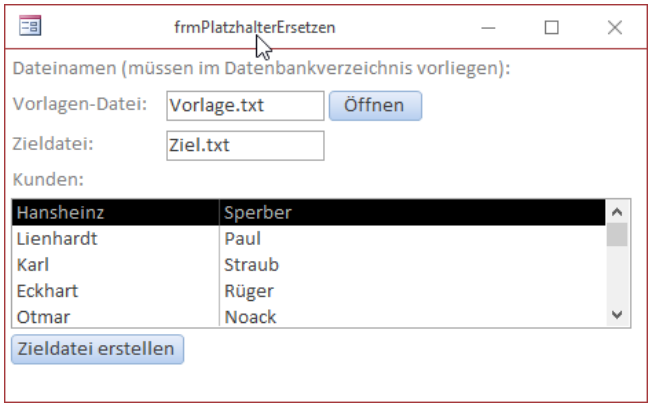


Bild 7: Das Formular **frmPlatzhalterErsetzen** in Aktion

Danach können Sie gleich loslegen – mit einem Klick auf die Schaltfläche **cmdErstellen**.

Textdatei-Platzhalter füllen

Die Prozedur, die durch den Mausklick auf die Schaltfläche **cmdErstellen** ausgelöst wird, finden Sie in Listing 3. Die Prozedur ermittelt zunächst den Primärschlüsselwert für den aktuell in der Liste markierten Kunden und speichert diesen in der Variablen **lngKundeID**.

Dann initialisiert sie die Variable **db** mit einem Verweis auf das aktuelle **Database**-Objekt und füllt die Variable **rst** mit einem Recordset auf Basis der Abfrage **qryKundenMitAbfrage**, wobei diese nach dem Wert der Variablen **lngKundeID** für das Feld **KundeID** gefiltert wird. Sofern das Recordset danach einen Datensatz enthält, folgen die Schritte innerhalb der **If...Then**-Bedingung der Prozedur.

Hier ermittelt die Prozedur den Pfad zur Vorlage und speichert diesen in der Variablen **strVorlage**. Dann

```
Private Sub cmdErstellen_Click()
    Dim lngKundeID As Long
    Dim strText As String
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim strZieldatei As String
    Dim strVorlage As String
    lngKundeID = Me!lstKunden
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM qryKundenMitAnrede WHERE ID = " & lngKundeID, dbOpenDynaset)
    If Not rst.EOF Then
        strVorlage = CurrentProject.Path & "\" & Me!txtVorlage
        strText = TextdateiEinlesen(strVorlage)
        If PlatzhalterPruefen(strText, rst) = True Then
            strText = PlatzhalterErsetzen(strText, rst)
            strZieldatei = CurrentProject.Path & "\" & Me!txtZiel
            TextdateiSchreiben strZieldatei, strText
            DateiOeffnen strZieldatei
        Else
            DateiOeffnen strVorlage
        End If
    End If
    rst.Close
    Set rst = Nothing
    Set db = Nothing
End Sub
```

Listing 3: Hauptprozedur zum Ersetzen der Platzhalter in einer Textdatei

liest sie mithilfe der Funktion **TextdateiEinlesen** den Inhalt der Vorlagen-Textdatei in die Variable **strText** ein. Bevor weitere Schritte folgen, prüfen wir noch, ob es für alle Platzhalter entsprechende Felder in der Datenherkunft gibt.

Es kann ja auch geschehen, dass der Benutzer durch einen Tippfehler einen Platzhalter eingibt, der so nicht in der Datenquelle abgebildet wird. Für diesen Fall rufen wir die Funktion **PlatzhalterPruefen** auf und übergeben dieser den zu durchsuchenden Text sowie das Recordset mit den Feldern.

Verläuft die Prüfung der Platzhalter erfolgreich, folgen weitere Schritte. Zunächst ruft die Prozedur die Funktion **PlatzhalterErsetzen** auf, welche sich um das Ersetzen der Platzhalter durch die tatsächlichen Werte aus dem Recordset **rst** kümmert. Deshalb übergibt die Prozedur dieser Funktion auch sowohl den Text der Vorlagendatei (**strText**) als auch einen Verweis auf das Recordset (**rst**).

Danach ermittelt sie die Zieldatei und schreibt diese in die Variable **strZieldatei**. Diese wird gleich als Parameter an die Prozedur **TextdateiSchreiben** weitergereicht – neben dem in der Variablen **strText** gespeicherten Text mit ersetzten Platzhaltern.

Schließlich öffnet die Prozedur noch die frisch erstellte Textdatei und zeigt diese mithilfe der Hilfsfunktion **DateiOeffnen** in einem Text-Editor an. Die hier vorgestellten Routinen schauen wir uns gleich im

Der Webbrowser als Bildanzeige

Die Handhabung von Bildern in Formularen war und ist ein Dauerbrennerthema in vielen Foren. Das genuine Bildsteuerelement von Access weist auch in der zeitweilig modernisierten Version etliche Beschränkungen auf. Das Anlage-Steuerelement ist zwar pfiffig, aber dessen Programmierung fällt reichlich umständlich aus. Doch zu unserem Favoriten haben wir schon früher einmal ein alternatives Steuerelement auserkoren: das Webbrowser Control.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1702_BildWeb.zip**.

Das Webbrowser Control als Universalcontainer

In der Ausgabe **04/2016** von **Access Basics**, **Elemente für die Bildanzeige**, verglichen wir die möglichen Methoden, um Bilder in Formularen und Berichten zur Anzeige zu bringen. Neben dem **Bildsteuerelement**, dem **Anlage**-Steuerelement, einer zweckentfremdeten **Schaltfläche** und dem **MSForms Image Control** kam dort auch ein **Webbrowser-Steuerelement** zum Einsatz. Es stellte sich heraus, dass Letzteres sich sowohl qualitativ, wie auch von der Programmierung her, sehr gut für diesen Zweck eignet.

Tatsächlich ist das **Webbrowser Control** als Host für den **Internet Explorer** eine Universalwaffe! Überall da, wo der Fundus von Access-Steuerelementen versagt, lässt sich über den Webbrowser fast alles realisieren. Schließlich gibt es ja inzwischen sehr komplexe Applikationen, die rein webbasiert arbeiten; und dorthin geht ohnehin der Trend. Liebäugeln Sie also eh schon mit der Portierung Ihrer Datenbank zu einer Weblösung, so ist der integrierte Webbrowser unter Access ein nützlicher Anfang, um sich mit Webprogrammierung vertraut zu machen.

Der Vorteil des **Webbrowser Controls** ist, dass es den gewaltigen Umfang der **HTML-Bibliothek** von Windows offenlegt. Über diese können Webseiten programmatisch nicht nur über HTML gesteuert werden, sondern auch über Objekte und deren Eigenschaften, sowie Methoden. Praktisch jedes **HTML-Tag** (Element) hat ein Pendant als Objektklasse in der HTML-Bibliothek von Microsoft. Über weitere Hilfsbibliotheken, auf die Sie verweisen können, wie die **Microsoft XML Library** oder die **Shell**-Bibliothek, haben Sie schließlich den **Internet Explorer** vollständig im Griff.

Webbrowser vs. Anlagesteuerelement

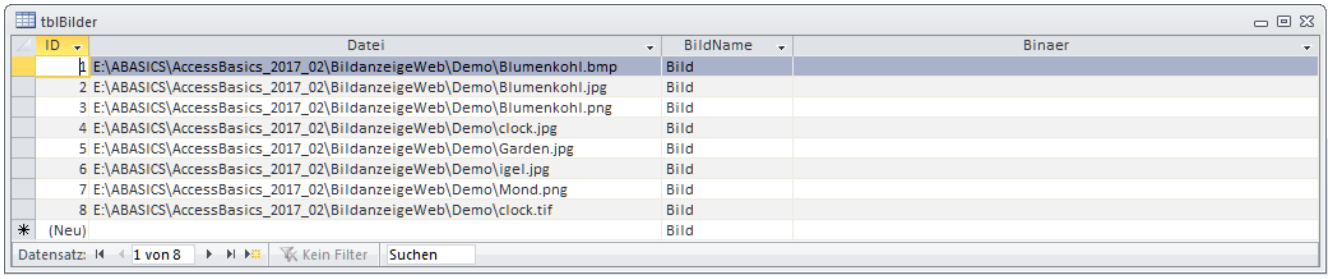
Das Anlagesteuerelement scheint alle Lösungen zur Bildanzeige dadurch auszustechen, dass es die Bilddaten binär aus einer Tabelle mit Anlagefeld hervorholen kann. Mit seinem Popup-Toolbar ist es zudem ohne jeglichen Programmcode möglich, Bilddateien zu laden und direkt im gebundenen Feld zu speichern.

Ein Bild im Webbrowser hingegen verlangt nach einer **URL**, die den Pfad zur Datei enthalten muss. Ein Abspeichern der Bilddatei selbst in einem Tabellenfeld scheint damit verwehrt. Bestenfalls könnte man die Datei binär in ein **OLE-Feld** laden, um dieses später wieder als temporäre Datei zu exportieren und deren Pfad als **URL** dem Webbrowser Control zu übergeben.

Tatsächlich jedoch gibt es eine Lösung, die diesen Umweg obsolet macht. Das Formular **frmBilder** der Beispieldatenbank zeigt, wie auf diese temporäre Datei über eine spezielle Kodierung der Bilddaten verzichtet werden kann. Damit steht diese Lösung dem Anlagesteuerelement in nichts nach. Sie erfordert lediglich einige überschaubare VBA-Routinen.

Bildtabelle

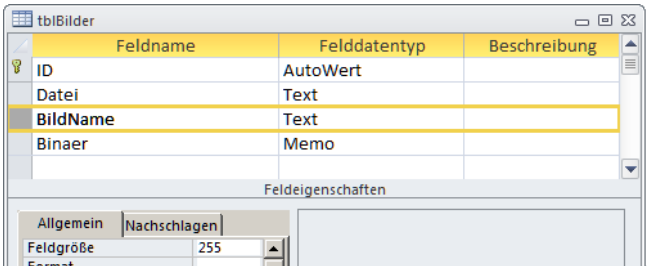
Die Tabelle zur Speicherung sowohl des Bildpfads, wie auch der Binärdaten, nennt sich **tblBilder** und zeigt sich in der Datenblattansicht wie in Bild 1. Das



ID	Datei	BildName	Binaer
1	E:\ABASICS\AccessBasics_2017_02\BildanzeigeWeb\Demo\Blumenkohl.bmp	Bild	
2	E:\ABASICS\AccessBasics_2017_02\BildanzeigeWeb\Demo\Blumenkohl.jpg	Bild	
3	E:\ABASICS\AccessBasics_2017_02\BildanzeigeWeb\Demo\Blumenkohl.png	Bild	
4	E:\ABASICS\AccessBasics_2017_02\BildanzeigeWeb\Demo\clock.jpg	Bild	
5	E:\ABASICS\AccessBasics_2017_02\BildanzeigeWeb\Demo\Garden.jpg	Bild	
6	E:\ABASICS\AccessBasics_2017_02\BildanzeigeWeb\Demo\igel.jpg	Bild	
7	E:\ABASICS\AccessBasics_2017_02\BildanzeigeWeb\Demo\Mond.png	Bild	
8	E:\ABASICS\AccessBasics_2017_02\BildanzeigeWeb\Demo\clock.tif	Bild	
(Neu)		Bild	

Bild 1: Die Tabelle **tblBilder** enthält zunächst nur die Pfade zu den Bilddateien im Feld **Datei** und einen **Default**-Namen

Feld **Datei** nimmt den vollen Pfad der Bilddatei auf. **BildName** dient zur optionalen Angabe einer Bezeichnung des Bilds und ist im Entwurf der Tabelle auf den Text **Bild** voreingestellt. Schließlich sollen die Binärdaten des Bilds im Feld **Binaer** aufgenommen werden, das hier aber nicht, wie eher üblich, als **OLE-Feld** daherkommt, sondern als **Memo**-Feld.



Feldname	Felddatentyp	Beschreibung
ID	AutoWert	
Datei	Text	
BildName	Text	
Binaer	Memo	

Feldeigenschaften: Allgemein | Nachschlagen — Feldgröße: 255

Bild 2: Entwurfsansicht der Tabelle **tblBilder**

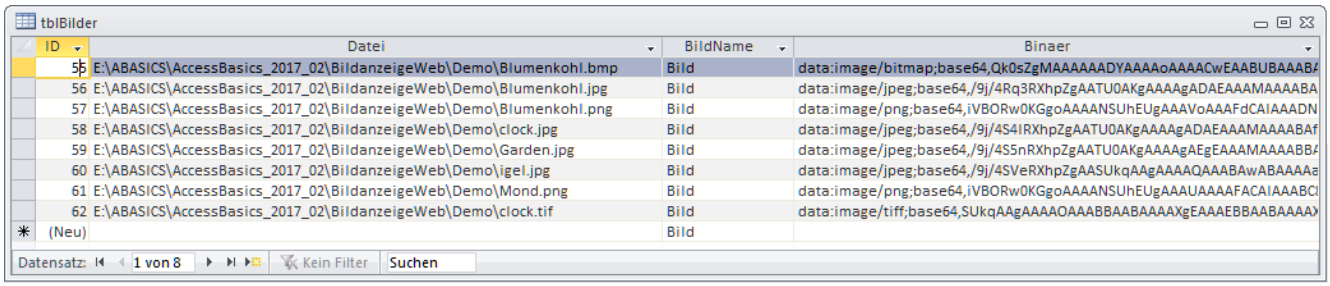
Den Grund erfahren Sie später. Der Entwurf der Tabelle ergibt sich damit gemäß Bild 2. Der Autowert **ID** als Primärschlüssel vervollständigt die Tabelle.

Die angezeigten Datensätze kommen übrigens beim Aufruf des **Intro**-Formulars zustande. Hier ist eine kleine Routine **UpdatePics** hinterlegt, die alle Bilddateien, die sich im Datenbankpfad befinden, enumeriert und in der Tabelle ablegt. Bei Ihnen wird der Inhalt der Tabelle deshalb wohl anders aussehen.

Die Tabelle stellt die Basis für das Formular **frmBilder** dar, wobei dort das Webbrowser-Steuerelement an das Feld **Datei** gebunden ist. Damit zeigt es die Bilder auch schon ordnungsgemäß an, falls die Pfadangaben stimmen. Gleichzeitig kodiert eine Prozedur die Bilddaten und speichert sie im Feld **Binaer** (siehe Bild 3). Bei erneutem Aufruf des Datensatzes im Formular schaut der Code im Ereignis **Beim Anzeigen** (**Form_Current**) nach, ob das Feld **Binaer** leer ist. Da das nun nicht mehr der Fall ist, verwendet es die Daten des Memo-Felds zur Bildanzeige, ohne eine Datei lesen zu müssen. Soweit der Überblick! Schauen wir uns das im folgend im Detail an...

Bild im Webbrowser im Formular

Der Entwurf des Formulars **frmBilder** (siehe Bild 4) ist so schlicht, wie die zugrundeliegende Tabelle. Die Textfelder oben im Formularkopf sind an die entsprechenden Tabellenfelder **BildName** für die Bezeichnung und **Datei** für den Pfad zur Bilddatei gebunden. Über den blauen Button **cmdLoad** kann ein Dateiauswahldialog geöffnet werden, um ein Bild aus dem Dateisystem laden zu können. Das rote Label mit der Beschriftung **Binär** ist auf Unsichtbar gestellt und



ID	Datei	BildName	Binaer
55	E:\ABASICS\AccessBasics_2017_02\BildanzeigeWeb\Demo\Blumenkohl.bmp	Bild	
56	E:\ABASICS\AccessBasics_2017_02\BildanzeigeWeb\Demo\Blumenkohl.jpg	Bild	
57	E:\ABASICS\AccessBasics_2017_02\BildanzeigeWeb\Demo\Blumenkohl.png	Bild	
58	E:\ABASICS\AccessBasics_2017_02\BildanzeigeWeb\Demo\clock.jpg	Bild	
59	E:\ABASICS\AccessBasics_2017_02\BildanzeigeWeb\Demo\Garden.jpg	Bild	
60	E:\ABASICS\AccessBasics_2017_02\BildanzeigeWeb\Demo\igel.jpg	Bild	
61	E:\ABASICS\AccessBasics_2017_02\BildanzeigeWeb\Demo\Mond.png	Bild	
62	E:\ABASICS\AccessBasics_2017_02\BildanzeigeWeb\Demo\clock.tif	Bild	
(Neu)		Bild	

Bild 3: Das Memo-Feld **Binaer** füllt sich später beim Aufruf eines Bilds im Webbrowser des Formulars **frmBilder**

wird erst per Code zum Vorschein gebracht.

Im Detailbereich des Formulars befindet sich in voller Ausdehnung nur das Webbrowser-Steuerelement **ctlWeb**. Seine Verankerung ist auf **Quer und nach unten dehnen** eingestellt, damit es alle Größenänderungen des veränderbaren Formularrahmens mitmacht. Links oben steht in ihm auch schon der Steuerelementinhalt, der jedoch nicht den Namen des Tabellenfelds anzeigt, sondern den Ausdruck **="about:blank"**.

Tatsächlich ist das Control nicht wirklich an die Tabelle gebunden, sondern nur indirekt über den VBA-Code des Formulars. Denn das Webbrowser-Steuerelement von Access **muss** nicht zwingend an ein Feld gebunden sein. Sie können auch eine feste **URL** im Eigenschaftenblatt unter **Steuerelementinhalt** eintragen, die dann standardmäßig angezeigt wird. Mit dem vorliegenden Eintrag erhalten Sie immer eine leere Seite.

Im Fußbereich des Formulars gibt es einige zusätzliche Elemente, die Einfluss auf die Anzeige des Bilds haben. Ist das Kontrollkästchen **Originalgröße** aktiviert, so wird das Bild in seiner vollen Ausdehnung gerendert. Andernfalls passt es sich, je nach Breite oder Höhe, genau in die Fläche des Webbrowser Controls ein.

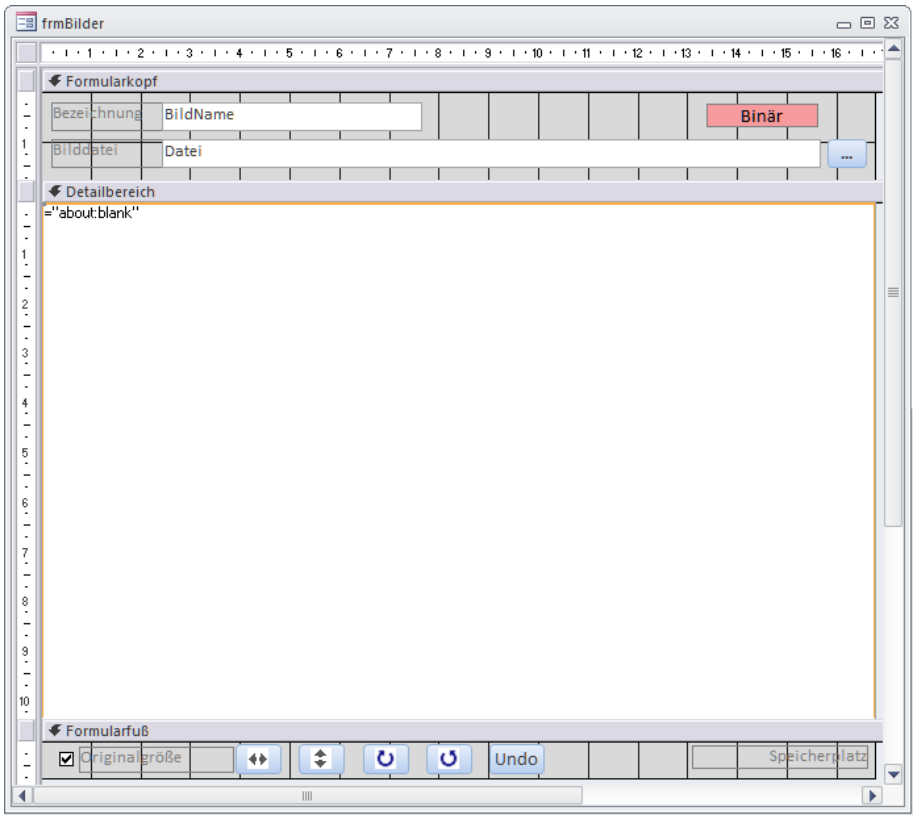


Bild 4: Das **Webbrowser-Control** in der Mitte des Formularentwurf von **frmBilder**

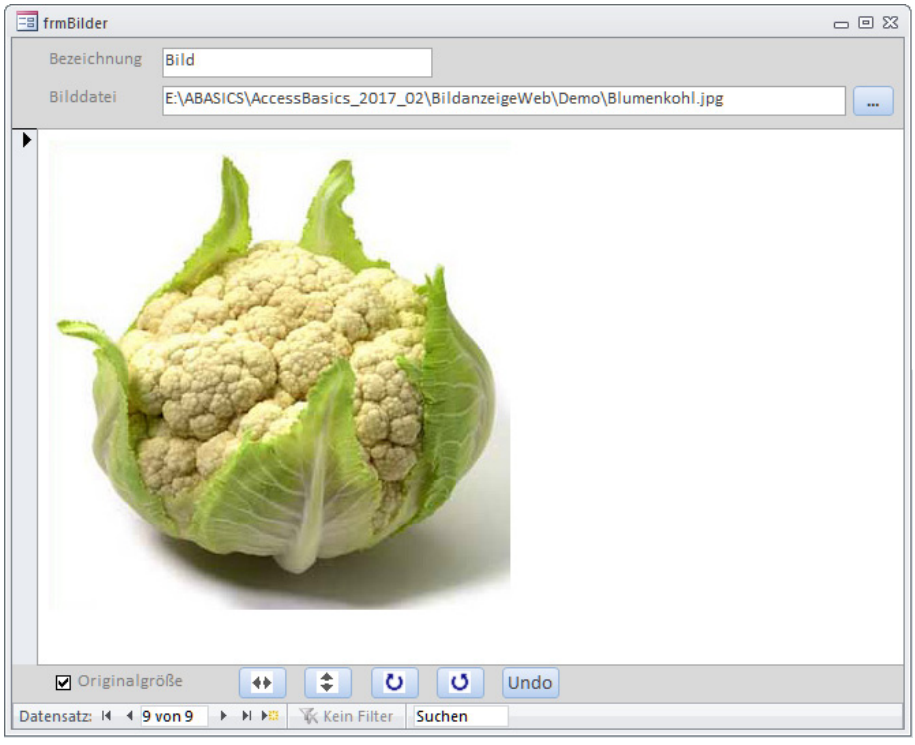


Bild 5: So präsentiert sich das Formular **frmBilder** dann zur Laufzeit

Die fünf Schaltflächen daneben demonstrieren, dass sich über das **HTML-Objektmodell** auch auf sehr einfache Weise Bildeffekte erzielen lassen. Hier können Sie das Bild horizontal oder vertikal spiegeln, sowie im oder gegen den Uhrzeigersinn um 90° drehen.

Der **Undo**-Button macht solche Änderungen rückgängig. In Bild 5 sehen Sie ein geladenes Beispielbild. Hier hat das Control das Bild aus dem Dateipfad eingelesen. Im Hintergrund aber wurden dessen Daten, wie erwähnt, in die Tabelle geschrieben.

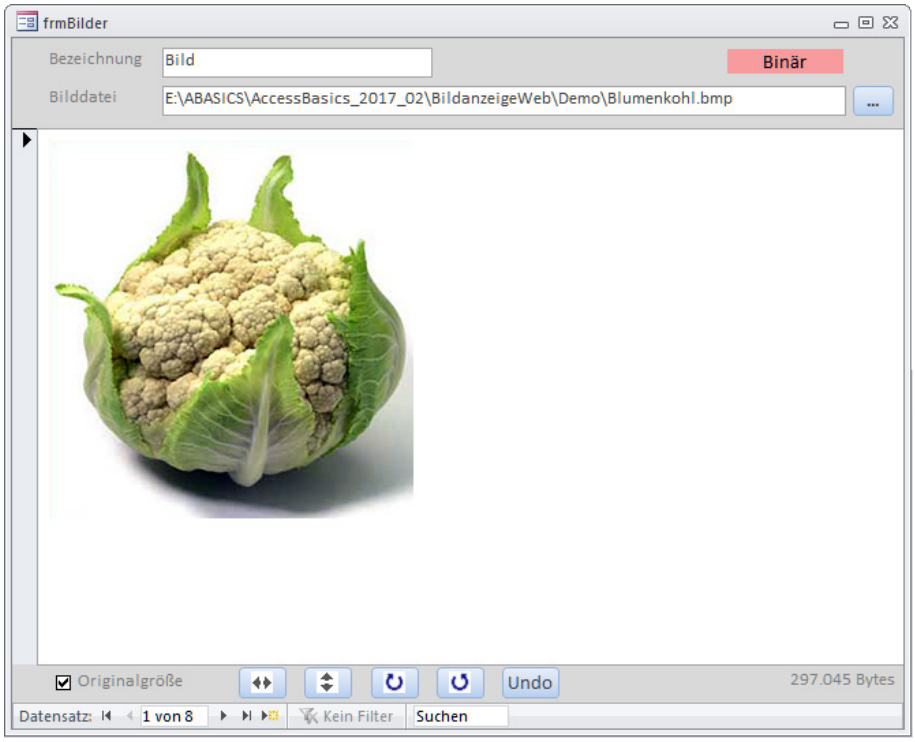


Bild 6: Ist ein Bild bereits eingelesen, so zeigt sich das Label **Binär** oben rechts

So kommt es bei erneutem Aufruf des Datensatzes zur Darstellung in Bild 6: Das **Binär**-Label oben wird eingeblendet und zusätzlich gibt rechts unten ein weiteres Label Auskunft über den Speicherbedarf des Bilds im Memo-Feld der Tabelle. Da es sich im Beispiel um ein Bitmap-Format handelt, im Textfeld **Bilddatei** zu erkennen, ist dieser ziemlich hoch.

Bei Linksklick auf das Bild erscheint eine Meldung, die direkt die aktuell im Webbrowser Control hinterlegte **URL** ausgibt. Bei aus dem Dateisystem geladenem Bild ergibt sich dabei etwa die Meldung in Bild 7. Hier hat der **Internet Explorer** den Pfad in das **file**-Protokoll umgewandelt und die **Backslashes** automatisch in **Slashes** umgewandelt.

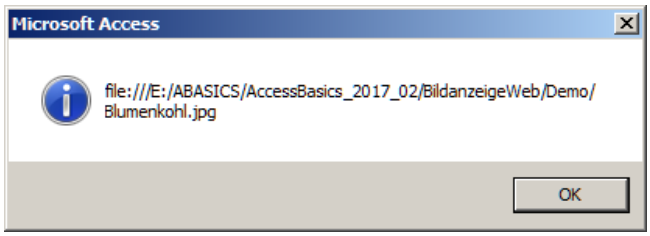


Bild 7: Info-Meldung beim Klick auf das Bild

Kommt das Bild hingegen binär aus der Tabelle, so zeigt die Meldung, wie in Bild 8, deren Memo-Text ausschnittsweise an. Dafür kommt keineswegs anderer Code zum Einsatz. Tatsächlich handelt es sich bei

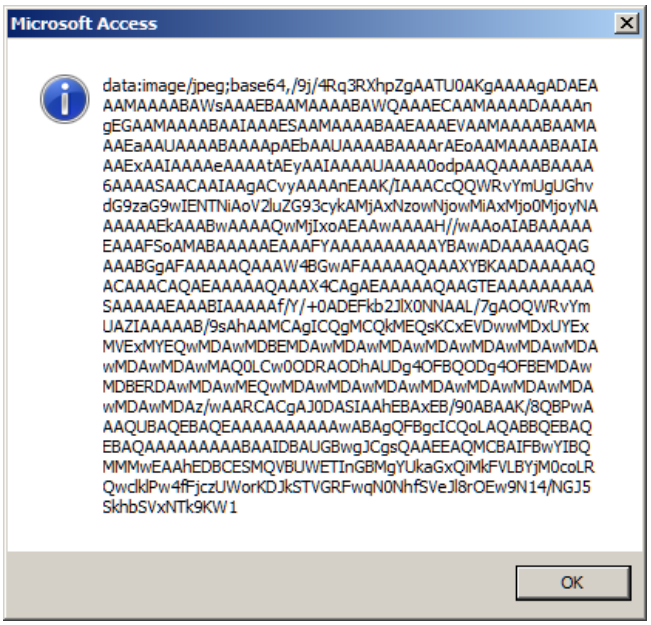


Bild 8: Info-Meldung bei binärem Inhalt des Bilds

dem langen String unmittelbar um die **Source-URL** des Browsers!

Bei Rechtsklick auf das Bild erscheint ein Kontextmenü, wie sonst im **Internet Explorer** auch (siehe Bild 9). Leider hat man auf dessen Einträge ohne größere Umstände keinen Einfluss. Man kann es allerdings mit wenig Code auch deaktivieren, wie wir noch sehen werden. Wir haben dies zunächst nicht getan, weil damit deutlich wird, welche Möglichkeiten das Webbrowser Control zur Verfügung stellt. So etwa lässt sich das Bild, egal, ob als Datei oder binär geladen, als Datei abspeichern, über den E-Mail-Client versenden oder ausdrucken.

Beim Speichern hat man die Wahl zwischen mehreren Dateiformaten, wie Bild 10 zeigt. Dabei scheint es keine Rolle zu spielen, welches Ursprungsformat das Bild hat. Auch bei **JPG** werden beim Abspeichern nur **PNG** und **BMP** angeboten.

Auch ein Kopieren des Bilds in die Zwischenablage ist möglich, so dass es danach etwa in **Word** wieder eingefügt werden kann. Allerdings kopiert das Control gleich mehrere Formate in die Zwischenablage, wie etwa neben dem Bitmap auch den Quelltext des Browsers. Beim Einfügen in Word etwa müssen Sie die Funktion **Einfügen | Inhalte einfügen...** im **Start**-Tab des Ribbon bemühen, was den Dialog in Bild 11 auf den Plan ruft. Hier ist **HTML** voreingestellt. Sie müssen stattdessen den Eintrag **Geräteunabhängige Bitmap** auswählen, um das Bild in ein Dokument zu bekommen.

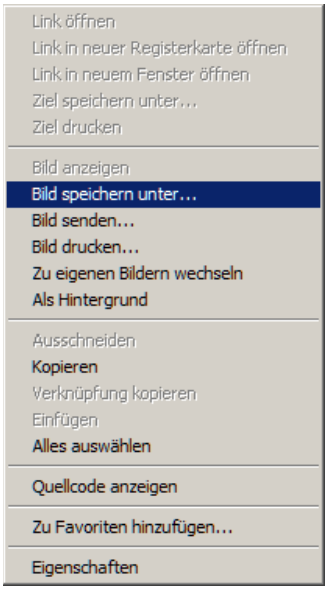


Bild 9: Das Kontextmenü des Webbrowser-Bilds

Am Störendsten sind im Kontextmenü sicherlich die Einträge weiter unten, die einen Anwender, der gar nicht weiß, dass er einen Browser für die Bildanzeige vor sich hat, verwirren. Deshalb wäre ein Abschalten des Kontextmenüs ratsam. Denn alle darin enthaltenen Anweisungen lassen sich gegebenenfalls auch recht einfach per Code über das Objektmodell des Browsers auslösen. Mit einigen Klimmzügen ließe sich das Kontextmenü sogar durch ein eigenes in der Datenbank angelegtes ersetzen. Da Microsoft leider seit mehreren Versionen keinen Editor für solche Menüs mehr in Access einbaut und deren Anlage deshalb rein programmtechnisch erfolgen muss, lassen wir diese Möglichkeit hier außen vor.

Programmierung der Bildanzeige

Gehen wir Schritt für Schritt vor. Schalten Sie im Formular auf einen neuen Datensatz. Die Bildfläche bleibt weiß. Über den Button für den Dateiauswahldialog (**cmdLoad**) öffnen Sie eine Bilddatei. Der dem **Click**-Ereignis hinterlegte Code steht in Listing 1 und

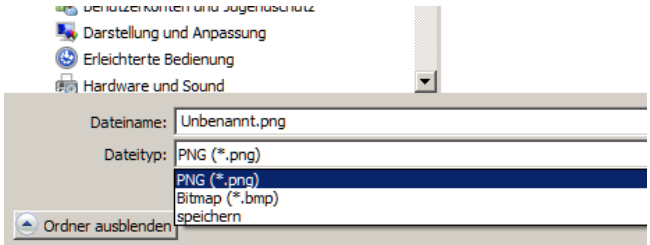


Bild 10: Beim Export des Bilds in eine Datei stellt der Webbrowser diese zwei Dateiformate zur Verfügung

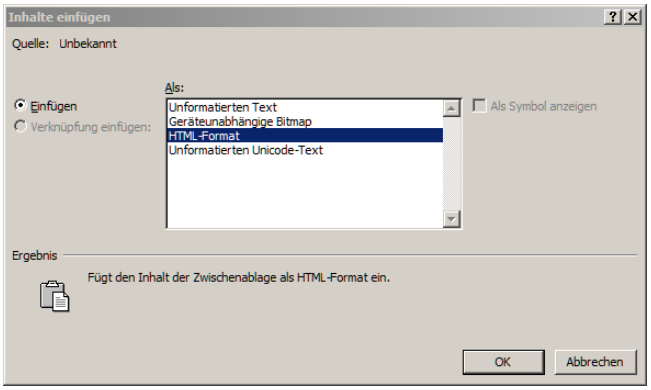


Bild 11: Einfügen des kopierten Bilds in MS Word

verwendet das **FileDialog**-Objekt von Access oder, genauer, der Office-Bibliothek.

Hier wird zunächst die Objektvariable **ODlg** auf das von Access zur Verfügung gestellte **FileDialog**-Objekt gesetzt, wobei gleich der Dateiauswahlmodus über die Konstante **msoFileDialogPicker** angefordert wird. Im folgenden **With**-Block werden nun einige Einstellungen für den Dialog vorgenommen. Mehrfachauswahl etwa ist untersagt (**AllowMultiSelect**), die Ansicht wird auf Details gesetzt (**InitialView**), der Baum navigiert zum Datenbankverzeichnis (**InitialFileName**) und die erlaubten Dateiendungen sind auf einige Bildformate beschränkt, mit denen der Internet Explorer klarkommt (**Filters**). Die Methode **Show** zeigt dann den Dialog erst an. Da der Dialog modal ist, hält der Code an dieser Stelle an. Hat der Anwender eine Datei ausgewählt, so quittiert die Funktion dies mit der Rückgabe **True**, wonach die Variable **sFile** mit dem ersten und einzigen Element der **SelectedItems**-Auflistung bestückt wird. Andernfalls ist die Länge des Strings in **sFile** gleich 0 und die Prozedur wird verlassen.

Weiter wird nun der Dateipfad in das Textfeld **txtDatei** des Formulars geschrieben. Die Dateiendung wird über einen Ausdruck mithilfe der **Mid**- und **InstrRev**-Funktionen ermittelt und in die Variable **sExt** gespeichert. Soweit die Vorarbeiten. Für das Verständnis der restlichen Zeilen der Routine, die im Listing ausgelassen wurden, müssen erst die Vorgänge beim Laden und Anzeigen des Formulars näher beleuchtet werden.

Initialisierung des Webbrowser Controls beim Laden des Formulars

In Form_Load steht nur eine Zeile:

```
Set oBrowser = Me!ctlWeb.Object
```

Das Webbrowser-Steuerelement hat im Formular den Namen **ctlWeb**. Seine **Object**-Eigenschaft gibt den eigentlichen Webbrowser zurück, die der Objektvariablen zugewiesen wird. Deren Deklaration befindet sich im Modulkopf des Formulars:

```
Private Sub cmdLoad_Click()
    Dim ODlg As Office.FileDialog
    Dim sFile As String
    Dim sExt As String
    Dim bin() As Byte

    Set ODlg = Application.FileDialog(msoFileDialogFilePicker)
    With ODlg
        .AllowMultiSelect = False
        .InitialView = msoFileDialogViewDetails
        .InitialFileName = CurrentProject.Path
        .Filters.Add "Bild-Dateien", "*.jpg,*.jpeg,*.png,*.
bmp,*.tif*"
        If .Show Then
            sFile = .SelectedItems(1)
        End If
    End With
    If Len(sFile) = 0 Then Exit Sub

    Me!txtDatei = sFile
    sExt = Mid(sFile, InStrRev(sFile, ".") + 1)
    ...
    ... (Fortsetzung folgt) ...
End Sub
```

Listing 1: Beim Laden einer Bilddatei über den Button **cmdLoad** spielt sich diese Prozedur ab (erster Teil)

```
Private WithEvents oBrowser As WebBrowser
```

Die Variable ist also vom Typ **Webbrowser** und das Statement **WithEvents** gibt an, dass dieses Objekt Ereignisse auslösen kann. Nun werden Sie eine Klasse **Webbrowser** im Objektkatalog vergeblich suchen. Die Access-Bibliothek enthält deren Definition nicht. Der Typ ist nämlich in einer Bibliothek zu finden, die Sie erst einmal in die Verweise des VBA-Projekts laden müssen: die **Microsoft Internet Controls** (**shdocvw**). Diese Bibliothek ist das Kernstück des **Internet Explorers**. Markieren Sie nun im Objektkatalog die Klasse **Webbrowser**, so stellen Sie fest, dass diese fast die gleichen Methoden und Eigenschaften aufweist, wie die Klasse **WebbrowserControl** von Access, nur dass jene viele zusätzliche Eigenschaften zeigt, die sich auf die Gestalt des Steuerelements im Formulars beziehen. Microsoft hat im **Webbrowser-**

Indizierung in der Praxis

In jedem Lehrbuch zur Datenbankentwicklung erfahren Sie, wie wichtig die Indizierung von Tabellenfeldern ist. Das betrifft jene, die in Vergleichs- und Filterabfragen eingeschlossen werden sollen und außerdem die Felder, welche mehrere Tabellen über Indexfelder über Schlüssel miteinander verknüpfen. Ziel sind möglichst performante Abfragen der Datenbank. Wir untersuchen hier, wie groß der Performancezuwachs in der Praxis tatsächlich ist.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1702_Indizes.zip**.

Wozu Indizierung?

In einer relationalen Datenbank gilt die Indizierung von mindestens den Verknüpfungsfeldern der Tabellen als das A und O. Sehen Sie sich dazu etwa das Datenmodell von Kunden und deren Bestellungen in Bild 1 an.

Zu jedem Kunden, identifiziert durch dessen **ID**, kann es mehrere Bestellungen geben. Das Feld **KundeID** der Tabelle bezieht sich auf die **ID** des Kunden. Damit ist die Verbindung hergestellt. Ähnlich verhält es sich mit den Tabellen zu Bestelldetails und Artikeln. Hier stellt jeweils die **ID** den Primärschlüssel dar, auf den sich die Datensätze einer Detailtabelle über einen Schlüssel **xxxID** verweisen. Die Bestelldetails machen das gleich zweimal.

Die Indizierung der beteiligten Schlüsselfelder ist immer dann zwingend, wenn Sie eine automatische **Lösch- und Aktualisierungsweitergabe** erreichen möchten. Access erlaubt die Einstellung dieser Optionen für eine Beziehung nur dann, wenn bereits Indizes auf die Felder gesetzt wurden. Das Löschen eines Kunden zieht dann das Löschen auch aller seiner Bestellungen samt Bestelldetails nach sich.

Doch oft ist **Lösch- oder Aktualisierungsweitergabe** gar nicht erforderlich oder erwünscht. Auch dann wird aber zu einer Indizierung geraten, weil dies der Performance zugutekommt. Sollen etwa die Bestellungen eines Kunden mit der **ID 79** gefunden werden, so muss die Datenbank-Engine alle Datensätze der Tabelle **tblBestellungen** durchlaufen und die Werte von **KundeID** mit dieser Zahl vergleichen.

Bei Millionen von Bestelldatensätzen dauert dies recht lange. Ist das Feld **KundeID** aber indiziert, so

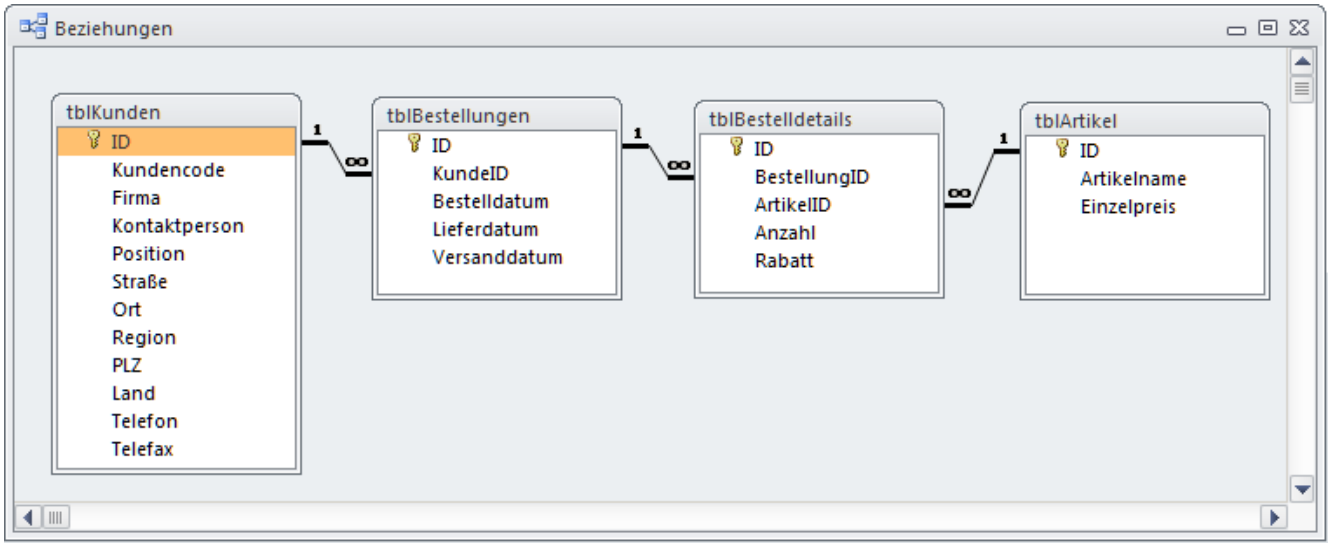


Bild 1: Die vier Tabellen der Kundendatenbank sind alle miteinander über Schlüsselfelder verknüpft

zieht Access dessen Index zurate. Dieser erlaubt eine viel schnellere Suche nach dem Wert **79**.

Soll gar zu einem Kunden ermittelt werden, welche Artikel er bereits bestellte, so ist die Tabelle **tblBestelldetails** und deren Feld **ArtikelID** zu durchsuchen. Da dies nur über den Umweg der Zwischentabelle **tblBestellungen** geschehen kann, sind nun schon zwei Tabellen verschachtelt komplett zu durchsuchen. Das potenziert die Suchdauer.

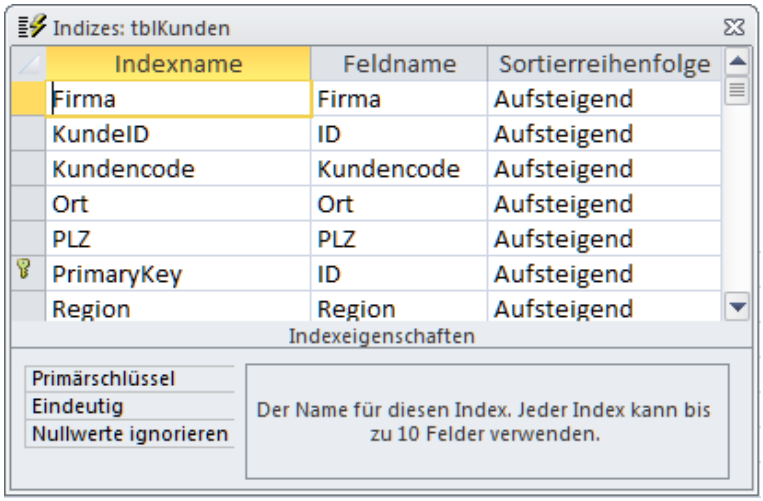


Bild 2: Der Indizes-Dialog zur Tabelle tblKunden im Entwurf

Der zweite Anwendungsfall für einen Index ist die unmittelbare Suche nach einem Feldinhalt. Möchten Sie etwa einen Kunden über dessen Namen finden, so muss Access wieder alle Datensätze scannen, um den oder die gewünschten zu erkennen. Sind die Namensfelder mit Indizes versehen, so verwendet die Engine diese zur Suche, was abermals schneller abläuft.

Soweit ist dies alles wahrscheinlich kein Fremdland für Sie!

Was ist ein Index?

Nehmen wir an, Sie hätten ein E-**Book**, wie das vorliegende, in dem Sie ein bestimmtes Thema finden möchten. Sie könnten nun eine Volltextsuche anstoßen, um nach einem Begriff zu suchen, und das dauert einige Zeit. Einfacher ist es da, sich eines Index im hinteren Teil des Buchs zu bedienen. Dort stehen zu jedem Begriff mehrere Seitenzahlen, über die Sie wahrscheinlich schneller zum Thema gelangen.

Ähnlich verhält es sich mit den Indizes einer Datenbank. Mache System weisen tatsächlich auch physisch einen Index auf. Zu einer **DBase**-Datenbank finden Sie etwa in der Regel neben der eigentlichen Datenbank (**dbf**) auch eine Datei mit der Endung **idx**, die lediglich die Indizes zu den Tabellen enthält. Intern speichert auch die **Access Database Engine** die Indizes in gesonderten Speicherbereichen, nur dass dies in der Oberfläche von Access nicht offenkundig ist. Indizieren Sie ein Tabellenfeld, so zeigt sich dies ja lediglich in der Entwurfsansicht am Eigenschaftenfeld **Indiziert** oder im **Indizes**-Dialog (Ribbon **Entwurf | Indizes**), wie in Bild 2. Intern hat Access aber Indextabellen angelegt.

Um zur Analogie eines Buches zurückzukehren: Suchen Sie etwa nach **Löschweitergabe**, so suchen Sie erstens im Inhaltsverzeichnis nach der ersten Seite des Index. Dort finden Sie die Begriffe alphanumerisch sortiert vor. Sie blättern zur Seite mit dem **L** als Überschrift und finden in der Liste den Begriff **Löschweitergabe** mit drei angefügten Seitenzahlen. Sie inspizieren diese Seiten nacheinander, um die gewünschten Informationen zu erhalten. So ähnlich arbeitet auch ein Index unter Access.

Allerdings sind diese internen Indextabellen wesentlich komplexer und hochoptimiert, damit sich die Zahl der Sprünge zu Einträgen und Untereinträgen so gering hält, wie möglich. Die hier verwendeten Algorithmen sind eine Wissenschaft für sich und tragen Namen, wie B-Tree. Ein mögliches vereinfachtes Modell bei aufsteigender Sortierung für einen Index:

```
K ->143
L -> 188
M -> 296
188 -> n -> 2354
188 -> o -> 2389
```

2389 -> d -> 7843
2389 -> e -> 7911

An der Speicherstelle **7911** sind wir immerhin schon bei **Loe** angekommen, den ersten drei Buchstaben von **Löschweitergabe**. Das Spiel setzt sich auf ähnliche Weise fort, bis der Begriff vollständig ermittelt ist. Dort stehen dann ein oder mehrere Zeiger auf den Speicherplatz der Datensätze innerhalb der eigentlichen Tabelle.

Für den Begriff braucht es deshalb vielleicht nur 15 Sprünge zwischen Speicherstellen, bis er im Index ermittelt ist. Dem entgegen stünde ein Durchlaufen aller Tausender Datensätze einer Tabelle, wenn auf einen Index für das Feld verzichtet würde.

Es ist also sehr plausibel, dass ein Index die Performance gewaltig erhöht. Zumindest gilt das für das Lesen von Tabellen. Bei Anfügen oder Ändern eines Datensatzes hingegen kommt es zu einem Nachteil: Nicht nur die Daten in der Tabelle selbst müssen geändert werden, sondern auch der Index! Sein ganzer Aufbau muss dem neuen Tabelleninhalt angepasst werden, damit die Suche effizient abläuft.

Imgrunde müssten dazu sämtliche Speicherstellenverweise neu angeordnet werden, damit die Buchstaben oder Zahlen wieder korrekt sortiert in ihm vorliegen. In der Realität passiert das so aber nicht, sondern nur in Teilen, weil es die Performance stark senken würde. Zu einer kompletten Neuordnung der Indizes kommt es erst dann, wenn Sie die Funktion **Komprimieren und Reparieren** für die Datenbank anstoßen!

Ein Test mit umfangreicher Tabelle

Um zu überprüfen, wie sich die Indizierung in der Praxis auswirkt, verwenden wir erst einmal eine Tabelle mit sehr vielen Datensätzen. Die Wahl fiel auf den **OpenThesaurus**, eine Tabelle, die synonyme Begriffe enthält. Die Tabelle der Beispieldatenbank kommt aus dem gleichnamigen **Open Source**-Projekt und enthält

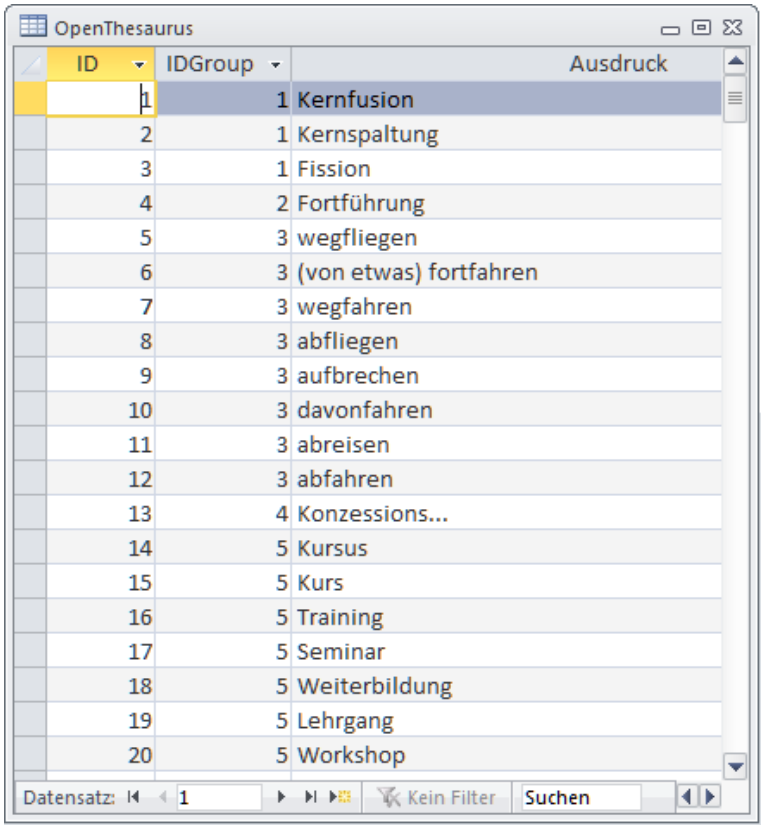


Bild 3: Die Tabelle **OpenThesaurus** in der Datenblattansicht

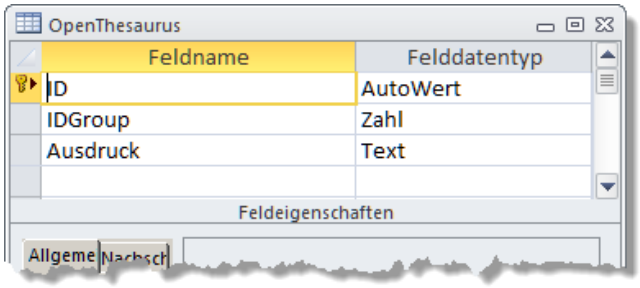


Bild 4: Die Tabelle **OpenThesaurus** in der Entwurfsansicht

etwa 125.000 Datensätze. Bild 3 zeigt einen Ausschnitt der Tabelle.

In Bild 4 finden Sie den Entwurf der Tabelle. Die Begriffe stehen in der Spalte **Ausdruck**. Jene Begriffe, die synonym sein sollen, haben alle die gleiche **IDGroup**. Man muss also nur nach einem Begriff suchen, die **IDGroup** desselben auslesen, um über diese und eine Abfrage alle anderen Begriffe zu erhalten. Jeder Datensatz enthält darüber hinaus eine eindeutige **ID**.

Es ist naheliegend, dass eine Indizierung der Felder **Ausdruck** (für die Suche nach einem Begriff) und **IDGroup** (für das Auffinden synonymer Begriffe) hier Pflicht ist. Wir haben dies für die Tabelle so gemacht und jeweils in die Eigenschaft **Indiziert** im Entwurf die Auswahl **Ja (Duplikate möglich)** eingestellt.

Um die Performance mit einer Tabelle ohne Indizes zu vergleichen, enthält die Beispieldatenbank zusätzlich die gleiche Tabelle ohne Indizierung, die den Namen **OpenThesaurus2** trägt. Und eine weitere nennt sich **OpenThesaurus3**. Sie leitet sich von der zweiten ab, hat aber für das Feld **Ausdruck** keine **Unicode-Kompression** eingestellt. Access stellt diese von Haus aus auf **Ja** ein. Alle Zeichen von Textfeldern werden unter Access ja als **Unicode** abgespeichert, um auch Sonderzeichen und ausländische Schriftzeichen abbilden zu können.

So ein **Unicode**-Zeichen nimmt zwei Bytes ein, was den Speicherbedarf gegenüber **ANSI** (ein Byte) verdoppelt. Aus diesem Grund wurde die Kompression dieser Doppelzeichen eingeführt, die den physischen Speicherbedarf reduziert. Man sollte nun annehmen, dass die Komprimierung und Dekomprimierung der Texte zusätzliche Rechenzeit benötigt und die Performance senken könnte. Auch das überprüfen wir im Folgenden. Eine Testprozedur im Modul **mdlTest** hat nun prinzipiell den Code aus Listing 1. Die Zeit wird über die Variable **T** gemessen. Sie kommt aus der VBA-Funktion **Timer**, die die aktuelle Zeit in Sekunden als **Single**-Wert mit drei Nachkommastellen zurückgibt. Das Recordset wird auf die Tabelle **OpenThesaurus** geöffnet und als Filter bestimmt, dass nur die Datensätze mit dem **Ausdruck Akronym** berücksichtigt werden sollen. In einer Schleife wird das Ergebnis durchlaufen und jeweils in der Variablen **V** abgespeichert. Nach dem Schließen des Recordsets gibt die letzte Zeile der Prozedur die Differenz der aktuellen Zeit zu der eingangs in **T** gespeicherten im VBA-Direktfenster aus.

Damit die Bedingungen für jeden Aufruf der Prozedur annähernd gleich sind, ist oben noch der Aufruf einer Hilfsroutine **InitEngine** eingeschoben (Listing 2). Was

```
Sub PerformanceRead()
    Dim T As Single
    InitEngine
    T = VBA.Timer
    Set rs = CurrentDb.OpenRecordset("SELECT * From ⤵
      OpenThesaurus WHERE Ausdruck='Akronym'", dbOpenDynaset)
    Do While Not rs.EOF
        V = rs!Ausdruck.Value
        rs.MoveNext
    Loop
    rs.Close
    Debug.Print  VBA.Timer - T
End Sub
```

Listing 1: Performance-Test für den OpenThesaurus

```
Sub InitEngine()
    DBEngine.Idle dbFreeLocks
    DBEngine.BeginTrans: DBEngine.CommitTrans dbForceOSFlush
    DBEngine.Idle dbRefreshCache
    DoEvents
End Sub
```

Listing 2: Initialisieren der Datenbank-Engine

macht diese? Die erste Zeile setzt die Anweisung **Idle** für die Engine von Access ab, wobei als Parameter ein **dbFreeLocks** übergeben wird. Das bewirkt, dass eventuell noch ausstehende Datensatzsperrungen aufgehoben werden. Bedenken Sie, dass die Engine nicht nur Ihre Tabellen bearbeitet, sondern auch die Systemtabellen von Access! Im Hintergrund hat sie unter Umständen also noch etwas zu erledigen.

Die zweite Zeile leitet eine sogenannte **Nulltransaktion** ein. Erst startet die Engine eine Datentransaktion mit **BeginTrans**. Dann beendet sie diese, ohne dazwischen irgendetwas getan zu haben, mit **CommitTrans**. Hier ist der Parameter **dbForceOSFlush** angegeben. Er weist die Engine an, alle noch ausstehenden Schreibvorgänge zu beenden und physisch in die Datenbankdatei zu schrieben. Sonst könnten solche Datenvorgänge möglicherweise nur im **Cache** der Engine stehen!

Die dritte Zeile löst wieder die Methode **Idle** aus, diesmal jedoch mit dem Parameter **dbRefreshCache**.

Sortieralgorithmen

In Datenbanken muss man sich über die Sortierung von Datensätzen gemeinhin keine Gedanken machen. Die Engines enthalten alles Benötigte, um Daten in Abfragen oder auch Recordsets über einfache SQL-Statements sortiert auszugeben. Doch hin und wieder steht man vor der Aufgabe, auch Daten in Arrays zu sortieren, etwa, um sie einem Treeview oder einem Listenfeld zu verabreichen. Über die möglichen Algorithmen erfahren Sie hier mehr.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1702_Sorting.zip**.

Sortieren per VBA

Einen kleinen Vorgeschmack lieferte bereits der Beitrag zu **BubbleSort** der Doppelausgabe 01/2016. Dieser Algorithmus ist ziemlich simpel und durchschaubar. Er hat aber seine Grenzen erreicht, sobald es um die Performance geht. Bei umfangreichen Datenbeständen ist er extrem langsam. Hier sind Alternativen gefragt.

Eigentlich kommt es ziemlich selten vor, dass Daten per VBA sortiert werden müssen. Sie liegen ja in der Regel in Tabellen vor, und die lassen sich über Abfragen sortieren. Manchmal jedoch kommen sie aus externen Quellen, wie etwa Textdateien. Natürlich könnte man diese in Tabellen importieren, dann einer Sortierabfrage unterziehen und das Ergebnis, so benötigt, wieder im VBA-Projekt über ein Recordset auslesen. Eleganter ist es aber, diesem Umweg zu vermeiden und die Daten direkt per VBA zu sortieren. Legen Sie sich dazu einfach ein Modul mit generellen Sortieralgorithmen an, wie in der Beispieldatenbank geschehen, das Sie später gegebenenfalls in ihre Zieldatenbank kopieren können.

Ausgangslage: String-Array

Der häufigste Fall ist sicher das Sortieren von Texten eines String-Arrays. Um diesen Anwendungsfall mit den im Folgenden vorgestellten Algorithmen zu testen, wird ein gefülltes String-Array mit einer nicht zu geringen Anzahl von Einträgen benötigt. Wir holen sie einfach aus einer vorliegenden Kundentabelle. Die Hilfsfunktion **CreateStringArray** (siehe Listing 1) stellt es zur Verfügung.

Hier werden die Felder **Nachname** und **Vorname** der Tabelle **tblKunden** in ein Recordset aufgenommen, das nach der **ID** der Datensätze sortiert ist. Die Namen selbst sind damit unsortiert. Eine Schleife durchläuft alle Datensätze und berücksichtigt nur jene, in der die Namen mindestens drei Buchstaben

```
Private Function CreateStringArray(Optional Limit As Long) As String()
    Dim rs As DAO.Recordset
    Dim S As String
    Dim n As Long
    Dim arrRet() As String
    Set rs = CurrentDb.OpenRecordset("SELECT Nachname, Vorname" & _
        "FROM tblKunden ORDER By ID", dbOpenDynaset)
    Do While Not rs.EOF
        If (Len(rs!Nachname.Value) > 2) And _
           (Len(rs!Vorname.Value) > 2) Then
            S = rs!Nachname.Value & ", " & rs!Vorname.Value
            ReDim Preserve arrRet(n)
            arrRet(n) = S
            n = n + 1
        End If
        If Limit <> 0 Then If n > Limit Then Exit Do
        rs.MoveNext
    Loop
    rs.Close
    CreateStringArray = arrRet
End Function
```

Listing 1: Über diese Funktion werden die Daten einer Kundentabelle in ein String-Array überführt

enthalten. In der Variablen **S** werden dann **Nachname** und **Vorname**, durch Komma getrennt, aneinandergefügt. Dieses Ergebnis wird jeweils einem Element des Arrays **arrRet** zugewiesen, dessen Dimension sich ständig über die Zählvariable **n** erweitert, wobei die **ReDim**-Anweisung den Inhalt nicht zerstört, weil sie eine **Preserve**-Klausel enthält. Optional können Sie der Prozedur per Parameter ein **Limit** übergeben, welches die Anzahl der Elemente des Ergebnis-Arrays begrenzt. Ohne Limit kommt es zu etwa 9.000 Elementen.

Sie rufen diese Funktion etwa so auf:

```
Dim arr() As String
arr = CreateStringArray()
... (Sortieren von arr() ) ...
```

Hier handelt es sich um ein eindimensionales Array. Nachname und Vorname stehen nicht in getrennten Feldern. Letzteres würde ein zweidimensionales Array erfordern und damit auch kompliziertere Sortieralgorithmen. Der Trick des Verkettens von Nach- und Vorname über ein Trennzeichen sortiert aber automatisch korrekt nach beiden Feldern, soweit zufällig nicht in einem Nachnamen ein Komma enthalten ist. Nun kann es an das Sortieren des Test-Arrays gehen.

Wizhook

Access-VBA hält an sich keine Funktion vor, um eine solches Array zu sortieren. Tatsächlich existiert jedoch eine Methode eines versteckten Klassenobjekts, die das ermöglicht. Blenden Sie im Objektkatalog per Kontextmenü die **Verborgenen Elemente** ein, so finden Sie die Klasse **Wizhook** in ihm, bei der es sich um eine Sammlung von Funktionen zu verschiedensten Aspekten handelt. Sie müssen kein Objekt dieser Klasse neu erzeugen, da Access sie von sich aus instanziiert. Der Beweis im VBA-Direktfenster:

```
? Wizhook Is Nothing
-> Falsch
```

Eine Methode der Klasse lautet **SortStringArray** mit dieser Definition:

```
Sub SortStringArray(Array() As String)
```

Zu übergeben ist ihr also ein String-Array, das allerdings eindimensional sein muss. Sie sortiert das Array auch wunschgemäß. Das aber erst, nachdem die Klasse freigegeben wurde! Denn ohne einen geheimen Schlüssel funktionieren keine der Funktionen des **Wizhook**-Objekts. Sie schalten sie erst über das Setzen der **Key**-Eigenschaft frei. Wozu Microsoft das implementiert hat, ist ein Rätsel, denn dieser Zahlenschlüssel ist natürlich keineswegs geheim:

```
Sub SortWizHook(sarr() As String)
    WizHook.Key = 51488399
    WizHook.SortStringArray sarr
End Sub
```

Die Methoden dieser Klasse wurden übrigens mit jeder neuen Access-Version erweitert. Sie sind nicht dokumentiert, und auch einige Seiten im Netz konnten nur einen Teil der Funktionen analysieren.

Der Key wird hier bei jedem Aufruf der Funktion **SortWizHook** gesetzt. Das ist allerdings nicht wirklich notwendig. Einmal gesetzt bleibt die Klasse für die gesamte VBA-Sitzung freigeschaltet. Die korrekte Funktion können Sie nun etwa so testen:

```
Dim arr() As String
Dim i As Long

arr = CreateStringArray()
SortWizHook arr
For i = 0 To Ubound(arr)
    Debug.Print arr(i)
Next i
```

Im VBA-Direktfenster listen sich nun alle Namen des Arrays alphanumerisch aufsteigend auf. Der **Wizhook**-Methode kann nicht angegeben werden, ob auf- oder absteigend sortiert werden soll. Das hat sie auch mit den meisten der folgenden Algorithmen gemein. Wäre eine absteigende Sortierung gewünscht, so müsste diese anschließend über ein erneutes Umsortierung mit einer Hilfsfunktion erreicht werden:

```
Sub SortReverse (arr() As String)
    Dim arrRet() As String
    Dim n As Long, i As Long

    n= UBound(arr)
    Redim arrRet(n)
    For i = 0 To n
        arrRet(i) = arr(n-i)
    Next i
End Sub
```

Das tut der Performance keinen Abbruch und verläuft ziemlich schnell.

BubbleSort

Der Vollständigkeit halber bilden wir den **BubbleSort**-Algorithmus nochmals in Listing 2 ab. Jedes Element des Arrays **sarr** wird ausnahmslos mit jedem verglichen. Ist das eine größer, als das andere, so kommt es zum Vertauschen der beiden über die temporäre Variable **S**. Eine absteigende Sortierung erhalten Sie hier durch Änderung der Vergleichszeile:

```
If sarr(i) <
sarr(j) Then
```

Bei den 9.000 Elementen des Beispiel-Arrays kommt es hier zu über 40 Millionen Iterationen. Das beansprucht auch auf einem aktuellen Rechner ziemlich viel Rechenzeit. Wir

```
Sub BubbleSortStrings(sarr() As String)
    Dim S As String
    Dim i As Long, j As Long, n As Long

    n = UBound(sarr)
    For i = 0 To n - 1
        For j = i + 1 To n
            If sarr(i) > sarr(j) Then
                S = sarr(i)
                sarr(i) = sarr(j)
                sarr(j) = S
            End If
        Next j
    Next i
End Sub
```

Listing 2: Funktion zum Sortieren über **BubbleSort**

```
Public Sub QuickSortStrings(sarr() As String, Optional ByVal LL As Long, Optional ByVal LR As Long)
    Dim n1 As Long, n2 As Long
    Dim sTmp As String, sSwap As String

    If LR = 0 Then   LL = LBound(sarr): LR = UBound(sarr)
    n1 = LL: n2 = LR
    sTmp = sarr((LL + LR) \ 2)
    Do
        Do While (sarr(n1) < sTmp) And (n1 < LR)
            n1 = n1 + 1
        Loop
        Do While (sTmp < sarr(n2)) And (n2 > LL)
            n2 = n2 - 1
        Loop
        If n1 <= n2 Then
            sSwap = sarr(n1)
            sarr(n1) = sarr(n2)
            sarr(n2) = sSwap
            n1 = n1 + 1
            n2 = n2 - 1
        End If
    Loop Until n1 > n2
    If LL < n2 Then QuickSortStrings sarr, LL, n2
    If n1 < LR Then QuickSortStrings sarr, n1, LR
End Sub
```

Listing 3: Funktion zum Sortieren über **Quicksort**

haben bei uns 13 Sekunden gemessen – ein völlig inakzeptabler Wert! Die **Wizhook**-Methode erreichte dasselbe Ergebnis in nur 0,03 Sekunden!

Sehen wir, ob sich dieser Wert durch andere Methoden noch unterbieten lässt.

QuickSort

Dieser Algorithmus ist zweifellos der verbreitetste überhaupt. Es ist davon auszugehen, dass auch die **Wizhook**-Methode, wie auch die **Database Engine**, ihn intern verwenden. Seine Effizienz ist kaum zu überbieten. Der Witz an ihm ist der Umstand, dass er ausgerechnet dann am schnellsten ist, wenn die Daten möglichst unsortiert daher kommen. Sind die Daten bereits weitgehend vorsortiert, dann ist er recht langsam!

Der Algorithmus in Listing 3 arbeitet **rekursiv**. Das heißt, dass sich die Funktion fortwährend selbst aufruft, wie an den letzten beiden Zeilen zu erkennen ist. Die jeweiligen Elementzeiger stehen in den Variablen **LR** und **LL**, stellvertretend für **Rechter Zeiger** und **Linker Zeiger**.

Das Prinzip besteht darin, dass die Daten zunehmend hälftig in Blöcke unterteilt und dann jeweils nur die Elemente eines Blocks untersucht werden. Und die Blöcke werden immer kleiner, bis sie nur noch aus zwei Elementen bestehen. Das schränkt die Anzahl der Vergleiche stark ein.

Sie übergeben der Funktion nur das Array und lassen die beiden optionalen Parameter unberücksichtigt:

```
Dim arr() As String

arr = CreateStringArray()
QuickSortStrings arr
```

Obwohl VBA nicht gerade die schnellste Programmiersprache der Welt ist, messen wir für das Beispiel eine Ausführungszeit von nur 0,039 Sekunden. Das kann schon mit der **Wizhook**-Methode konkurrieren!

Doch wozu eine benutzerdefinierte Funktion einsetzen, wenn es schon eine eingebaute gibt?

Das Modul **mdlSortStrings** enthält eine Variante des Quicksort-Algorithmus (**QuickSortVariants**), dem Sie auch ein **Variant**-Array übergeben können. Sein Aufbau ist ansonsten identisch zu dem in **Listing 2**. Sie können der Routine also ein Array übergeben, das beliebige Datentypen enthält, wie etwa Zahlenwerte (**Long**) oder Datumswerte (**Date**).

Die Elemente müssen in der Deklaration des Arrays allerdings vom Typ **Variant** sein. Wie sich herausstellt, ist diese Alternative genauso schnell, wie die String-Variante. Und das ist ein Vorteil, weil die **Wizhook**-

```
Public Sub ShellSortStrings(sarr() As String)
    Dim lHold As Long
    Dim lGap As Long
    Dim i As Long
    Dim iMin As Long
    Dim iMax As Long
    Dim sSwap As String

    iMin = LBound(sarr)
    iMax = UBound(sarr)
    lGap = iMin
    Do
        lGap = 3 * lGap + 1
    Loop Until lGap > iMax
    Do
        lGap = lGap \ 3
        For i = lGap + iMin To iMax
            sSwap = sarr(i)
            lHold = i
            Do While sarr(lHold - lGap) > sSwap
                sarr(lHold) = sarr(lHold - lGap)
                lHold = lHold - lGap
                If lHold < iMin + lGap Then Exit Do
            Loop
            sarr(lHold) = sSwap
        Next i
    Loop Until lGap = 1
End Sub
```

Listing 4: Funktion zum Sortieren über **Shellsort**

Methode nun mal nur Strings als Input akzeptiert.

Shellsort

Auch dieser Algorithmus gehört zu den performanteren. Eine Implementation in der Beispieldatenbank zeigt Listing 4. Das Verfahren ähnelt dem **Quicksort**-Algorithmus, weil er auch hier die Elemente des Arrays in Blöcke aufteilt, die den Vergleichsoperationen unterzogen werden. Er arbeitet aber nicht rekursiv, sondern verwendet stattdessen verschachtelte Schleifen. Sie können die Routine auch für **Variants** abändern, indem Sie die String-Angaben in **Variant**-Deklarationen umwandeln. Eigentlich würde man erwarten, dass VBA mit **Variant**-Datentypen langsamer ist, als mit diskreten Datentypen. Zumindest kann man derlei auf verschiedenen Seiten zur Performanceoptimierung lesen. Unsere Erfahrungen bestätigen dies nicht! Heutzutage sind die Windows-Bibliotheken, die für die interne Umwandlung von **Variant**-Typen angesprochen werden, dermaßen optimiert, dass der geringe Mehraufwand auf aktuellen Rechnern nicht mehr ins Gewicht fällt.

Unser Test sieht nun so aus:

```
Dim arr() As String

arr = CreateStringArray()
ShellSortStrings arr
```

```
Public Sub ShakerSortStrings(sarr() As String)
    Dim i As Long, j As Long, k As Long, iMin As Long, iMax As Long
    Dim sSwap As String, b As Boolean

    iMin = LBound(sarr): iMax = UBound(sarr)
    i = (iMax - iMin) * 1999 \ 2000 + iMin
    Do
        j = i: b = False
        Do While j > iMin
            For k = iMin To i - j
                If sarr(k) > sarr(k + j) Then
                    sSwap = sarr(k): sarr(k) = sarr(k + j): sarr(k + j) = sSwap: b = True
                End If
            Next
            If b Then
                j = j * 7 \ 10
            ElseIf j > k \ 2 Then j = j \ 2
            Else: Exit Do
            End If
        Loop
        If i > iMin + 3 Then
            i = i \ 20
        ElseIf i > iMin + 2 Then i = iMin + 2
        ElseIf i > iMin + 1 Then i = iMin + 1
        Else: Exit Do
        End If
    Loop: iMax = iMax - 1
    Do
        b = False
        For i = iMin To iMax
            If sarr(i) > sarr(i + 1) Then
                sSwap = sarr(i): sarr(i) = sarr(i + 1): sarr(i + 1) = sSwap: b = True
            End If
        Next i
        If b Then
            b = False: iMax = iMax - 1
            For i = iMax To iMin Step -1
                If sarr(i) > sarr(i + 1) Then
                    sSwap = sarr(i): sarr(i) = sarr(i + 1): sarr(i + 1) = sSwap: b = True
                End If
            Next i: iMin = iMin + 1
        End If
    Loop Until Not b
End Sub
```

Listing 5: Funktion zum Sortieren über **Shakersort**

Lokaler Webshop, Teil III

Die Auswahl der Artikel ist getätigt, der Warenkorb gefüllt, der Kassenvorgang abgeschlossen und eine Bestellbestätigung versandt worden. Welche Vorgänge nun noch für den in Access fingierten Webshop zu vollziehen sind, findet sich hier im letzten Beitrag zur Reihe. Vornehmlich hat der Vertrieb nun das Wort, damit die Ware zum Versand kommt.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1702_WebshopIII.zip**.

Der Beitrag setzt die Serie aus den Ausgaben **08/2016** und **10/2016** fort.

Alte Bestellungen einsehen

Beim Öffnen des zentralen Shop-Formulars **frmShop** wird, wie früher bereits ausgeführt, aus einem Cookie die **ID** des Kunden ausgelesen, der den Shop zuletzt besuchte. Das übernimmt die vom **Beim Laden**-Ereignis (**Form_Load**) aufgerufene Funktion **LoadTempVar**:

```
LoadTempVar "KundeID"
```

Sie liest aus der Datei **cookie.txt** im Datenbankverzeichnis die beim letzten Schließen des Shops abgelegte **ID**, um sie in der **TempVar** "**KundeID**" zwischenzuspeichern. Diese **ID** steht forthin für die Dauer der Sitzung allen weiteren Operationen zur Verfügung.

```
Private Sub cmdBestellungen_Click()
    If TempVars("KundeID") = 0 Then
        DoCmd.OpenForm "frmLogin", , , , , acDialog
    Else
        If MsgBox("Neu einloggen?", vbQuestion Or vbYesNo, sLogo) = vbYes Then
            DoCmd.OpenForm "frmLogin", , , , , acDialog
        End If
    End If
    If TempVars("KundeID") = 0 Then Exit Sub
    ShowKunde
    DoCmd.OpenForm "frmBestellungen", , , "KundeID=" & TempVars("KundeID")

Private Sub ShowKunde()
    Dim sKunde As String

    sKunde = DLookup("[Vorname] & ' ' & [Nachname]", _
        "tblKunden", "ID=" & TempVars("KundeID"))
    Me!LblKunde.Caption = "Hallo, " & sKunde
    Me!LblKunde.Visible = True
End Sub
```

Listing 1: Aufruf der getätigten Bestellungen über den Button **cmdBestellungen**

Sie wirkt sich etwa aus, wenn in der rechten oberen Ecke des Shops der Button **Meine Bestellungen...** angeklickt wird. Der für das **Click**-Ereignis hinterlegte Code findet sich in Listing 1. Zunächst erfolgt hier eine Überprüfung, ob die **TempVar** überhaupt schon gefüllt ist, was genau dann **nicht** der Fall ist,

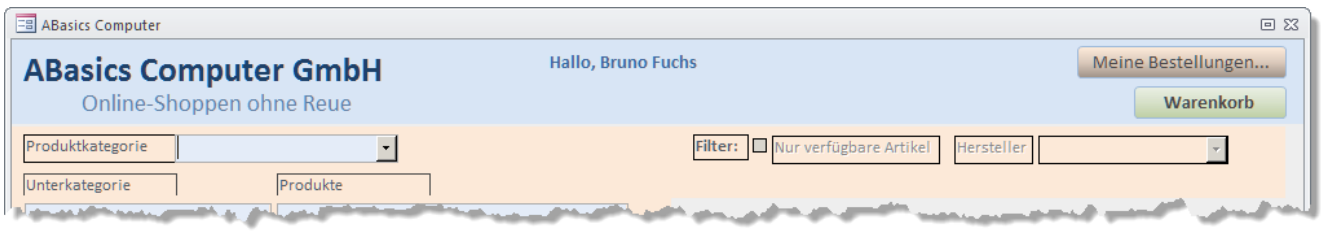


Bild 1: Im Kopf des Shop-Forumlar **frmShop** blendet sich eine Begrüßungformel mit dem Namen des Kunden ein

wenn sich ein Kunde zum ersten Mal im Shop bewegt. Ihr Wert beträgt dann **0**. In diesem Fall erscheint das Dialog-Formular für einen **Login**, das nach Beendigung die **TempVar KundeID** selbst setzt, falls die Aktion nicht abgebrochen wird.

Andernfalls kommt es zur Nachfrage nach einem neuen Login. Das könnte dann Sinn machen, falls der Kunde sich mit einer anderen E-Mail-Adresse anmelden möchte.

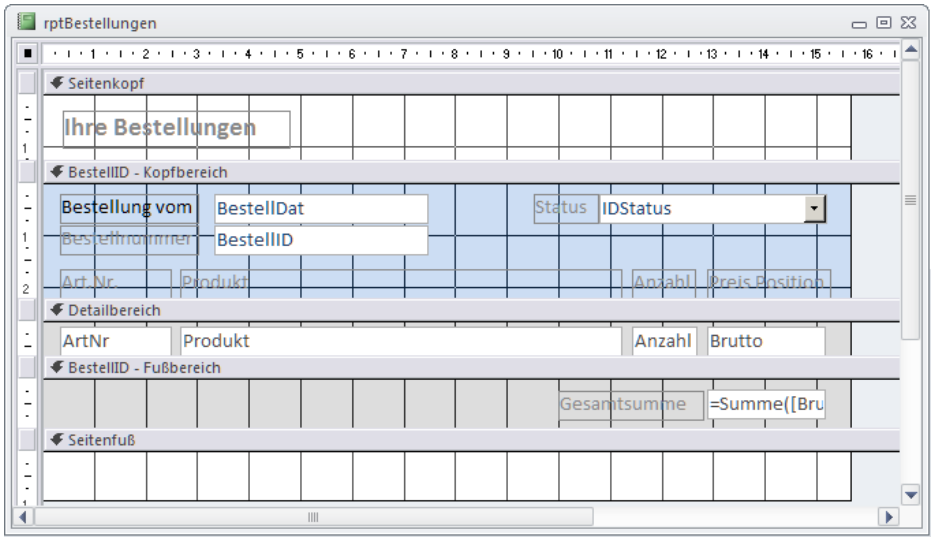


Bild 2: Die Entwurfsansicht des Berichts zur Einsicht in die getätigte Bestellungen

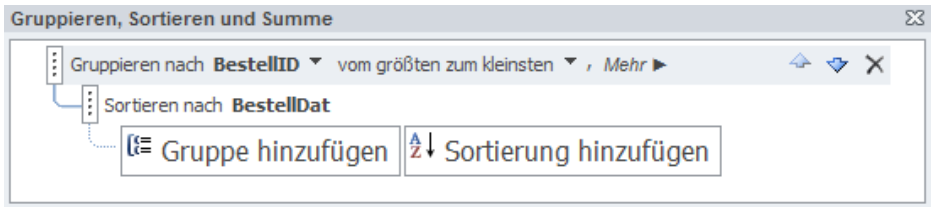


Bild 3: Der im Entwurf des Berichts unten eingeblendete Gruppen-Dialog

Wie auch immer: Nach dem Schließen der Dialoge kann **KundeID** immer noch den Wert **0** besitzen, worauf die Routine sang- und klanglos verlassen wird. Sonst aber öffnet sich das Formular für die Bestellungen. Zuvor allerdings wird noch ein Label im Shop-Formular über die Prozedur **ShowKunde** mit dem Namen des Users beschrieben, wofür eine **DLookup**-Funktion zum Einsatz kommt (siehe Bild 1).

Das über die **OpenForm**-Anweisung geladene Bestellungen-Formular **frmBestellungen** wird gleich im Aufruf über den optionalen Parameter **WhereCondition** nach der **ID** des Kunden gefiltert. Die Datenbasis für dieses Formular ist direkt die Tabelle **tblBestellungen**.

Allerdings spielt dies nur eine untergeordnete Rolle, denn das Formular selbst zeigt gar keine Daten an. Dies übernimmt ein im Detailbereich eingebetteter Unterbericht mit der Herkunft **rptBestellungen** in Berichtsansicht (siehe Bild 2).

Lediglich die Verknüpfung von Formularfeld **KundeID** zum Berichtsfeld **KundeID** ist vorzunehmen. Natürlich hätte man statt des Berichts auch ein Unterformular verwenden können, doch ersterer gestattet mehr Gestaltungsspielraum. Schließlich geht es hier nur um die passive Darstellung von Daten, wofür ein Bericht in der Regel die erste Wahl ist.

Der Bericht enthält im Detailbereich die Felder für die Artikelnummer (**ArtNr**), das Produkt, die Anzahl und den Bruttopreis einer Bestellposition. Das alles wird jedoch nach der Bestellung über deren Merkmal **BestellID** gruppiert, wie der Gruppen-Dialog des Berichts in Bild 3 zeigt.

Außerdem sortieren dessen Einstellungen noch die Bestellungen nach dem Bestelldatum (**BestellDat**). Im Fußbereich wird zu jeder Bestellung noch die **Gesamtsumme** über den hinterlegten Ausdruck **Summe([Brutto])** ausgegeben.

Im Ergebnis finden wir also alle Bestellungen des eingeloggten Kunden in der Reihenfolge der Bestellungen nach Datum absteigend angezeigt.

In Bild 4 finden Sie die Bestellhistorie mit dem Bericht im Formular dargestellt. Sowohl Bericht, wie auch Formular, kommen ohne jeglichen VBA-Code aus. Lediglich die Schaltfläche zum Schließen des Formulars (**Zurück zum Shop**) ist mit einer **Close**-Anweisung hinterlegt.

Selbstverständlich sind alle Felder des Berichts schreibgeschützt. Der Status einer Bestellung steht in einem Kombinationsfeld auf Basis der Tabelle **tblBearbeitungsStati**, gebunden an das Datenfeld **IDStatus**.

Nach Abschluss des Kassenvorgangs hat der Shop hier automatisch den Wert **1** für **Bestellung aufgegeben** eingesetzt. Die Aufgabe des Vertriebs ist es dann, diesen Wert zu ändern.

Bestellbestätigung

Bevor wir zu diesen Vorgängen kommen, fehlt noch der Bestätigungsbericht **rptBestaetigung**, der nach Abschluss der Bestellung per **Outlook** an den Kunden versandt wurde (siehe Bild 5).

Sein Entwurf (siehe Bild 6) ähnelt mit den Gruppierungsebenen im Prinzip jedem des Berichts für die Bestellhistorie, doch hier kommt zusätzlich noch die Kundenadresse ins Spiel, die im Seitenkopf ihren Platz findet.

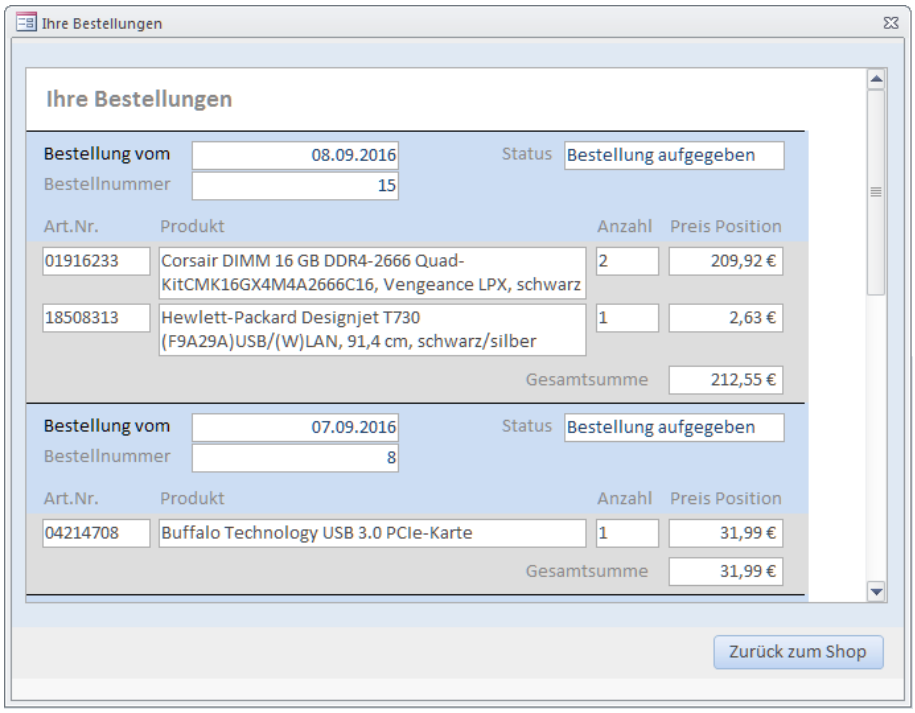


Bild 4: Das Formular **frmBestellungen** zeigt alle Bestellungen im Überblick

Bild 5: Der Bericht zur Bestellbestätigung per E-Mail

Die Datenherkunft des Berichts (Abfrage **qry_Kundenbestellung**) kommt entsprechend komplexer daher. Bild 7 zeigt die Abfrage mit ihren vier verknüpften Tabellen. Genaugenommen sind es nur drei, denn die Adresse des Kunden kommt selbst aus einer Abfrage **qry_Kundenadressen**, welche die Aufgabe hat, aus Namen, Strasse PLZ und Ort den kompletten Adressblock per String-Verkettung zu bilden.

Übersichtlicher als im Abfrageeditor lässt sich diese Abfrage in der SQL-Ansicht anlegen (siehe Listing 2).

Ohne Filterung würde dieser Bericht dann alle Bestellungen aller Kunden ausgeben. Damit nur genau eine Bestellung zur Bestätigung kommt, ist der Filter gleich in das **Open**-Ereignis integriert, der einzigen Code-Zeile des Berichts:

```
Private Sub Report_
Open(Cancel As Integer)
    Me.Filter = "
     "BestellID=" & BestID
    Me.FilterOn = True
End Sub
```

Die **BestellID** wird mit dem Wert verglichen, der in der globalen Variablen **BestID** steht. Dieser Wert wurde nämlich beim Abschluss der Bestellung zugewiesen. Die Filterung im **Open**-Ereignis ist deshalb nötig, weil

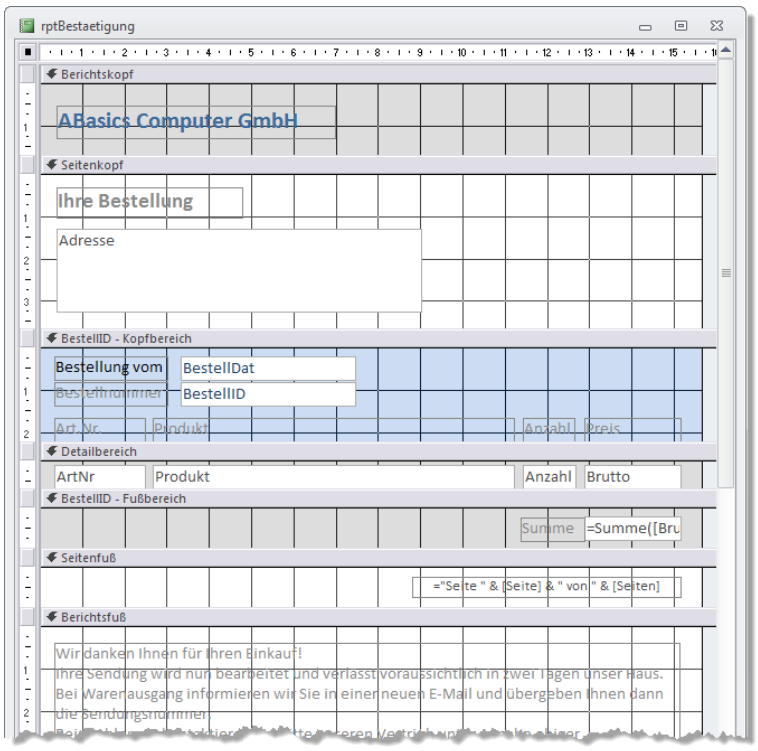


Bild 6: Der Aufbau des Bestätigungsberichts gleicht dem der Bestellhistorie

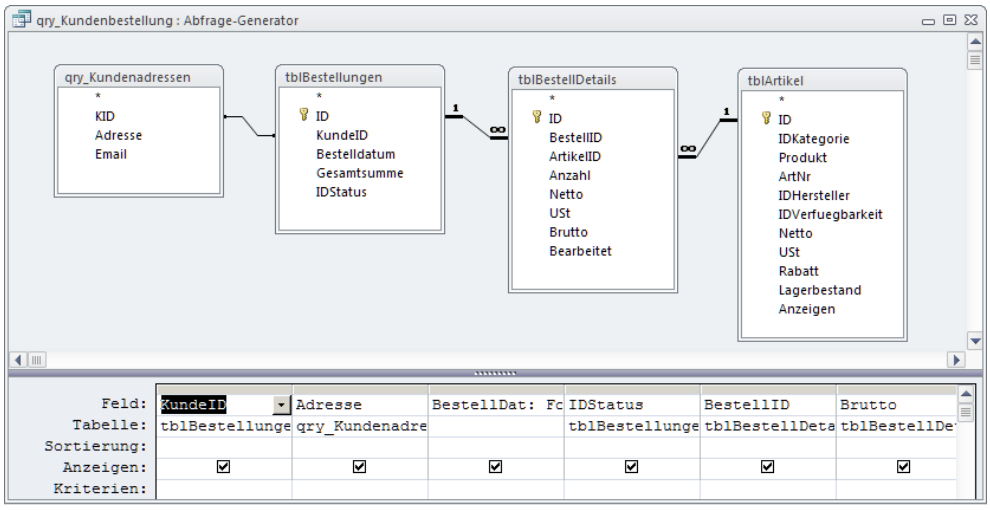


Bild 7: Die Datenherkunftsabfrage für den Bestätigungsbericht benötigt eine weitere Abfrage **qry_Kundenadressen**

```
SELECT
tblKunden.ID AS KID, [tblAnreden].[Anrede] & " " & [tblKunden].[Vorname] & " " & [tblKunden].[Nachname] & Chr$(13) & Chr$(10)
& [tblKunden].[Strasse] & " " & [tblKunden].[NrStrasse] & Chr$(13) & Chr$(10) & [tblKunden].[PLZ] & " " & [tblOrte].[Ort]
AS Adresse, tblKunden.Email FROM tblOrte INNER JOIN (tblAnreden INNER JOIN tblKunden ON tblAnreden.ID = tblKunden.IDAnrede)
ON tblOrte.ID = tblKunden.IDOrt;
```

Listing 2: Die Abfrage **qry_Kundenadressen** in der SQL-Ansicht

der Bericht über die Anweisung **SendObject** als E-Mail versandt wird.

Bearbeitung der Bestellung

Im Vertrieb werden die eingegangenen Bestellungen bearbeitet. Wir können hier kein komplettes Warenwirtschaftssystem abbilden. Unser Bearbeitungsformular **frmVertrieb** ist deshalb ziemlich einfach gestrickt und zeigt einfach zu jedem Kunden dessen Bestellungen an (siehe Bild 8).

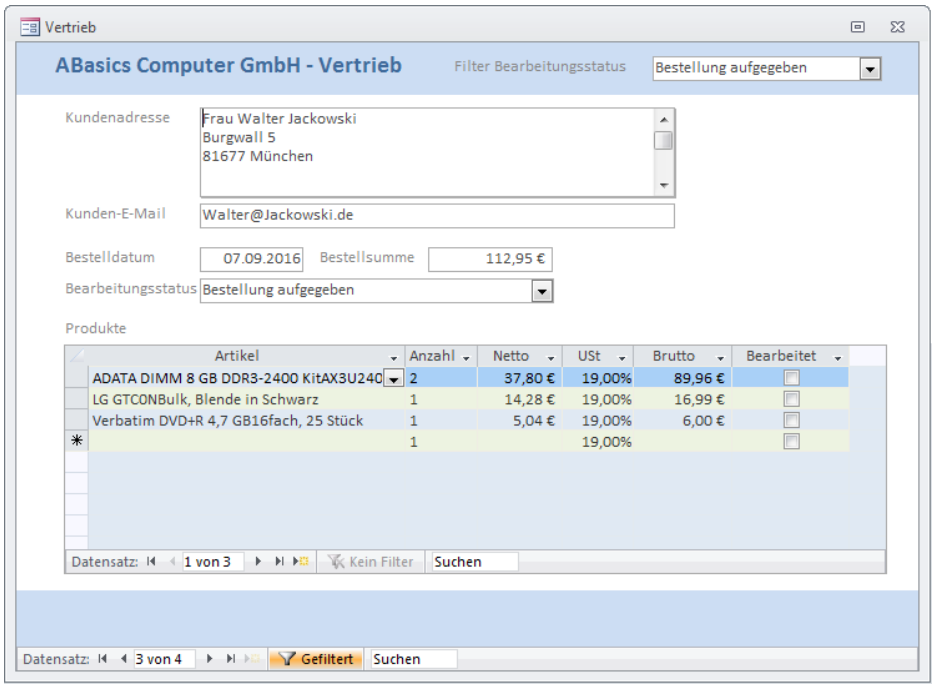


Bild 8: Der Vertrieb arbeitet die eingehenden Bestellungen über dieses Formular ab

Sie schalten über die Navigationsschaltflächen zwischen den Kunden und deren Bestellungen hin und her. Eine weitergehende Navigation existiert nicht. Die wesentlichen Daten stehen im Formularkopf, das im Detailbereich eingebaute Unterformular zeigt die Bestellpositionen.

Dennoch gibt es mit dem **Filter Bearbeitungsstatus** eine Möglichkeit im Formular, die Datenmenge einzuschränken. Aus dieser Combobox könnte der Vertriebler etwa den Status **Bestellung aufgegeben** auswählen, um nur jene Datensätze angezeigt zu bekommen, die aktuell bearbeitet werden müssen. Die Combobox führt im Ereignis **Nach Aktualisierung** (**After_Update**) diesen Code aus:

```
Me.Filter = "IDStatus=" & Me!cbFilterStatus.Value
Me.FilterOn = True
```

```
Private Sub cbStatus_AfterUpdate()
    If Me!cbStatus.Value = 2 Then
        If MsgBox("Soll die Versandbestätigung abgeschickt werden?", vbYesNo Or vbQuestion, _
                sLogo) = vbYes Then
            BestID = Me!ID.Value
            DoCmd.SendObject acSendReport, "rptVersand", acFormatPDF, _
                 Me!Email.Value, , , "Versandbestätigung", _
                "Sehr geehrter Kunde," & vbCrLf & vbCrLf & _
                "Ihre Bestellung bei uns wurde auf den Versandweg gebracht." & vbCrLf & _
                "Weitere Details finden Sie im Anhang." & vbCrLf & vbCrLf & _
                "Mit freundlichen Grüßen," & vbCrLf & _
                "ABasics Computer GmbH", True
            End If
    End If
End Sub
```

Listing 3: Absenden der PDF-Versandbestätigung bei Versandt-Status

Nun kann für einen Kunde die Ware kommissioniert werden, wonach im Unterformular für jede Bestellposition jeweils ein Häkchen bei **Bearbeitet** gesetzt wird.

Sind alle Waren beisammen und verpackt, so kann der **Bearbeitungsstatus** in der Combobox in der Mitte des Formulars auf **Bestellung versandt** eingestellt werden. Das ruft wiederum die Routine aus Listing 3 auf den Plan, welche mithilfe einer **Messagebox** nach Absenden der Versandbestätigung fragt.

Bei Bejahung wird zunächst der globalen Variablen **BestID** die **ID** der angezeigten Bestellung zugewiesen. Wir brauchen diesen Wert wieder bei Filterung des Versandberichts. Anschließend kommt es, analog zur Bestellbestätigung, zum Versenden des Berichts **rpt-Versand** über die **SendObject**-Methode des **DoCmd**-Objekts. Als Anhangsformat wird **acFormatPDF** gewählt, der Betreff auf **Versandbestätigung** eingestellt, die E-Mail-Adresse aus dem gleichnamigen Datenfeld entnommen und der Text der E-Mail hartkodiert aus Strings zusammengesetzt. Der im System installierte Standard-E-Mail-Client, wahrscheinlich **Outlook**, wird nun von Access aufgerufen und diese Daten übergeben. Das abschließende **True** der **SendObject**-Methode sagt aus, dass vor dem eigentlichen Absenden noch der E-Mail-Editor der Anwendung geöffnet werden soll. Das gibt Gelegenheit zur Kontrolle und gegebenenfalls Änderung des Textes.

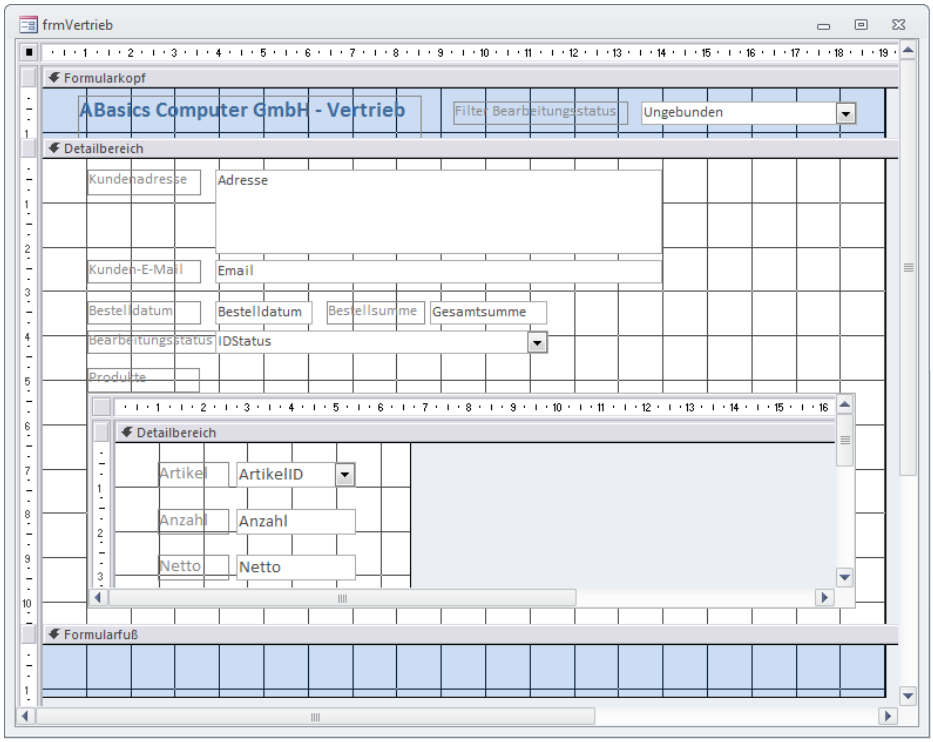


Bild 9: Das Formular **frmVertrieb** in der Entwurfsansicht mit Bestell-Unterformular

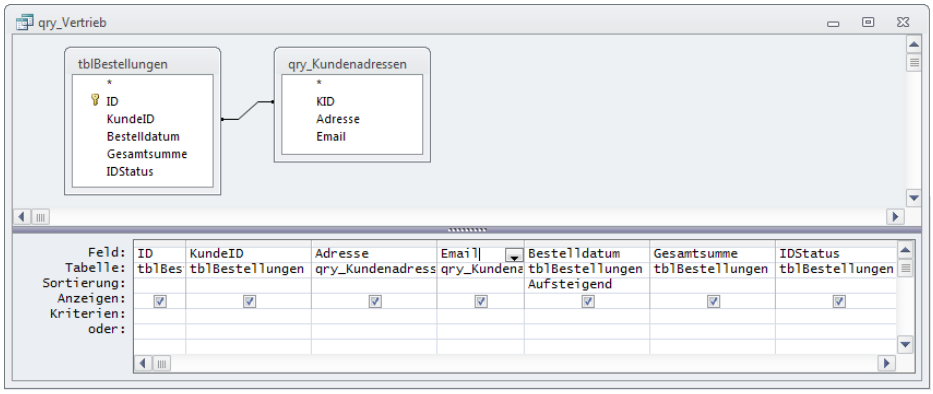


Bild 10: Die Abfrage **qry_Vertrieb** kombiniert Kundenadressen und Bestellungen

Bild 9 zeigt den Entwurf des Vertriebsformulars. Im Detailbereich sind sowohl die Felder der Adressdaten, wie auch das Unterformular für die Bestellpositionen eingebaut.

Die Datenherkunft ergibt sich aus der Auswahlabfrage **qry_Vertrieb** (siehe Bild 10), die nur Bestellungen und Kundenadressen miteinander verbindet.

Unter- und Hauptformular sind über die Felder **ID** der Bestellung und über **BestellID** der Bestellpositionen

Mit Binärzahlen arbeiten

Binärzahlen – da arbeiten doch nur Computer mit, oder auch mit Hexadezimalzahlen! Das ist doch uninteressant für mich! Mitnichten: Einige Grundkenntnisse im Bereich des Umgangs mit Binärzahlen haben noch keinem geschadet. Schon gar nicht, wenn man mit Access arbeitet und mit VBA programmiert. Dort ergibt sich doch das eine oder andere Mal Gelegenheit, mit einfachen Funktionen größere Probleme zu lösen – oder die Rechenoperationen wie die logische Konjunktion oder Disjunktion einzusetzen. Dieser Artikel liefert die wichtigsten Grundlagen zu diesem Thema.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1703_Binaerzahlen.accdb**.

Dezimalzahlen und Binärzahlen

Dezimalzahlen stammen aus dem Stellenwertsystem, mit dem wir täglich umgehen – dem Dezimalsystem. Die erste Stelle entspricht dem Einer, die zweite dem Zehner, die dritte dem Hunderter und so weiter. Dementsprechend gibt es für jede Stelle zehn verschiedene Ziffern von 0 bis 9. Anders sieht es beim Dualsystem oder Binärsystem aus. Hier gibt es nur zwei verschiedene Zahlen, nämlich 0 und 1. Die erste Stelle steht wie beim Dezimalsystem für den Einer, aber die zweite erfasst direkt die Zweier, die dritte die Vierer und so weiter. Die Zahl 5 im Dezimalsystem entspricht dann etwa der Zahl 101 im Binärsystem (ein Vierer, kein Zweier, ein Einer).

Konviertieren

Nun kommt es vor, dass Sie Zahlen im Binärsystem erhalten und diese gegebenenfalls in das Dezimalsytem umwandeln wollen oder umgekehrt. Dafür benötigen Sie kleine Funktionen, die Sie sich nicht selbst ausdenken müssen – wir stellen Sie ihnen in den folgenden Abschnitten vor.

Umrechnen von Dezimal zu Binär

Die Funktion zum Umrechnen von dezimalen Zahlen in binäre Zahlen heißt **Dec2Bin**. Sie erwartet einen **Long**-Wert und liefert einen String zurück:

```
Public Function Dec2Bin(lngDecimal As Long) As String
    Dim strBinaer As String
    Do While lngDecimal > 0
        If lngDecimal Mod 2 = 1 Then
            strBinaer = "1" & strBinaer
            lngDecimal = lngDecimal - 1
            lngDecimal = lngDecimal / 2
        Else
            strBinaer = "0" & strBinaer
            lngDecimal = lngDecimal / 2
        End If
    Loop
    Dec2Bin = strBinaer
End Function
```

Sie können diese Funktion beispielsweise mit den folgenden Aufrufen im Direktbereich des VBA-Editors ausprobieren:

```
? dec2bin(4096)
1000000000000

? dec2bin(4095)
111111111111
```

Die Funktion prüft zunächst, ob die übergebene Zahl beim Teilen durch den Wert **2** einen Rest von **1** liefert. Ist dies der Fall, besetzt sie schon einmal die ganz rechte Stelle der Binärzahl mit dem Wert **1**. Von der Zahl subtrahiert die Funktion dann den Wert **1**, da wir die **1** ja schon in **strBinaer** abgebildet haben.

Außerdem teilen wir die Zahl durch den Wert **2**, sodass wir im folgenden Durchlauf wieder prüfen können, ob die zweite Stelle der Binärzahl belegt werden kann. Dies ist im ersten Teil der **If...Then**-Bedingung

der Fall, der zweite Teil kümmert sich darum, dass beim Teilen durch den Wert **2** kein Rest übrigbleibt. In diesem Fall wird der Wert **0** vorn an die Binärzahl in **strBinaer** angehängt. Auf diese Weise durchläuft die Funktion alle Zweierpotenzen, trägt entweder 1 oder 0 vorn in strBinaer ein und teilt die Zahl nochmals, nachdem gegebenenfalls der Wert 1 subtrahiert wurde.

Wenn Sie die Funktion einmal im Debugging-Modus für einfache Werte wie 1, 2 oder 3 durchlaufen, wird der Ablauf deutlicher.

Umrechnen von binär zu dezimal

Die folgende Funktion namens Bin2Dec wandelt eine als Variant übergebene Binärzahl in eine Dezimalzahl um:

```
Public Function Bin2Dec(varBinaer As Variant) As Long
    Dim lngDecimal As Long
    Dim intPos As Integer
    Dim bytVal As Byte
    Dim i As Integer
    For i = Len(varBinaer) To 1 Step -1
        bytVal = Mid(varBinaer, i, 1)
        lngDecimal = lngDecimal + bytVal * 2 ^ intPos
        intPos = intPos + 1
    Next i
    Bin2Dec = lngDecimal
End Function
```

Die Aufrufe sehen wie folgt aus:

```
? Bin2Dec("11111111")
 255
? Bin2Dec("100000000")
 256
```

Da wir einen Variant als Eingangsparameter nutzen, können Sie die Binärzahl auch als Zahl übergeben:

```
? Bin2Dec(11111111)
 255
? Bin2Dec(100000000)
 256
```

Mit diesen beiden Funktionen können Sie nun von Binär nach Dezimal und umgekehrt konvertieren.

Mit Binärzahlen rechnen

Wenn Sie mit Binärzahlen rechnen, gelingt dies in schriftlicher Form genau wie bei den Dezimalzahlen – nur dass die Anzahl der Ziffern je Stelle auf zwei begrenzt ist. 1 plus 1 ergibt nicht 2, sondern 10. 10 plus 1 ergibt 11. 11 plus 1 ergibt 100 und so weiter. Die Addition oder auch die übrigen Rechenoperationen wie Subtraktion, Multiplikation oder Division sollen uns in diese Artikel nicht weiter interessieren. Viel spannender sind zwei Operationen, die sich logische Konjunktion und logische Disjunktion nennen. Die logische Konjunktion (**And**) liefert ein Ergebnis, bei dem zwei Binärzahlen stellenweise verglichen werden. Das Ergebnis für jede Stelle ist das Ergebnis eines Und-Vergleichs. Das Ergebnis der logischen Disjunktion (**Or**) entspricht dem Ergebnis eines stellenweisen Oder-Vergleichs.

Schauen wir uns ein Beispiel für einstellige Binärzahlen an, zunächst für die logische Disjunktion:

```
0 Or 0 = 0
0 Or 1 = 1
1 Or 0 = 1
1 Or 1 = 1
```

Für die logische Konjunktion folgen diese Ergebnisse:

```
0 And 0 = 0
0 And 1 = 0
1 And 0 = 0
1 And 1 = 1
```

Welche Stellen kommen vor?

Wir gehen einen Schritt weiter und betrachten größere Zahlen damit. Was kommt beispielsweise heraus, wenn wir die logische Konjunktion auf größere Zahlen im dezimalen Format anwenden – beispielsweise wie folgt?

```
? 21 And 7
 5
```

Transaktionen in Access

Wenn Sie Datenbankaktionen per Code ausführen, also beispielsweise mit der Methode **Execute** des **Database**-Objekts oder mit den DAO-Methoden AddNew oder Edit, setzen Sie in der Regel immer nur eine einzige Anweisung ab und ändern so Daten oder auch Tabellen. Was aber, wenn Sie mehrere Aktionen ausführen wollen, die nur dann ausgeführt werden sollen, wenn die Ausführung jeder einzelnen Aktion erfolgreich ist? Dann kommen Transaktionen ins Spiel. Hierbei handelt es sich um einen Kontext über eine oder mehrere Datenbankänderungen, der nach dem Ausführung aller Änderungen entweder komplett abgeschlossen oder verworfen werden kann.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1703_Transaktionen.accdb**.

Möglichkeiten der Änderung von Daten

Unter DAO gibt es verschiedene Möglichkeiten, per Code Daten zu ändern. Die erste Variante nutzt etwa die **AddNew**-Methode, um einen neuen Datensatz zu einem Recordset hinzuzufügen und schließt den Vorgang nach dem Festlegen der Feldinhalte mit der **Update**-Methode ab:

```
Public Sub Datenaenderungen()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("tblKategorien", dbOpen-
Dynaset)
    rst.AddNew
    rst!Kategoriename = "Neue Kategorie"
    rst.Update
End Sub
```

Hier gibt es keine offenen Fragen: Wenn keine Fehlermeldung erscheint, speichert die Prozedur den Datensatz wie gewünscht in der Datenbank. Ein Fehler tritt beispielsweise auf, wenn Sie zwei Mal den gleichen Wert für das Feld Kategoriename in der Datenbank speichern und den eindeutigen Index für dieses Feld verletzen. In diese Fall wird der Datensatz dann einfach nicht gespeichert.

Etwas anders sieht es aus, wenn Sie die Aufgaben mit der **Execute**-Methode ausführen und dazu eine entsprechende SQL-Abfrage formulieren. Ein ganz einfaches Beispiel sieht wie folgt aus:

```
Public Sub NeuerDatensatzExecute()
    Dim db As DAO.Database
    Set db = CurrentDb
    db.Execute "INSERT INTO tblKategorien
        (Kategoriename) VALUES('Noch eine Kategorie')"
End Sub
```

Auch hier gilt: Der Datensatz wird entweder eingetragen oder auch nicht. Über letzteres werden wir allerdings nicht informiert, denn wir haben den Wert **dbFailOnError** für den zweiten Parameter der **Execute**-Methode vergessen. Erst damit setzt es eine Fehlermeldung, wenn Sie beispielsweise versuchen, den gleichen Kategorienamen zwei Mal hintereinander einzufügen (siehe Bild 1).

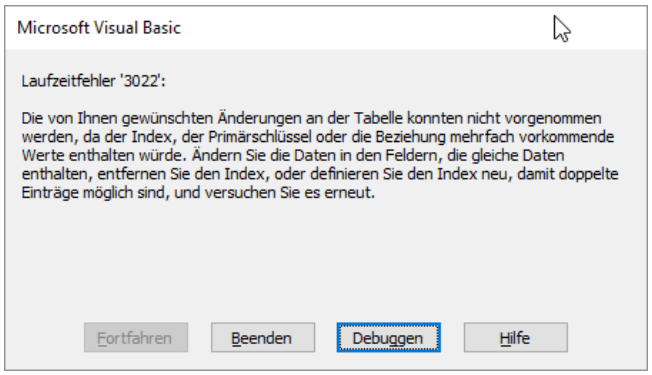


Bild 1: Fehlermeldung beim Anlegen eines bereits vorhandenen Kategorienamens

Probleme mit mehreren betroffenen Datensätzen

Nun schauen wir uns an, was passiert, wenn wir mit einer Aktionsabfrage gleich mehrere Datensätze ändern wollen – beispielsweise, indem wir versuchen, alle Einzelpreise in der Tabelle **tblArtikel** um 10 EUR zu senken:

```
Public Sub MehrereDatensaetze()
    Dim db As DAO.Database
    Set db = CurrentDb
    db.Execute "UPDATE tblArtikel SET Einzelpreis = _
                                    Einzelpreis - 10"
    Debug.Print "Betroffene Datensätze: " _
        & db.RecordsAffected
End Sub
```

Wir verwenden bewusst noch einmal die Variante der **Execute**-Methode ohnen den Wert **dbFailOnError**. Was geschieht? Es wird natürlich keine Fehlermeldung ausgelöst und wir erhalten im Direktfenster die folgende Ausgabe:

```
Betroffene Datensätze: 38
```

Das heißt, dass von den 77 Datensätzen dieser Tabelle nur 38 geändert wurden. Bei den übrigen hat die Gültigkeitsregel gegriffen, die besagt, dass nur Werte größer oder gleich **0** in das Feld eingegeben werden dürfen. Welche Erkenntnis ziehen wir daraus?

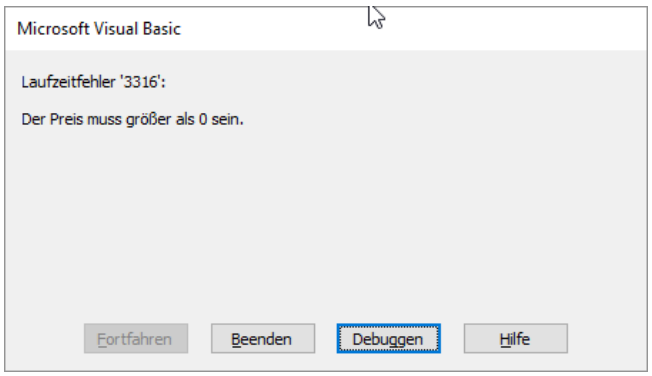


Bild 2: Fehlermeldung bei Verletzung der Gültigkeitsregel bei mindestem einem der betroffenen Datensätze

Aktionsabfragen, die sich auf mehrere Datensätze beziehen, werden unter Umständen nicht komplett ausgeführt.

Fügen wir dem Aufruf der **Execute**-Methode hingegen wieder den Wert **dbFailOnError** als zweiten Parameter hinzu, wird die Abfrage insgesamt nicht ausgeführt, sondern die Gültigkeitsmeldung erscheint (siehe Bild 2):

```
db.Execute "UPDATE tblArtikel SET Einzelpreis = Einzel-
preis - 10", dbFailOnError
```

Wenn Sie eine solche Abfrage über die Benutzeroberfläche über den Abfrageentwurf formulieren und starten, erhalten Sie eine Fehlermeldung wie in Bild 3. Hier erhalten Sie zwar keine genaue Angabe, warum

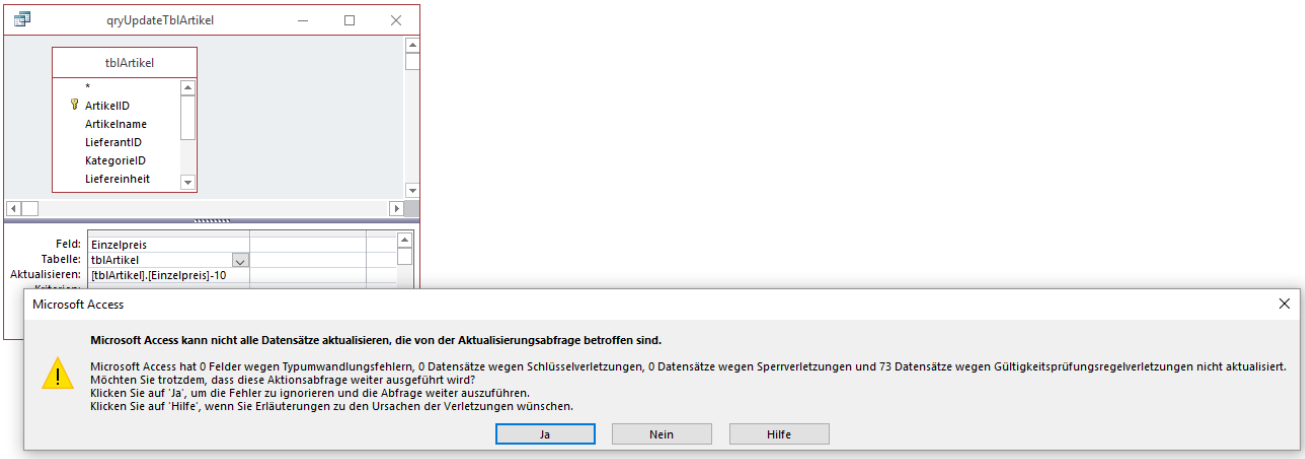


Bild 3: Fehlermeldung, wenn nicht alle Daten geändert werden können

die Daten nicht aktualisiert werden können, aber immerhin haben Sie die Möglichkeit, die Aktualisierung abzubrechen.

Sie sehen: Datenänderungen, die Sie nicht mit den DAO-Methoden **AddNew**, **Edit** und **Update** durchführen, sondern die Sie als SQL-Abfrage formulieren und mit der **Execute**-Methode ausführen, sind längst nicht in sich abgeschlossen und werden durchaus auch nur teilweise ausgeführt, wenn ein Teil der Änderungen nicht durchgeführt werden kann.

Hier können Sie auch mit Transaktionen nichts ändern – beim Einsatz der **Execute**-Methode müssen Sie zwingend mit dem Parameter **dbFailOnError** arbeiten, um beim Ausführen auftretende Fehler abzufangen.

Transaktionen in gespeicherten Abfragen

Wenn Sie eine Aktionsabfrage im Abfrageentwurf anlegen und diese speichern, können Sie im Eigenschaftsfenster der Abfrage die Eigenschaft **Transaktionen verwenden** einstellen (siehe Bild 4). Standardmäßig ist diese Option aktiviert. Das bedeutet, dass Access je nach dem Umfang der durchzuführenden Änderung Daten in eine temporäre Datenbank auslagert und wieder einliest. Wenn Sie **Transaktionen verwenden** auf **Nein** einstellen, wird keine temporäre Datenbank erstellt, was in der Regel zu einer schnelleren Ausführung beitragen sollte.

Transaktionen

An dieser Stelle kommen Transaktionen ins Spiel. Eine Transaktion ist prinzipiell ein Rahmen, in dessen Kontext eine oder mehrere Änderungen an den Daten der betroffenen Tabellen durchgeführt werden. Der Ablauf bezüglich der oben beschriebenen Fehlermeldungen ist der Gleiche, als wenn Sie die Anweisungen wie oben durchführen.

Der Unterschied ist: Sie können nach dem Durchführen der Änderungen festlegen, ob diese wirklich in die Tabellen geschrieben werden oder ob sie verworfen werden sollen. Sprich: Sie starten eine Transaktion, führen Änderungen an den Daten der Datenbank durch, prüfen, ob alles nach Ihren Wünschen geändert wurde und führen die Änderungen dann endgültig aus oder verwerfen diese.

Oben haben Sie gesehen, dass Sie eine einfache Aktualisierungsabfrage, die Sie per **Execute** ausführen, quasi in eine Transaktion umwandeln können, wenn Sie den Fehlerparamter **dbFailOnError** anhängen. Wenn Sie beispielsweise mehrere Abfragen hintereinander ausführen sollen und diese nur in die Datenbank geschrieben werden sollen, wenn alle erfolgreich waren, benötigen Sie eine echte Transaktion, wie Sie aber auch von Access zur Verfügung gestellt wird.

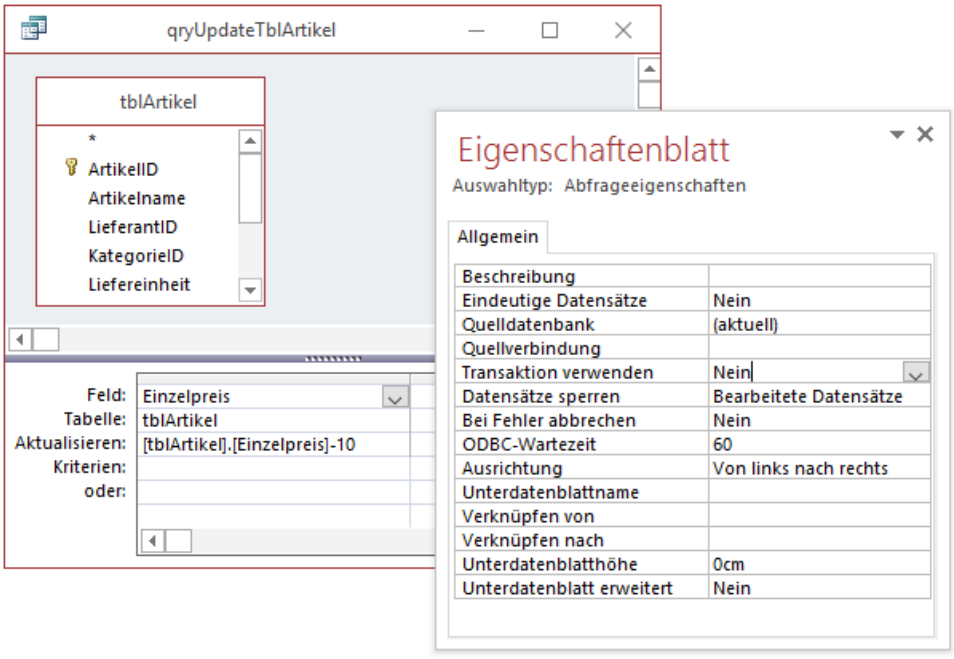


Bild 4: Die Eigenschaft **Transaktionen verwenden**

Selektion im Datenblatt

Im Listenfeld kann man mit den Einstellungen Mehrfach oder Erweitert auch leicht mehrere Datensätze auswählen. Was aber ist, wenn man diese Funktion in der Datenblattansicht bereitstellen will? Klar, anklicken kann man die einzelnen Datensätze, man kann auch mehrere zusammenhängende Datensätze markieren – aber was ist, wenn Sie beispielsweise den ersten, dritten und fünften Datensatz in der Datenblattansicht markieren wollen? Dann hilft nur ein spezieller Trick, den wir in diesem Artikel vorstellen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1703_SelektionImDatenblatt.accdb**.

Datenselektion speichern

Wenn Sie die Daten einer Tabelle in der Datenblattansicht anzeigen, gibt es verschiedene Möglichkeiten, einen oder mehrere Datensätze zu markieren. Die erste Variante ist die Markierung eines einzelnen Datensatzes (siehe links in Bild 1). Hier klicken Sie einfach für den gewünschten Datensatz auf den grauen Bereich links im Datenblatt, also auf den sogenannten Datensatzmarkierer.

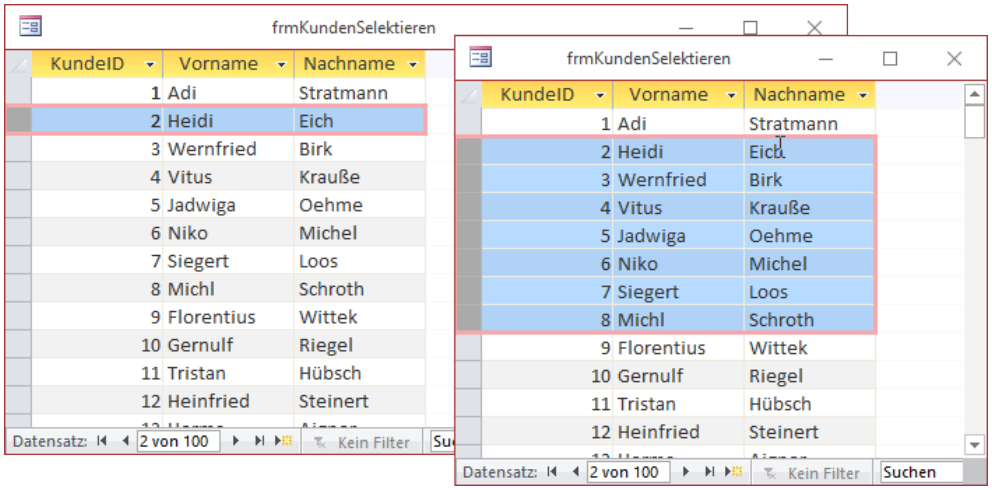


Bild 1: Beispiele für die Selektion von Datensätzen im Datenblatt

Die zweite Variante ist, mehrere zusammenhängende Datensätze zu markieren. Dazu markieren Sie zuerst den obersten Datensatz, halten dann die Umschalttaste gedrückt und markieren dann den unteren Datensatz. Das Ergebnis sehen Sie rechts im Bild.

Selektion abfragen

Um den oder die ausgewählten Datensätze abfragen zu können, fügen wir das Formular, das die Daten unserer Beispieltabelle in der Datenblattansicht anzeigt, als Unterformular in ein weiteres Formular ein. Im neuen Hauptformular legen wir dann ein paar Schaltflächen an, mit denen wir die selektierten Datensätze auslesen können (siehe Bild 2). Um einen einzelnen aktuell markierten Datensatz auszulesen, hinterlegen wir die folgende Prozedur für die Ereigniseigenschaft **Beim Klicken** der Schaltfläche **cmdSelektionAusgebenEinzeln**:

```
Private Sub cmdSelektionAusgebenEinzeln_Click()
    Dim sfm As Form
    Set sfm = Me!frmKundenSelektieren.Form
    MsgBox "Selektierte Kunden-ID: " & sfm!KundeID
End Sub
```

Wenn Sie nun bei gedrückter Umschalttaste mehr als einen Eintrag auswählen, liefert das Meldungsfenster die ID des zuerst ausgewählten Kunden. Wenn Sie also zuerst den Eintrag mit dem Wert **7** im Feld **KundeID** wählen und dann bei gedrückter Umschalttaste den Kunden mit der ID **3** anklicken, zeigt die Meldung den Wert **7** an.

Nun fügen wir eine zweite Schaltfläche hinzu, welche alle markierten Datensätze liefern soll. Hier gibt es nur ein Problem: Wenn wir in der Datenblattansicht einen oder mehrere Datensätze markierten und dann auf eine Schaltfläche klicken, wird die Markierung wieder gelöscht. Wir müssen also die Informationen zur Markierung speichern, bevor wir diese ausgeben. Dazu eignet sich, so dachten wir, am besten eine der Ereignisprozeduren, die beim Selektieren der Einträge im Datenblatt ausgelöst werden.

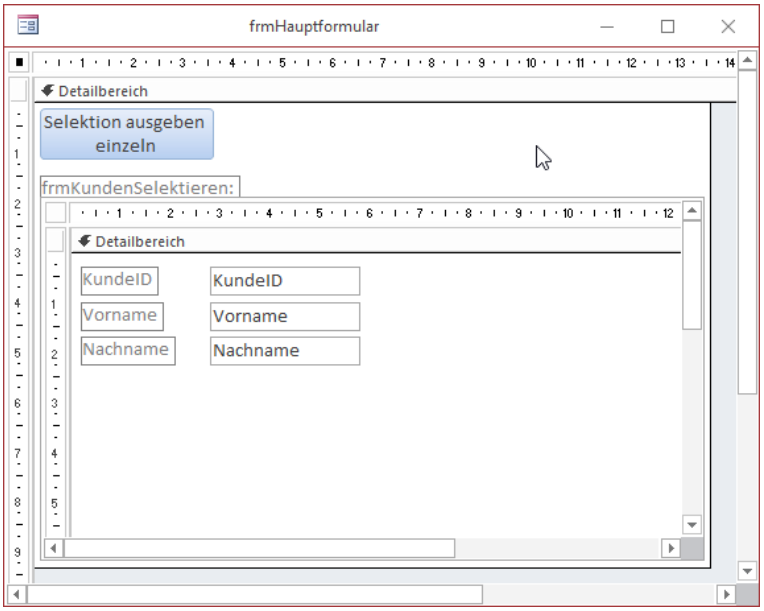


Bild 2: Formular zum Auswerten der aktuellen Selektion

Allerdings stellte sich heraus, dass es keine einfache Methode gab, auf das Selektieren eines Bereichs von Datensätzen zu reagieren. Was nun? Ein Ereignis, dass vor dem Anklicken der Schaltlfäche, aber nach dem Selektieren der Datensätze ausgelöst wird, ist das **Beim Verlassen**-Ereignis des Unterformular-Steuerelements. Und das trifft sich besonders gut, denn das Unterformular-Steuerelement ist ja ein Steuerelement des Hauptformulars, was bedeutet, dass wir das Ereignis auch gleich im Klassenmodul des Hauptformulars implementieren können.

Anderenfalls hätten wir ein Ereignis des Unterformulars verwenden müssen, was aus folgendem Grund ungünstig ist: Wir wollen ja die aktuelle Selektion des Unterformulars erfassen und möglichst gleich in entsprechenden Variablen im Hauptformular speichern, wo wir dann per Klick auf die Schaltlfäche etwas mit dem markierten Datensätzen erledigen. Mit einem Ereignis im Unterformular hätten wir die gesuchten Werte, welche die Selektion repräsentieren, erst noch irgendwie in das Hauptformular bekommen müssen. Das können wir uns nun sparen.

Welche Eigenschaften benötigen wir, um die Selektion auszuwerten? Dabei handelt es sich um die beiden Eigenschaften **SelTop** und **SelLength**. Langjährige Leser kennen diese beiden Eigenschaften vermutlich schon vom Textfeld-Steuerelement – dort wurde mit ähnlichen Eigenschaften (**SelStart** und **SelLength**) der markierte Text ermittelt. Im Falle der Datenblattansicht liefert **SelTop** den Index der obersten markierten Spalte und **SelLength** liefert die Anzahl der selektierten Zeilen. Um diese beiden Werte im Klassenmodul des Hauptformulars speichern zu können, legen wir die folgenden beiden Variablen im Kopf des Moduls an:

```
Dim intSelHeight As Integer
Dim intSelTop As Integer
```

Damit diese beim Verlassen des Unterformular-Steuerelements gefüllt werden, legen wir für das Ereignis **Bei Verlassen** dieses Elements die folgende Ereignisprozedur an:

```
Private Sub frmKundenSelektieren_Exit(Cancel As Integer)
    intSelHeight = Me!frmKundenSelektieren.Form.SelHeight
    intSelTop = Me!frmKundenSelektieren.Form.SelTop
End Sub
```

Wenn wir nun auf die Schaltfläche **cmdSelektionAusgebenAlle** klicken, wollen wir die folgende Ereignisprozedur auslösen:

```
Private Sub cmdSelektionAusgebenAlle_Click()
    MsgBox "Erste markierte Zeile: " & intSelTop _
```

```
        & vbCrLf _
        & "Anzahl markierter Zeilen: " & intSelHeight
End Sub
```

Diese gibt dann einfach die Werte der beiden Variablen **intSelTop** und **intSelHeight** per Meldungsfenster aus.

Von der Position zum Datensatz

Nun wollen wir allerdings nicht nur auf die jeweiligen Zeilen der Datenblattansicht zugreifen, sondern auch auf die darin enthaltenen Daten. Das erledigen wir, indem wir die Prozedur **cmdSelektionAusgebenAlle_Click** wie in Listing 1 erweitern. Dazu fügen wir ein **Form**-, ein **Recordset**- und eine **Integer**-Variable hinzu. Die Form-Variable referenziert das Unterformular mit der Datenblattansicht.

```
Private Sub cmdSelektionAusgebenAlle_Click()
    Dim sfm As Form
    Dim rst As dao.Recordset
    Dim i As Integer
    Set sfm = Me!frmKundenSelektieren.Form
    Set rst = sfm.RecordsetClone
    For i = intSelTop - 1 To intSelTop + intSelHeight - 2
        rst.AbsolutePosition = i
        Debug.Print rst!KundeID
    Next i
End Sub
```

Listing 1: Ausgeben der Werte des Feldes **KundeID** für alle selektierten Datensätze des Datenblatts

Das **Recordset**-Objekt füllen wir mit einem Verweis auf den **RecordsetClone** des Recordsets des Formulars. Warum **RecordsetClone**? Warum greifen wir nicht direkt auf das Recordset zu? Weil wir im Folgenden die markierten Datensätze des Unterformulars durchlaufen wollen, die Position des Datensatzzeigers im Datenblatt aber nicht ändern wollen.

Mit dem **RecordsetClone** holen wir uns eine Kopie, die wir nach Lust und Laune durchlaufen können, ohne dass es die Darstellung des Datenblatts beeinflusst. Die Prozedur durchläuft dann in einer **For... Next**-Schleife alle Werte von der ersten bis zur letzten Position bezogen auf die markierten Datensätze. Da **SelTop** für die oberste Zeile den Wert **1** liefert, müssen wir diesen noch um **1** vermindern, da wir über die Eigenschaft **AbsolutePosition** auf die im Datenblatt angezeigten Datensätze zugreifen wollen.

Und diese Eigenschaft wiederum erwartet die Position als null-basierten Wert. Nachdem wir die Datensatzzeiger-Position des RecordsetClones auf die richtige Position eingestellt haben, können wir mit **rst!KundeID** leicht auf den Primärschlüsselwert des aktuellen Eintrags der Markierung zugreifen.

Wir könnten nun noch die Markierung im Unterformular wieder herstellen, allerdings müssten wir dann auch noch den Fokus zurück auf dieses Element verschieben, damit die Markierung wieder sichtbar ist. Dies können wir aber ebenfalls per Code erledigen, indem wir die Prozedur cmdSelektionAusgebenAlle_Click nochmals erweitern:

```
Private Sub cmdSelektionAusgebenAlle_Click()
    ...
    Me.frmKundenSelektieren.SetFocus
    sfm.SelTop = intSelTop
    sfm.SelHeight = intSelHeight
    sfm.SelWidth = 99
End Sub
```

Wir müssen hier allerdings nicht nur die oberste Position und die Höhe der Markierung einstellen, sondern auch die Breite – sonst wird nur die erste Spalte für die betroffenen Zeilen markiert.

Markierung nicht zusammenhängender Zeilen

Damit haben wir die beiden leichteren Varianten der Markierung im Datenblatt erledigt, nämlich die Markierung eines einzelnen Eintrag und die Markierung mehrerer zusammenhängender Einträge. Dies wird nicht mit eingebauten Mitteln möglich sein – Access erlaubt schlicht und einfach nicht das Markieren nicht zusammenhängender Zeilen in der Datenblattansicht.

Also müssen wir uns mit ein paar Tricks behelfen. Als Erstes benötigen wir eine Möglichkeit, irgendwie abzuspeichern, ob ein Datensatz markiert ist oder nicht. Dazu erweitern wir die Tabelle **tblKunden** einfach um ein Feld namens **Selektiert** mit dem Datentyp **Ja/Nein** (siehe Bild 3). Natürlich können Sie das nicht machen, wenn die Tabelle beispielsweise von einer SQL Server-Datenbank stammt oder von einem Backend, dessen Entwurf Sie nicht ändern können. Für diesen Fall gibt es Alternativen, die wir später besprechen können.

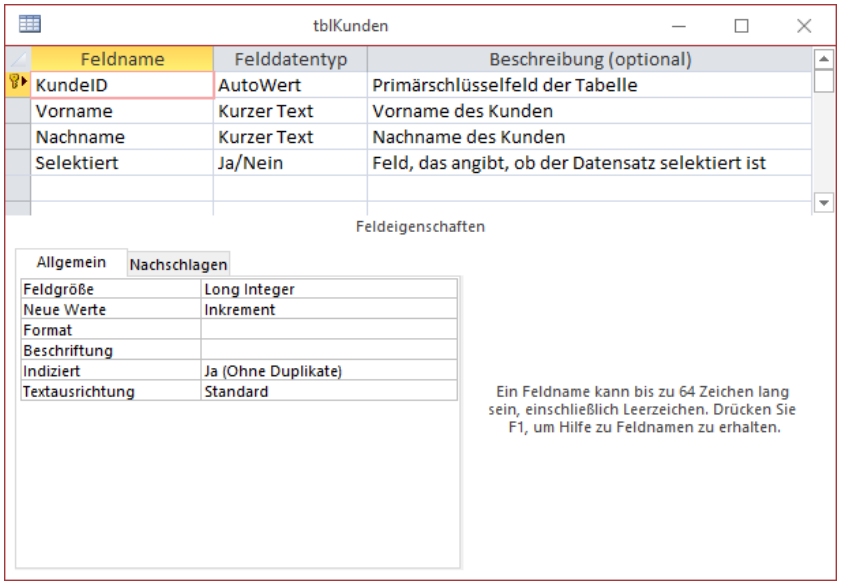


Bild 3: Tabelle der Beispieldatenbank

Markierung speichern

Im zweiten Schritt müssen wir dafür sorgen, dass dieses **Ja/Nein**-Feld auf den Wert **Ja** eingestellt wird, wenn wir das erste Mal auf einen Eintrag klicken und auf **Nein**, wenn wir diesen erneut betätigen. Wir wollen uns hier auf das Anklicken des Datensatzmarkierers beschränken, damit wir nicht entsprechende Ereignisprozeduren für alle Steuerelemente der Datenblattansicht implementieren müssen.

Für dieses neue Beispiel kopieren wir die beiden bereits verwendeten Formulare in neue Formulare, deren Namen wir um **...Mehrfachauswahl** erweitern. Außerdem fügen wir dem Unterformular noch das Feld **Selektiert** hinzu (siehe Bild 4).

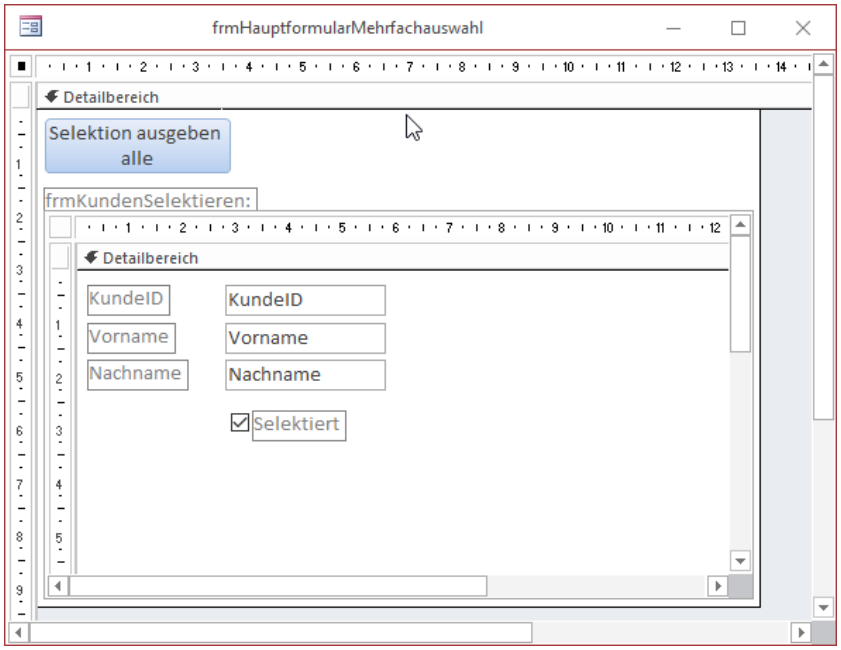


Bild 4: Entwurf für das zweite Beispiel mit Mehrfachauswahl

Für die Ereignisprozedur **Beim Klicken** des Formulars selbst hinterlegen wir die Ereignisprozedur aus Listing 2. Diese erstellt ein **Database**-Objekt auf Basis der aktuellen Datenbank, ermittelt den Wert des Feldes **KundeID** für den angeklickten Datensatz und stellt den Wert des Feldes **Selektiert** für diesen Datensatz auf den Wert **Ja** oder **Nein** ein – je nachdem, welchen Wert das Feld vorher hatte.

Auf diese Weise sehen wir nun schon einmal, dass wir einzelne Datensätze durch Anklicken des Datensatzmarkierers als selektiert oder nicht selektiert markieren können (siehe Bild 5).

Nun wollen wir noch einen Weg finden, um die Selektion anders als durch das Kontrollkästchen der Spalte **Selektiert** anzuzeigen.

```
Private Sub Form_Click()
    Dim lngKundeID As Long
    Dim db As DAO.Database
    Set db = CurrentDb
    lngKundeID = Me!KundeID
    db.Execute "UPDATE tblKunden SET Selektiert = NOT Selektiert WHERE KundeID = " & lngKundeID, dbFailOnError
End Sub
```

Listing 2: Aktualisieren der Daten für alle zu markierenden Datensätze

Selektion per bedingter Formatierung hervorheben

Hier greifen wir nun auf eine Funktion der neueren Version von Access zurück: die bedingte Formatierung. Damit können wir festlegen, unter welcher Bedingung ein oder mehrere Felder mit einer bestimmten Formatierung versehen werden. In diesem Fall wollen wir dafür sorgen, dass die markierten Datensätze, also solche mit dem Wert **Ja** im Feld **Selektiert**, auch entsprechend hervorgehoben werden, damit wir dieses Feld ausblenden können.

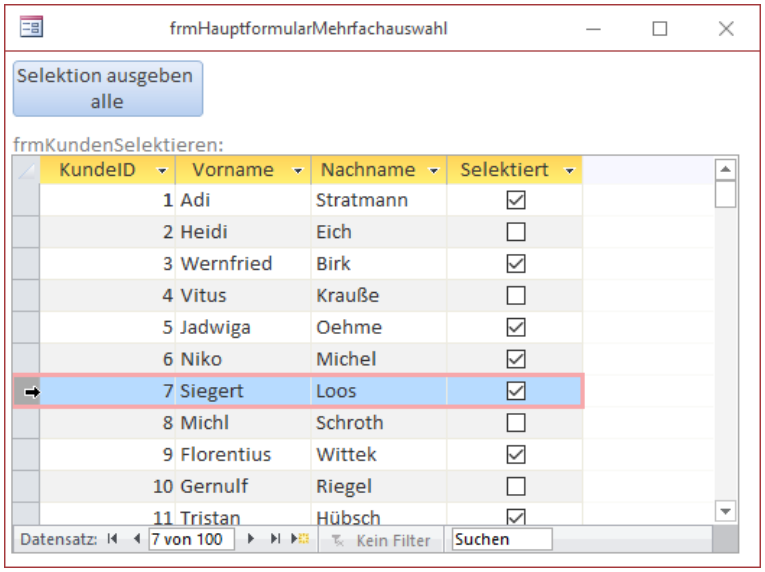


Bild 5: Das Aus- und Abwählen einzelner Einträge funktioniert bereits.

Also klicken Sie, während sich das Formular noch in der Formularansicht befindet, in das Datenblatt des Unterformulars und betätigen dann den Befehl **Datenblatt|Formatierung|Bedingte Formatierung** des Ribbons. Es erscheint der Dialog aus Bild 6, wo Sie auf die Schaltfläche **Neue Regel** klicken.

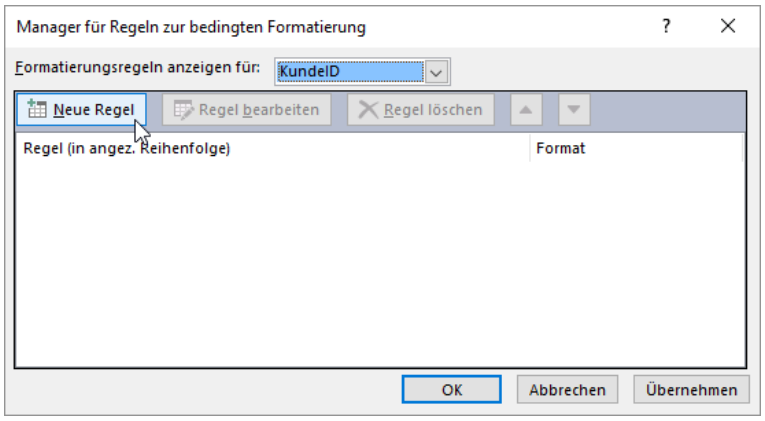


Bild 6: Hinzufügen einer neuen Formatierungsregel

Danach finden Sie den Dialog aus Bild 7 vor. Hier behalten Sie den angegebenen Regeltyp bei, wählen im ersten Kombinationsfeld aber den Wert **Ausdruck ist** aus. Dadurch erscheint rechts ein leeres Textfeld, in das Sie den als Bedingung zu verwendenden Ausdruck eingeben. Dieser lautet:

```
[Selektiert] = -1
```

Das heißt, dass die nachfolgend festgelegten Formatierungen nur angewendet werden, wenn das Feld **Selektiert** den Wert **-1** aufweist. Damit legen wir nun noch die gewünschte Formatierung an, beispielsweise eine hellblaue Hintergrundfarbe wie in der Abbildung.

Schließen Sie diesen Dialog nun und haben bereits einige Datensätze mit dem Wert **Ja** für das Feld

Individuelle Auswahl mit eigener Tabelle

In den beiden Artikeln Datensätze individuell auswählen (Ausgabe 10/2012) und Selektion im Datenblatt (aktuelle Ausgabe) nutzen wir ein zusätzliches Feld in der betroffenen Tabelle, um Datensätze individuell auswählen zu können. Die Methode ist sehr praktisch, kommt jedoch an ihre Grenzen, wenn Sie keinen Zugriff auf die betroffene Tabelle haben. Das kann passieren, wenn sich diese in einem Backend befindet, dessen Daten Sie nicht ändern können, weil Sie keinen Zugriff haben oder keine Berechtigung. Ein weiterer wichtiger Grund, die Selektionsdaten in einer eigenen Tabelle zu speichern, sind Mehrbenutzeranwendungen. Da bietet es sich an, die Daten bezüglich der Ansicht im Frontend zu speichern.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1703_IndividuelleAuswahl.accdb**. Wir setzen auf der im Artikel **Selektion im Datenblatt** beschriebene Lösung auf. Dabei entfernen wir einfach das Feld Selektiert aus der Tabelle tblKunden und fügen dieses dann über eine zusätzliche Tabelle wieder hinzu. Die übrigen Schritte werden in diesem Artikel erläutert.

Tabelle hinzufügen

Wir gehen also nun davon aus, dass die Tabelle **tblKunden** sich in einer Backend-Datenbank befindet, deren Entwurf wir nicht ändern können. Daher entfernen wir das Feld **Selektiert** aus dieser Tabelle und speichern die Tabelle.

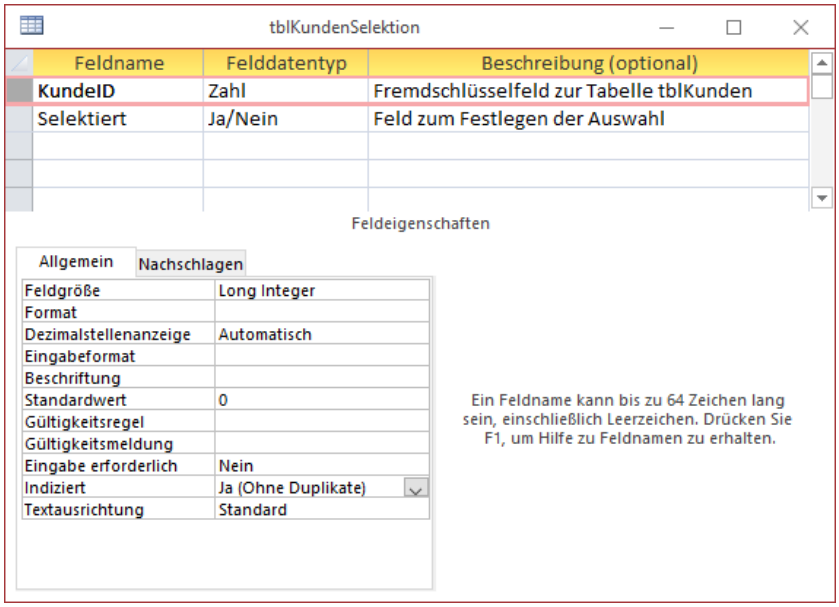


Bild 1: Die Tabelle mit dem Selektionsfeld

Die Tabelle, in der wir die Selektionsdaten speichern wollen, soll den Namen **tblKundenSelektion** erhalten und zwei Felder enthalten. Das erste heißt **KundeID** und wird als Zahlenfeld mit einem eindeutigen Index versehen. Das zweite Feld ist das Feld, das wir aus der ursprünglichen Tabelle entnommen haben – es heißt dementsprechend **Selektiert** (siehe Bild 1).

Damit wir das Feld **Selektiert** sauber mit der Tabelle **tblKunden** verbinden können, müssen wir eine 1:1-Beziehung zwischen den beiden Tabellen anlegen. Das bedeutet, dass es für jeden Datensatz der Tabelle **tblKunden** nur maximal einen Datensatz in der Tabelle **tblKundenSelektion** geben darf, dessen Wert im Feld **KundeID** mit dem Feld **KundeID** in **tblKunden** übereinstimmt.

Dazu stellen wir die Verbindung zwischen den beiden Tabellen her, und zwar im Beziehungen-Fenster von Access. Diesem fügen Sie die beiden Tabellen **tblKunden** und **tblKundenSelektion** hinzu. Anschließend ziehen Sie den Beziehungspfeil vom Feld **KundeID** der Tabelle **tblKunden** auf das gleichnamige Feld der Tabelle **tblKundenSelektion** und legen Sie für

die Beziehung **Referenzielle Integrität** und **Löschweitergabe** fest (siehe Bild 2). Versuchen Sie dies andersherum, erhalten Sie beim Versuch, referenzielle Integrität festzulegen eine Fehlermeldung. Hier geht es um die Richtung, in der die Beziehung angelegt wird: Access prüft im letzteren Fall, der einen Fehler auslöst, ob die Tabelle **tblKunden** Datensätze mit Werten im Feld **KundeID** enthält, die nicht in der Tabelle **tblKundenSelektion** enthalten sind. Und das ist zu diesem Zeitpunkt definitiv der Fall, da wir die Tabelle **tblKundenSelektion** ja soeben erst angelegt haben.

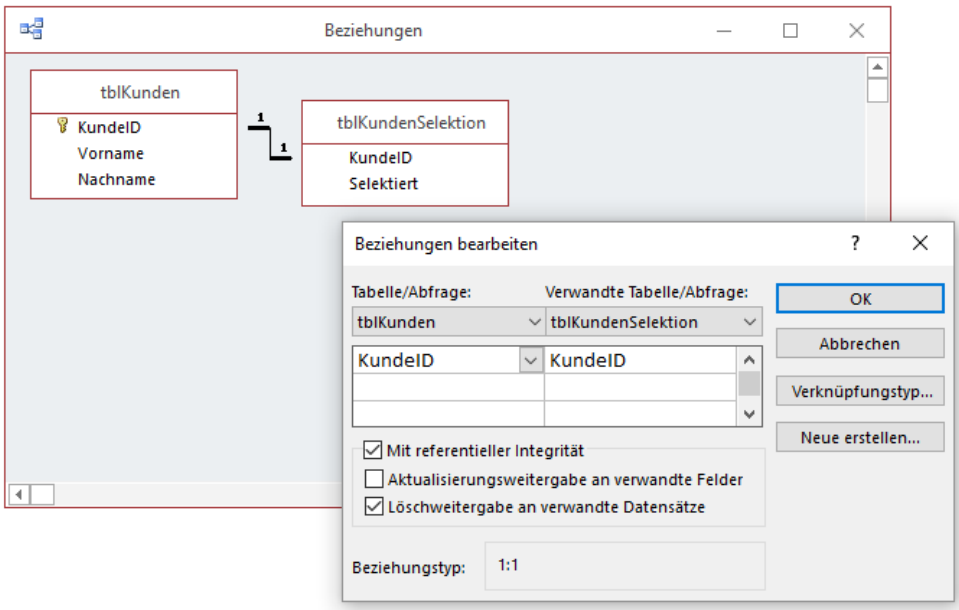


Bild 2: Herstellen der Beziehung zwischen den beiden Tabellen

In der Abbildung erkennen Sie auch, dass Access keine 1:n-Beziehung angelegt hat, wie dies üblicherweise der Fall ist, sondern eine 1:1-Beziehung. Woran erkennt Access dies? Ganz einfach: Wir haben auf beiden Seiten Felder, für die ein eindeutiger Index festgelegt ist. Und wenn keines der Felder einen Wert mehr als einmal enthalten kann, dann kann man zwischen diesen Feldern auch keine 1:n-Beziehung anlegen.

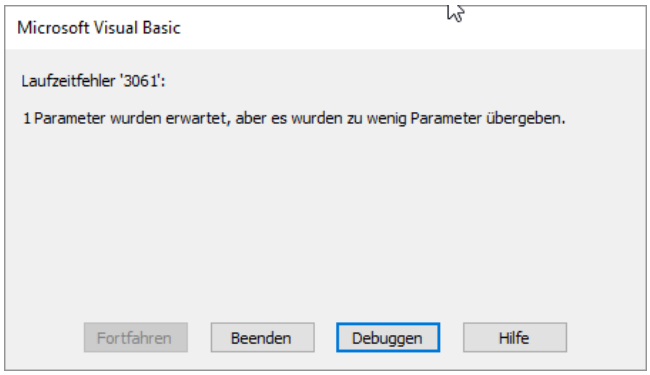


Bild 3: Fehler beim Versuch, auf das Feld **Selektiert** zuzugreifen

Sie können nun testweise einen Datensatz in der Tabelle **tblKundenSelektion** anlegen, der in der Tabelle **KundeID** einen Wert enthält, der auch in der Tabelle **tblKunden** vorkommt. Wenn Sie im Feld **KundeID** der Tabelle **tblKundenSelektion** einen Wert anlegen, der nicht im Feld **KundeID** der Tabelle **tblKunden** vorkommt, löst dies einen Fehler aus.

Und wozu benötigen wir die Löschtweitergabe? Diese sorgt dafür, dass wenn ein Datensatz in der Tabelle **tblKunden** gelöscht wird, auch ein eventuell mit diesem Datenstaz der Tabelle **tblKunden** verknüpfter Datensatz der Tabelle **tblKundenSelektion** gelöscht wird. Auf diese Weise verhindern wir, dass die Tabelle Karteileichen speichert.

Anpassung der zugreifenden Instanzen

Nun haben wir zwar das Feld **Selektiert** aus der Tabelle **tblKunden** in die Tabelle **tblKundenSelektion** ausgelagert und eine Beziehung zwischen den beiden Tabellen hergestellt – dies hilft uns aber nicht weiter, wenn wir nun über das Formular auf das Feld **Selektiert** zugreifen wollen. Ganz im Gegenteil: Wenn wir das Formular frmHauptformularErweitert der Beispieldatenbank öffnen, erhalten wir den Fehler aus Bild 3. Dieser tritt auf, wenn wir in der Ereignis-

Access-Add-Ins programmieren

Sicher haben Sie im Laufe der Zeit schon einmal ein Add-In benutzt. Diese tauchen zu verschiedenen Gelegenheiten auf: Beim Erstellen von Tabellen (nicht mehr in aktuellen Access-Versionen) oder Abfragen, beim Entwurf von Formularen und Berichten, beim Hinzufügen von Steuerelementen oder auch global als Menü-Add-In. In diesem Artikel zeigen wir Ihnen, wie Sie selbst den Grundstein für die Erstellung eines Add-Ins für Ihre eigenen Zwecke legen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **AccessAddIn.accda**.

Beispiele für eingebaute Add-Ins

Wenn Sie mit Access arbeiten, haben Sie vermutlich, zumindest zu Beginn, reichlich Gebrauch von den zahlreichen eingebauten Add-Ins von Access gemacht.

Da gibt es beispielsweise die Abfrage-Assistenten, die erscheinen, wenn Sie den Ribbon-Befehl **Erstellen|Abfragen|Abfrage-Assistent** bedienen. Es erscheint dann das Formular aus Bild 1, welches die verfügbaren Assistenten anzeigt.

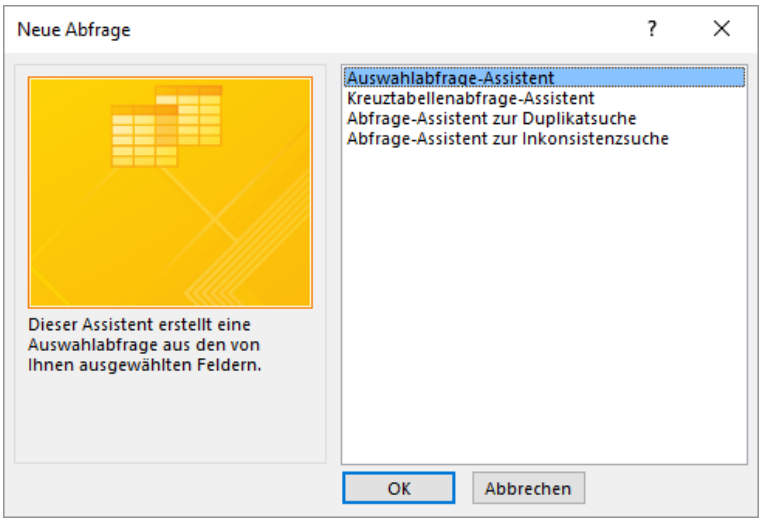


Bild 1: Übersicht der Abfrageassistenten

Diese dienen beispielsweise zum Erstellen von Auswahlabfragen, Kreuztabellenabfragen, Abfragen zur Duplikatsuche oder zur Inkonsistenzsuche.

Oder Sie wählen den Ribbon-Eintrag **Erstellen|Formulare|Formular-Assistent** aus, der den Dialog aus Bild 2 öffnet. Diese bietet keine Auswahl verschiedener Assistenten an, sondern öffnet gleich den Formular-Assistent, mit dem Sie beispielsweise die Datenherkunft auswählen und weitere Schritte durchführen.

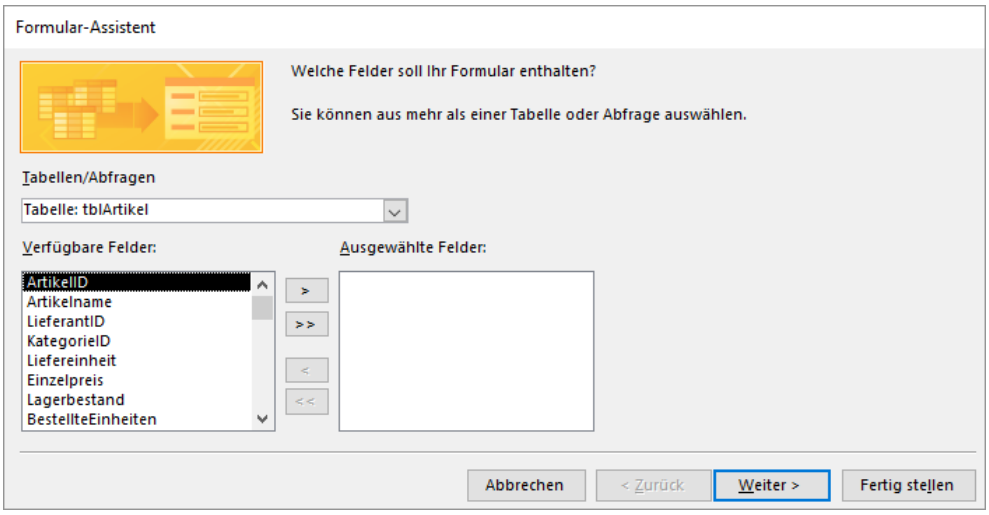


Bild 2: Der Formular-Assistent von Access

Wenn Sie einmal ein Formular oder einen Bericht in der Entwurfsansicht geöffnet haben, ist die erste Aufgabe meist das Erstellen von Steuerelementen. Auch hierbei erhalten Sie Unterstützung von verschiedenen Assistenten. Diese müssen

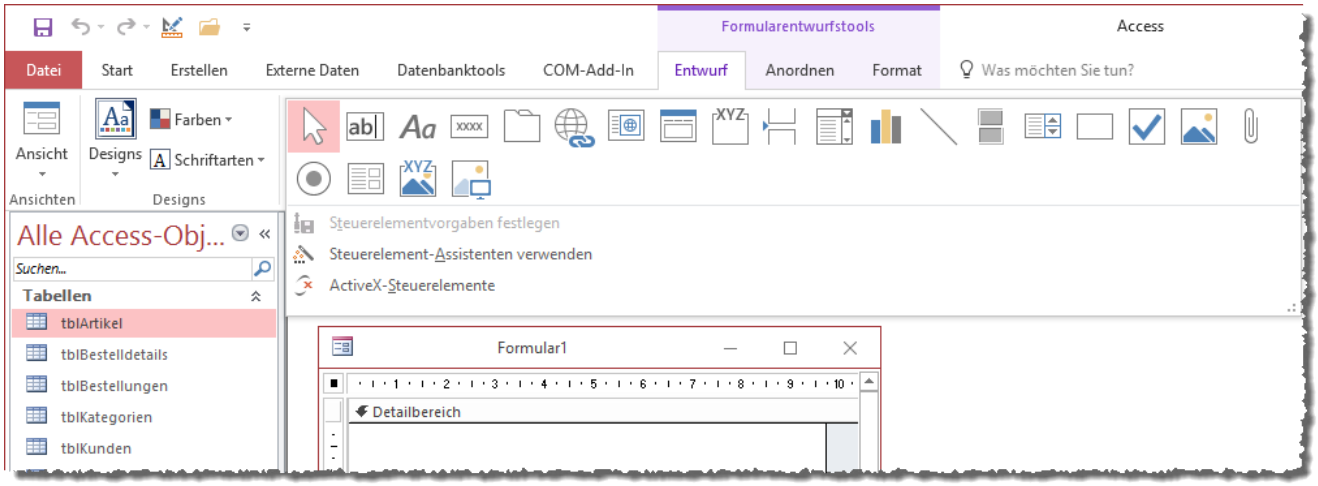


Bild 3: Aktivieren des Steuerelement-Assistenten

Sie allerdings erst aktivieren, indem Sie im Ribbon die Liste unter den Steuerelement aufklappen und, sofern noch nicht geschehen, den Eintrag **Steuerelement-Assistenten verwenden** aktivieren.

Wenn Sie danach beispielsweise die Schaltfläche im Ribbon markieren und dann auf die Stelle im Formularentwurf klicken, an der Sie diese platzieren wollen, erscheint der Befehlsschaltflächen-Assistent aus Bild 4, der Sie beim Anlegen der gewünschten Schaltfläche unterstützt.

Solche Assistenten gibt es auch noch für viele weitere Steuerelemente.

Darüber hinaus gibt es noch Eigenschafts-Assistenten, die erscheinen, wenn Sie auf eine Eigenschaft klicken und dann auf die Schaltfläche mit den drei Punkten klicken, sofern diese verfügbar ist.

Für die Steuerelement- und Eigenschaftsassistenten gilt, dass immer, wenn es mehr als einen Assistenten für das Steuerelement oder die Eigenschaft gibt, eine

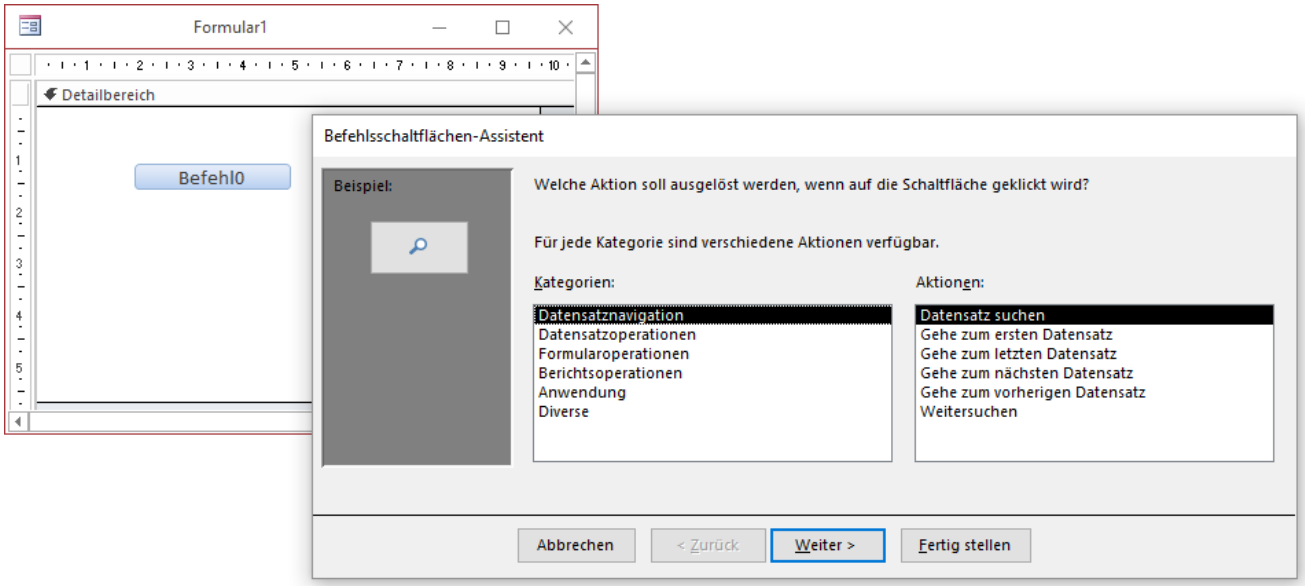


Bild 4: Unterstützung durch den Befehlsschaltflächen-Assistenten

Auswahlliste der verfügbaren Assistenten erscheint oder, wenn es nur einen Assistenten gibt, dieser direkt gestartet wird.

Schließlich gibt es noch die allgemeineren Add-Ins, die früher allesamt in einem Add-Ins-Menüpunkt zu finden waren, aber seit der Einführung des Ribbons an anderen Stellen im Ribbon untergebracht wurden. Es gibt allerdings immer noch einen Add-In-Menüpunkt, den Sie im Ribbon unter **Datenbanktools|Add-Ins** finden. Meine Access-Installation beherbergt an dieser Stelle einige Add-Ins, so zum Beispiel den Ribbon-Admin und einige weitere Assistenten, die mir das Leben bei Entwickeln von Access-Datenbanken erleichtern (siehe Bild 5). In diesem Menü finden Sie in der Basisinstallation zumindest den Eintrag **Add-In-Manager**, mit dem Sie die installierten Add-Ins verwalten können.

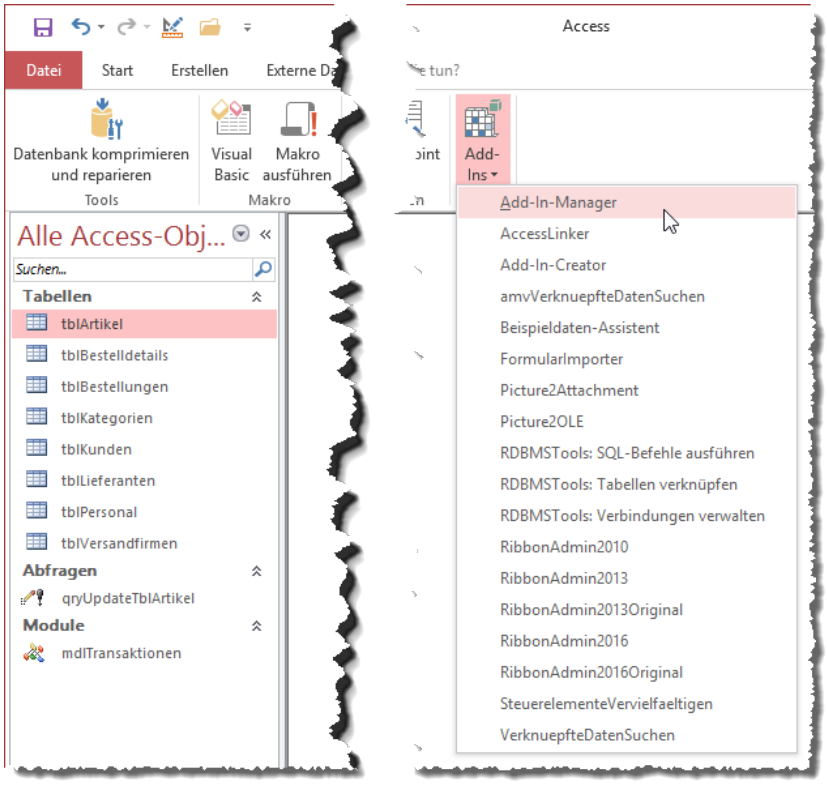


Bild 5: Das Add-In-Menü mit einigen benutzerdefinierten Einträgen

Eigene Add-Ins hinzufügen

Nun fragen Sie sich, wo Access doch schon eine Menge eingebauter Assistenten anbietet, wo Sie noch eigene Assistenten hinzufügen sollen. Der erste Ansatzpunkt dürfte das Add-In-Menü sein, wo Sie global verfügbare Add-Ins unterbringen können. Ein Blick in die Add-In-Liste meiner Installation liefert ein paar Anregungen: Der Access-Linker beispielsweise unterstützt Sie, eine neue Verknüpfung zur aktuellen Access-Datenbank zu erstellen.

Der Add-In-Creator hilft, Add-Ins zu erstellen – also bei den Schritten, die Sie in diesem Artikel zunächst manuell durchführen. Sehr oft nutze ich den Beispieldaten-Assistent, der die Auswahl einer Tabelle und das Füllen der Spalten dieser Tabelle nach verschiedenen Vorlagen erledigt. Auch den Formular-Importer nutze ich sehr oft: Mit diesem speichere ich oft verwendete Formular wie etwa ein Spash-Screen für Beispieldatenbanken und füge es dann per Mausklick zu einer Datenbank hinzu – natürlich inklusive Füllen von vordefinierten Platzhaltern.

Nach der Lektüre des vorliegenden Artikels haben Sie die Grundlagen, um eigene Add-Ins zu erstellen. Dabei kann es sich allerdings auch um Add-Ins der anderen Typen handeln, also Abfrage-, Formular- oder Berichts- und Steuerelement-Assistenten.

Erstellen einer leeren Add-In-Datenbank

Der erste Schritt ist der einfachste: Sie müssen einfach eine herkömmliche Datenbank erstellen, in diesem Beispiel namens **BeispielAddIn.accdb**, und diese schließen und die Dateiendung in **.accda** ändern. Alle übrigen Änderungen, die eine Add-In-Datenbank von einer herkömmlichen Datenbank unterscheiden, müssen Sie manuell durchführen.

Registrierung von Add-Ins

Woher weiß Access überhaupt, ob es an irgendeiner Stelle eine Ihrer Datenbanken als Add-In oder Assistent anbieten soll? Dazu gibt es spezielle Registry-

Bereiche, die Access beim Starten prüft. Die dort angegebenen Datenbanken werden dann erfasst und an der entsprechenden Stelle, also etwa im Add-In-Menü, angezeigt.

Wenn Sie sich diese Bereiche einmal ansehen wollen, öffnen Sie einfach die Registry, indem Sie im Suchen-Feld von Windows den Wert RegEdit eingeben und die Eingabetaste betätigen. Es sollte nun ein neues Fenster mit dem Titel Registrierungs-Editor erscheinen. Hier wollen wir uns zunächst den Bereich für die Menü-Add-Ins ansehen, den Sie unter dem Schlüssel **HKEY_CURRENT_USER\SOFTWARE\Microsoft\Office\16.0\Access\Menu Add-Ins** finden. Hier sind in meiner Installation die gleichen Einträge vorhanden, die Sie auch weiter oben in der Liste im Ribbon finden konnten (siehe Bild 6).

Wenn Sie noch ein wenig weitersuchen, finden Sie unter dem Eintrag **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Office\ClickToRun\REGISTRY\MACHINE\Software**

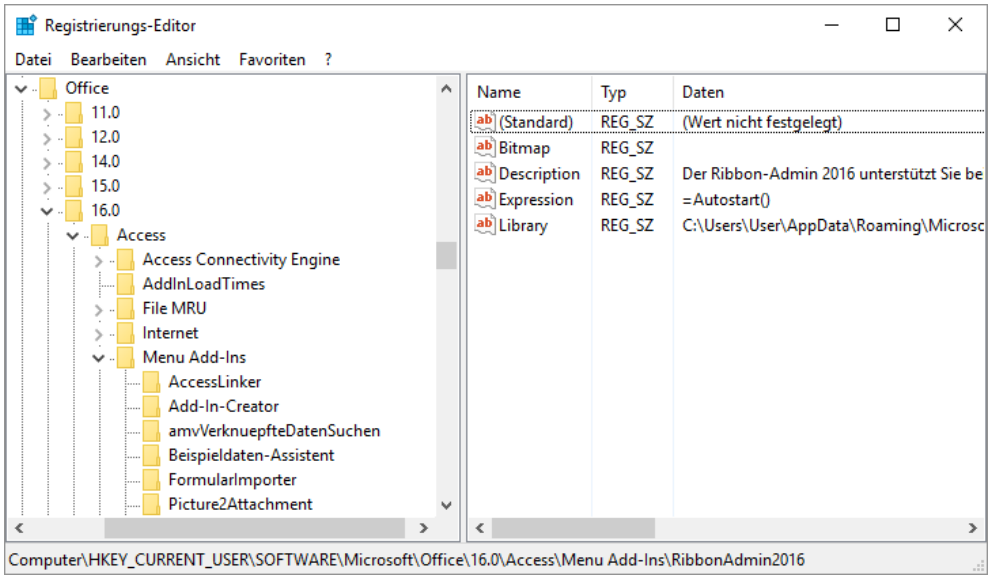


Bild 6: Die Registry mit den Einträgen für die Menü-Add-Ins

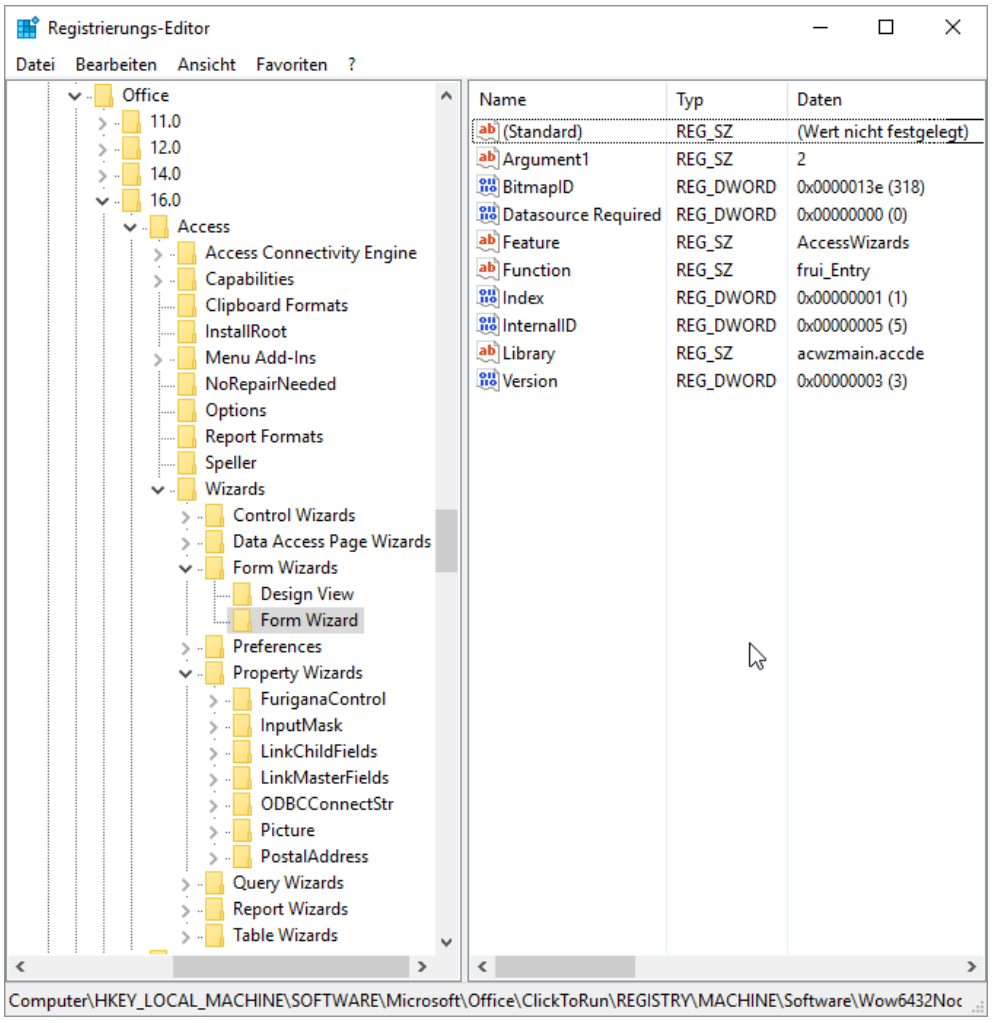


Bild 7: Einträge für die eingebauten Assistenten für Tabellen, Abfragen, Formulare, Berichte, Eigenschaften und Steuerelemente

Access-Objekt-Assistenten programmieren, Teil I

Während ein Add-In, wie wir es im Artikel »Access-Add-Ins programmieren« demonstriert haben, auch so genannt und im entsprechenden Ribbon-Menü verfügbar ist, nennen sich die übrigen Add-Ins »Assistenten«. Schauen wir uns also nun an, wie die Assistenten sich von den Add-Ins unterscheiden und wie Sie diese programmieren. Den Start machen wir dabei mit den Objekt-Assistenten, also solchen Assistenten, die Sie für die verschiedenen Objekttypen wie Tabellen, Abfragen, Formulare und Berichte über das Ribbon-Tab »Erstellen« aufrufen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **Abfrageassistent.accdb**.

Tabellen-Assistenten

Tabellen-Assistenten, wie es sie in früheren Access-Versionen einmal gab, finden Sie in den neuen Access-Versionen nicht mehr.
Es gibt schlicht keinen Eintrag namens Tabellen-Assistent mehr im Ribbon unter **Erstellen|Tabellen** (für Abfragen, Formulare und Berichte gibt es diese noch). Aber können wir überhaupt noch eigene Tabellen-Assistenten anlegen, wenn wir gar keinen Eintrag namens Tabellen-Assistenten mehr in Access haben? Dazu schauen wir uns zunächst die Vorgehensweise für einen Abfrage-Assistenten an, denn dort wissen wir, dass wir den Assistenten auch an der gewünschten Stelle vorfinden.

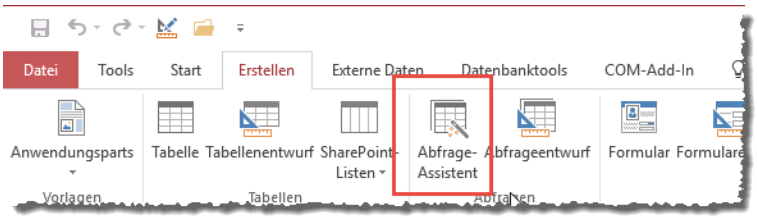


Bild 1: Aufruf eines Abfrage-Assistenten

Abfrage-Assistenten

Die Abfrage-Assistenten von Access öffnen Sie, indem Sie auf den Eintrag **Erstellen|Abfragen|Abfrage-Assistent** im Ribbon klicken (siehe Bild 1).

Es erscheint dann der Dialog aus Bild 2, der die vier in Access eingebauten Assistenten zum Erstellen von Abfragen für verschiedene Anforderungen anbietet. Diesem wollen wir nun einen eigenen Eintrag hinzufügen.

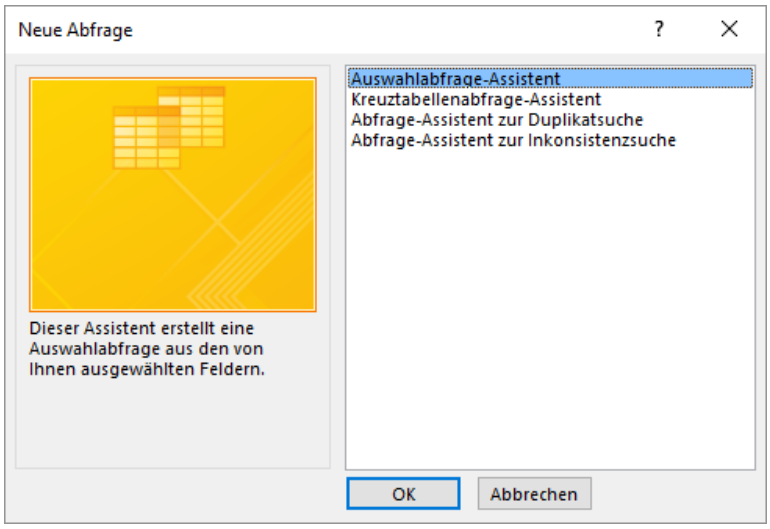


Bild 2: Der Dialog mit den verschiedenen Abfrage-Assistenten

Die Tabelle USysRegInfo

Diese Tabelle haben Sie bereits im Artikel Access-Add-Ins programmieren kennengelernt und Sie wissen, dass wir dort die Informationen speichern, die beim Installieren des Assistenten in die Registry eingetragen werden. Diese liest Access beim Start und stellt die dort registrierten Assistenten an den entsprechenden Stellen zur Verfügung – so wie etwa im oben vorgestellten Dialog **Neue Abfrage**.

Im Gegensatz zu einfachen Menü-Add-Ins gibt es bei den Objekt-Assistenten, also den Assistenten für die Erstellung von Tabellen, Abfragen, Formularen und Berichten eine

Reihe weiterer Einstellungen, die in die Registry geschrieben werden müssen. Diese schauen wir uns nun im Überblick an:

- **Bitmap**: Legt eine Bilddatei fest, die in dem Dialog zur Auswahl des Assistenten angezeigt werden soll, wenn der Benutzer den Assistenten markiert hat – siehe auch weiter oben im Bild des Dialogs **Neue Abfrage** (Zeichenfolge).

USysRegInfo

Subkey	Type	ValName	Value
HKEY_CURRENT_ACCESS_PROFILE\Wizards\Query Wizards\AbfrageAssistent	0		
HKEY_CURRENT_ACCESS_PROFILE\Wizards\Query Wizards\AbfrageAssistent	1	Function	Autostart
HKEY_CURRENT_ACCESS_PROFILE\Wizards\Query Wizards\AbfrageAssistent	1	Description	Dies ist ein Beispiel für einen Abfrage-Assistent.
HKEY_CURRENT_ACCESS_PROFILE\Wizards\Query Wizards\AbfrageAssistent	1	Library	|ACCDIR\Abfrageassistent.accda
HKEY_CURRENT_ACCESS_PROFILE\Wizards\Query Wizards\AbfrageAssistent	4	Datasource Required	0
HKEY_CURRENT_ACCESS_PROFILE\Wizards\Query Wizards\AbfrageAssistent	4	BitmapID	327
	0		

Datensatz: 1 von 6 Kein Filter Suchen

Bild 3: Die Tabelle **USysRegInfo** für unseren Abfrage-Add-In-Dummy

- **BitmapID**: Angabe einer ID, über die Sie die in der Datei **MSAccess.exe** enthaltenen Icons referenzieren und als Icon des Assistenten verwenden können (Zahlenwert).

- **Datasource Required**: Gibt an, ob für diesen Assistenten die Auswahl einer Datenquelle, also einer Tabelle oder Abfrage, erforderlich ist. In diesem Fall stellen Sie diese Einstellung auf den Wert **1** ein, anderenfalls auf den Wert **0** (Zahlenwert).

- **Description**: Wie bei den Add-Ins die Beschreibung des Add-Ins (Zeichenfolge).

- **Function**: Die beim Aufrufen des Assistenten aufzurufende VBA-Funktion (Zeichenfolge).

- **Index**: Einstellung für die Position des Assistenten in der Liste der verfügbaren Assistenten (Zahlenwert).

- **Library**: Pfad zu der Access-Datei, welche die aufzurufende Funktion enthält. In der Tabelle **USysRegInfo** können Sie den Ausdruck **|ACCDIR** als Platzhalter für den Add-In-Ordner auf dem jeweiligen System angeben.

Die Tabelle USysRegInfo

Die zuvor genannten Informationen tragen wir wie in Bild 3 in eine Tabelle namens **USysRegInfo** ein. Der Aufbau dieser Tabelle wurde bereits im Artikel **Access-Add-Ins programmieren** erläutert. Es gibt ein paar zusätzliche Informationen, die Sie für einen Wizard/Assistenten angeben können, aber zumindest die Angabe des Subkey ist identisch: Hier geben Sie den Platzhalter **HKEY_CURRENT_ACCESS_PROFILE/...** für den zu erstellenden Registry-Key an. In meiner Installation hat Access daraus einen recht wilden Registry-Pfad erzeugt:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Office\ClickToRun\REGISTRY\MACHINE\Software\Wow6432Node\Microsoft\Office\16.0\Access\Wizards\Query Wizards\AbfrageAssistent
```

Den Wert **327** für die Eigenschaft **BitmapID** finden Sie übrigens nicht ohne Weiteres heraus, da es sich um die interne Nummer einer in der Datei **MSAccess.exe** gespeicherte Bilddatei handelt. Diese

ermitteln Sie beispielsweise mit dem Tool **ResourceHacker**, indem Sie die Datei **MSAccess.exe** laden und die Einträge unter dem Element Bitmap durchlaufen (siehe Bild 4).

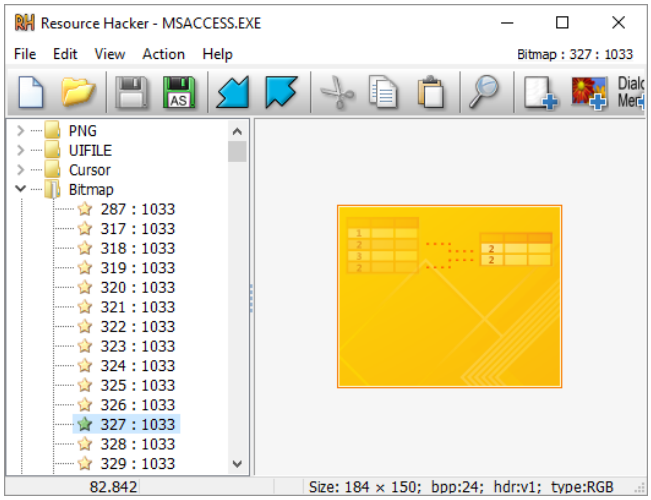


Bild 4: Ermitteln der **BitmapID** für die eingebauten Bilder

Der tatsächlich durch diese **USysRegInfo**-Tabelle bei der Installation des Add-Ins über den Add-In-Manager (Ribbon-Befehl **Datenbanktools|Add-Ins|Add-Ins|Add-In-Manager**) vorgenommene Registry-Eintrag sieht wie in Bild 5 aus.

Nachdem Sie diesen Dummy installiert haben, können Sie den Ribbon-Eintrag **Erstellen|Abfragen|Abfrage-Assistent** aufrufen. Hier

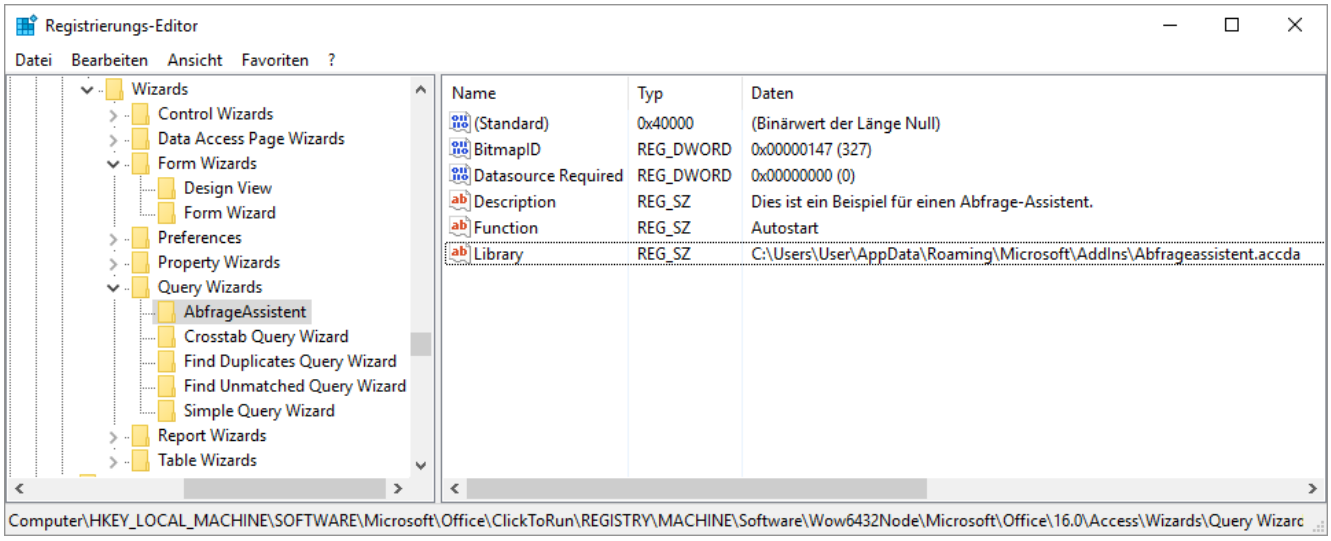


Bild 5: Registry-Eintrag für das Abfrage-Add-In

scheint nun der Dialog aus Bild 6 – mit unserem Eintrag! Ein Klick auf diesen verläuft allerdings recht unspektakulär, denn es erscheint natürlich nur das von uns programmierte Meldungsfenster.

Sehr wichtig für den korrekten Aufruf des Assistenten ist übrigens, dass Sie im Gegensatz zu einem Add-In nicht den Parameter **Expression** im Format **=<Expression>()**, sondern den Parameter **Function** im Format **<Function>** angeben.

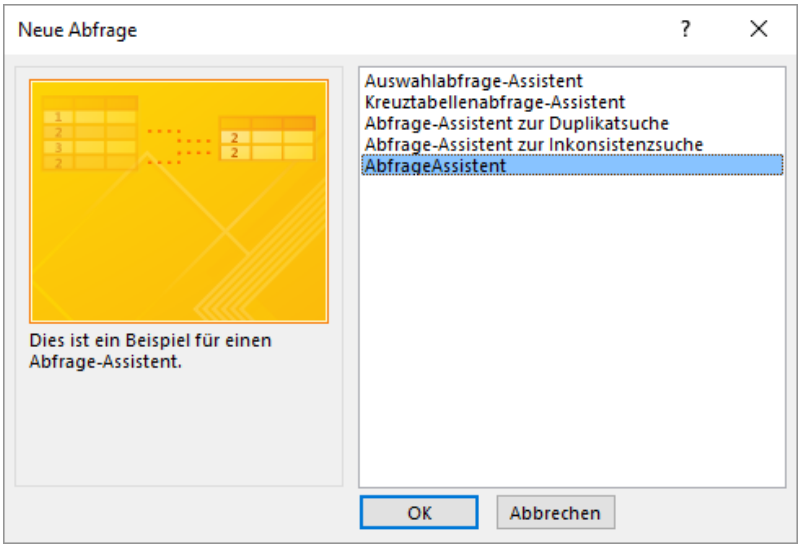


Bild 6: Unser Dummy-Addin in der Liste der Abfrage-Assistenten

Tabellen erstellen mit SQL und DDL

Tabellen erstellen Sie normalerweise über die Entwurfsansicht für Tabellen. Dort fügen Sie Felder hinzu, legen Datentypen fest und wählen das Primärschlüsselfeld aus. Was aber, wenn Sie einmal eine Änderung an einer Tabelle vornehmen müssen, ohne dass Sie Zugriff auf die Datenbank haben – beispielsweise, weil die Datenbank schon beim Kunden in Betrieb ist? Oder Sie sogar komplette neue Tabellen anlegen wollen? Dann können Sie dies auch per Code erledigen. Dieser Artikel zeigt, wie das per Data Definition Language (DDL) gelingt – dem Teil von SQL, der sich um das Definieren der Objekte der Datenbank kümmert.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1704_TabellenErstellenSQL.accdb**.

Was ist DDL?

Access-SQL besteht aus zwei Teilen: **DML** und **DDL**. **DML** ist die **Data Manipulation Language**, mit der Sie Daten auswählen und ändern können. Die Auswahl erfolgt dabei in der Regel durch die **SELECT**-Anweisung, das Ändern durch die **INSERT INTO**-Anweisung zum Anlegen von Datensätzen, die **UPDATE**-Anweisung zum Ändern vorhandener Daten und die **DELETE**-Anweisung zum Löschen von Daten.

Die **DDL**, also die **Data Definition Language**, dient dazu, die Tabellen, Felder, Indizes, Fremdschlüssel und so weiter anzulegen, zu ändern oder zu löschen. In diesem Artikel schauen wir uns den DDL-Teil von SQL an und werden beispielsweise neue Tabellen anlegen, Tabellen löschen, Felder hinzufügen, ändern und löschen oder die Indizes und die Beziehungen zwischen den Tabellen bearbeiten.

Parallel schauen wir uns an, wie sich die jeweiligen Anweisungen auf den Entwurf der entsprechenden Tabelle auswirken.

Anzulegende Tabellen

Um in der Entwurfsansicht eine neue Tabelle anzulegen, betätigen Sie den Ribbon-Eintrag **Erstellen|Tabellen|Tabellenentwurf**. Es erscheint dann der Entwurf der neuen, noch leeren Tabelle. Hier fügen Sie nun die gewünschten Felder hinzu und speichern die Tabelle unter dem gewünschten Namen, beispielsweise **tblKunden**. Das Ergebnis sieht dann etwa wie in Bild 1 aus. Diese Tabelle wollen wir nun komplett per VBA mit den entsprechenden SQL-Befehlen erstellen. Außerdem wollen wir eine weitere Tabelle namens **tblAnreden** hinzufügen, welche die beiden Felder **AnredeID** (als Primärschlüsselfeld) sowie

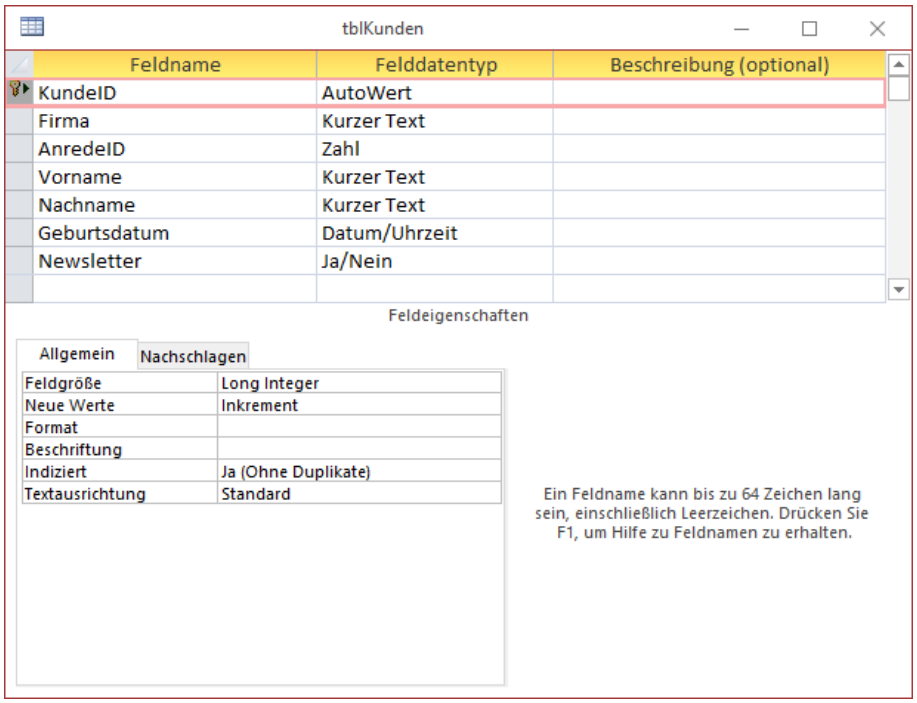


Bild 1: Diese Tabelle wollen wir mit VBA und SQL nachbauen.

Anrede enthält. Die Tabelle **tblKunden** soll dann über das Fremdschlüsselfeld **AnredeID** mit dem Primärschlüsselfeld der Tabelle **tblAnreden** verknüpft werden. Die Beziehung soll mit referenzieller Integrität definiert werden, damit in das Fremdschlüsselfeld **AnredeID** der Tabelle **tblKunden** nur Werte eingegeben werden können, die bereits im gleichnamigen Primärschlüsselfeld der Tabelle **tblAnreden** vorhanden sind.

Die Beziehung soll anschließend die Eigenschaften wie in Bild 2 aufweisen.

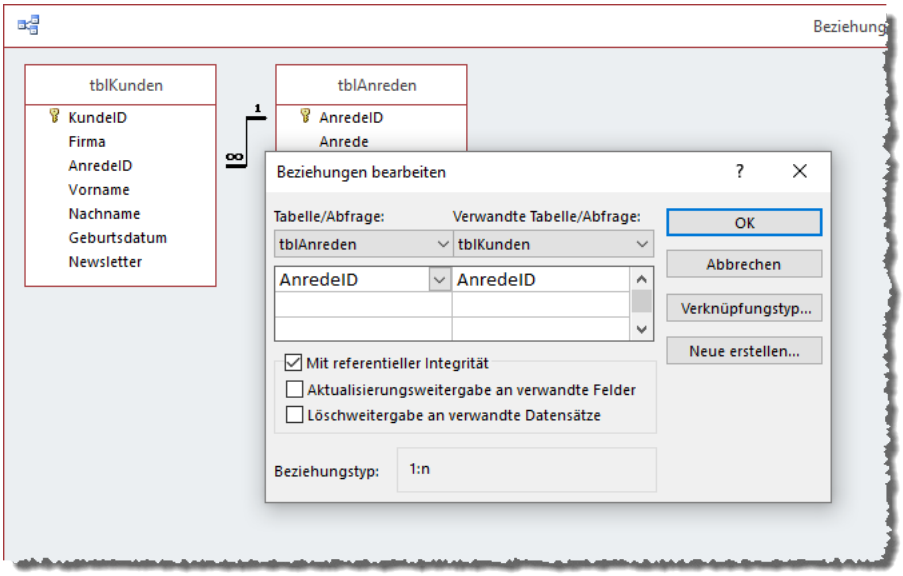


Bild 2: Beziehung zwischen den Tabellen **tblKunden** und **tblAnreden**

Die Primärschlüsselfelder der beiden Tabellen **tblKunden** und **tblAnreden** sollen außerdem als Autowert-Felder definiert werden.

Ausführen von DDL-Befehlen

Um einen DDL-Befehl auszuführen, benötigen Sie wenige Zeilen VBA-Code. Die einzige Methode, die wir anwenden, heißt **Execute** und gehört zum **Database**-Objekt der DAO-Bibliothek (**Microsoft DAO 3.6 Object Library**, in neueren Access-Versionen **ACE**, **Microsoft Office x.0 Access database engine Object Library**). Für dieses deklarieren wir zuvor eine entsprechende Objektvariable:

```
Dim db as DAO.Database
```

Mit der folgenden Anweisung füllen wir die Objektvariable, wobei die Funktion **CurrentDb** einen Verweis auf das aktuelle **Database**-Objekt liefert:

```
Set db = CurrentDb
```

Schließlich rufen wir die Methode **Execute** dieses Objekts auf:

```
db.Execute "CREATE TABLE ...", dbFailOnError
```

Wichtig ist hier, dass Sie neben der auszuführenden SQL-Anweisung als ersten Parameter noch einen zweiten Parameter übergeben, welcher die Konstante **dbFailOnError** enthält. Nur so löst eine fehlerhafte SQL-Anweisung auch einen Laufzeitfehler aus.

Diese Anweisung definieren Sie beispielsweise in einem Standardmodul, das Sie im VBA-Editor (zu öffnen mit **Alt + F11**) mit dem Menübefehl **Einfügen|Modul** hinzufügen. Hier können Sie die nachfolgend vorgestellten Anweisungen in jeweils eigenen Prozeduren hinzufügen. Zum testweisen Ausführen positionieren Sie die Einfügemarke irgendwo innerhalb der auszuführenden Prozedur und betätigen die Taste **F5**. Die Inhalte der Prozeduren können Sie natürlich auch etwa in die Prozeduren einfügen, die durch die Schaltflächen eines Formulars ausgelöst werden. Im Rahmen dieses Artikels wollen wir uns jedoch auf die Tests vom VBA-Editor aus beschränken.

Anlegen der Tabelle tblAnreden

Als Erstes legen wir die Tabelle **tblAnreden** an. Diese Tabelle hat nur zwei Felder, nämlich **AnredeID** und **Anrede**. Schon der erste Versuch, die Tabelle einfach mit der Anweisung **CREATE TABLE tblAnreden** anzulegen, führt zu einem Fehler. Aber nicht etwa, weil wir keine Felder angegeben haben, sondern weil die Ta-

belle schlicht und einfach bereits vorhanden ist – wir haben sie ja zum Veranschaulichen der üblicherweise verwendeten Vorgehensweise bereits über die Benutzeroberfläche angelegt (siehe Bild 3).

Tabelle löschen

Wir kümmern uns also bereits vor dem ersten Anlegen einer Tabelle per DDL um das Löschen einer bereits vorhandenen Tabelle gleichen Namens. Das ist zugegebenermaßen auch viel leichter, als die Tabelle zu erstellen und sieht wie folgt aus:

```
db.Execute "DROP TABLE tblAnreden",
dbFailOnError
```

Dummerweise führt dies gleich zum nächsten Fehler (siehe Bild 4). Wir müssen uns nun allerdings nicht um das Löschen der Beziehung kümmern, sondern wir entfernen gleich die komplette Tabelle **tblKunden**, bevor wir **tblAnreden** löschen:

```
db.Execute "DROP TABLE tblKunden", dbFailOnError
db.Execute "DROP TABLE tblAnreden", dbFailOnError
```

Danach können wir dann die Tabelle **tblAnreden** anlegen. Das klappt sogar ohne Angabe eines einzigen Feldes:

```
db.Execute "CREATE TABLE tblAnreden", dbFailOnError
```

Die Abfrage erscheint dann auch im Navigationsbereich, allerdings führt der Versuch, die Tabelle zu öffnen, zu dem Fehler aus Bild 5. Wir behalten für die folgenden Beispiele, die wir in der Prozedur **tblAnredenAnlegen** entwickeln, die beiden Anweisungen zum Löschen der Tabellen **tblKunden** und **tblAnreden** bei – so löschen wir gleich immer die zuvor angelegten Exemplare der beiden Tabellen.

Der nächste Versuch soll die beiden Felder **AnredeID** und **Anrede** mit den jeweiligen Datentypen zur Tabelle **tblAnreden** hinzufügen. Dabei haben wir vor den beiden Anweisungen zum Löschen der Tabellen

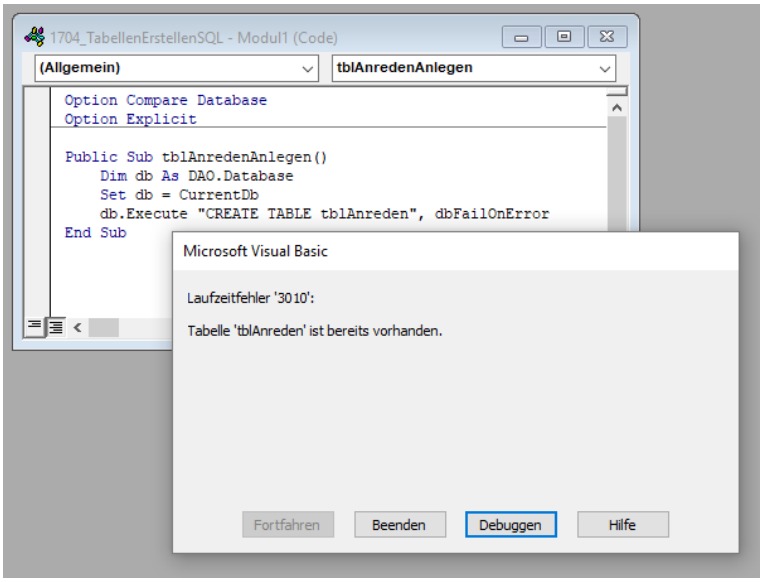


Bild 3: Fehler beim Versuch, eine bereits vorhandene Tabelle anzulegen

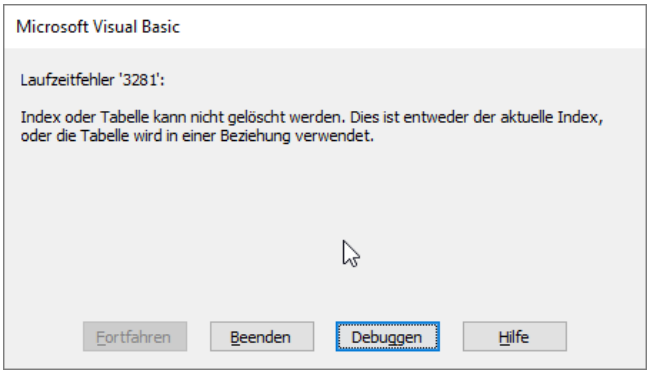


Bild 4: Die Beziehung verhindert das Löschen der Tabelle **tblAnreden**.

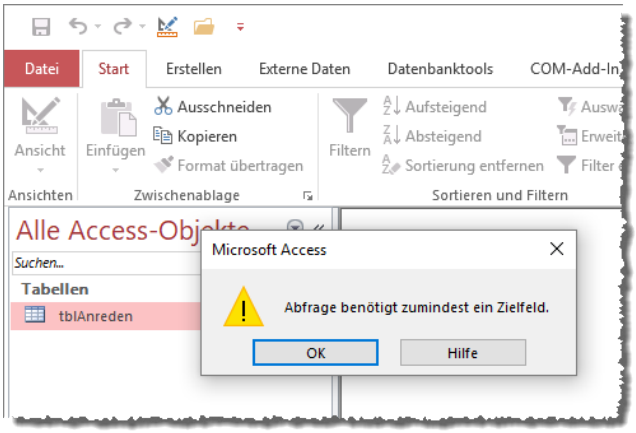


Bild 5: Tabellen ohne Felder können nicht geöffnet werden.

tblKunden und **tblAnreden** die Fehlerbehandlung deaktiviert, da diese beim Versuch, eine nicht vorhandene Tabelle zu löschen, einen Fehler auslösen würden:

```
Public Sub tblAnredenAnlegen()
    Dim db As DAO.Database
    Set db = CurrentDb
    On Error Resume Next
    db.Execute "DROP TABLE tblKunden", dbFailOnError
    db.Execute "DROP TABLE tblAnreden", dbFailOnError
    On Error GoTo 0
    db.Execute "CREATE TABLE tblAnreden(
     AnredeID INTEGER, Anrede TEXT(50))", dbFailOnError
End Sub
```

Das Ergebnis sehen Sie in Bild 6. Die beiden Felder werden mit den gewünschten Datentypen angelegt, allerdings natürlich noch ohne Primärschlüsselfeld und Autowert-Eigenschaft. Dies können wir nun nachträglich erledigen oder aber durch erneutes Löschen und Anlegen der Tabelle. Zunächst werfen wir jedoch noch einen Blick auf die **CREATE TABLE**-Anweisung:

```
CREATE TABLE tblAnreden(
    AnredeID INTEGER,
    Anrede TEXT(50)
)
```

Hier wird der grundsätzliche Aufbau schnell deutlich: Nach dem Tabellennamen folgt in Klammern eine komma-separierte Liste der anzulegenden Felder, wobei nach dem Feldnamen jeweils der Datentyp angegeben wird. Wir wollen die gängigen Datentypen abbilden, also diejenigen, die im Tabellenentwurf zur Auswahl stehen (siehe Bild 7).

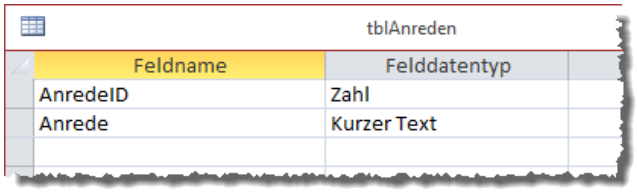


Bild 6: Die Tabelle **tblAnreden** mit den beiden Feldern **AnredeID** und **Anrede** – allerdings noch ohne Primärschlüssel- und Autowert.

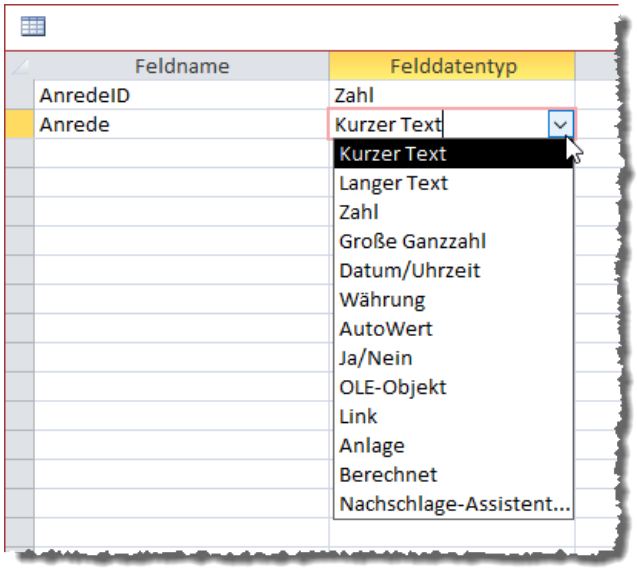


Bild 7: Datentypen für die Felder einer Access-Tabelle

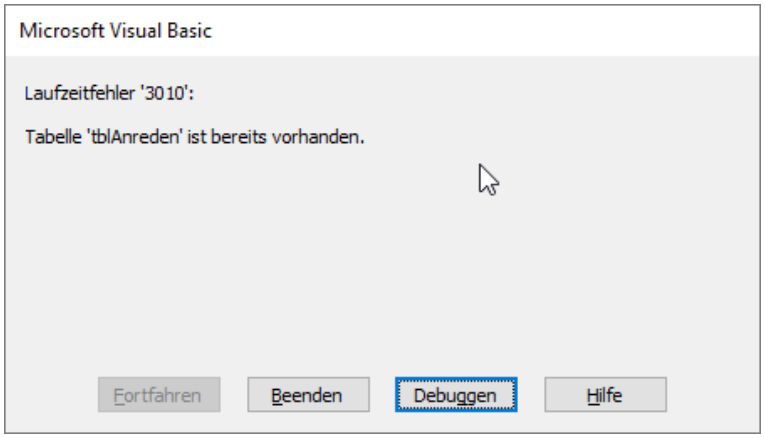


Bild 8: Datentypen für die Felder einer Access-Tabelle

Tabelle trotz Löschen noch vorhanden?

Es kann sein, dass trotz des Löschens einer Tabelle eine Fehlermeldung beim erneuten Erstellen einer Tabelle erscheint (siehe Bild 8). Dies ist dann der Fall, wenn die Tabelle beim Versuch, diese zu löschen, geöffnet ist – egal, ob in der Datenblatt- oder Entwurfsansicht. Sie müssen die Tabelle dann zunächst schließen. Auch dies könnten Sie noch in Form einer VBA-Anweisung zur Prozedur hinzufügen:

```
DoCmd.Close acTable, "tblAnreden"
```

Datentypen unter Access-SQL

Die SQL-Namen für die verschiedenen Datentypen des Tabellenentwurfs lauten wir folgt:

- **Kurzer Text: TEXT(n)** mit **n** als Anzahl der Zeichen (bis zu 255)
- **Langer Text: LONGTEXT**
- **Zahl**: Feldgröße Single: **SINGLE**, Feldgröße **Double: DOUBLE**, Feldgröße **Integer: SMALLINT**, Feldgröße **Long Integer: INTEGER**
- **Große Ganzzahl: DECIMAL** (nur per ADODB – siehe unten)
- **Datum/Uhrzeit: DATETIME**
- **Währung: MONEY, CURRENCY**
- **Autowert: COUNTER**
- **Ja/Nein: BIT**
- **OLE-Objekt: IMAGE**

Anreden mit Primärschlüssel

Um der Tabelle **tblAnreden** gleich einen Primärschlüssel für das Feld **AnredeID** zuzuweisen, verwenden Sie die folgende Anweisung als Parameter der **Execute**-Methode:

```
CREATE TABLE tblAnreden(
    AnredeID INTEGER CONSTRAINT PK PRIMARY KEY,
    Anrede TEXT(50)
)
```

Das Ergebnis finden Sie in Bild 9. Für das Feld **AnredeID** wird nun das Primärschlüssel-Symbol angezeigt.

Anreden mit Autowert

Wenn Sie statt eines einfachen Primärschlüsselfeldes ein Autowert-Feld einrichten wollen, also eines, bei dem beim Anlegen eines neuen Datensatzes automatisch ein noch nicht vergebener Zahlenwert für das Feld festgelegt wird, brauchen Sie einfach nur den Datentyp von **INTEGER** auf **COUNTER** zu ändern (Ergebnis siehe Bild 10):

```
CREATE TABLE tblAnreden(
    AnredeID COUNTER CONSTRAINT PK PRIMARY KEY,
    Anrede TEXT(50)
)
```

tblAnreden

Feldname	Felddatentyp	
AnredeID	Zahl	Primärschlüsselfeld der Tabelle
Anrede	Kurzer Text	Bezeichnung der Anrede

Bild 9: **AnredeID** mit Primärschlüssel

tblAnr

Feldname	Felddatentyp	
AnredeID	AutoWert	Primärschlüsselfeld mit Autowert
Anrede	Kurzer Text	Bezeichnung der Anrede

Bild 10: **AnredeID** mit Primärschlüssel und Autowert

Feld nachträglich hinzufügen

Wenn Sie sich nach dem Anlegen einer Tabelle entscheiden, ein neues Feld hinzuzufügen, haben Sie zwei Möglichkeiten: Entweder, Sie erstellen die Tabelle neu und berücksichtigen diesmal das fehlende Feld. Das ist jedoch in den meisten Fällen nicht möglich, weil die Tabelle bereits Datensätze enthält. Also gehen wir ein wenig feiner vor und fügen nur das fehlende Feld hinzu. In diesem Fall wollen wir die Tabelle **tblAnreden**

Alle Ereignisse im Hauptformular

Wenn Sie mit Haupt- und Unterformularen arbeiten und Ereignisprozeduren sowohl in der Code behind-Klasse des Haupt- und des Unterformulars angelegt haben, kann es manchmal unübersichtlich werden. Das ist vor allem dann der Fall, wenn Ereignisse im Unterformular sich auf die Abläufe im Hauptformular auswirken. Dieser Artikel zeigt, wie Sie die Ereignisse, die durch ein Unterformular ausgelöst werden, im Klassenmodul des Hauptformulars implementieren. Damit können Sie Ereignisse, Objekte und Variablen in nur noch einem Modul nutzen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1704_EreignisseImHauptformular.accdb**.

Beispielkonfiguration

In unserem Beispiel wollen wir die Einträge einer Tabelle namens **tblArtikel** in einem Unterformular anzeigen. Das Hauptformular soll Steuerelemente zum Hinzufügen eines neuen Artikels, zum Löschen eines Artikels oder zum Bearbeiten eines bestehenden Artikels enthalten. Dazu ist es aus Gründen der Ergonomie notwendig, dass die Schaltflächen zum Anzeigen eines Datensatzes zum Bearbeiten und zum Löschen des aktuell markierten Datensatzes nur aktiviert sind, wenn der Benutzer auch einen Datensatz im Datenblatt markiert hat.

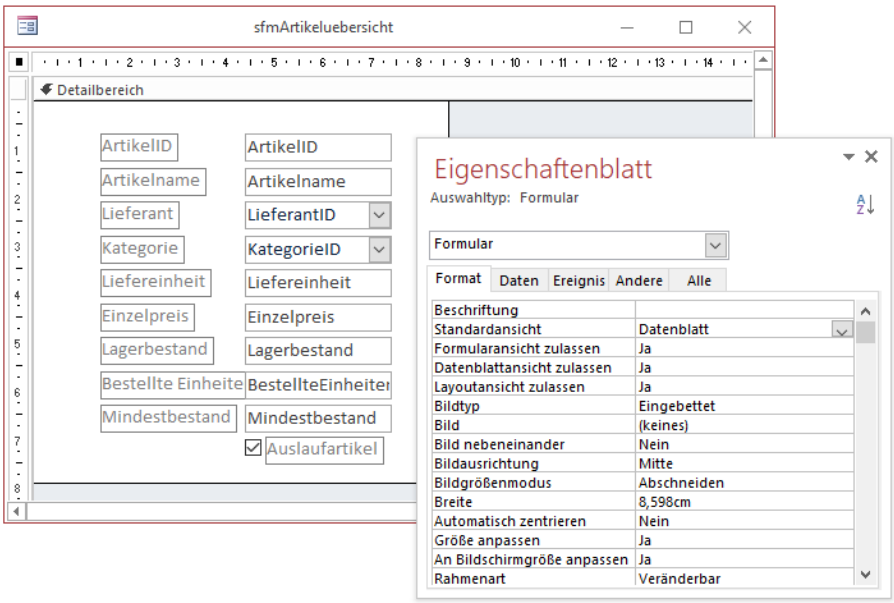


Bild 1: Entwurf des Unterformulars mit der Datenblattansicht

Das Unterformular versehen wir daher mit der Tabelle **tblArtikel** als Datenherkunft und ziehen alle Felder dieser Tabelle aus der Feldliste in den Detailbereich des Entwurfs. Außerdem stellen wir die Eigenschaft **Standardansicht** des Formulars auf **Datenblatt** ein (siehe Bild 1). Speichern Sie das Unterformular unter dem Namen **sfmArtikeluebersicht** und schließen Sie es. Dann legen Sie ein neues Formular namens **frmArtikeluebersicht** an und ziehen das Unterformular **sfmArtikeluebersicht** aus dem Navigationsbereich in den Detailbereich des neuen Formulars.

Außerdem fügen Sie noch drei Schaltflächen namens **cmdNeu, cmdBearbeiten** und **cmdLoeschen** zu diesem Hauptformular hinzu. Ordnen Sie die Steuerelemente wie in Bild 2 an und stellen Sie außerdem die beiden Eigenschaften **Horizontaler Anker** und **Vertikaler Anker** des Unterformular-Steuerelements aus **Beide** ein. Für die unter diesem Steuerelement befindlichen Schaltflächen müssen Sie entsprechend **Horizontaler Anker** auf **Unten** einstellen, damit sich das Unterformular beim Vergrößern des Hauptformulars nicht über die Schaltflächen legt.

Da das Hauptformular selbst keine Daten anzeigt, können Sie dessen Eigenschaften **Navigationsschaltflächen**, **Datensatzmarkierer**, **Trennlinien** und **Bildlaufleisten** auf **Nein** einstellen sowie **Automatisch zentrieren** auf **Ja**.

Nun wollen wir dafür sorgen, dass die beiden Schaltflächen cmdBearbeiten und cmdLoeschen deaktiviert werden, wenn der Datensatzzeiger im Unterformular nicht auf einen Datensatz zeigt – also beispielsweise der neue Datensatz markiert ist wie in Bild 3.

Klassisch erledigen wir das wie folgt: Wir nutzen das Ereignis **Beim Anzeigen** des Unterformulars, indem wir die entsprechende Eigenschaft den Wert **[Ereignisprozedur]** auswählen und dann auf die Schaltfläche mit den drei Punkten rechts von der Eigenschaft klicken. Dann ergänzen wir die im VBA-Editor angelegte Ereignisprozedur wie in Listing 1. Das Ereignis wird jeweils beim Verschieben des Datensatzzeigers auf einen der Datensätze ausgelöst, auch beim Verschieben auf den leeren, neuen Datensatz.

In der Ereignisprozedur stellen wir die Eigenschaft **Enabled** der beiden Schaltflächen jeweils auf den Wert des Ausdrucks **Not IsNull(Me!ArtikelID)** ein. **Me!Artikel** hat immer einen Wert, wenn der Datensatzzeiger auf einen Datensatz zeigt, im Falle des leeren, neuen Datensatzes liefert dies jedoch den Wert **Null**. In diesem Fall werden die beiden Schaltflächen dann wie in der Abbildung deaktiviert. Beim Wechsel zu einem anderen Datensatz aktiviert die

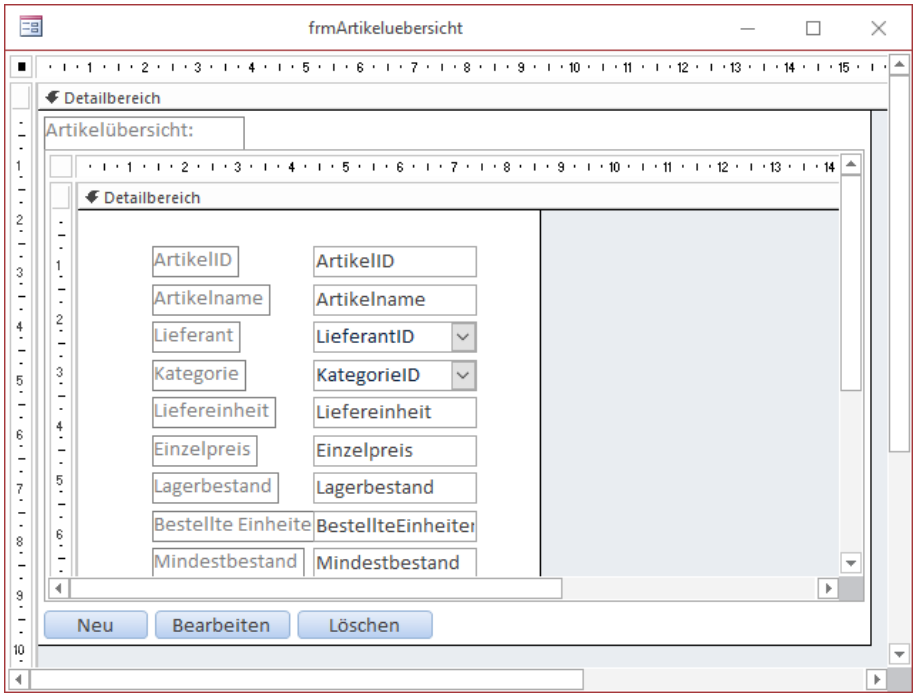


Bild 2: Haupt- und Unterformular in der Entwurfsansicht

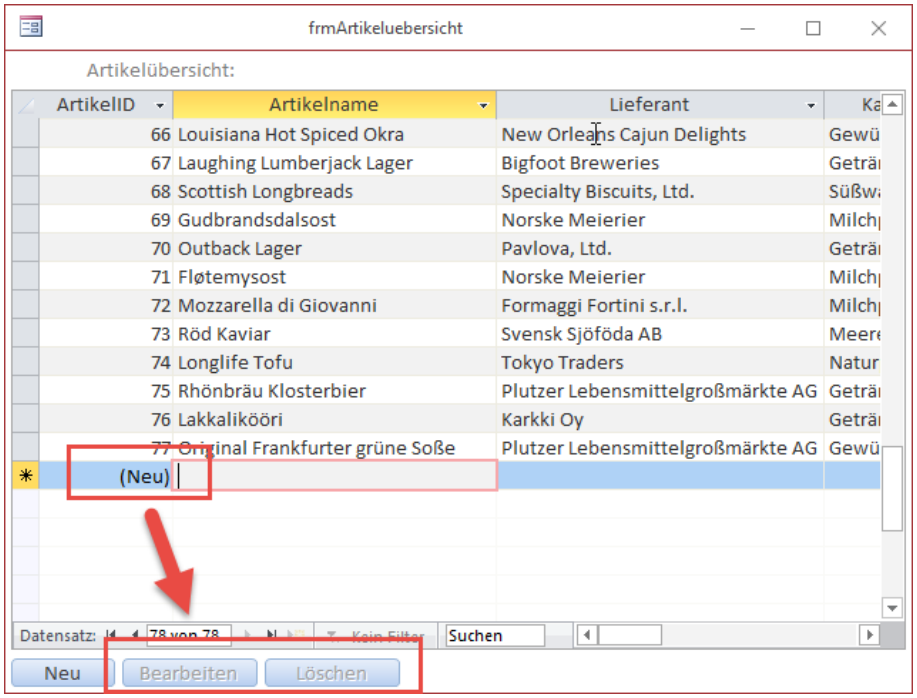


Bild 3: Die beiden Schaltflächen **cmdBearbeiten** und **cmdLoeschen** sollen deaktiviert sein, wenn im Unterformular kein Datensatz markiert ist.

Text- und Bezeichnungsfelder

Wenn Sie in einem Formular ein Feld aus der Feldliste hinzufügen oder auch ein Textfeld oder andere Steuerelemente wie ein Kombinationsfeld oder ein Kontrollkästchen aus der Liste der Steuerelemente im Ribbon verwenden, fügt Access direkt ein entsprechendes Bezeichnungsfeld zum Steuerelement hinzu. Manch einer nimmt dies einfach so hin. Dabei gibt es einige Einstellungen, mit denen man das Verhalten beim Hinzufügen anpassen kann und man kann die an das Steuerelement gebundenen Bezeichnungsfelder sogar von diesem lösen und wieder hinzufügen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1704_Bezeichnungsfelder.accdb**.

Automatische Bezeichnungsfelder

Wenn Sie ein Textfeld, ein Kombinationsfeld, ein Optionsfeld, ein Kontrollkästchen, ein Listenfeld, ein Unterformular, ein gebundenes Objektfeld oder – als Sonderfall – eine Optionsgruppe zu einem Formulare in der Entwurfsansicht hinzufügen, legt Access standardmäßig auch gleich ein passendes Bezeichnungsfeld mit an (siehe Bild 1).

Die Optionsgruppe betrachten wir zunächst als Sonderfall, weil sich das Bezeichnungsfeld hier an einer ganz anderen Stelle befindet als bei den anderen Steuerelementen, nämlich nicht links daneben, sondern auf der oberen Begrenzungslinie. Auch beim Kontrollkästchen und bei der Optionsgruppe wird die Bezeichnung rechts vom Steuerelement angebracht (siehe Bild 2).

Verhalten der Bezeichnungsfelder

Es handelt sich bei diesen Bezeichnungsfeldern jedoch nicht einfach um lieblos eingefügte Bezeichnungsfelder, sondern diese haben eine spezielle Eigenschaft: Sie haften nämlich an dem Steuerelement, mit dem Sie zum Formular oder Bericht hinzugefügt wurden. Wenn Sie das Steuerelement verschieben, verschieben Sie auch das Bezeichnungsfeld. Wenn Sie das Steuerelement löschen, verschwindet auch das Bezeichnungsfeld. Sie können das Steuerelement lediglich unabhängig vom Bezeichnungsfeld

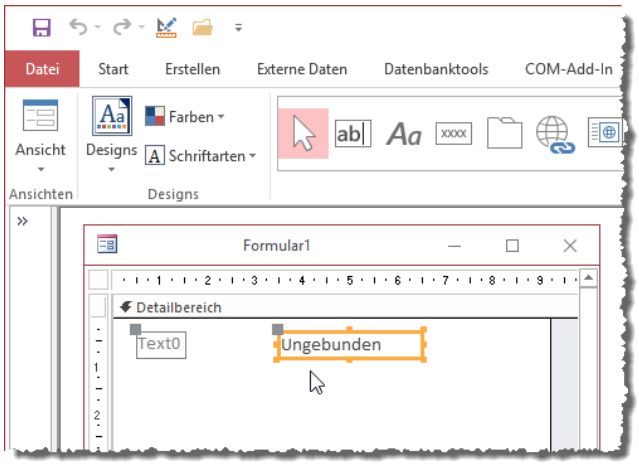


Bild 1: Ein frisch angelegtes Textfeld mit Bezeichnungsfeld

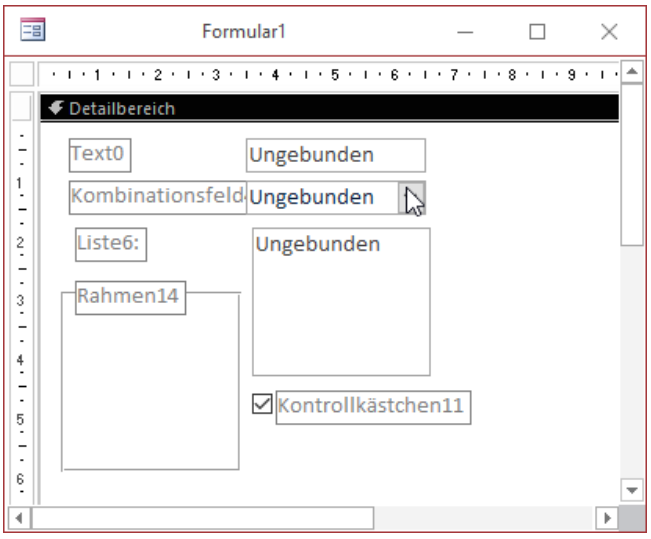


Bild 2: Einige Steuerelemente, zu denen automatisch ein Bezeichnungsfeld hinzugefügt wird.

bewegen, wenn Sie dieses an dem Kästchen oben links mit der Maus greifen und verschieben. Das gilt nicht nur für das Steuerelement, sondern auch, wenn Sie das Bezeichnungsfeld unabhängig vom Steuerelement verschieben wollen (siehe Bild 3).

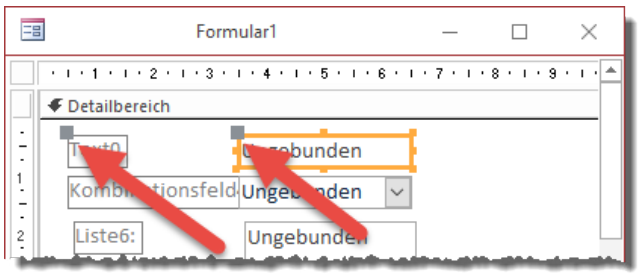


Bild 3: An diesen Punkten müssen Sie ziehen, wenn Sie Bezeichnungsfeld oder Steuerelement ohne den Partner umpositionieren wollen.

Bezeichnungsfeld lösen

Wenn Sie ein Bezeichnungsfeld vom Steuerelement lösen wollen, um diese unabhängig von diesem zu bewegen, gibt es eine einfache Möglichkeit: Sie müssen lediglich das Bezeichnungsfeld markieren, es mit der Tastenkombination **Strg + X** ausschneiden und mit der Tastenkombination **Strg + V** wieder einfügen.

Leider wird das Bezeichnungsfeld danach nicht direkt an der gleichen Stelle wieder eingefügt, sondern oben links im jeweiligen Formularbereich. Access zeigt dann ein Aufklapp-Menü mit einem Ausrufezeichen an, das beim Anklicken die Optionen aus Bild 4 bietet.

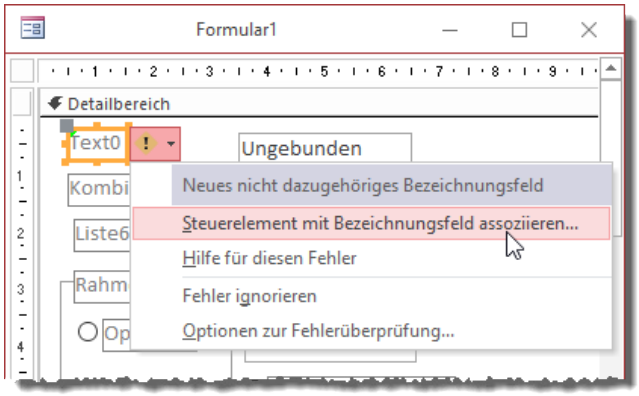


Bild 4: Nach dem Trennen eines Bezeichnungsfeldes von einem Steuerelement bietet Access verschiedene Möglichkeiten an.

Es hat ein neues, nicht zu einem Steuerelement gehörendes Bezeichnungsfeld gefunden und bietet die Möglichkeit, das Bezeichnungsfeld einem Steuerelement zuzuweisen. Wenn Sie diese Option wählen, erscheint der Dialog aus Bild 5, mit dem Sie das Steuerelement auswählen können, dem Sie das Bezeichnungsfeld zuweisen wollen.

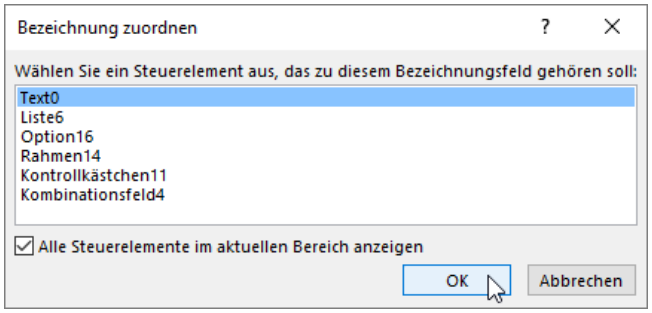


Bild 5: Verbinden eines Bezeichnungsfeldes mit einem Steuerelement

Wenn Sie hier ein Steuerelement auswählen, dass bereits mit einem Bezeichnungsfeld assoziiert ist, dann wird die Verbindung gelöscht und das Steuerelement mit dem neuen Bezeichnungsfeld verbunden.

Das Aufklapp-Menü erscheint übrigens auch, wenn Sie ein neues Bezeichnungsfeld zu einem Formular hinzufügen.

Verbindungen erkennen

Ob und mit welchem Steuerelement ein Bezeichnungsfeld verbunden ist, erkennen Sie übrigens daran, für welches Steuerelement beim Markieren des Bezeichnungsfeldes das Quadrat in der linken oberen Ecke des Steuerelements eingeblendet wird. Gleiches gilt übrigens umgekehrt auch, wenn Sie ein Steuerelement markieren. Hat dieses ein assoziiertes Bezeichnungsfeld, dann wird dessen Quadrat in der linken oberen Ecke markiert.

Bezeichnungsfeld verbinden

Neben der Methode über das Aufklappmenü können Sie ein Bezeichnungsfeld auch noch auf einem ande-

Handyverwaltung I: Datenmodell

Vor ein paar Jahren war das Leben noch einfach. Jeder hatte zuhause ein Telefon und, wenn er einen festen Arbeitsplatz hatte, noch eine dienstliche Nummer. Heute hat jedes Kind ein Handy in der Tasche, wenn es das Haus verlässt, und manch ein Arbeitnehmer läuft mit einem privaten Handy in der linken und dem Diensthandy in der rechten Hosentasche herum. Wer soll da noch durchblicken, vor allem, wenn man als Arbeitgeber oder auch als Familienoberhaupt für die Kosten, die Anschaffung und die Verwaltung der Verträge verantwortlich ist? Die in diesem Beitrag vorgestellte Handyverwaltung kann nicht nur von Firmen, sondern auch von Familien eingesetzt werden und hilft hoffentlich, den Durchblick bei Verträgen und Geräten zu bewahren.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1704_Handyverwaltung.accdb**.

Tabellen der Anwendung

Die Anwendung benötigt auf jeden Fall eine Tabelle, mit der wir die Benutzer der Handys speichern. Da es sich sowohl um Familienmitglieder als auch um Mitarbeiter handeln kann, verwenden wir einfach die Bezeichnung **Person** für unsere Handybenutzer. Die Tabelle, in der wir diese Personen speichern, heißt dementsprechend **tblPersonen**. Da die Anwendung auch in Firmen zum Einsatz kommen soll und diese sich vielleicht die Rechnung komplett oder in Teilen erstatten lassen möchte, gibt es vielleicht eine schriftliche Korrespondenz, weshalb wir auch das Feld **AnredeID** in die Tabellen einbeziehen wollen – nebst einer entsprechenden Lookup-Tabelle namens **tblAnreden**.

Die Tabelle tblAnreden

Diese enthält die einfache Form der Anrede (**Herr/Frau**) sowie die Anrede für den Adressblock (**Herrn/Frau**) und die Briefanrede (**Sehr geehrter Herr/Sehr geehrte Frau**) in den drei Feldern **Anrede**, **Adressanrede** und **Briefanrede**. Der Entwurf dieser Tabelle ist in Bild 1 abgebildet.

Für die drei Felder **Anrede**, **Adressanrede** und Briefanrede haben wir jeweils einen eindeutigen Index festgelegt.

Die Tabelle tblPersonen

Die Tabelle **tblPersonen** enthält neben dem Fremdschlüsselfeld **AnredeID** zur Auswahl eines der Datensätze noch die Felder **Vorname** und **Nachname**. Für die Verwaltung der Mobiltelefone einer Familie reicht dies aus, wenn Sie die Tabelle in einer Firma nutzen wollen, werden Sie vermutlich noch weitere Felder hinzufügen wollen, mit denen Sie die Eigenschaften bezüglich der Anstellung des Mitarbeiters verwalten

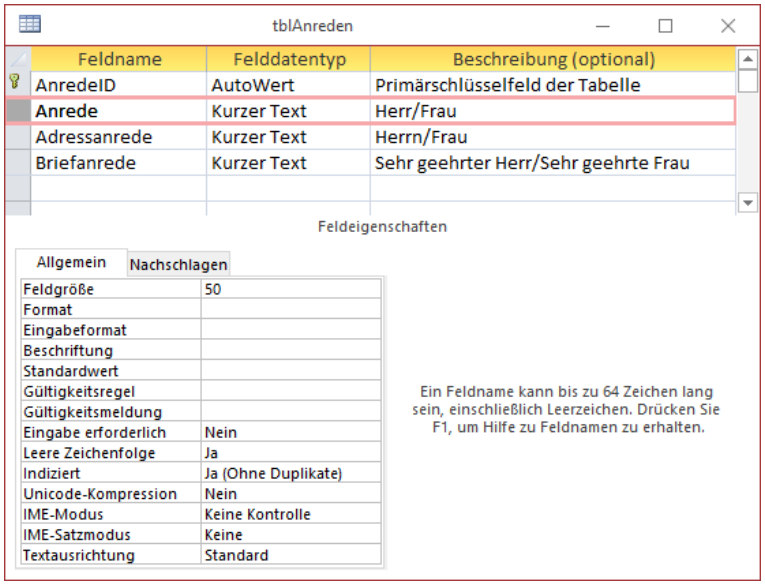


Bild 1: Tabelle mit den verschiedenen Anreden

können – also etwa Abteilung, Funktion, Jobtitel, Anstellungsdatum et cetera. Vor allem das Auflösungsdatum der Anstellung dürfte von Interesse sein, denn mit dem Ende des Beschäftigungsverhältnisses soll in der Regel auch das Mobilfunkgerät zurückgegeben werden. Aus Gründen der Übersicht lassen wir solche Felder jedoch außen vor.

Fremdschlüsselfeld zur Auswahl einer Anrede

Damit Sie die Anrede für die Person einfach auswählen können, stellen wir das Feld **AnredeID** der Tabelle **tblPersonen** als Nachschlagefeld ein. Dazu wählen Sie als Datentyp für das Feld den Wert Nachschlagefeld aus. Dies startet einen Assistenten, der Sie beim Einrichten des Nachschlagefeldes unterstützt:

- Dieser fragt zunächst, ob die Daten aus einer Tabelle stammen sollen oder ob Sie die Werte selbst eingeben wollen. Hier wählen Sie die erste Option (siehe Bild 2).

- Danach wählen Sie mit dem folgenden Dialog die Quelltabelle für die auszuwählenden Daten aus, in diesem Fall die Tabelle **tblAnreden** (siehe Bild 3).

- Aus dieser Tabelle wollen wir nicht alle Felder im Nachschlagefeld anzeigen, sondern wir benötigen nur die beiden Felder **AnredeID** und **Anrede** (siehe Bild 4).

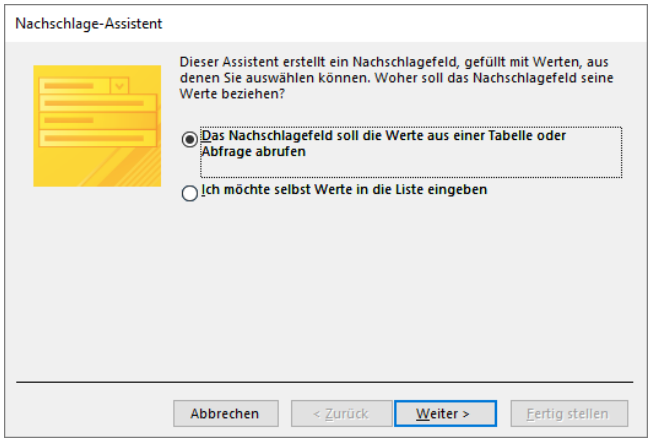


Bild 2: Nachschlagefeld einrichten

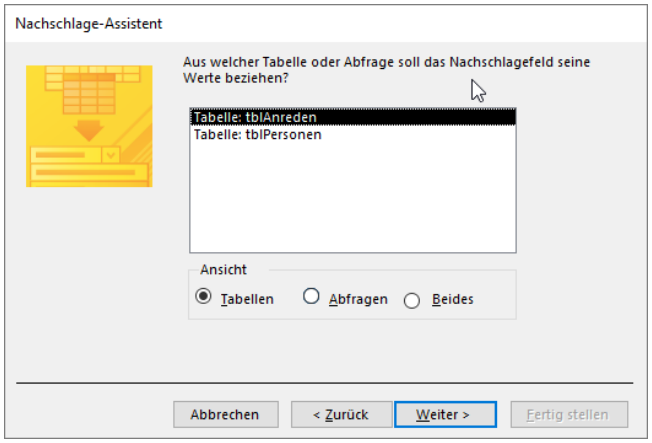


Bild 3: Auswahl der zu verknüpfenden Tabelle

- Das zuoberst in der Liste der ausgewählten Felder angezeigte Feld interpretiert der Assistent als Schlüsselspalte, also als die Spalte, deren Wert letzlich im Feld **AnredeID** der Tabelle **tblPersonen**

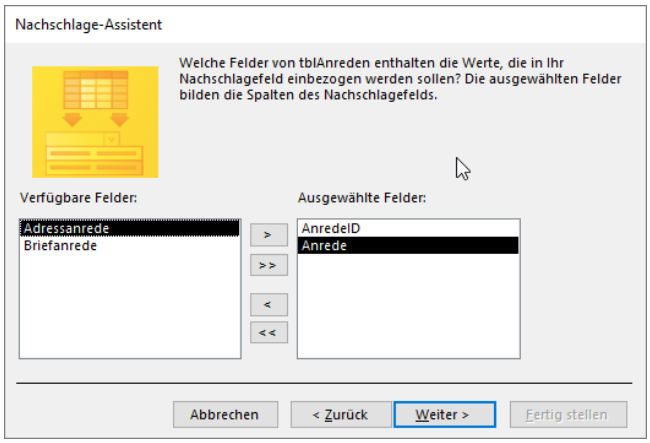


Bild 4: Festlegen der gewünschten Felder

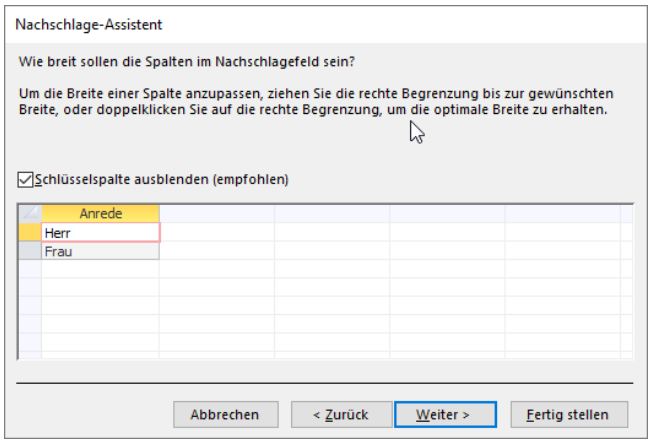


Bild 5: Ausblenden der Schlüsselspalte

eingetragen wird. Damit diese ausgeblendet wird und nur der Inhalt des Feldes **Anrede** im Nachschlagefeld angezeigt wird, aktivieren Sie die Option **Schlüsselspalte ausblenden** aus Bild 5.

- Damit fehlt noch der letzte Schritt, nämlich das Festlegen der Löschweitergabe. Hier wählen wir die zweite Option, nämlich **Löschbeschränkung**. Das sorgt dafür, dass Sie einen Datensatz der Tabelle **tblAnreden** nur löschen können, wenn kein Datensatz der Tabelle **tblPersonen** mehr damit verknüpft ist. Dies ist nur zur Sicherheit – die Datenbank dürfte ohnehin kein Element der Benutzeroberfläche enthalten, mit dem Sie einen Datensatz der Tabelle **tblAnreden** löschen können (siehe Bild 6).

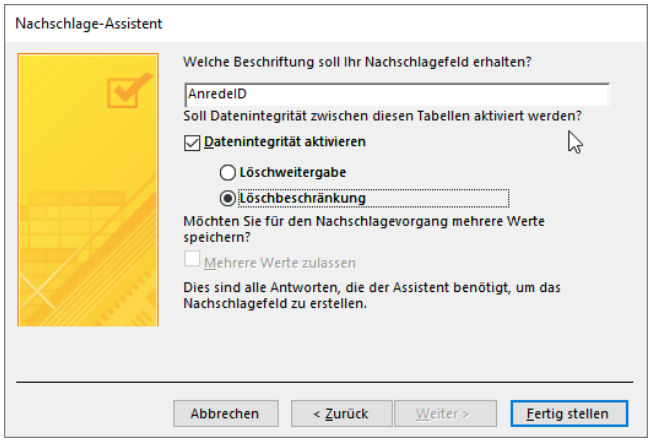


Bild 6: Festlegen der Löschbeschränkung

Damit ist das Nachschlagefeld eingerichtet. Der Vorteil hiervon ist, dass wenn Sie das Feld **AnredeID** der Tabelle **tblPersonen** später in den Entwurf eines Formulars ziehen, direkt ein Kombinationsfeld statt eines Textfeldes angelegt wird. Dieses erlaubt dann das einfache Auswählen der gewünschten Anrede. Das ist recht komfortabel, denn so brauchen Sie nicht immer das Textfeld erst noch in ein Kombinationsfeld umzuwandeln und die entsprechenden Eigenschaften manuell einstellen.

Verträge und Handys – wie speichern und verknüpfen?

Das Anlegen der Tabellen für die Personen und die Anreden war einfach. Jetzt wird es etwas interessanter: Wir wollen sowohl die Handys speichern, die in der Familie beziehungsweise in der Firma existieren als auch die Verträge. In welcher Beziehung sollen die Handy mit den Personen stehen? Spielen wir einmal die Konstellationen durch. Kann eine Person/ein Mitarbeiter mehr als ein Handy besitzen? Ja, warum nicht: Vielleicht verwendet man ein hochwertiges Handy, das man im Alltagsbetrieb nutzt, und ein weiteres, das man zum Sport verwendet, weil es dort auf den Boden fallen kann oder anderweitig stärker beansprucht wird. Möglicherweise teilen sich auch mehrere Personen wie etwa die Kinder einer Familie ein Mobiltelefon?

Warum ein Mitarbeiter mehr als ein Handy nutzen sollte fällt mir auf Anhieb nicht ein, aber es gibt bestimmt mehr oder weniger abwegige Gründe, dies zu tun. Was aber gar nicht abwegig ist, dass ein Familienmitglied oder auch ein Arbeitnehmer neben dem Mobiltelefon noch ein weiteres Gerät wie etwa ein Tablet nutzt, dass ja auch mit einer SIM-Karte ausgestattet werden kann.

Kann ein Handy von mehr als einer Person oder mehr als einem Mitarbeiter genutzt werden? Auch das ist theoretisch möglich. Vielleicht wird das soeben erwähnte Handy, das beim Sport zum Einsatz kommt, nicht nur von einem Familienmitglied genutzt. Ob das auch im beruflichen Umfeld geschehen kann, ist auch nicht auszuschließen.

Wie wahrscheinlich die oben genannten Fälle auch sein mögen: Tatsache ist, dass wir das Datenmodell ohne Probleme so gestalten können, dass alle Varianten möglich sind, und auch die Benutzeroberfläche wird nicht weiter darunter leiden. Wir werden also eine eigene Tabelle namens **tblMobiltelefone** erstellen, die wir über eine Verknüpfungstabelle mit der Tabelle **tblPersonen** verknüpfen. Auf diese Weise können wir dann jedes Mobiltelefon mehreren Personen zuordnen als auch jede Person mehreren Mobiltelefonen.

Damit kommen wir zu den Verträgen. Wie kommen die ins Spiel – werden diese den Personen zuge-

ordnet oder den Mobiltelefonen? Tatsache ist: Ein Vertrag ist mindestens einer SIM-Karte zugeordnet. Andersherum kann es aber zu einem Vertrag auch mehr als eine SIM-Karte geben. Dabei handelt es sich um sogenante Multi-SIM. Diese sind beispielsweise geeignet, um den oben erwähnten Fall der Person abzudecken, die ein "Ausgeh-Handy" und eines für rauhere Einsatzfälle wie Sport, Wandern et cetera verwendet. Oder die ein Tablet für den mobilen Datenverkehr über den gleichen Tarif nutzen möchte.

Sprich: Wir müssen vorsehen, dass es zu einem Vertrag mehrere SIM-Karten gibt, aber zum Glück nicht umgekehrt mehrere Verträge zu einer SIM-Karte.

Nun können aber beispielsweise zwei Geräte, die zwei SIM-Karten mit dem selben Vertrag verwenden, auch von zwei verschiedenen Personen genutzt werden. So kann es beispielsweise sein, dass der Vater der Familie einen Vertrag mit hohem Datenvolumen abgeschlossen hat, dessen zwei SIM-Karten er in ein Mobiltelefon und ein Tablet gesteckt hat. Das Tablet nutzt er aber nicht selbst, sondern seine Tochter, die wiederum nur ein Billig-Handy mit einem günstigen Tarif verwendet.

Es sieht also danach aus, als ob wir mit der Zuweisung der Mobiltelefone zu den Personen richtig lagen. Aber wie bringen wir die Verträge in das Datenmodell ein? Die erste Variante wäre, die Tabelle der Verträge so mit der Tabelle der Mobilfunkgeräte zu verknüpfen, dass für jedes Mobilfunkgerät über ein Fremdschlüsselfeld der Vertrag festgelegt werden kann. Damit wäre abgedeckt, dass jedes Mobilfunkgerät nur einem Vertrag zugeordnet werden kann, aber jedem Vertrag mehrere Mobilfunkgeräte.

Damit könnte aber wiederum ein Vertrag über die enthaltenen Mobilfunkgeräte theoretisch mehreren Personen zugeordnet werden. Hier geht es nun um die Kostenfrage: Wer bezahlt die laufenden Kosten für den Vertrag? In einer Familie mit mehreren Personen bezahlt vielleicht der Hauptverdiener die Mobilfunkverträge, aber möglicherweise sind ja Vater und Mutter beide selbstständig und bezahlen ihre Verträge über ihr jeweiliges Geschäftskonto. Und die Kinder sind vielleicht angehalten, ihre Handyverträge vom eigenen Taschengeld zu bezahlen. Es wird immer komplizierter!

In der Berufswelt könnte man naiverweise davon ausgehen, dass der Arbeitgeber die Mobiltelefone bezahlt. Aber vielleicht hat er auch mit dem Angestellten ausgemacht, dass dieser das gestellte Handy auch für private Zwecke nutzen darf und hat dafür eine anteilige Zahlung der Mobilfunkgebühren vereinbart.

Ein Mobilfunkvertrag kann also über die damit verknüpften SIM-Karten für ein oder mehrere Geräte genutzt werden und wird vielleicht auch noch von mehreren Personen bezahlt.

Und können wir überhaupt Mobilfunkgeräte den Verträgen zuordnen – müssen wir nicht vielmehr noch über die SIM-Karten reden? Denn immerhin kann es ja auch einmal sein, dass ein Mobilfunkgerät die SIM-Karte wechselt. In einer Familie, die sich nicht ständig mit den neuesten Handys eindeckt, kauft sich vielleicht der Vater ein neues iPhone, während er sein vorheriges Gerät der Tochter vererbt. Natürlich behalten beide ihre SIM-Karten und setzen diese in ihre neuen Handys ein.

Theoretisch ist noch nicht einmal eine Telefonnummer auf alle Ewigkeit auf eine SIM-Karte geprägt, denn es kann ja sein, dass man einmal den Mobilfunk-Anbieter wechselt und die Nummer vom alten Anbieter zum neuen Anbieter übernimmt – die dann allerdings einer neuen SIM-Karte zugeordnet werden muss.

Fest steht jedenfalls, dass jede SIM-Karte zu einem Zeitpunkt nur in einem Mobilfunkgerät stecken kann. Die Frage ist nur: Müssen wir wissen, wenn SIM-Karten von Zeit zu Zeit das Gerät wechseln, in welchen Geräten eine SIM-Karte vorher steckte – oder reicht uns der aktuelle Zustand? Ich würde sagen: Der aktuelle Zustand reicht. Und da jede SIM-Karte nur einem Mobilfunkgerät zugeteilt werden kann, brauchen wir sogar noch nicht einmal eine 1:n-Beziehung für die Zuteilung der beiden Objekte, sondern es reicht eine

Handyverwaltung II: Personen verwalten

Im zweiten Teil der Artikelreihe zum Thema »Handyverwaltung« kümmern wir uns um die Formulare, die zur Verwaltung der Personen notwendig sind. Dazu gehören das Formular zur Anzeige der Personenübersicht mit allen in der Datenbank gespeicherten Personen sowie ein Detailformular, mit dem der Benutzer die Details der gespeicherten Personen bearbeiten sowie neue Personen anlegen kann.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1704_Handyverwaltung.accdb**.

Vorbereitungen

Im Artikel **Handyverwaltung I: Datenmodell** haben wir die Tabellen für die hier vorgestellte Lösung definiert. Wenn Sie die Tabellen nun als Datenherkunft von Formularen und Berichten nutzen und die Felder der Tabellen und der darauf basierenden Abfragen in den Entwurf der Zielobjekte ziehen, werden üblicherweise Beschriftungen zu den resultierenden Steuerelementen wie Textfeldern, Kombinationsfeldern oder Kontrollkästchen hinzugefügt, die genau den Feldnamen entsprechen, die in den Tabellen definiert wurden. In den meisten Fällen können Sie die Feldnamen gleich weiterverwenden, aber manchmal sind Korrekturen erforderlich – beispielsweise, weil der Feldname wegen der üblichen Konventionen keinen Bindestrich wie in **Person-ID** enthält, sondern mit **PersonID** benannt wird. Andere gute Beispiele sind das Feld zum Speichern der E-Mail-Adresse, welches oft **EMail** genannt wird, oder das Fremdschlüsselfeld **AnredeID**, das wir mit **Anrede** bezeichnen wollen. Im Bezeichnungsfeld zu diesem Feld soll aber dann der Text **E-Mail** stehen. Es geht weiter mit Feldern, deren Umlaute beispielsweise durch ae statt ä ersetzt wurden.

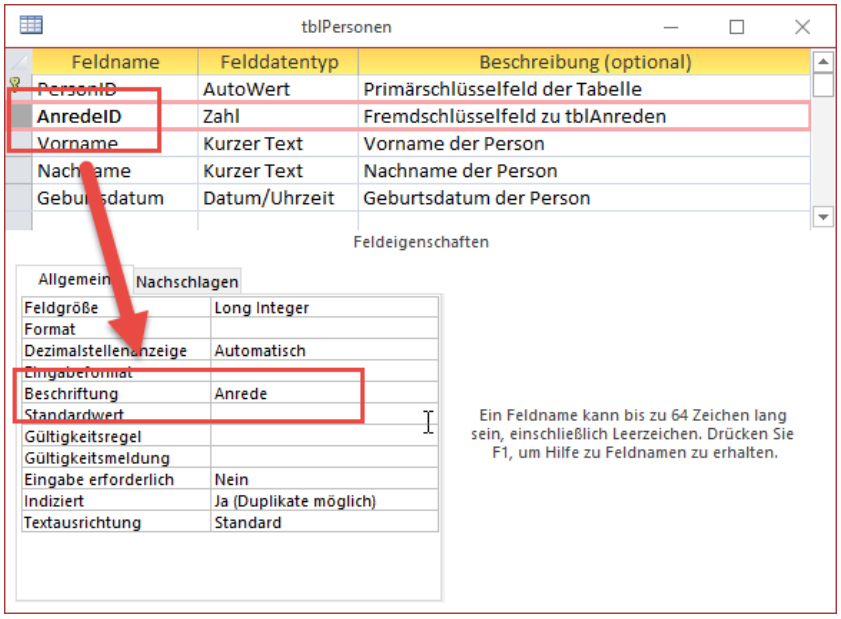


Bild 1: Anpassen der Beschriftung für zu erstellende Bezeichnungsfelder

Wenn Sie sich die Arbeit ersparen wollen, die Bezeichnungsfelder zu diesen Feldern nachträglich anzupassen, können Sie dies auch direkt im Tabellenentwurf erledigen. Dazu müssen Sie lediglich die Eigeschaft **Beschriftung** der jeweiligen Felder auf den gewünschten Wert einstellen, also beispielsweise für das Feld **AnredeID** den Wert **Anrede** hinzufügen (siehe Bild 1).

Die neuen Beschriftungen werden dann übrigens gleich in der Datenblattansicht angezeigt (siehe Bild 2) – und später in allen Bezeichnungsfeldern von

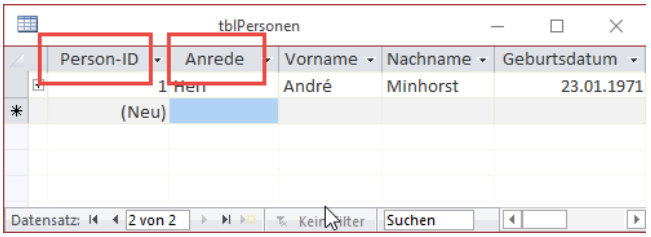


Bild 2: Die neuen Beschriftungen werden bereits in der Datenblattansicht angezeigt.

Steuerelementen, die an die entsprechenden Felder der Datenherkunft gebunden sind.

Es gibt noch weitere Stellen, an denen wir das Aussehen der hinzugefügten Bezeichnungsfelder manipulieren können.

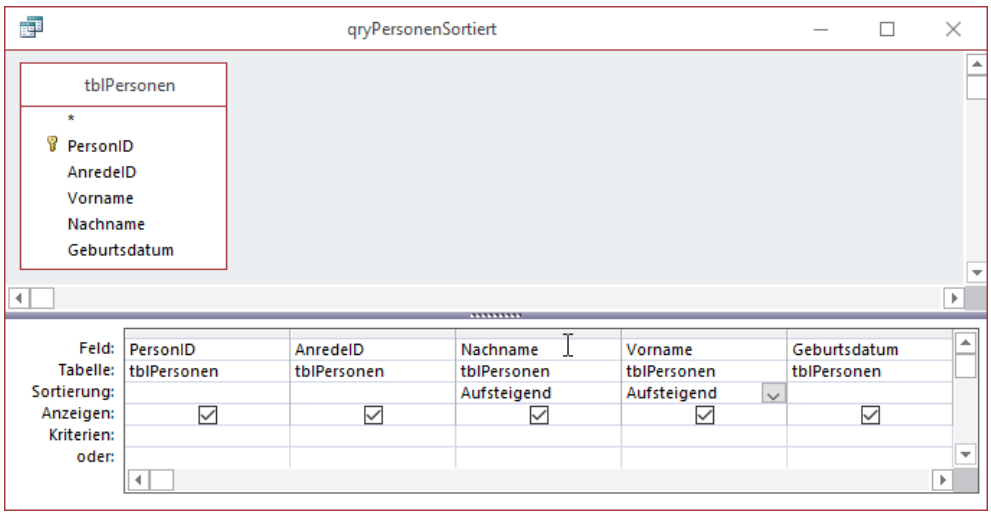


Bild 3: Entwurf des Unterformulars mit der Datenblattansicht

Personenübersicht erstellen

Als Erstes fügen wir der Datenbank eine Übersicht der Personen hinzu. Diese besteht aus einem Hauptformularnamens **frmPersonenuebersicht** und einem Unterformular namens **sfmPersonenuebersicht**. Das Unterformular erhält eine Abfrage auf Basis der Tabelle **tblPersonen** als Datenherkunft, die wir wie in Bild 3 definieren. Dabei ziehen wir alle Felder der Tabelle in das Entwurfsraster der Abfrage (beachten Sie die Reihenfolge der Felder **Nachname** und **Vorname**) und legen dann für die Felder **Nachname** und **Vorname** als Sortierung den Wert **Aufsteigend** fest. Speichern Sie die Abfrage dann unter dem Namen **qryPersonenSortiert**.

Danach fügen wir diese Abfrage dem Unterformular **sfmPersonenuebersicht** als Datenherkunft hinzu und ziehen alle Felder aus der Feldliste in den Detailbereich des Formularentwurfs. Stellen Sie die Eigenschaft **Standardansicht** auf den Wert **Datenblatt** ein und speichern und schließen Sie das Formular.

Danach öffnen Sie das neue Hauptformular **frmPersonenuebersicht** in der Entwurfsansicht und ziehen das Unterformular **sfmPersonenuebersicht** aus dem Navigationsbereich in den Detailbereich des Hauptformulars. Stellen Sie die Eigenschaften **Horizontaler Anker** und **Vertikaler Anker** des Unterformular-Steuerelements auf **Beide** ein. Falls Sie das automatisch mit hinzugefügte Bezeichnungsfeld verwenden wollen, stellen Sie die Eigenschaft **Horizontaler Anker** auf **Oben** und **Vertikaler Anker** auf **Links** ein. Außerdem erhalten die Eigenschaften **Navigationsschaltflächen**, **Datensatzmarkierer**, **Trennlinien** und **Bildlaufleisten** des Hauptformulars jeweils den Wert **Nein** und die Eigenschaft **Automatisch zentrieren** den Wert **Ja**.

Der Entwurf des Hauptformulars inklusive Unterformular sollte nun wie in Bild 4 aussehen.

Wenn Sie nun in die Formularansicht des Hauptformulars wechseln, finden Sie die Eingabemaske von

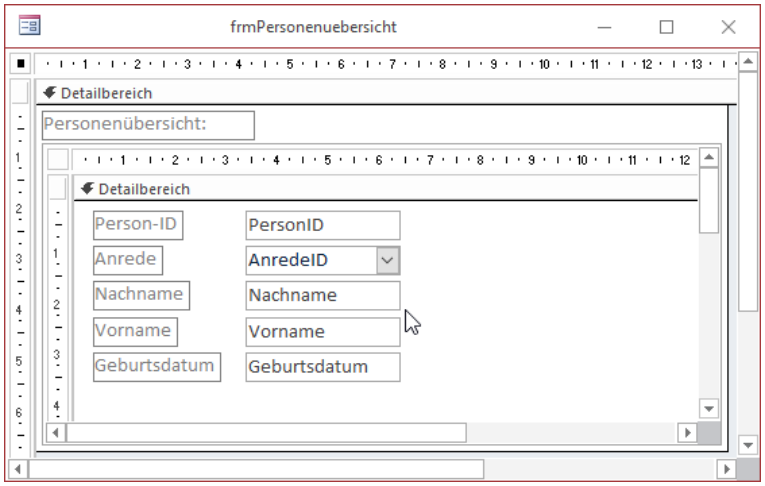


Bild 4: Entwurf des Unterformulars mit der Datenblattansicht

Bild 5 vor. Hier fehlen nun noch ein paar Optionen, um die Details zu diesem Datensatz anzuzeigen. Die in der Tabelle **tblPersonen** gespeicherten Daten zur jeweiligen Person lassen sich zwar direkt hier ändern, aber wir erhalten keinen Einblick etwa auf die der Person zugeordneten Verträge oder Mobilfunkgeräte. Dies wollen wir einem eigenen Formular abbilden. Um dieses zu öffnen, müssen wir jedoch noch geeignete Steuerelemente zum Formular **frmPersonenuebersicht** hinzufügen.

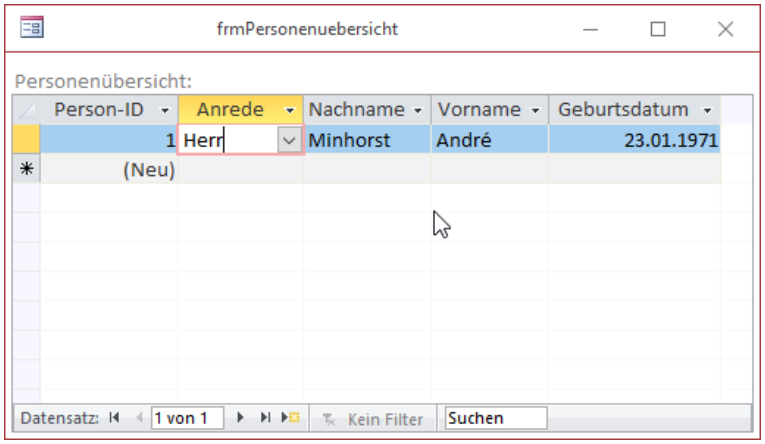


Bild 5: Datenblattansicht von Haupt- und Unterformular

Das Formular frmPersonendetails

Zunächst einmal wollen wir jedoch das Detailformular erstellen – zunächst ohne die Daten aus den verknüpften Tabellen. Dazu erstellen Sie ein neues Formular namens **frmPersonendetails**. Binden Sie es wie das Unterformular **sfmPersonenuebersicht** über die Eigenschaft **Datenherkunft** an die Tabelle **tblPersonen**. Ziehen Sie dann alle Felder dieser Tabelle in den Detailbereich des Entwurfs. Außerdem wollen wir dem Formular bereits eine Schaltfläche zum Schließen des Formulars hinzufügen. Dieses beschriften wir mit dem Text **OK** und weisen ihm den Namen **cmdOK** zu (siehe Bild 6).

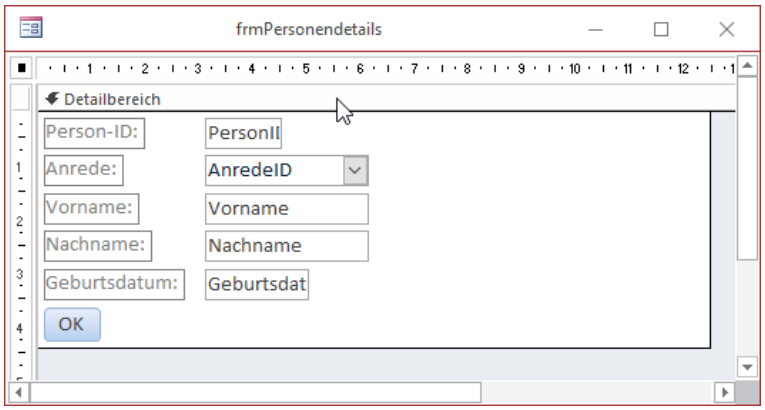


Bild 6: Entwurf des Formulars **frmPersonendetails**

Dieses Formular soll es nicht ermöglichen, etwa mit den Navigationsschaltflächen durch einzelne Datensätze zu navigieren oder auch den aktuellen Datensatz mit dem Datensatzmarkierer zu markieren, um diesen dann löschen zu können. Also entfernen wir diese und andere Steuerelemente durch Einstellen der Eigenschaften **Datensatzmarkierer**, **Navigationsschaltflächen**, **Trennlinien** und **Bildlaufleisten** auf den Wert **Nein**. Die Eigenschaft **Automatisch zentrieren** soll den Wert **Ja** erhalten.

Öffnen der Personendetails von der Personenübersicht aus

Nun soll der Benutzer, wenn er das Formular **frmPersonenuebersicht** vor sich sieht, einen der angezeigten Einträge in der Detailansicht öffnen können, also im Formular **frmPersonendetails**. Dazu gibt es verschiedene Möglichkeiten. Die offensichtlichste wäre, eine Schaltfläche mit der Beschriftung **Details anzeigen** hinzuzufügen, die dann per Mausklick den aktuell markierten Datensatz öffnet. Eine andere Alternative wäre, die Steuerelemente der Datenblattansicht mit dem Ereignis **Beim Doppelklicken** auszustatten, damit der Benutzer den gewünschte Eintrag per Doppelklick in der Detailansicht anzeigen kann.

Beginnen wir mit der Schaltfläche **cmdDetailsAnzeigen**. Diese legen wir im Hauptformular direkt unter dem Unterformular an. Da wir für das Unterformular die Eigenschaft **Vertikaler Anker** auf **Beide** eingestellt haben, wird dieses beim Vergrößern der Höhe des Formulars nach unten vergrößert. Also müssen wir die Eigenschaft **Vertikaler Anker** für die Schaltfläche auf **Unten** einstellen, damit dieses nicht unter

```
Private Sub cmdDetailsAnzeigen_Click()
    Dim lngPersonID As Long
    lngPersonID = Me!sfmPersonenuebersicht.Form!PersonID
    DoCmd.OpenForm "frmPersonendetails", WindowMode:=acDialog, WhereCondition:="PersonID = " & lngPersonID
End Sub
```

Listing 1: Öffnen der Detailansicht einer Person

dem Unterformular verschwindet. Schließlich legen Sie eine Ereignisprozedur für diese Schaltfläche an, indem Sie für die Ereigniseigenschaft **Beim Klicken** den Wert **[Ereignisprozedur]** auswählen und auf die Schaltfläche mit den drei Punkten (**...**) klicken. Die nun im VBA-Editor angezeigte Prozedur ergänzen Sie wie in Listing 1.

Die Prozedur ermittelt in der ersten Anweisung den Primärschlüsselwert des aktuell markierten Datensatzes des Unterformulars, und zwar über den Ausdruck **Me!sfmPersonenuebersicht.Form!PersonID**, und speichert diesen in der Variablen **lngPersonID**. Diesen Wert verwendet die zweite Anweisung, um einen Parameter namens **WhereCondition** zusammenzustellen, welcher an das aufgerufene Formular **frmPersonendetails** übergeben gleich als Filter angewendet wird. Der erste Parameter **WindowMode** erhält den Wert **acDialog**, damit das aufgerufene Formular solange den Fokus behält, bis der Benutzer dieses wieder schließt.

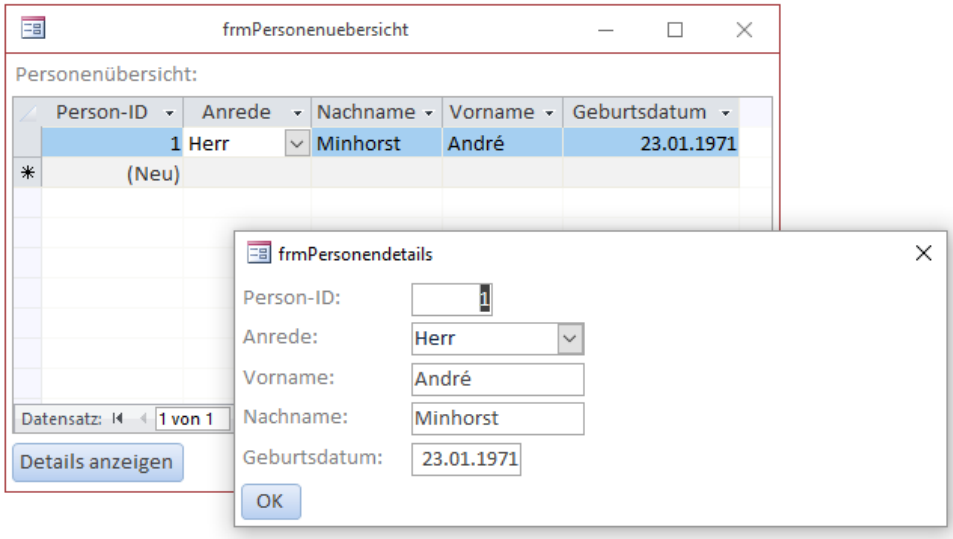


Bild 7: Entwurf des Formulars **frmPersonendetails**

Das Ergebnis sieht dann wie in Bild 7 aus. Das als modaler Dialog geöffnete Formular **frmPersonendetails** zeigt genau den gewünschten Datensatz an.

Für die Schaltfläche **cmdOK** des Formulars **frmPersonendetails** legen wir die folgende Ereignisprozedur an, damit der Benutzer dieses später wieder schließen kann:

```
Private Sub cmdOK_Click()
    DoCmd.Close acForm, Me.Name
End Sub
```

Wechseln zu anderen Datensätzen im Detailformular verhindern

Damit der Benutzer tatsächlich nur den aufgerufenen Datensatz im Formular **frmPersonendetails** anzeigen kann und nicht etwa zu einem leeren, neuen Datensatz wechselt, indem er etwa mit der Tabulator-Taste alle Felder durchläuft und dann nach dem Verlassen des letzten Feldes beim neuen Datensatz landet, stellen Sie die Eigenschaft **Zyklus** des Formulars **frmPersonendetails** auf den Wert **Aktueller Datensatz** ein.

Personendetails per Doppelklick öffnen

Die zweite Alternative zum Öffnen der Details zum gewünschten Personen-Datensatz soll ein Doppelklick auf einen der Einträge im Unterformular **sfmPersonenuebersicht** sein. Dazu legen wir für alle

```
Private Sub DblClick(cancel As Integer)
    Dim lngPersonID As Long
    lngPersonID = Me!PersonID
    DoCmd.OpenForm "frmPersonendetails", WindowMode:=acDialog, WhereCondition:="PersonID = " & lngPersonID
End Sub
```

Listing 2: Öffnen der Detailansicht einer Person vom Unterformular aus

gebundenen Steuerelemente im Unterformular die Ereignisprozedur **Beim Doppelklicken** an, die sämtlich mit der gleichen Code-Zeile füllen:

```
Private Sub AnredeID_DblClick(cancel As Integer)
    Call DblClick(cancel)
End Sub

Private Sub Geburtsdatum_DblClick(cancel As Integer)
    Call DblClick(cancel)
End Sub

Private Sub Nachname_DblClick(cancel As Integer)
    Call DblClick(cancel)
End Sub

Private Sub PersonID_DblClick(cancel As Integer)
    Call DblClick(cancel)
End Sub

Private Sub Vorname_DblClick(cancel As Integer)
    Call DblClick(cancel)
End Sub
```

Die dadurch aufgerufene Prozedur **DblClick** definieren wir wie in Listing 2. Sie enthält annähernd die gleichen Anweisungen wie die Prozedur, die durch die Schaltfläche **cmdDetailsAnzeigen** im Hauptformular, allerdings wird der Primärschlüsselwert des anzuzeigenden Datensatzes auf andere Art und Weise ermittelt – nämlich durch einfaches Abfragen von **Me!PersonID**. Das Resultat des Doppelklicks auf einen der Einträge ist jedoch das Gleiche wie beim Klick auf die Schaltfläche **cmdDetailsAnzeigen** – das Formular **frmPersonendetails** wird mit dem gewünschten Datensatz geöffnet.

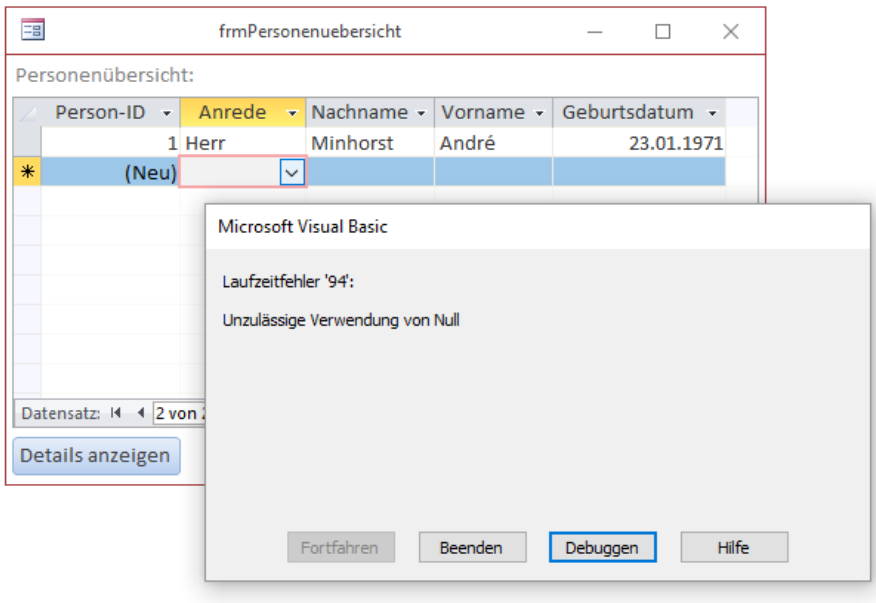


Bild 8: Entwurf des Formulars **frmPersonendetails**

Fehler beim Doppelklick auf den neuen, leeren Datensatz

Es tritt allerdings ein Fehler auf, wenn der Benutzer den leeren, neuen Datensatz im Unterformular markiert und dann auf die Schaltfläche **Details anzeigen** klickt oder auch direkt doppelt auf den neuen, leeren Datensatz klickt (siehe Bild 8). Der Fehler tritt auf, weil der neue, leere Datensatz für das Feld **PersonID** den Wert **Null** enthält. Wenn Sie diesen der **Long**-Variablen **lngPersonID** zuweisen, dann tritt logischerweise der Fehler aus der Abbildung auf. Um diesen zu unterbinden, gibt es mehrere Möglichkeiten. Der einfachs-

Meldungen bei Datensatzänderungen

Access bietet einige Sicherheitsmaßnahmen, um den Benutzer vor unbewussten und nicht wiederherstellbaren Änderungen von Daten zu schützen. Dabei erscheinen zu bestimmten Gelegenheiten wie etwa vor dem Löschen von Datensätzen entsprechende Warnmeldungen. Das kann auch geschehen, wenn der Benutzer Daten durch den Aufruf von VBA-Anweisungen löschen will oder dies per Aktionsabfrage erledigt. Dieser Artikel zeigt, wo Access eingreift und wie Sie das Verhalten für Ihre Zwecke anpassen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1705_MeldungenBeiDatensatzaenderungen.accdb**.

Wo tauchen Warnmeldungen auf?

Als Erstes sehen wir uns an, wo überhaupt Warnmeldungen auftauchen, wenn der Benutzer Datensatzänderungen durchführt.

Dies bereits in der Datenblattansicht einer Tabelle der Fall. Versuchen Sie hier, einen Datensatz zu löschen, erscheint die Meldung aus Bild 1. Diese taucht auf, nachdem der zu löschende Datensatz bereits verschwunden ist. Klicken Sie hier auf **Nein**, wird der Datensatz wiederhergestellt.

Die gleiche Meldung erscheint auch bei allen weiteren Datenblattansichten, also etwa von Abfragen oder in Formularen.

Die zweite Gelegenheit zum Anzeigen von Meldungen ist das Ausführen von Aktionsabfragen wie zum Beispiel die Abfrage zum Löschen von Daten aus Bild 2. Auch hier bewirkt ein Klick auf

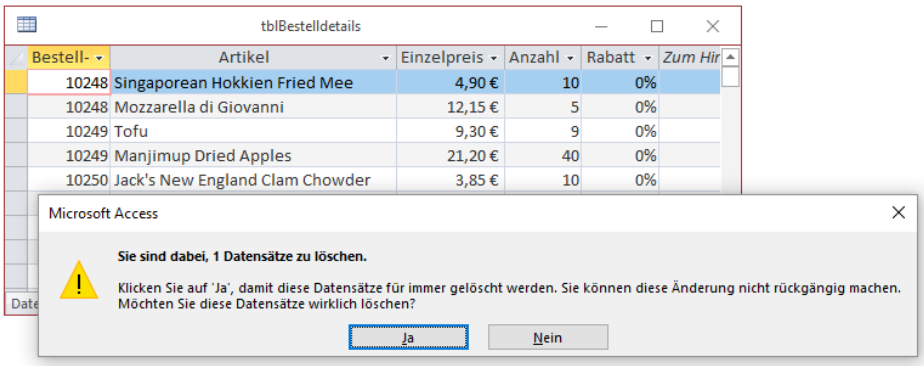


Bild 1: Meldung beim manuellen Löschen aus der Datenblattansicht

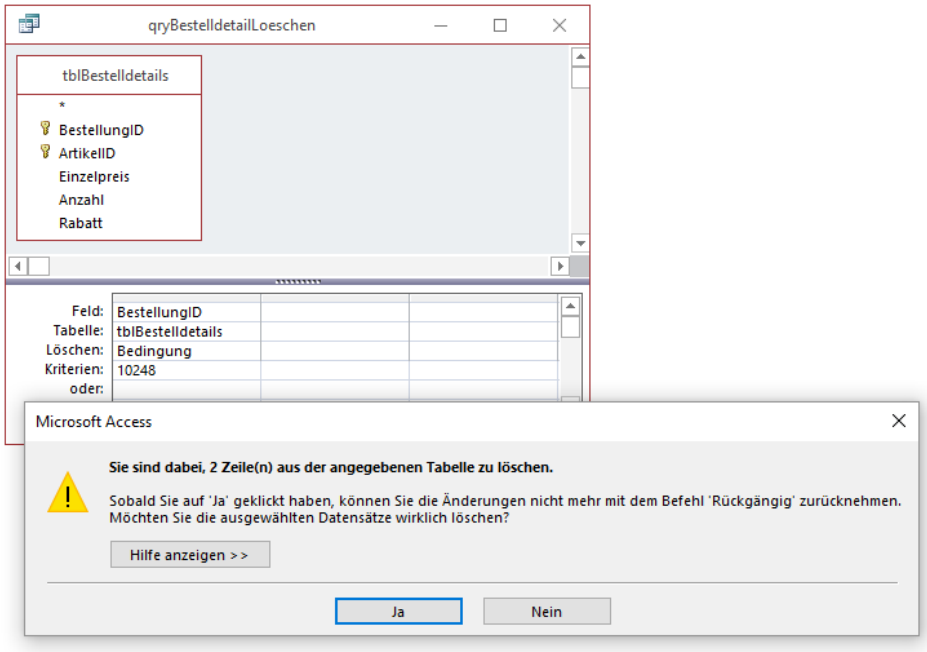


Bild 2: Meldung beim Löschen per Aktionsabfrage

Ja, dass die Abfrage tatsächlich ausgeführt wird und **Nein**, dass die Daten erhalten bleiben.

Nächste Gelegenheit ist das Löschen von Datensätzen über DAO, und zwar mit der **Delete**-Methode für den aktuell im Recordset markierten Datensatz.

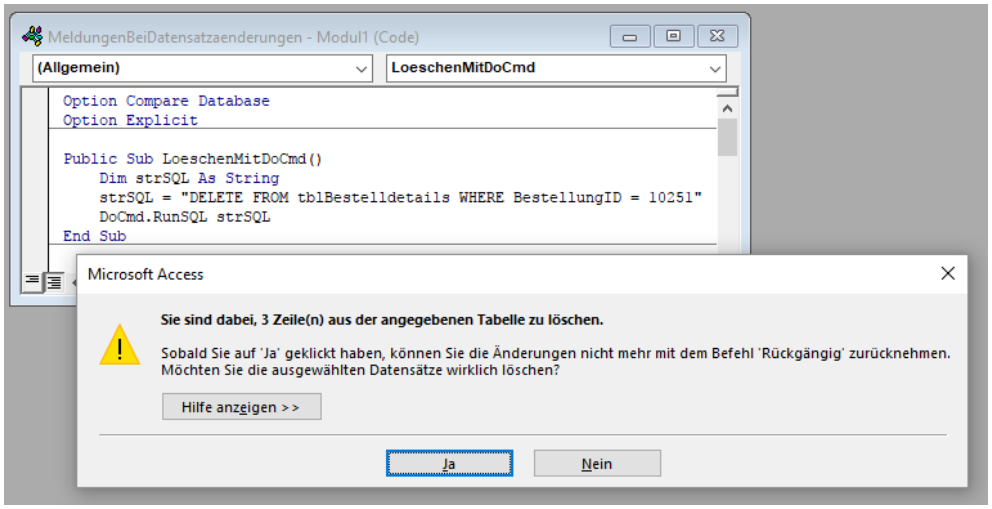


Bild 3: Meldung beim Löschen per **DoCmd.RunSQL**

Folgende Prozedur öffnet ein Recordset auf Basis der Tabelle **tblBestelldetails**, bewegt den Datensatzzeiger zum ersten und wieder zum ersten Datensatz, damit **rst.RecordCount** die korrekte Anzahl Datensätze ausgibt, löscht dann den aktuellen Datensatz und gibt erneut die Anzahl der Datensätze aus:

```
Public Sub LoeschenMitDAO_I()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM ↵
                    tblBestelldetails", dbOpenDynaset)
    rst.MoveLast
    rst.MoveFirst
    Debug.Print rst.RecordCount
    If Not rst.EOF Then
        'ersten Datensatz löschen
        rst.Delete
    End If
    Debug.Print rst.RecordCount
End Sub
```

Im Direktbereich können Sie ablesen, dass genau ein Datensatz gelöscht wurde – wobei allerdings diesmal keine Meldung mit einer Rückfrage an den Benutzer ausgegeben wurde.

Aktionen im Recordset führen also direkt zum Löschen des Datensatzes.

Wie sieht es mit dem Löschen per **Execute**-Anweisung aus? Schauen wir es uns anhand des folgenden Beispiels an:

```
Public Sub LoeschenMitDAO_II()
    Dim db As DAO.Database
    Set db = CurrentDb
    db.Execute "DELETE FROM tblBestelldetails ↵
        WHERE BestellungID = 10250", dbFailOnError
    Debug.Print db.RecordsAffected
End Sub
```

Hier haben wir die **Execute**-Methode des **Database**-Objekts mit einer **DELETE**-Anweisung aufgerufen. Es werden Datensätze gelöscht, aber auch hier gibt es keine Rückfrage an den Benutzer.

Hat der Benutzer gar keine Möglichkeit, bei Datenänderungen über VBA über die anstehenden Änderungen informiert zu werden – außer gegebenenfalls über selbst programmierte Meldungen? Doch, wenn Sie die Methode **RunSQL** des **DoCmd**-Objekts wie in Bild 3 aufrufen.

Klicken Sie hier auf **Ja**, wird die Abfrage durchgeführt, **Nein** führt allerdings zu einem Laufzeitfehler (siehe Bild 4). In diesem Fall müssen Sie noch eine Fehlerbehandlung zum Code hinzufügen, die beispielsweise wie folgt aussieht:

Handyverwaltung III: Provider verwalten

Im dritten Teil der Artikelreihe zum Thema »Handyverwaltung« setzten wir die Arbeit an den Formularen der Lösung fort. In diesem Fall wollen wir die Provider verwalten. Um innerhalb der Lösung nicht immer die gleichen Techniken zu präsentieren, nutzen wir diesmal ein Listenfeld zur Auswahl der unterschiedlichen Datensätze, das sich allerdings im gleichen Formular wie die Darstellung des aktuell ausgewählten Providers befindet.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1705_Handyverwaltung.accdb**.

Vorbereitungen

Im Gegensatz zum vorherigen Teil der Artikelreihe, in dem wir beschrieben haben, wir wir die Personen der Handyverwaltung in entsprechenden Formularen darstellen, gibt es zwei wichtige Unterschiede: Erstens kümmern wir uns hier lediglich nur um eine einzige Tabelle ohne weitere Lookup-Felder, nämlich um **tblProvider**. Zweitens wollen wir die Daten der Tabelle nicht in zwei Formularen darstellen, wie es bei den Personen der Fall war: Dort haben wir nämlich ein Übersichtsformular genutzt, das ein Unterformular mit allen Datensätzen der Tabelle **tblPersonen** enthielt sowie ein Detailformular, dass nach der Auswahl eines der Datensätze geöffnet wurde und dann diesen Datensatz anzeigte. Nein, wir wollen diesmal nur ein einziges Formular nutzen: Dieses soll im oberen Bereich ein Listenfeld mit allen Einträgen der Tabelle **tblProvider** anzeigen. Das gleiche Formular soll an die Tabelle **tblProvider** gebunden sein und die Daten aus den Feldern der Tabelle in daran gebundenen Steuerelementen anzeigen.

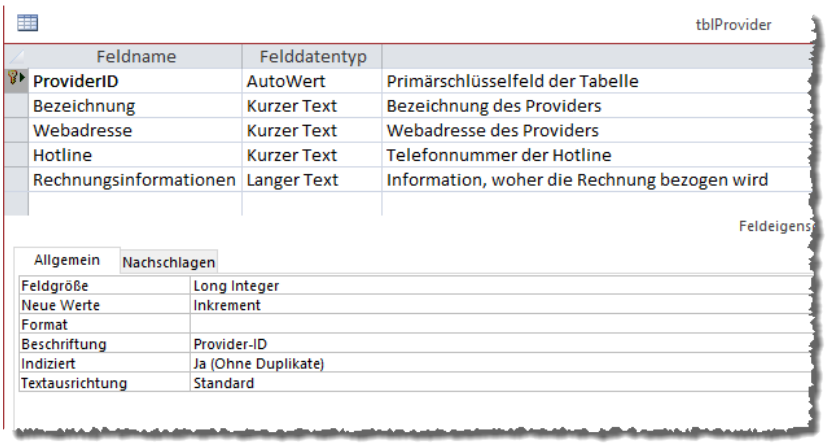


Bild 1: Anpassen der Beschriftung des Feldes **Provider-ID** für das Bezeichnungsfelder

Tabelle tblProvider vorbereiten

Wir wollen die Nutzung der Felder der Tabelle **tblProvider** in den Tabellen und Steuerelementen möglichst einfach gestalten. Deshalb schauen wir uns zuvor an, wo die Feldnamen nicht mit den Bezeichnungen übereinstimmen, die wir in den Abfragen, Formularen, Berichten und Steuerelementen als Bezeichnungsfelder nutzen wollen und passen diese über die Feldeigenschaft **Beschriftung** entsprechend an. In dieser Tabelle passen die Feldbeschriftungen **Bezeichnung**, **Webadresse**, **Hotline** und **Rechnungsinformationen** bereits, sodass wir nur für das Primärschlüsselfeld **ProviderID** die Feldeigenschaft **Beschriftung** in den Wert **Provider-ID** ändern müssen (siehe Bild 1).

Formular erstellen

Das Formular soll **tblProvider** heißen. Im Gegensatz zum vorherigen Teil der Artikelreihe, wo wir die beiden Formulare **frmPersonenuebersicht** und **frmPersonendetails** verwendet haben, zeigen wir ja hier die Übersicht und die Details in einem Formular an. Also brauchen wir auch keine Erweiterung mehr wir **...uebersicht** oder **...details**, sondern können das Formular schlicht nach dem bennen, was es auch anzeigt – nämlich Provider.

Datenherkunft zuweisen

Das Formular **frmProvider** soll die Daten der Tabelle **tblProvider** anzeigen, allerdings nicht unbedingt in

der dort per Primärschlüssel vorgegebenen Reihenfolge, sondern alphabetisch nach der Bezeichnung des Providers sortiert. Dazu erstellen Sie mit dem Ribbon-Befehl eine neue Abfrage namens **qryProviderNachBezeichnung**, der Sie alle Felder der Tabelle hinzufügen und außerdem für das Feld **Bezeichnung** in der Zeile **Sortierung** den Wert **Aufsteigend** auswählen (siehe Bild 2).

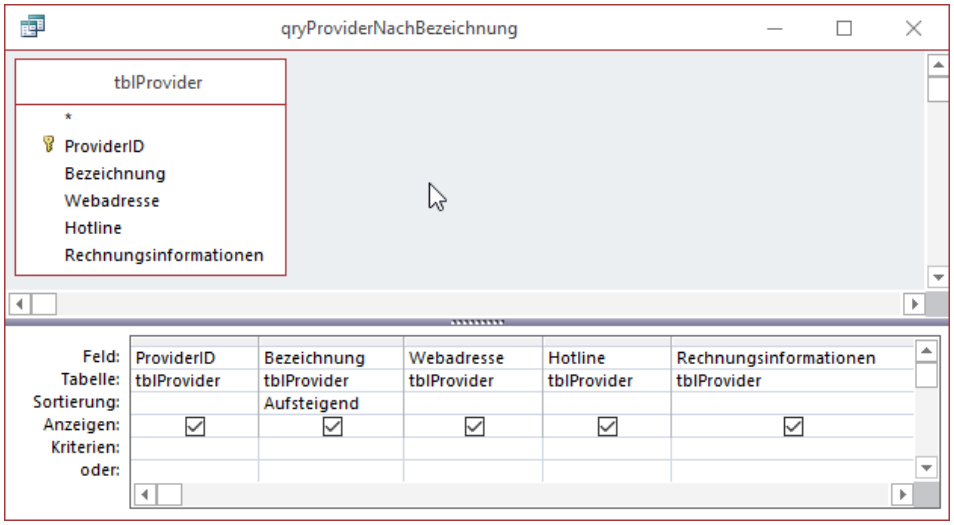


Bild 2: Abfrage, welche die Provider nach der Bezeichnung sortier

Diese Abfrage weisen Sie dem Formular nun für die Eigenschaft **Datenherkunft** zu. Danach wechseln Sie über den Ribbon-Eintrag **Entwurf|Tools|Vorhandene Felder hinzufügen** zur Feldliste und ziehen von dort aus alle Einträge direkt in den Detailbereich des Formularentwurfs (siehe Bild 3).

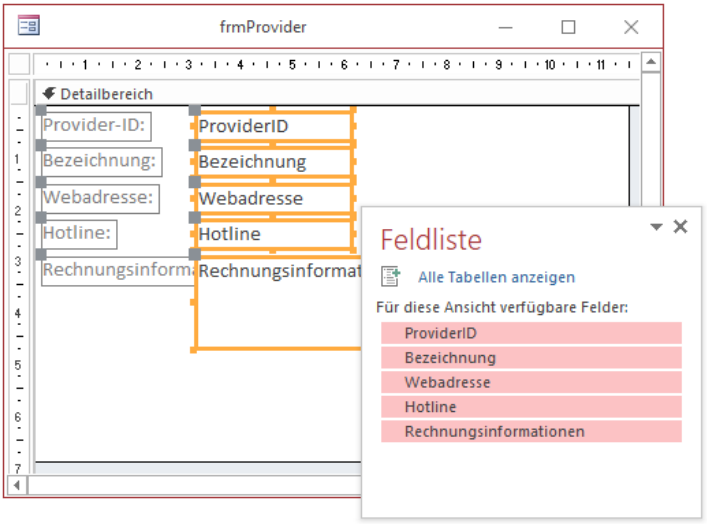


Bild 3: Hinzufügen der gebundenen Felder zum Formular **frmProvider**

Wir wollen jeweils nur einen Datensatz in diesem Formular anzeigen und nur über das Listenfeld einen anderen Datensatz auswählen können, dessen Details in den gebundenen Steuerelementen des Formulars angezeigt werden. Also benötigen wir standardmäßig im Formular aktivierten Elemente wie **Navigationsschaltflächen**, **Datensatzmarkierer**, **Trennlinien** und **Bildlaufleisten** nicht und können diese durch Einstellen der gleichnamigen Eigenschaften auf den Wert **Nein** deaktivieren. Wenn das Formular geöffnet wird, soll es jedoch in der Mitte des Access-Fensters erscheinen, weshalb wir die Eigenschaft **Automatisch zentrieren** auf den Wert **Ja** einstellen.

Listenfeld als Übersicht hinzufügen

Nun fehlt noch das Listenfeld, welches die Übersicht der Einträge anzeigt und die Auswahl eines der Einträge zur Anzeige der Details im Formular erlaubt. Dazu verschieben Sie zunächst die gebundenen Felder um einige Zentimeter nach unten, sodass Sie das Listenfeld darüber platzieren können. Dazu markieren Sie einfach alle Elemente mit der Tastenkombination **Strg + A** und verschieben diese dann beispielsweise mit der **Nach unten**-Taste nach unten.

Haben Sie genügend Platz geschaffen, fügen Sie über das Ribbon ein neues Listenfeld zum Formular hinzu. Dies sieht dann beispielsweise wie in Bild 4 aus. Hier haben wir das Listenfeld noch etwas angepasst. Dazu haben wir es zunächst unterhalb des dazugehörigen Bezeichnungsfeldes positioniert. Dazu markieren Sie das Listenfeld und ziehen es an dem

viereckigen Kästchen oben links an die gewünschte Stelle. Legen Sie nun auch gleich den Namen des Listenfeldes fest, und zwar mit dem Wert **lstProviderauswahl** für die Eigenschaft **Name**.

Zuweisen einer Datensatzherkunft zum Listenfeld

Das Listenfeld zeigt bisher noch keine Daten an, was wir ändern, indem wir die Eigenschaft **Datensatzherkunft** auf den Namen der bereits für das Formular als Datenherkunft verwendeten Abfrage **qryProviderNachBezeichnung** einstellen. Nun müssen wir uns überlegen, welche Felder der Datensatzherkunft das Listenfeld anzeigen soll. Grundsätzlich reicht es aus, wenn es nur den Namen des Providers anzeigt – die Webadresse, die Hotline und die Rechnungsinformationen benötigen wir ja nie alle gleichzeitig.

Die Datensatzherkunft liefert aber nun gleich alle fünf Felder der Tabelle. Damit das Listenfeld nur das Feld **Bezeichnung** anzeigt und das Feld **ProviderID** als gebundene Spalte verwendet, sind Einstellungen in zwei Eigenschaften nötig. Der Wert **2** für die Eigenschaft **Spaltenanzahl** des Listenfeldes sorgt dafür, dass diese nur die ersten beiden Felder der Datensatzherkunft, also der Abfrage **qryProviderNachBezeichnung** anzeigt. Aber wir wollen doch nur ein Feld anzeigen? Ja, allerdings ist standardmäßig das erste Feld der Datensatzherkunft als das Feld für die gebundene Spalte festgelegt (Eigenschaft **Gebundene Spalte** mit dem Wert **1**) und zweitens stellen wir im zweiten Schritt ja noch die Spaltenbreiten für die Anzeige der beiden Spalten ein.

Die dazu verwendete Eigenschaft heißt **Spaltenbreiten** und erhält den Wert **0**. Aber warum **0**, wir wollen doch die Breite für zwei Spalten einstellen – und welche Einheit wird denn dann überhaupt verwendet? Ganz einfach: Die Einheit ergänzt Access automatisch mit **cm**, also Zentimeter, und wenn wir nur die Breite für die erste Spalte angeben, hier also **0cm**, dann nimmt die zweite Spalte einfach den verbleibenden Platz im Listenfeld ein. Das Ergebnis sieht dann

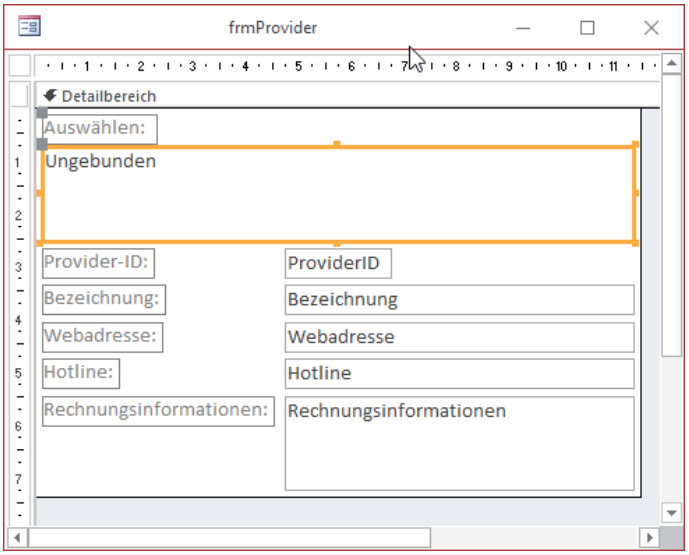


Bild 4: Hinzufügen des Listenfeldes zur Auswahl des anzuzeigenden Providers

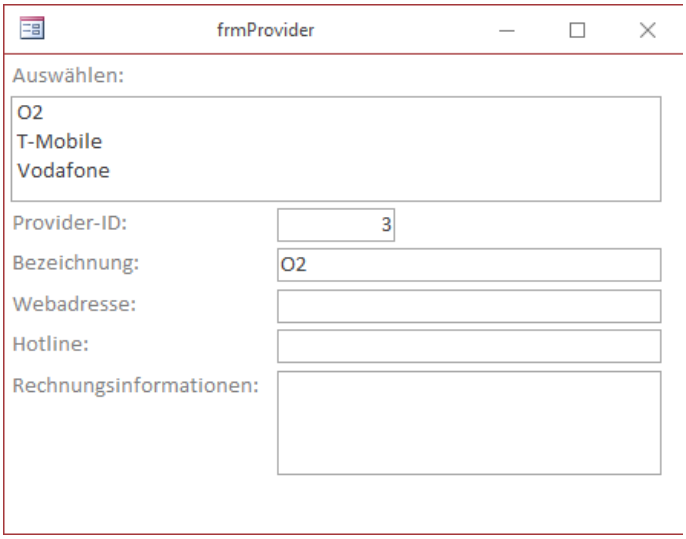


Bild 5: Das Listenfeld zeigt nur den Wert des Feldes **Bezeichnung** der zugrunde liegenden Datensatzherkunft an.

wie in Bild 5 aus. Das Listenfeld enthält in der ersten Spalte, welche mit der Breite **0cm** angezeigt wird, den Primärschlüsselwert der Tabelle **tblProvider**. Damit kann dann leicht der aktuell ausgewählte Wert ermittelt werden.

Listenfeld zur Auswahl nutzen

Nun wollen wir dafür sorgen, dass die Auswahl eines der Einträge im Listenfeld auch den gewünschten Datensatz im Formular anzeigt. Dazu fügen Sie dem Lis-

tenfeld eine Ereignisprozedur hinzu, die durch das Ereignis **Nach Aktualisierung** ausgelöst wird. Dazu tragen Sie für die Eigenschaft **Nach Aktualisierung** den Wert **[Ereignisprozedur]** ein und klicken auf die Schaltfläche mit den drei Punkten. Die nun im VBA-Editor erscheinende leere Ereignisprozedur füllen Sie wie folgt:

```
Private Sub lstProviderauswahl_AfterUpdate()
    Dim lngProviderID As Long
    Dim strSQL As String
    lngProviderID = Me!lstProviderauswahl
    strSQL = "ProviderID = " & lngProviderID
    Me.Filter = strSQL
    Me.FilterOn = True
End Sub
```

Die erste Anweisung nach den beiden Deklarationszeilen füllt den aktuellen Wert des Listenfeldes in die Variable **lngProviderID** ein. Die zweite setzt eine SQL-Filterbedingung zusammen und speichert diese in der Variablen **strSQL**. Der Wert könnte etwa so aussehen:

```
ProviderID = 1
```

Dieser Ausdruck wird dann der Eigenschaft **Filter** des Formulars zugewiesen, danach aktiviert die Prozedur den Filter durch Einstellen der Eigenschaft **FilterOn** auf den Wert **True**. Das Ergebnis zeigt Bild 6. Wenn wir etwa den dritten Eintrag auswählen, erscheinen die passenden Detaildaten im Unterformular.

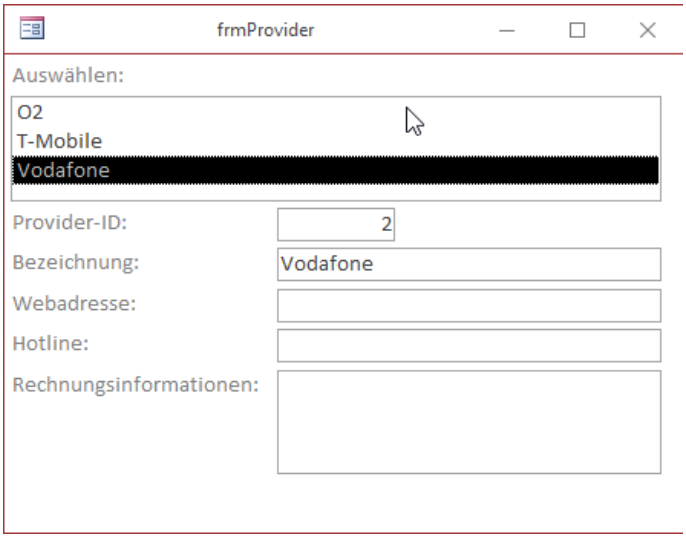


Bild 6: Nach der Auswahl im Listenfeld zeigt das Formular in den übrigen Steuerelementen die Details zu diesem Provider an.

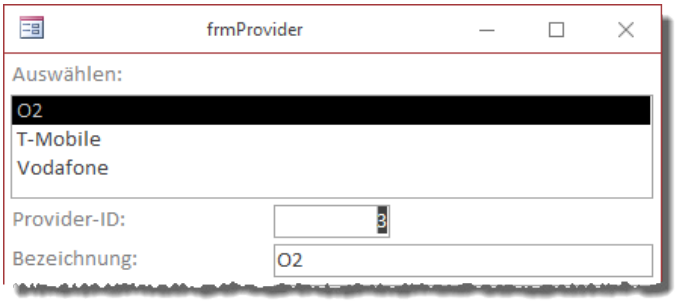


Bild 7: Das Listenfeld zeigt nun gleich nach dem Laden den gleichen Datensatz an, den auch das Formular enthält.

Listenfeld und Datenherkunft beim Öffnen synchronisieren

Nun sollen das Listenfeld und die übrigen Steuerelemente des Formulars beim Öffnen des Formulars den gleichen Datensatz anzeigen. Das ist nicht automatisch der Fall, denn das Listenfeld weist beim Öffnen des Formulars zunächst den Wert **Null** auf, da kein Eintrag markiert ist. Also sorgen wir mit einer weiteren Ereignisprozedur dafür, dass das Listenfeld beim Öffnen auf den gleichen Datensatz eingestellt wird, den auch das Formular anzeigt. Die Ereigniseigenschaft, für die wir eine Prozedur hinterlegen wollen, heißt **Beim Laden** und gehört zum Formular selbst. Dafür legen Sie die folgende Ereignisprozedur an:

```
Private Sub Form_Load()
    Me!lstProviderauswahl = Me!ProviderID
End Sub
```

Das Ergebnis liefert Bild 7. Wie gewünscht, ist im Listenfeld der gleichen Eintrag wie im Formular selbst markiert.

Datensatzwechsel nur per Listenfeld

Wenn der Benutzer im aktuellen Zustand des Formulars mehrfach die Tabulator-Taste verwendet, landet

Steuerelemente verankern

Mit Access 2007 hat Microsoft die beiden Eigenschaften »Horizontaler Anker« und »Vertikaler Anker« für Steuerelemente eingeführt. Mit diesen beiden Eigenschaften können Sie die Steuerelemente in Formularen nicht nur wie bisher oben und links verankern, sondern auch unten und rechts. Außerdem können Sie auch den Wert »Beide« auswählen und somit dafür sorgen, dass das so definierte Steuerelemente beim Vergrößern des Formulars nach rechts oder unten mit vergrößert wird.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1705_SteuerelementeVerankern.accdb**.

»Horizontaler Anker« und »Vertikaler Anker«
Die beiden neuen Eigenschaften, die es für alle Steuerlemente gibt, die Sie in Access-Formularen einsetzen können, heißen **Horizontaler Anker** und **Vertikaler Anker**. Für die Eigenschaft **Horizontaler Anker** können Sie die folgenden Werte einstellen (siehe Bild 1):

- **Links**: Verankert das Steuerelement auf der linken Seite. Die Größe und die Position bleiben beim Ändern der Größe des Formulars konstant.
- **Rechts**: Verankert das Steuerelement auf der rechten Seite. Wenn die Breite des Formulars vergrößert wird, wird das rechts verankerte Steuerelement nach rechts verschoben, beim Verkleinern entsprechend nach links (aber nicht weiter nach links als in der Ausgangsposition).
- **Beide**: Verankert das Steuerelement auf der linken und der rechten Seite. Wenn der Benutzer die Breite des Formulars vergrößert, verbreitert er damit auch das an beiden Seiten verankerte Steuerelement, umkehrt gilt es für das verkleinern der Breite – hier wird das Steuerelement allerdings nicht schmaler als im Entwurf festgelegt.

Für die Eigenschaft **Vertikaler Anker** gibt es diese Werte:

- **Oben**: Verankert das Steuerelement am oberen Rand. Das heißt, dass sich die Größe des Steuerelements beim Ändern der Höhe des Formulars nicht ändert.
- **Unten**: Verankert das Steuerelement am unteren Rand. Wenn der Benutzer die Höhe des Formulars vergrößert, verschiebt dies das Steuerelement nach unten.
- **Beide**: Verankert das Steuerelement an der oberen und unteren Seite. Wenn der Benutzer also

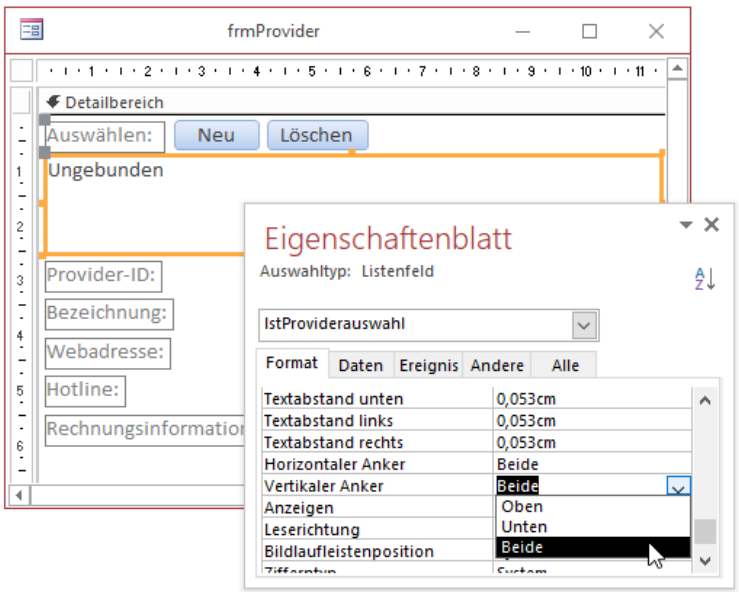


Bild 1: Einstellung der beiden Eigenschaften **Horizontaler Anker** und **Vertikaler Anker**

die Höhe des Formulars verändert, ändert sich das Steuerelement analog.

Verknüpfte Bezeichnungsfelder

Wenn das Steuerelement automatisch mit einem Bezeichnungsfeld angelegt wird, dann beeinflusst die Änderung der Eigenschaften **Horizontaler Anker** und **Vertikaler Anker** des Steuerelements auch die entsprechenden Einstellungen des Bezeichnungselements. Bei einem Textfeld beispielsweise ist die Eigenschaft **Horizontaler Anker** für das Textfeld sowie für das verknüpfte Bezeichnungsfeld standardmäßig auf **Links** eingestellt. Wenn Sie die Eigenschaft für das Steuerelement nun etwa auf **Rechts** oder **Beide** einstellen, ändert sich der Wert für das verknüpfte Bezeichnungsfeld auf **Rechts**. Damit das Bezeichnungsfeld nun nicht wie in Bild 2 immer beim Verbreitern des Formulars nach rechts verschoben wird, müssen Sie dessen Eigenschaft **Horizontaler Anker** wieder auf **Links** einstellen.

Anker über das Ribbon einstellen

Neben den beiden Eigenschaften Horizontaler Anker und Vertikaler Anker können Sie ein Steuerelement auch über die Einträge des Ribbons verankern. Diese finden Sie, wenn das Formular in der Entwurfsansicht geöffnet ist, im Ribbon unter **Anordnen|Position|Anker** (siehe Bild 3). Die Einstellungen haben die folgenden Werte für die Eigenschaften **Horizontaler Anker** und **Vertikaler Anker** zur Folge:

- **Oben links: Links, Oben**
- **Quer nach oben dehnen: Beide, Oben**

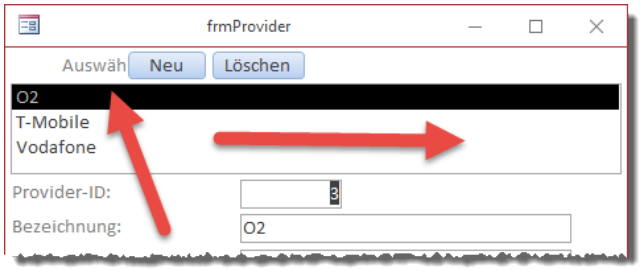


Bild 2: Das Bezeichnungsfeld des Listenfeldes wird beim Verbreitern des Formulars nach rechts verschoben.

- **Oben rechts: Rechts, Oben**
- **Nach unten dehnen: Links, Beide**
- **Nach unten und quer dehnen: Beide, Beide**
- **Nach unten und rechts dehnen: Rechts, Beide**
- **Unten links: Links, Unten**
- **Quer nach unten dehnen: Beide, Unten**
- **Unten rechts: Rechts, Unten**

Steuerelemente unterhalb und rechts

Wenn Sie eine der beiden Eigenschaften **Horizontaler Anker** oder **Vertikaler Anker** auf **Rechts/Unten** oder **Beide** einstellen, müssen Sie beachten, dass die Elemente, die sich rechts oder unterhalb des so eingestellten Steuerelements befinden, durch den sich nach rechts oder unten verschiebenden rechten oder unteren Rand verdeckt werden können. Wenn Sie also etwa das Listenfeld aus Bild 4 so einstellen, dass es nach rechts und nach unten vergrößert wird, dann müssen Sie für alle darunter liegenden Steuer-

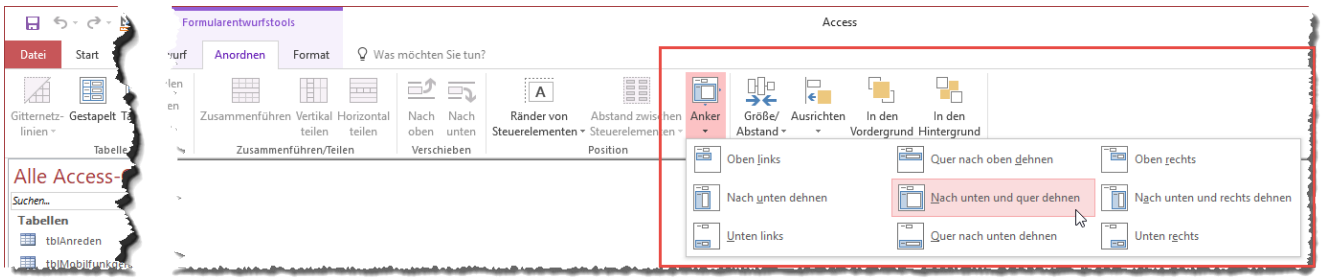


Bild 3: Einstellen der Verankerung über das Ribbon

Handyverwaltung IV: Verträge und SIM-Karten

Im vierten Teil der Artikelreihe zum Thema »Handyverwaltung« setzen wir die Arbeit an den Formularen der Lösung fort. Diesmal fügen wir die Formulare zum Verwalten gleich zweier Objekttypen hinzu: für die Verträge und die SIM-Karten. Die Verträge und die SIM-Karten stehen in einem 1:n-Verhältnis, wobei wir jedem Vertrag mehrere SIM-Karten zuordnen können wollen. Sehen Sie selbst, wie dies umgesetzt wird!

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1705_Handyverwaltung.accdb**.

Vorüberlegungen

In diesem Teil der Artikelreihe bilden wir die Formulare für den Ausschnitt des Datenmodells ab, den Sie in Bild 1 erkennen können. Die Tabelle **tblProvider** haben wir ja bereits im Artikel **Handyverwaltung III: Provider verwalten** über ein Formular für den Benutzer verfügbar gemacht. Hinzu kommen nun die beiden Tabellen **tblVertraege** und **tblSIMKarten**. Für jeden Vertrag wollen wir dabei zunächst einen Provider auswählen lassen, und zwar über ein Kombinationsfeld.

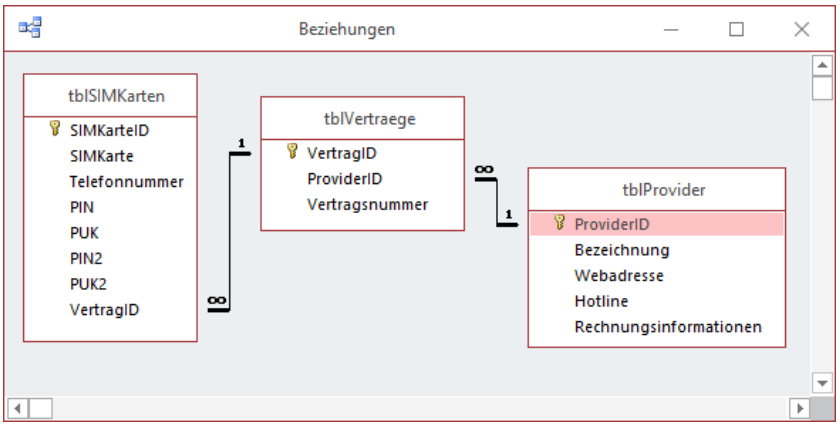


Bild 1: Teil des Datenmodells, den dieser Teil der Artikelreihe abdeckt

Die Vorbereitungen dazu haben wir ja bereits getroffen, indem wir im Entwurf der Tabelle **tblVertraege** ein entsprechendes Nachschlagefeld angelegt haben. Als weiteres Datum für die Verträge wollen wir die Vertragsnummer speichern. Später kümmern wir uns um weitere interessante Eigenschaften eines Vertrags, nämlich zum Beispiel die Laufzeit, das Vertragsdatum, die einmalig und monatlich anfallenden Kosten und so weiter.

Die Tabelle **tblSIMKarten** enthält die Daten für die jeweilige SIM-Karte wie die Kartennummer, die Telefonnummer, die PINs und PUKs sowie den Vertrag, dem die Karte zugeordnet ist. Auch hier verwenden wir wieder ein Fremdschlüsselfeld als Basis für das Kombinationsfeld, mit dem der Benutzer den zugehörigen Vertrag auswählen kann.

In diesem Fall wollen wir allerdings nicht nur in dem Formular, das die Details einer SIM-Karte anzeigt, die verknüpften Verträge anzeigen (es kann ja für Verträge auch einmal mehr als eine SIM-Karte geben – zum Beispiel für ein zweites Gerät wie ein Tablet). Wir wollen auch im Formular für die Vertragsdetails alle mit dem Vertrag verknüpften SIM-Karten anzeigen, und zwar in einem Unterformular.

Das Formular frmSIMKarten

Beginnen wir also mit dem einfacher zu entwerfenden Formular, nämlich dem für die SIM-Karten. Dieses soll die Tabelle **tblSIMKarten** als Datenherkunft verwenden. Wollen wir zur Verwaltung der SIM-Karten wieder zwei Formular erstellen, also ein Übersichtsformular und ein Detailformular wie bei den Personen, oder fassen wir die Übersicht und die Details in einem Formular zusammen wie bei den Providern? Wir entscheiden uns für die letztere Variante. Allerdings soll diesmal kein Listenfeld, sondern ein Unterformular mit allen Datensätzen der Tabelle **tblSIMKarten** zum Einsatz kommen. Wir wollen mit der Lösung ja auch

einmal verschiedene Ansätze zur Darstellung der auf unterschiedliche Art verknüpften Daten liefern. Also legen wir zunächst eine Abfrage namens **qrySIMKartenNachKartennummer** an, welche die alle Felder der Tabelle **tblSIMKarten** liefert und die Einträge nach dem Feld **SIMKarte** sortiert (siehe Bild 2).

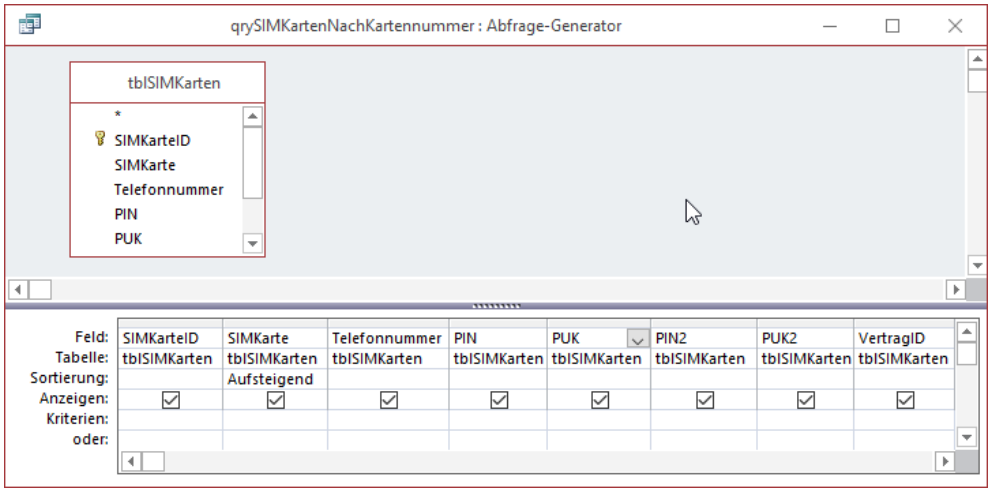


Bild 2: Abfrage, welche die Felder der Tabelle **tblSIMKarten** nach Kartennummer sortiert liefert

Diese Abfrage wählen Sie dann für das neu erstellte Formular **sfmSIMKarten** für die Eigenschaft **Datenherkunft** ein. Wechseln Sie dann zur Anzeige der Feldliste und ziehen Sie die drei Felder **SIMKarteID**, **SIMKarte** und **Telefonnummer** dieser Abfrage in den Detailbereich des Formularentwurfs (siehe Bild 3). Anschließend stellen Sie die Eigenschaft **Standardansicht** auf **Datenblatt** ein. Schließen Sie das Formular nun.

Das Hauptformular frmSIMKarten

Nun folgt das Hauptformular, dem Sie auch die Abfrage **qrySIMKartenNachKartennummer** zuweisen. Auch wenn dieses Formular immer nur einen Datensatz anzeigt, verwenden wir diese sortierte Variante der Abfrage – warum, werden Sie weiter unten sehen. Und welcher beim Öffnen des Formulars angezeigt wird, legen wir später fest.

Erstellen Sie also ein neues Hauptformular namens **frmSIMKarten**. Stellen Sie als Datenherkunft die Abfrage **qrySIMKartenNachKartennummer** ein. Da wir immer nur einen Datensatz anzeigen und die Navigation über die im Unterformular angezeigten Datensätze erfolgen soll, wollen wir hier die Eigenschaften **Datensatzmarkierer**, **Navigationsschaltflächen**, **Trennlinien** und **Bildlaufleisten** auf **Nein** einstellen sowie die Eigenschaft **Automatisch zentrieren** auf den Wert **Ja**.

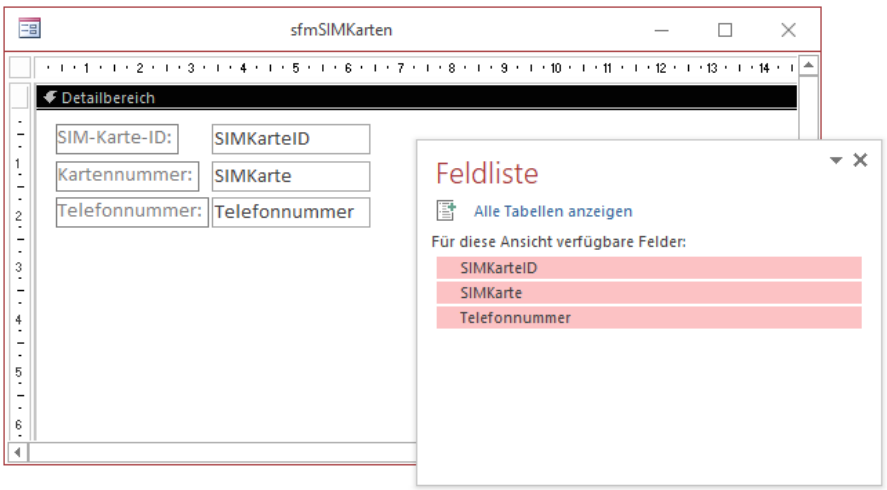


Bild 3: Unterformular für die Anzeige aller SIM-Karten

Ziehen Sie dann als Erstes das Unterformular **sfmSIMKarten** aus dem Navigationsbereich in den Entwurf des Hauptformulars **frmSIMKarten**. Positionieren Sie es am linken Rand, sodass sie die übrigen Steuerelemente, welche die Details zum aktuell im Unterformular ausgewählten Datensatz anzeigen sollen, rechts noch ausreichend Platz haben.

Nun blenden Sie die Feldliste ein und ziehen alle Felder der

Datenherkunft so in das Formular, dass diese rechts vom Unterformular angezeigt werden. Das Ergebnis soll dann etwa wie in Bild 4 aussehen.

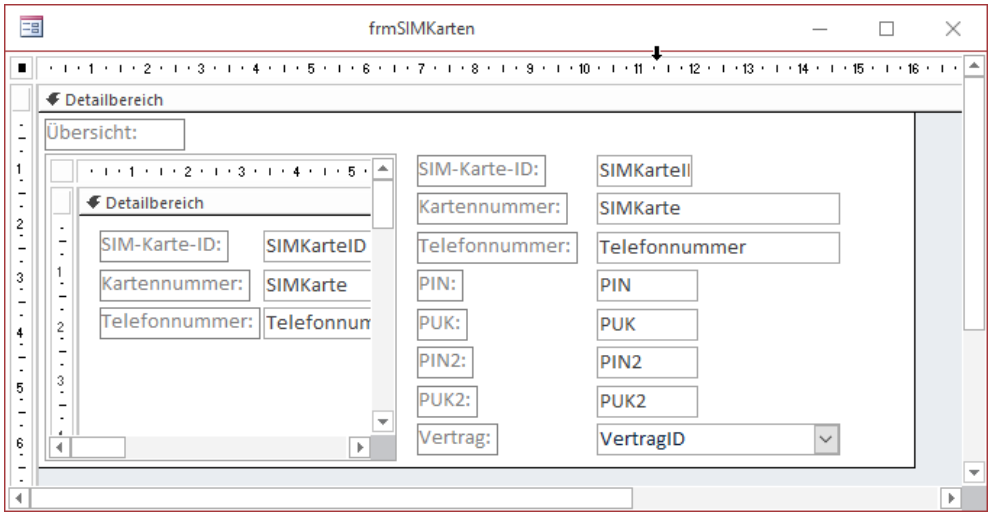


Bild 4: Hauptformular mit Unterformular

Wenn Sie nun in die Formularansicht wechseln, erhalten Sie die Ansicht aus Bild 5. Hier ist noch nicht alles Gold, was glänzt – selbst mit fehlenden Daten noch nicht. Zunächst einmal scheint die Anzeige des Feldes **SIMKarteID** im Unterformular in der Datenblattansicht nicht unbedingt nötig zu sein. Wenn diese wegfällt, haben die übrigen beiden Felder vermutlich auch schon genug Platz. Aber können wir dieses Feld einfach entfernen? Im Artikel **Handyverwaltung III: Provider verwalten**, wo wir ein Listenfeld zur Anzeige aller Provider verwendet haben, musste die **ProviderID** ja explizit aufgeführt werden, damit wir diese als gebundene Spalte nutzen und ihren Wert ermitteln konnten. Also sollen wir das Feld SIMKarteID nun aus dem Entwurf des Unterformulars entfernen oder können wir dieses irgendwie ausblenden?

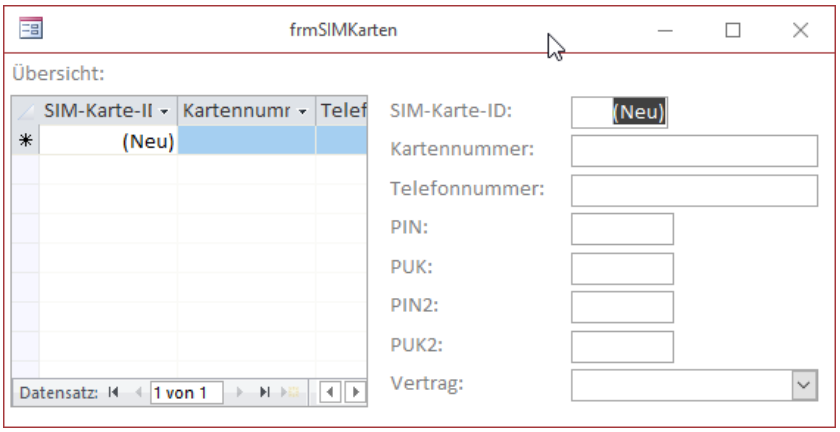


Bild 5: Formularansicht ohne Daten

Ein Einstellen der Eigenschaft **Sichtbar** für das Textfeld auf den Wert **Nein** hilft jedenfalls in der Datenblattansicht nichts: Die Spalte wird nach wie vor angezeigt. Um zu prüfen, was geschieht, wenn wir das Feld entfernen, müssen wir zunächst die synchrone Darstellung der Daten im Haupt- und Unterformular realisieren.

Haupt- und Unterformular synchronisieren

Um dafür zu sorgen, dass das Hauptformular immer den aktuell im Unterformular markierten Datensatz anzeigt, haben wir mehrere Möglichkeiten. Man mag vermuten, dass hier der Effekt eintritt, der auch beim Erstellen von Haupt- und Unterformular für die Darstellung von Daten aus 1:n-Beziehungen eintritt: Hier wird ja das Unterformular-Steuerelement, dass dann schon an die entsprechende Datenherkunft gebunden sein sollte, in das Hauptformular gezogen. Wenn diese an eine Datenherkunft gebunden ist, für das Access eine Beziehung erkennen kann, werden die beiden Eigenschaften **Verknüpfen von** und **Verknüpfen nach** des Unterformular-Steuerelements automatisch mit dem Primärschlüsselfeld und dem Fremdschlüsselfeld der Beziehung gefüllt.

Wechseln wir einmal in die Entwurfsansicht und markieren das Unterformular-Steuerelement wie in Bild 6,

erkennen wir, dass Access hier automatisch für beide Eigenschaften den Wert **SIMKarteID** eingetragen hat.

Das bedeutet, dass das Formular im Unterformular nur diejenigen Datensätze anzeigt, deren Wert im Feld **SIMKarteID** mit dem entsprechenden Wert im übergeordneten Formular übereinstimmt. Was wiederum heißt, dass das Unterformular immer nur den Datensatz anzeigt, der momentan im Hauptformular angezeigt wird (siehe Bild 7). So soll es natürlich nicht aussehen, also leeren wir in einem ersten Schritt die beiden Eigenschaften Verknüpfen von und Verknüpfen nach des Unterformular-Steuerelements. Wichtig ist dabei, dass Sie tatsächlich das Unterformular-Steuerelement markieren und nicht das Unterformular selbst. Das erreichen Sie beispielsweise, indem Sie oben links im Access-Fenster den Namen des Unterformular-Steuerelements, hier **sfmSIMKarten**, auswählen und so das Unterformular-Steuerelement markieren (siehe Bild 8).

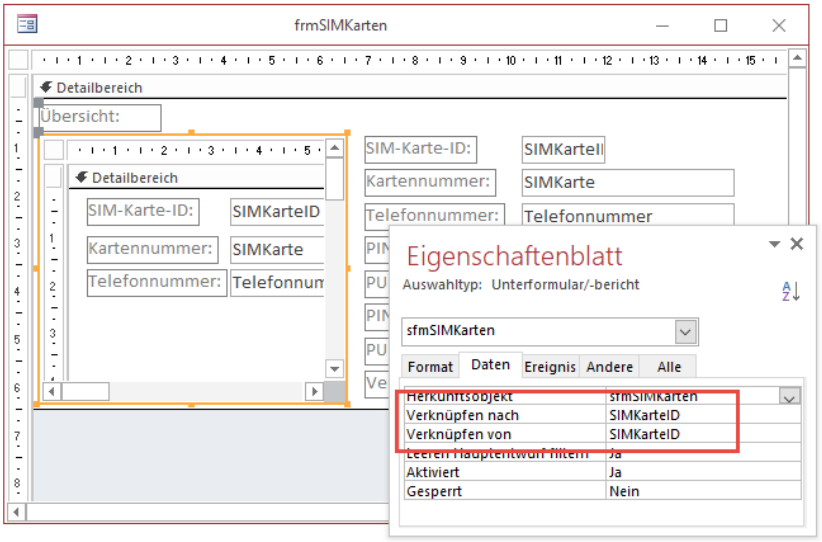


Bild 6: Verknüpfungseigenschaften des Unterformular-Steuerelements

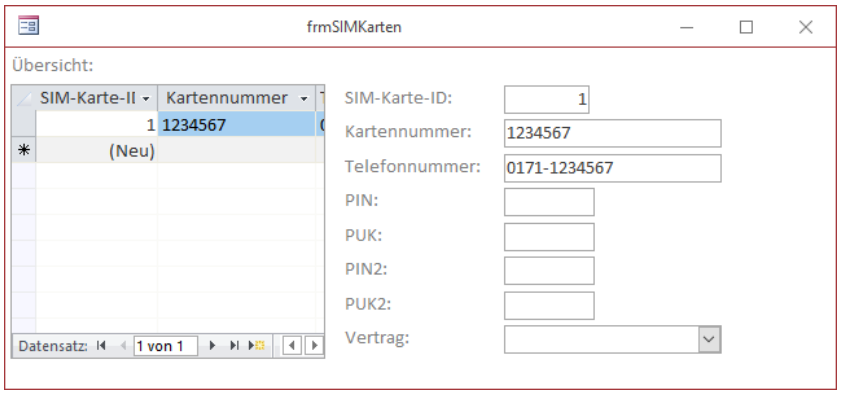


Bild 7: Das Unterformular zeigt nur den aktuellen Datensatz des Hauptformulars an.

Wenn Sie nach dem Leeren der beiden Eigenschaften **Verknüpfen von** und **Verknüpfen nach** wieder zur Formularansicht wechseln, zeigt dieses nun alle Datensätze im Unterformular an. Wenn Sie einen davon auswählen, liefert das allerdings nicht die Details des Datensatzes im Hauptformular. Kein Wunder: Es gibt ja auch nun keinerlei Beziehung mehr zwischen den Daten im Haupt- und im Unterformular. Also sorgen wir zunächst dafür, dass das Hauptformular die Daten des im Unterformular ausgewählten Datensatzes anzeigt.

Dazu ist nur eine einzige Anweisung nötig. Diese fügen Sie in einer Ereignisprozedur ein, die durch das Ereignis **Beim Laden** des Hauptformulars ausgelöst wird. Geben Sie dazu für die Eigenschaft **Beim Laden** des Hauptformulars den Wert **[Ereignisprozedur]** ein und klicken Sie auf die Schaltfläche mit den drei Punkten. Die nun erscheinende Ereignisprozedur füllen Sie wie folgt auf:

```
Private Sub Form_Load()
    Set Me!sfmSIMKarten.Form.Recordset _
                        = Me.Recordset
End Sub
```

Der nächste Wechsel zur Formularansicht zeigt, dass wir unser Ziel erreicht haben (siehe Bild 9). Was geschieht hier? Wir haben die Eigenschaft **Recordset** des Unterformulars im Unterformularsteuerelement **sfmSIMKarten** auf das **Recordset**-Element des

Hauptformulars eingestellt. Auf diese Weise zeigen das Haupt- und das Unterformular nun immer den gleichen Datensatz an, egal, ob Sie diesen im Haupt- oder im Unterformular wechseln.

Unterformular aus der Reihenfolge entfernen

Das können Sie vom Unterformular aus testen, indem Sie dort durch die Datensätze navigieren und betrachten, wie da Hauptformular jeweils die passenden Detaildaten anzeigt. Andersherum ist es nicht so einfach zu testen, da wir bereits die Navigationsschaltflächen ausgeblendet haben.

So können wir nur noch mit der Tabulator-Taste durch die Steuerelemente navigieren und würden so nach dem Verlassen des letzten Steuerelements zum nächsten Datensatz des Hauptformulars wechseln. Allerdings landen wir vor dem Verlassen des ersten Datensatzes im Unterformular-Steuerelement, weil dieses zuletzt zum Formular hinzugefügt wurde und somit in der Reihenfolge die letzte Position belegt. Das können wir ändern, indem wir das Unterformular-Steuerelement einfach aus der Reihenfolge herausnehmen. Dazu stellen Sie die Eigenschaft **In Reihenfolge** des Steuerelements auf den Wert **Nein** ein. Von nun an können Sie testweise durch die Steuerelemente des Hauptformulars navigieren, bis Sie das letzte Steuerelement des aktuellen Datensatzes verlassen und im nächsten Schritt zum ersten Feld des nächsten Datensatzes wechseln. Dann zeigt auch das Unterformular den nächsten Datensatz an.

Da der Benutzer aber nicht mit den Tabulator-Tasten durch die Datensätze navigieren soll, sondern die Datensätze über die Auswahl im Unterformular ansteuern soll, legen wir noch für die Eigenschaft **Zyklus** den Wert **Aktueller Datensatz** fest. So kann der Benutzer mit den Tabulator-Tasten lediglich durch den aktuell angezeigten Datensatz navigieren.

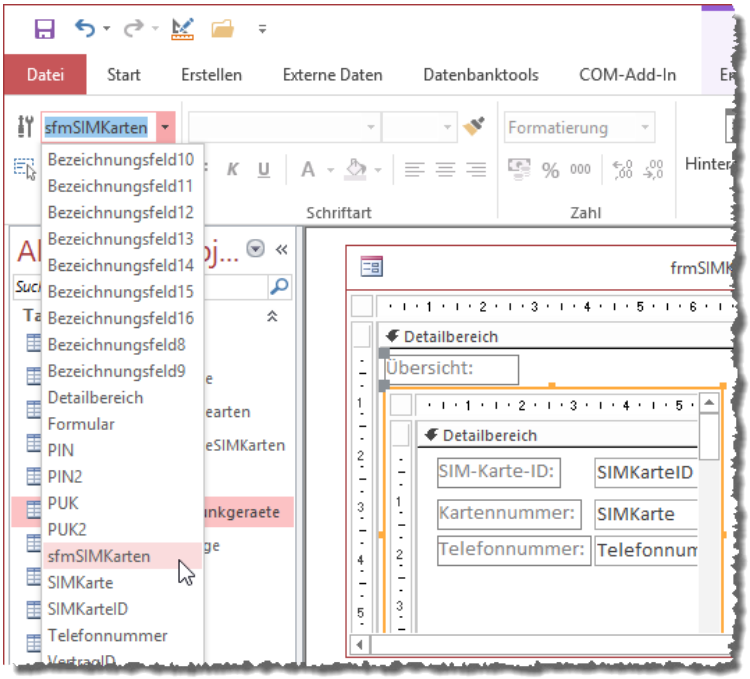


Bild 8: Auswählen des Unterformular-Steuerelements

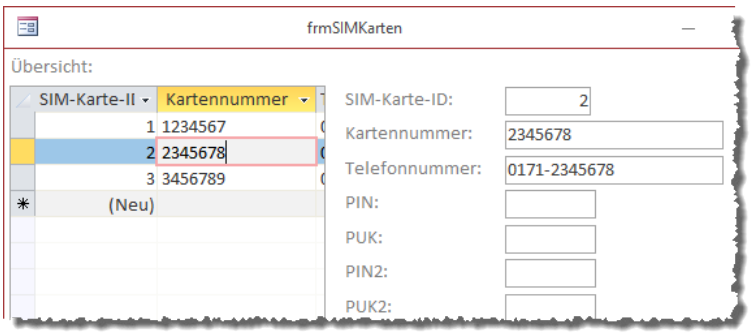


Bild 9: Das Hauptformular zeigt nun die Daten des im Unterformular ausgewählten Datensatzes an.

Primärschlüsselfeld aus Unterformular ausblenden

Nun haben wir weiter oben begonnen, die Anzeig des Primärschlüsselfelds im Unterformular loszuwerden. Mit dem Einstellen der Eigenschaft **Sichtbar** auf den Wert **Nein** für das Steuerelement **SIMKarteID** hat es nicht geklappt. Nun wollen wir testen, ob wir das Feld einfach löschen können oder ob dies Auswirkungen auf die Funktion des Formulars hat. Wie Bild 10 zeigt, führt dies zum Erfolg: Die Primärschlüsselfeldspalte ist aus dem Unterformular verschwunden, aber wir können damit immer noch den Datensatz auswählen,

dessen Details im Hauptformular angezeigt werden sollen.

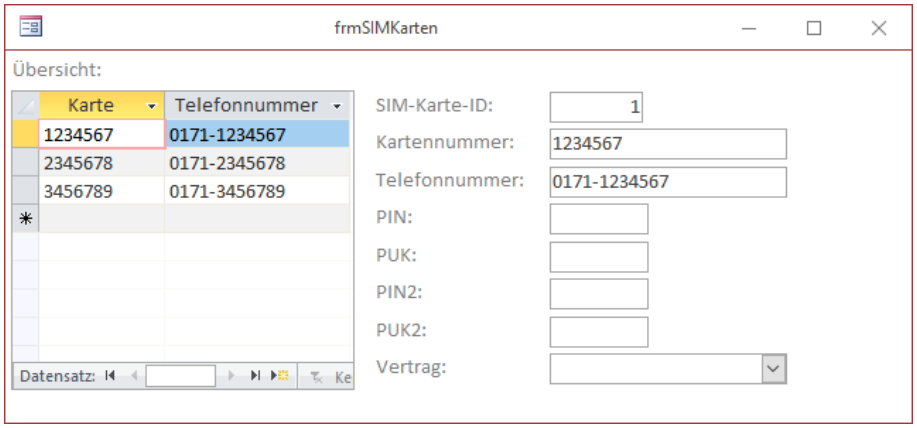


Bild 10: Unterformular ohne störendes Primärschlüsselfeld

Neuen Datensatz anlegen

Wie aber legen wir in diesem Formular einen neuen Datensatz an? Über das Hauptformular können wir nicht mehr zu einem neuen, leeren Datensatz navigieren. Im Unterformular wird zwar in der untersten Zeile zwar ein solcher Datensatz angezeigt, mit dem wir auch einen neuen, leeren Datensatz im Hauptformular hervorrufen können. Allerdings ist das nicht gerade benutzerfreundlich. Also fügen wir zunächst eine Schaltfläche namens **cmdNeu** mit der Beschriftung **Neu** unter dem Unterformular zum Formular hinzu. Für diese Schaltfläche hinterlegen wir die folgende Prozedur:

```
Private Sub cmdNeu_Click()
    DoCmd.GoToRecord Record:=acNewRec
    Me!SIMKarte.SetFocus
End Sub
```

Diese zeigt im aktuellen Formular einen neuen, leeren Datensatz an und verschiebt den Fokus auf das Textfeld **SIMKarte**. Geben wir nun die Daten für die neue SIM-Karte ein und durchlaufen mit der Tabulator-Taste alle Felder, bis der Datensatz gespeichert wird, aktualisiert dies sogar das Unterformular und zeigt den neuen Datensatz dort direkt an. Da auch dieser Weg nicht für jeden Benutzer offensichtlich ist, fügen wir diesem Formular noch eine **Speichern**- und eine **Abbrechen**-Schaltfläche unterhalb der gebundenen Steuerelemente des Formulars hinzu. Die Schaltfläche zum Speichern soll die folgende Prozedur auslösen:

```
Private Sub cmdSpeichern_Click()
    Me.Dirty = False
End Sub
```

Die **Dirty**-Eigenschaft wird auf **True** eingestellt, sobald der Benutzer mit der Bearbeitung entweder eines bestehenden oder eines neuen Datensatzes beginnt. Stellt man diese per Code wieder auf den Wert **False** ein, speichert dies den aktuellen Datensatz.

Die **Abbrechen**-Schaltfläche soll eine Ereignisprozedur auslösen, die lediglich die **Undo**-Methode des aktuellen Formulars aufruft und damit die seit dem letzten Speichern am aktuellen Datensatz durchgeführten Änderungen zurücknimmt:

```
Private Sub cmdAbbrechen_Click()
    Me.Undo
End Sub
```

Nun müssen wir noch dafür sorgen, dass die beiden Schaltflächen nur dann aktiviert werden, wenn der Benutzer auch tatsächlich gerade speicherbare Änderungen an den aktuell angezeigten Daten vornimmt. Dazu stellen wir zunächst einmal die Eigenschaft **Aktiviert** der beiden Schaltflächen auf den Wert **Nein** ein, damit diese beim Öffnen des Formulars noch nicht aktiviert sind. Dies erledigen Sie über das Eigenschaftsfenster der jeweiligen Steuerelemente. Tipp: Sie können beide Steuerelemente durch aufziehen eines geeigneten Rahmens gleichzeitig markieren und die Eigenschaft für beide Steuerelemente gleichzeitig ändern. Dann fügen wir für die Ereigniseigenschaft **Bei Geändert** die folgende Ereignisprozedur hinzu:

```
Private Sub Form_Dirty(Cancel As Integer)
    Me!cmdAbbrechen.Enabled = True
```

Detailformular öffnen und abfragen

Wenn Sie von einem Übersichtsformular oder einem Listenfeld- oder Kombinationsfeld eines Formulars ein Detailformular öffnen, um einen der Datensätze aus der Übersicht oder einer Liste im Detail anzuzeigen, lässt sich das mit einem Einzeiler programmieren. Kniffliger wird es, wenn Sie in dem Detailformular Änderungen vornehmen oder gar einen anderen Datensatz auswählen und die Änderungen oder die Auswahl in das aufrufende Formular übernehmen wollen. Wir stellen zwei Techniken vor – die gängige und eine nicht so oft verwendete, aber grundsätzlich bessere Technik.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1705_Handyverwaltung.accdb**.

Beispiel

Wir wollen uns am Beispiel der Formular **frmSIMKarten und frmVertragsdetails** der Beispieldatenbank Handyverwaltung ansehen, wie Sie die Detaildaten zu einem Datensatz in einem Kombinationsfeld des Formulars **frmSIMKarten** per Doppelklick im Formular **frmVertragsdetails** öffnen (siehe Bild 1).

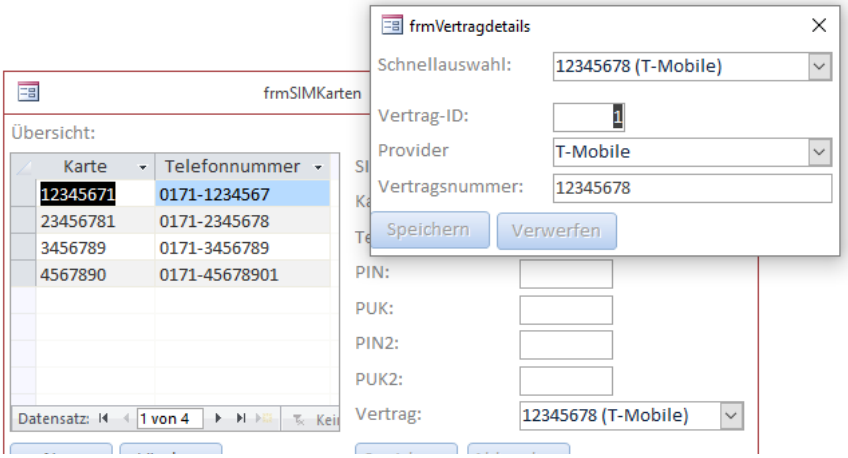


Bild 1: Beispiel für das Aufrufen eines Detailformulars

Hier gibt es nun im Detailformular die Möglichkeit, einen anderen Datensatz, also einen anderen Vertrag, auszuwählen. Die Auswahl in diesem Formular soll dann beim Schließen in das aufrufende Formular übernommen werden – gegebenenfalls auch erst nach Rückfrage an den Benutzer, da dieser sich möglicherweise nicht im Klaren darüber ist, dass er damit auch den Eintrag im Kombinationsfeld manipuliert.

Herkömmliche Technik

Normalerweise würden Sie, wenn Sie wie hier per Doppelklick auf ein Kombinationsfeld von einem Formular aus ein Detailformular öffnen, eine bewährte Vorgehensweise nutzen.

Damit wir die Originalformulare der Beispieldatenbank nicht bearbeiten müssen, haben wir die beiden Formulare unter den Namen **frmSIMKarten_Herkoemmlich** und **frmVertragsdetails_Herkoemmlich** kopiert.

Die Vorgehensweise sieht so aus, dass wir beim Doppelklick auf das Kombinationsfeld das Formular **frmVertragsdetails_Herkoemmlich** als modalen Dialog öffnen.

Die beiden Schaltflächen **cmdSpeichern** und **cmdVerwerfen** statten wir dabei mit entsprechenden Funktionen aus: **cmdSpeichern** speichert den Datensatz und macht das Formular dann unsichtbar. **cmdVerwerfen** verwirft die Änderungen und schließt das Formular.

Warum? Weil die Prozedur, die das Formular **frmVertragsdetails_Herkoemmlich** als modalen Dialog geöffnet hat, erst nach dem Fokusverlust von **frm-**

```
Private Sub cboVertragID_DblClick(Cancel As Integer)
    Dim strFormular As String
    strFormular = "frmVertragdetails_Herkoemmlich"
    If Not IsNull(Me!cboVertragID) Then
        DoCmd.OpenForm strFormular, WindowMode:=acDialog, OpenArgs:=Me!cboVertragID
        If IstFormularGeoeffnet(strFormular) Then
            If Not Me!cboVertragID = Forms(strFormular)!VertragID Then
                If MsgBox("Es wurde ein anderer Vertrag ausgewählt. Übernehmen?", vbYesNo) = vbYes Then
                    Me!cboVertragID = Forms(strFormular).VertragID
                End If
            End If
            DoCmd.Close acForm, strFormular
        End If
    End If
End Sub
```

Listing 1: Aufrufen des Formulars **frmVertragsdetails_Herkoemmlich** und Auswerten des dort selektierten Vertrags

Vertragsdetails_Herkoemmlich fortgesetzt wird. Der Fokusverlust tritt sowohl beim Schließen als auch beim Verbergen des Formulars ein. In der aufrufenden Prozedur prüfen wir dann, ob das Formular noch geöffnet ist. Falls nicht, ist der Fall klar: Der Benutzer hat die Änderungen verworfen und es sind keine weiteren Schritte im aufrufenden Formular nötig. Ist das Formular jedoch noch geöffnet, muss es zumindest unsichtbar sein, denn sonst würde die aufrufende Prozedur nicht weiter ausgeführt werden.

In diesem Fall haben wir von dieser Prozedur aus vollen Zugriff auf das Formular – auch wenn dieses nicht mehr sichtbar ist. Wir können dann prüfen, ob der Benutzer einen anderen Datensatz im Formular ausgewählt hat und diesen dann entweder in das Kombinationsfeld im aufrufenden Formular übernehmen oder Rücksprache mit dem Benutzer halten, ob dies gewünscht ist.

Schauen wir uns den benötigten Code an. Wir starten mit der Prozedur, die durch das Ereignis **Beim Doppelklicken** auf das Kombinationsfeld **cboVertragID** ausgelöst wird und die wie in Listing 1 aussieht.

Hier ist zunächst nur interessant, wie wir das Formular **frmVertragsdetails_Herkoemmlich** aufrufen. Wir übergeben dabei nämlich keinen **WhereCondition**-Parameter, denn dies würde die Datenherkunft des Formulars einschränken und man könnte nicht mehr leicht über das Kombinationsfeld einen anderen Vertrag auswählen.

Stattdessen übergeben wir mit dem Parameter **OpenArgs** den Wert der gebundenen Spalte des Kombinationsfeldes **cboVertragID**, also die **VertragID** des anzuzeigenden Vertrags.

Entscheidend ist auch, dass wir für den Parameter **WindowMode** den Wert **acDialog** angeben, wodurch das Formular **frmVertragsdetails_Herkoemmlich** als modaler Dialog geöffnet wird – die aufrufende Prozedur wird erst dann fortgesetzt, wenn das Formular geschlossen oder unsichtbar gemacht wird.

Damit gelangen wir zur Ereignisprozedur **Form_Load**, die beim Laden des Formulars **frmVertragsdetails_Herkoemmlich** ausgelöst wird.

Diese prüft, ob **Me.OpenArgs** einen Wert enthält (**Not IsNull(Me.OpenArgs)**) und verschiebt dann den Datensatzzeiger zu dem Datensatz, dessen Feld **VertragID** den Wert von **Me.OpenArgs** aufweist:

```
Private Sub Form_Load()
    If Not IsNull(Me.OpenArgs) Then
```

```
        Me.Recordset.FindFirst "VertragID = " & Me.OpenArgs
    End If
End Sub
```

Damit zeigt das Formular beim Öffnen den beim Doppelklick in das Kombinationsfeld ausgewählten Vertrag an. Nun kann der Benutzer die Werte der gebundenen Felder im Formular **frmVertragsdetails_Herkoemmlich** ändern oder auch mit dem Kombinationsfeld **cboSchnellauswahl** einen ganz anderen Vertrag auswählen.

In jedem Fall sollte er eine der Schaltflächen **cmdSpeichern** oder **cmdVerwerfen** anklicken, um die Änderungen zu übernehmen oder zu verwerfen. Die Schaltfläche **cmdSpeichern_Click** löst die folgende Ereignisprozedur aus:

```
Private Sub cmdSpeichern_Click()
    Me.Dirty = False
    SchaltflaechenDeaktivieren
    Me.Visible = False
End Sub
```

Sie sorgt also im letzten Schritt dafür, dass das Formular unsichtbar geschaltet wird.

Die Prozedur, die beim Anklicken der Schaltfläche **cmdVerwerfen** ausgelöst wird, schließt das Formular-hingegen:

```
Private Sub cmdVerwerfen_Click()
    Me.Undo
    SchaltflaechenDeaktivieren
    DoCmd.Close acForm, Me.Name
End Sub
```

In beiden Fällen verliert das Formular den Fokus und die aufrufende Prozedur wird fortgesetzt:

```
If IstFormularGeoeffnet(strFormular) Then
    If Not Me!cboVertragID = _
                    Forms(strFormular)!VertragID Then
        If MsgBox("Es wurde ein anderer Vertrag _
         ausgewählt. Übernehmen?", vbYesNo) = vbYes Then
            Me!cboVertragID = Forms(strFormular).VertragID
        End If
    End If
    DoCmd.Close acForm, strFormular
End If
```

Die Funktion **IstFormularGeoeffnet** ermittelt dabei den Zustand des Formulars beziehungsweise liefert eine Information darüber zurück, ob das Formular noch geöffnet ist (siehe Listing 2).

Falls ja, prüft die nächste **If...Then**-Bedingung, ob der Wert des Kombinationsfeldes **cboVertragID** des aufrufenden Formulars mit dem Wert des Feldes **VertragID** des Formulars **frmVertragsdetails_Herkoemmlich** übereinstimmt.

Falls nicht, hat der Benutzer einen anderen Vertrag ausgewählt. In diesem Fall prüfen wir weiterhin per **MsgBox**-Funktion, ob der Benutzer den im Formular **frmVertragsdetails_Herkoemmlich** ausgewählten Vertrag in das Kombinationsfeld **cboVertragID** des aufrufenden Formulars **frmSIMKarte_Herkoemmlich** übernehmen möchte (siehe Bild 2).

Ist dies der Fall, stellt die Prozedur die ID des ausgewählten Vertrags im aufrufenden Formular **frmSIMKarte_Herkoemmlich** ein.

Nachteile dieser Methode

So oder ähnlich funktionieren einige Konstellationen, egal, ob Sie die Daten über den Doppelklick auf ein Kombinationsfeld, über die Auswahl über ein Listenfeld oder auch eine Datenblattansicht vornehmen.

```
Public Function IstFormularGeoeffnet(strFormular As String) As Boolean
    IstFormularGeoeffnet = SysCmd(acSysCmdGetObjectState, acForm, strFormular) > 0
End Function
```

Listing 2: Die Funktion **IstFormularGeoeffnet** ermittelt, ob ein Formular noch geöffnet ist.

Tipps und Tricks für Memofelder

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1705_Memofelder.accdb**.

Anfügen in Memofeldern

Memofeldern haben in den neueren Access-Versionen eine Eigenschaft namens **Nur anfügen**, die Sie in den Feldeigenschaften im Tabellenentwurf finden und welche die Werte **Ja** und **Nein** annehmen kann (siehe Bild 1).

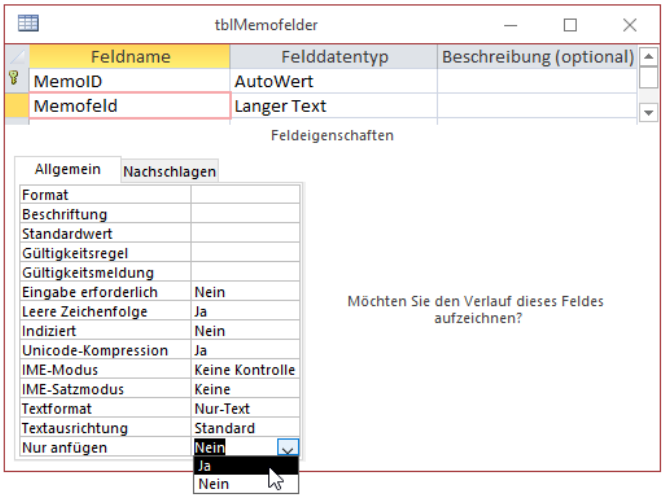


Bild 1: Die Eigenschaft **Nur anfügen** von Memofeldern

Was bringt diese Eigenschaft? Probieren wir es aus, indem wir sie vom Standardwert **Nein** auf **Ja** ändern. Bei der Eingabe von verschiedenen Texten, auch nacheinander im zeitlichen Abstand, ändert sich zunächst jedoch nichts. Erst wenn wir mit der rechten Maustaste auf das Memofeld klicken, erscheint ein Kontextmenü-Eintrag, den es vorher noch nicht gab: **Spaltenverlauf anzeigen** (siehe Bild 2).

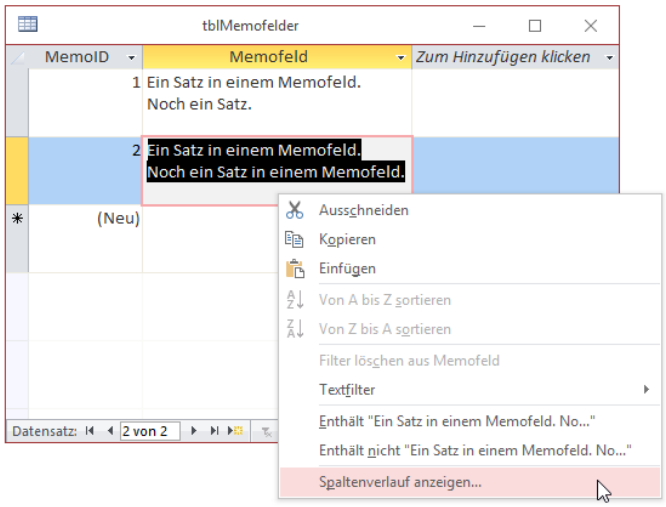


Bild 2: Keine Änderungen bei der Eingabe, jedoch ein neuer Kontextmenü-Eintrag

Wenn wir diesen Befehl anklicken, erscheint das Fenster aus Bild 3. Dieses zeigt den Inhalt des Memofeldes zu verschiedenen Zeitpunkten an. Sie können also nun nachvollziehen, wann welche Änderungen stattgefunden haben.

Diese Historie können Sie sogar per VBA ausgeben. Dazu nutzen Sie die neue Methode ColumnHistory des Application-Objekts, dem Sie den Namen der Tabelle, den Namen des Feldes sowie ein Filterkriterium übergeben, mit dem Sie den Datensatz festlegen. Das Ergebnis sieht dann so aus:

```
? Application.ColumnHistory("tblMemofelder", 7
                            "Memofeld", "MemoID = 2")
[Version: 28.09.2017 21:44:45 ]
Ein Satz in einem Memofeld.
[Version: 28.09.2017 21:44:53 ]
Ein Satz in einem Memofeld.
Noch ein Satz in einem Memofeld.
```

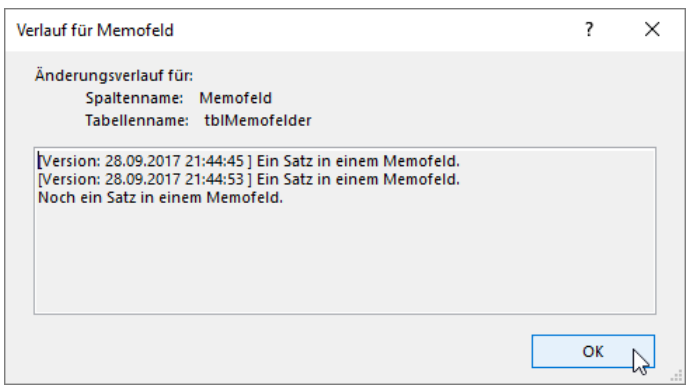


Bild 3: Der Verlauf in einem Memofeld

Suche in Lookupfeldern

Die Suche nach Datensätzen mit bestimmten Werten in Text- oder Zahlenfeldern haben wir in Acces [basics] bereits bearbeitet. Aber was ist, wenn es um Kombinationsfelder geht, welche die Daten aus einer Lookup-Tabelle anzeigen? So wie im einfachsten Fall bei den Anreden einer Kundentabelle? Dann werden Sie in der Regel erstmal nicht an die Eingabe eines Suchetextes denken, sondern dem Benutzer die Möglichkeit geben wollen, den gesuchten Datensatz auch im Suchfeld per Kombinationsfeld auszuwählen. Dieser Artikel zeigt, wie das gelingt.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1706_Lookupsuche.accdb**.

Artikel nach Kategorie durchsuchen
Die Datenblattansicht von Access bietet seit einigen Access-Versionen die Möglichkeit, über den Pfeil nach unten ein Menü zu öffnen, mit dem Sie alle Einträge des aktuellen Feldes anhaken und damit die Suche nach Datensätzen mit diesem Inhalt im entsprechenden Feld starten können (siehe Bild 1).

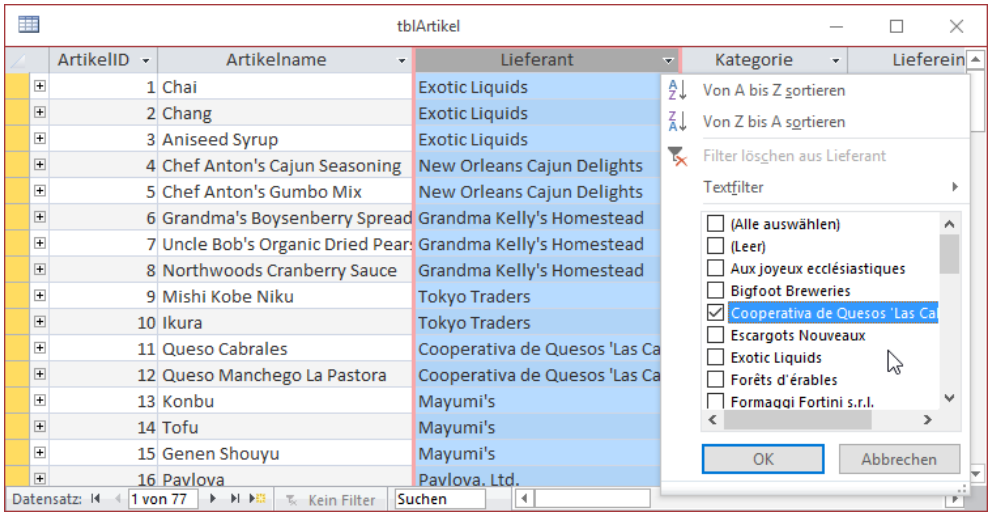


Bild 1: Eingebaute Suche nach einem Eintrag

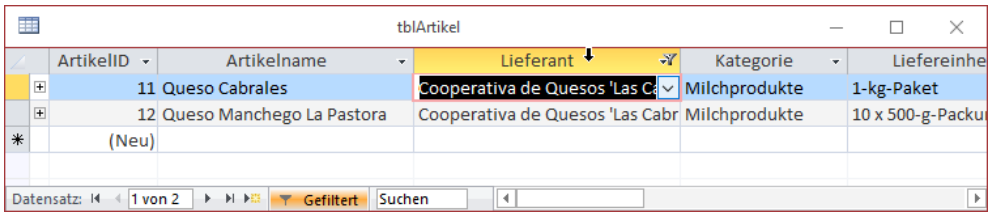


Bild 2: Suchergebnis mit eingebauter Suche

Das Ergebnis entspricht dann wie in Bild 2 genau den Erwartungen. Was aber nun, wenn wir diese Suche nicht anbieten können oder wollen und eine eigene Suche implementieren müssen, die so ähnlich arbeitet – und in diesem Fall zumindest ein Kombinationsfeld anbieten soll, um die passenden Datensätze zu ermitteln?

Dann benötigen wir zunächst ein Formular, welches das Datenblatt mit den zu durchsuchenden Datensätzen in einem Unterformular anzeigt. Dazu erstellen Sie zunächst ein Unterformular namens **sfmLookupsuche** und weisen der Eigenschaft **Datenherkunft** die Tabelle **tblArtikel** zu.

Ziehen Sie dann alle Felder dieser Tabelle aus der Feldliste in den Detailbereich des Formularentwurfs und stellen Sie noch die Eigenschaft Standardansicht auf den Wert **Datenblatt** ein. Speichern und schließen Sie das Formular nun.

Legen Sie ein neues Formulars namens **frmLookupsuche** an, welches als Hauptformular dienen soll.

Ziehen Sie das Unterformular **sfmLookupsuche** aus dem Navigationsbereich in den Detailbereich des neuen Formulars. Stellen Sie die Eigenschaften **Navigationsschaltflächen**, **Datensatzmarkierer**, **Bildlaufleisten** und **Trennlinien** des Hauptformulars auf **Nein** und die Eigenschaft **Automatisch zentrieren** auf **Ja** ein. Fügen Sie über dem Unterformular schließlich noch das Kombinationsfeld hinzu, mit dem wir die Suchbegriffe festlegen wollen (siehe Bild 3). Das Kombinationsfeld nennen wir **cboSucheLieferant**.

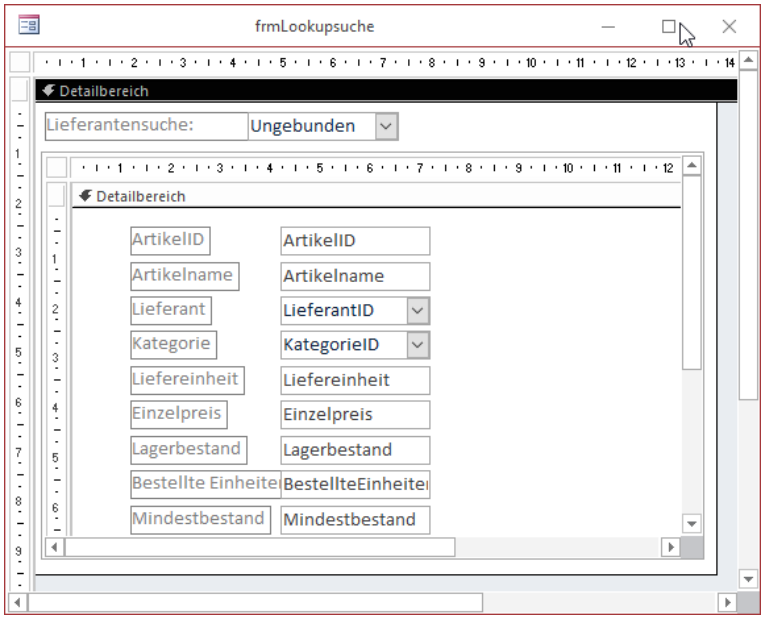


Bild 3: Suchformular mit Kombinationsfeld

Filterausdruck ermitteln

Wenn Sie wissen wollen, welchen Filterausdruck Access automatisch einstellt, wenn Sie mit der eingebauten Filtermöglichkeit arbeiten wollen, welche wir eingangs vorgestellt haben, brauchen Sie nur das Formular zu öffnen und den Filter festzulegen.

Nun öffnen Sie den VBA-Editor und geben im Direktbereich einen Ausdruck ein, der den aktuellen Wert der **Filter**-Eigenschaft des Unterformulars ausgibt:

```
? Forms(0).sfmLookupsuche.form.filter
```

Die Ausgabe lautet wie folgt:

```
([Lookup_LieferantID].[Firma]=
"Bigfoot Breweries")
```

Was ist nun **[Lookup_LieferantID].[Firma]** für ein Ausdruck? Wir konnten es nicht herausfinden. Selbst wenn wir den Code des Formulars mit der folgenden Methode in eine Textdatei exportiert haben, konnten wir diese Zeichenfolge nicht finden:

```
SaveAsText acForm, "sfmLookupsuche", CurrentProject.Path & "\form.txt"
```

Fest steht jedenfalls, dass Access wohl hier irgendwie auf ein Recordset zugreift, das sich hinter dem Kombinationsfeld zur Auswahl des Lieferanten verbirgt und diesen als Feld im Filterausdruck einsetzt.

Kombinationsfeld mit Daten füllen

Wie auch immer: Wir können uns nicht auf irgendwelche nirgends offensichtlich definierten Feldnamen beziehen, wenn wir programmieren, und füllen

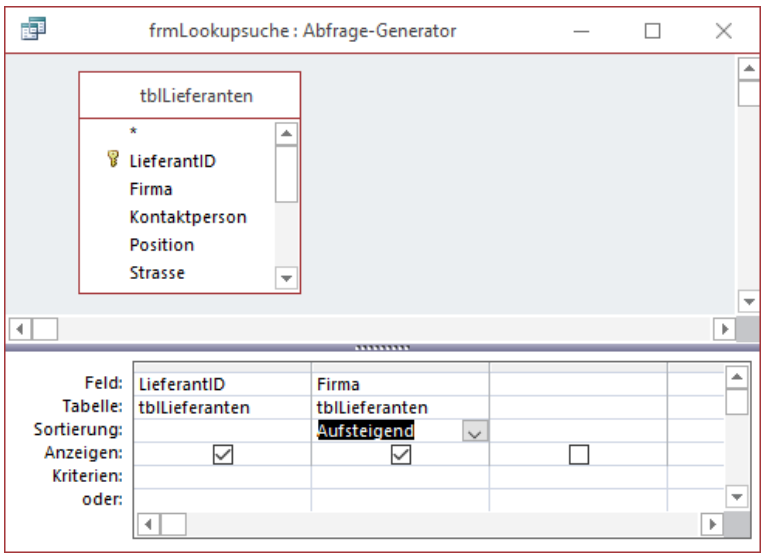


Bild 4: Abfrage als Datensatzherkunft für das Such-Kombinationsfeld

zunächst unser Kombinationsfeld mit den Daten, die wir zur Suche nach den Lieferanten auswählen wollen.

Dazu stellen wir die Eigenschaft **Datensatzherkunft** des Kombinationsfeldes auf die Abfrage aus Bild 4 ein. Damit es das Primärschlüsselfeld als gebundene Spalte verwendet und diese aber ausblendet, stellen sie die Eigenschaft **Spaltenanzahl** auf den Wert **2** und die Eigenschaft **Spaltenbreiten** auf den Wert **0cm** ein. Das Zwischenergebnis sieht dann wie in Bild 5 aus.

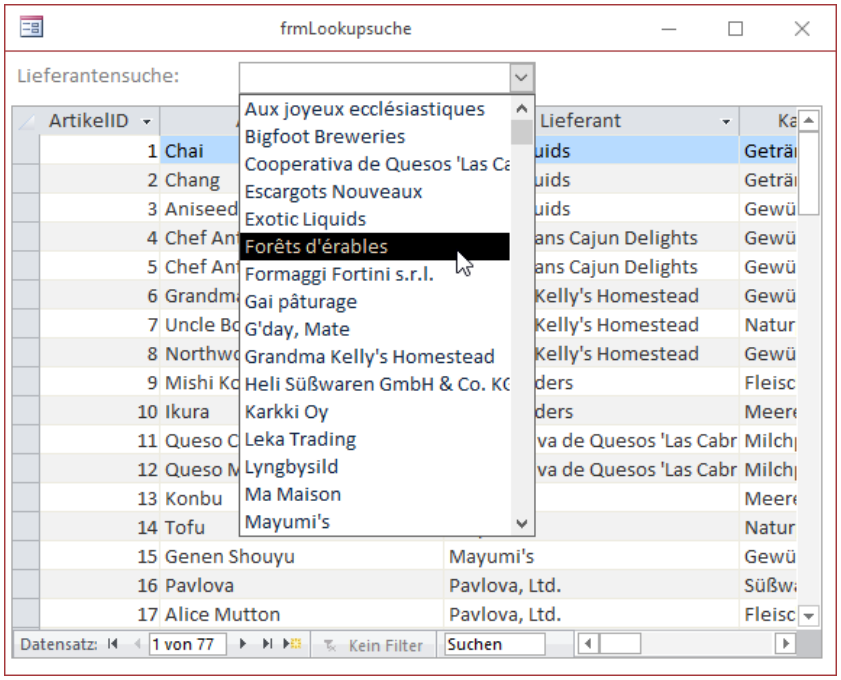


Bild 5: Abfrage als Datensatzherkunft für das Such-Kombinationsfeld

Nun müssen wir noch eine Ereignisprozedur hinzufügen, die nach der Auswahl eines der Einträge des Kombinationsfeldes ausgelöst wird und die Daten des Datenblatts entsprechend des Filterkriteriums filtert.

Dazu stellen Sie für die Eigenschaft Nach **Aktualisierung** des Kombinationsfeldes den Wert **[Ereignisprozedur]** ein und klicken auf die Schaltfläche mit den drei Punkten (...) neben der Eigenschaft.

Dies öffnet den VBA-Editor und zeigt gleich die neue, leere Prozedur **cboSucheLieferanten_AfterUpdate** an (siehe Bild 6). Diese füllen wir nun wie folgt:

```
Private Sub cboSucheLieferant_AfterUpdate()
    Dim lngLieferantID As Long
    Dim strSQL As String
    lngLieferantID = Nz(Me!cboSucheLieferant, 0)
    If Not lngLieferantID = 0 Then
        strSQL = "LieferantID = " & lngLieferantID
        Me!sfmLookupsuche.Form.Filter = strSQL
        Me!sfmLookupsuche.Form.FilterOn = True
    Else
        Me!sfmLookupsuche.Form.Filter = ""
    End If
End Sub
```

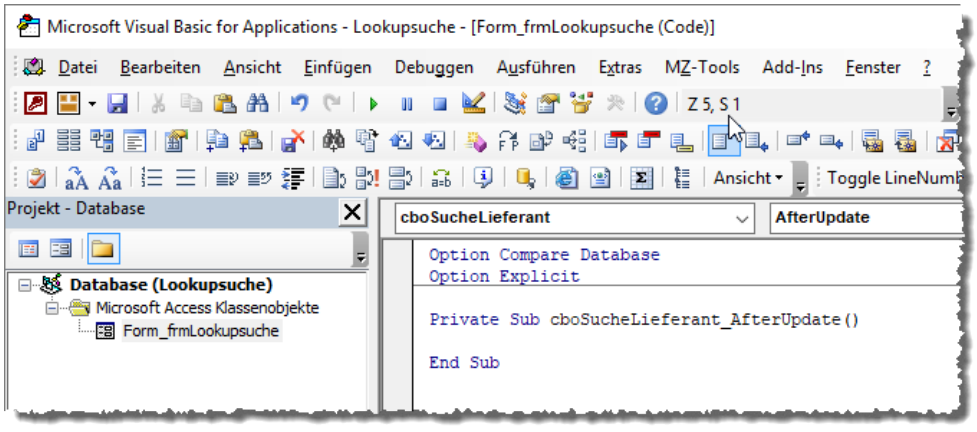


Bild 6: Neue Ereignisprozedur im VBA-Editor

Diese Prozedur deklariert zunächst zwei Variablen namens **lngLieferantID** und **strSQL**. Die Variable **lngLieferantID** wird gleich mit dem aktuellen Wert des Feldes **cboSuche** gefüllt. Allerdings kann es vorkommen, dass der Benutzer das Kombinationsfeld geleert hat, und dann liefert die-

Zeilenvorschub in Berichten steuern

Es gibt einige Möglichkeiten, den Zeilenvorschub bei der Ausgabe in Berichten zu steuern. Dazu gehört die Eigenschaft **MoveLayout**, die Sie in entsprechenden Ereignissen eines Berichts setzen können, um die Ausgabe des Detailbereichs zu steuern. Ein schönes Beispiel dazu ist die Ausgabe eines Monatskalenders, bei dem jeweils eine Zeile eine Woche abbildet und jede neue Woche in einer weiteren Zeile beginnt.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1706_BerichtsausgabeSteuern.accdb**.

Vorbereitungen

Als Beispiel für die Eigenschaften eines Berichts, um mehrere Datensätze in einer Zeile anzuzeigen oder einen Zeilenvorschub zu bewirken, verwenden wir eine Tabelle namens **tblDatumswerte**. Diese soll im Feld Datumswert einige hundert Datumswerte enthalten, mit denen wir im Bericht komfortabel experimentieren können. Der Entwurf dieser Tabelle ist einfach gehalten und umfasst neben dem obligatorischen (und hier noch nicht einmal nötigen) Primärschlüsselfeld noch das Feld **Datumswert** (siehe Bild 1).

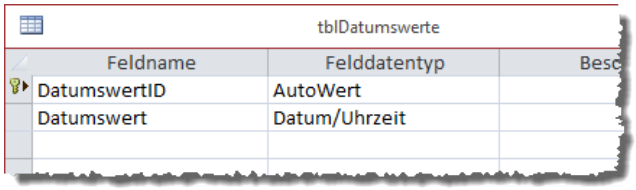


Bild 1: Entwurf der Datumstabelle

Diese Tabelle füllen wir mit der Prozedur **DatumsangabenSchreiben**, welche Sie im Modul **mdlTools** der Beispieldatenbank finden. Die Prozedur sieht wie in Listing 1 aus und durchläuft eine Schleife, in der jeweils ein Datensatz zur Tabelle hinzugefügt wird, bis 1.001 Datumsangaben in der Tabelle liegen. Die Schleife läuft über eine Datumsvariable namens **datCurrent**, der Startwert ist der 1.1.2017, der Endwert ist dieses Datum plus 1.000.

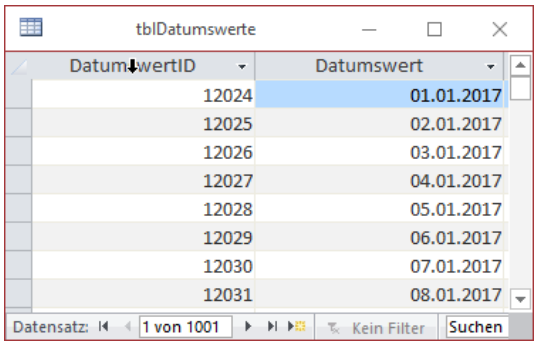


Bild 2: Gefüllte Datumstabelle

Das Ergebnis sieht in der Tabelle wie in Bild 2 aus.

Datum im Bericht

Nun wollen wir den Bericht erstellen, für den wir vorerst

```
Public Sub DatumsangabenSchreiben()
    Dim db As DAO.Database
    Dim datCurrent As Date
    Dim datStart As Date
    Set db = CurrentDb
    datStart = "1.1.2017"
    db.Execute "DELETE * FROM tblDatumswerte", dbFailOnError
    For datCurrent = datStart To datStart + 1000
        db.Execute "INSERT INTO tblDatumswerte(Datumswert) VALUES(" & Format(datCurrent, "\#yyyy\-mm\-dd\#") & ")", dbFailOnError
    Next datCurrent
End Sub
```

Listing 1: Füllt die Tabelle **tblDatumswerte** mit 1.000 Datensätzen

die folgende Vorgabe festlegen wollen: Er soll in einer Zeile sieben Datumsangaben anzeigen und dann in der nächsten Zeile mit den Datumsangaben fortfahren. Um die Wochentage et cetera kümmern wir uns vorerst nicht.

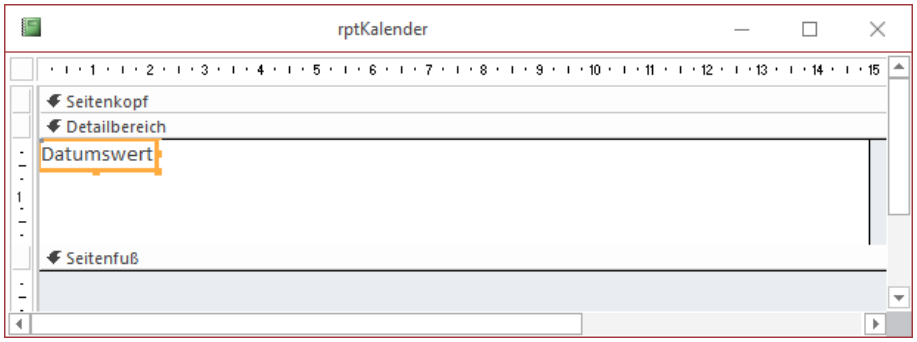


Bild 3: Einfügen des Feldes zur Anzeige des Datums

Als Erstes legen wir einen neuen Bericht namens **rptKalender** an, dem wir als Datenherkunft die Tabelle **tblDatumswerte** hinzufügen. Dann ziehen Sie das Feld **Datumswert** aus der Feldliste in den Detailbereich des Berichtsentwurfs und entfernen das Bezeichnungsfeld dieses Steuerelements. Schieben Sie das Steuerelement dann nach ganz links oben. Ändern Sie seine Größe so, dass es sieben Mal nebeneinander in den Bericht passt (siehe Bild 3).

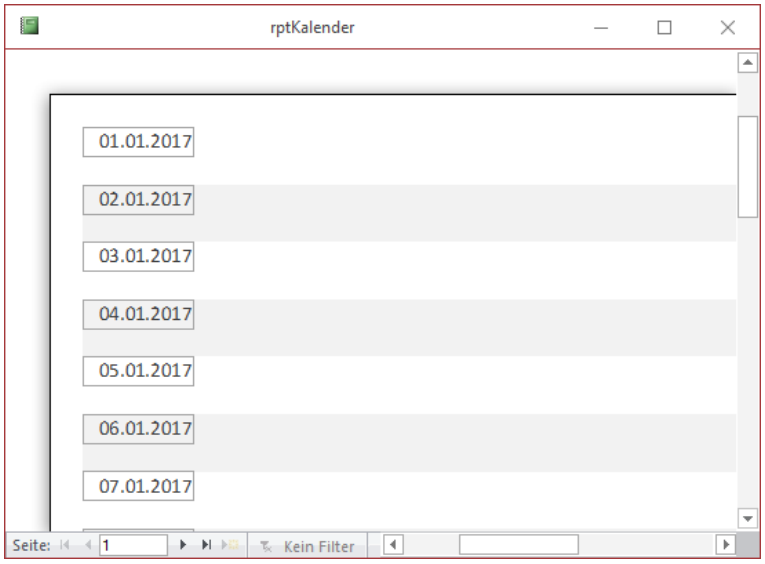


Bild 4: Bericht in der herkömmlichen Ansicht mit einem Datensatz je Zeile

Wenn Sie den Bericht nun in der Seitenansicht öffnen, werden die Datumswerte natürlich alle untereinander angezeigt (siehe Bild 4). Bevor wir dafür sorgen, dass sieben Datensätze je Zeile angezeigt werden, entfernen wir noch die alternierenden Hintergrundfarben und die Textrahmen. Stellen Sie dazu für den Detailbereich die Eigenschaft **Alternative Hintergrundfarbe** auf den gleichen Wert ein wie die Eigenschaft **Hintergrundfarbe** sowie die Eigenschaft **Rahmenart** des Textfeldes auf den Wert **Transparent**.

Mehrere Datensätze pro Zeile

Nun kümmern wir uns um die Wiederholung der Datensätze pro Zeile. Dazu fügen Sie für den Detailbereich des Berichts eine Ereignisprozedur für das Ereignis **Beim Formatieren** hinzu (siehe Bild 5).

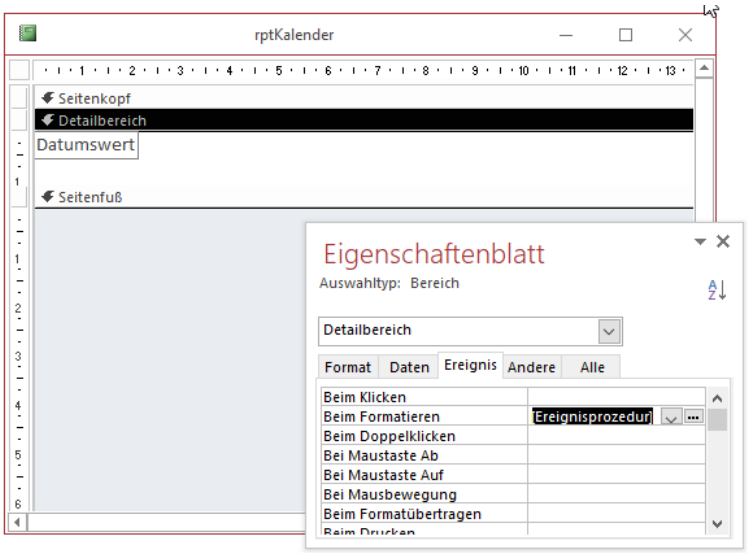


Bild 5: Einfügen einer Ereignisprozedur für **Beim Formatieren**

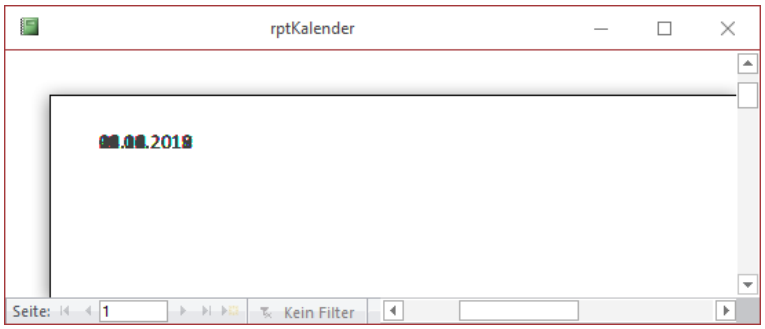


Bild 6: Alle Datensätze werden übereinander geschrieben.

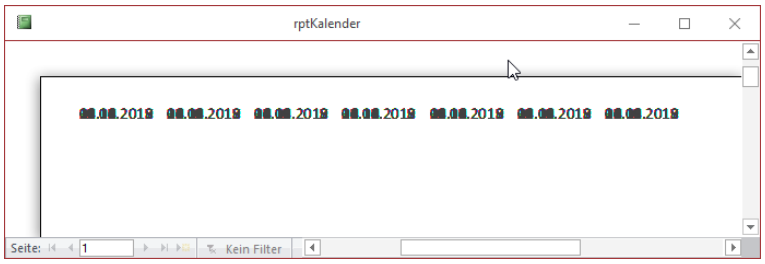


Bild 7: Nun sind die Datensätze immerhin auf sieben Spalten aufgeteilt.

In einem ersten Versuch stellen wir in dieser Prozedur die Eigenschaft **MoveLayout** auf den Wert **False** ein:

```
Private Sub Detailbereich_Format(Cancel As Integer, _
        FormatCount As Integer)
    Me.MoveLayout = False
End Sub
```

Dies führt uns bereits in die richtige Richtung, aber noch nicht vollständig: Wie Bild 6 zeigt, erfolgt nun kein Zeilenwechsel mehr. Allerdings werden nun alle Datensätze der Tabelle übereinander abgebildet. Wir sorgen nun im nächsten Schritt dafür, dass die Elemente immer eine Position weiter nach rechts rücken.

Dazu behelfen wir uns der Einstellung der Eigenschaft **Left**, welche den Abstand eines Steuerelements vom linken Rand angibt. Wir führen eine modulweit zugängliche Variable namens **intPosition** ein, welche die Position des Textfeldes für den aktuellen Datensatz speichern soll. Dieser Wert wird immer beim Aufruf der **Beim Formatieren**-Ereignisprozedur um eins erhöht. Da dies wie in Listing 2 immer zu Beginn der Prozedur geschieht, lautet dieser Wert beim ersten Durchgang also beispielsweise **1**. Wenn **intPosition** größer als 7 wird, soll es wieder auf den Wert **1** eingestellt werden. Immer nach der Ermittlung des aktuellen Wertes für **intPosition** stellt die Prozedur die Eigenschaft **Left** ein, und zwar auf die über die Eigenschaft **Width** ermittelte Breite multipliziert mit **intPosition - 1**.

Für den ersten Datensatz beträgt der Wert für **Left** also **0**, da **intPosition - 1** gleich **0** ist und auch das Produkt somit zu **0** wird. Die Breite des Steuerelements beträgt aktuell **1206**, also wird das Steuerelement durch Einstellen der Eigenschaft **Left** auf diesen Wert für den zweiten Datensatz genau rechts neben dem ersten Textfeld platziert. Auf diese Weise geht es weiter, bis das siebte Feld mit dem Wert **1206 * 6** für die Eigenschaft **Left** ganz rechts angelegt wird und **intPositon** für den folgenden Datensatz wieder auf den Wert **1** gesetzt wird.

```
Dim intPosition As Integer

Private Sub Detailbereich_Format(Cancel As Integer, FormatCount As Integer)
    Me.MoveLayout = False
    intPosition = intPosition + 1
    If intPosition > 7 Then
        intPosition = 1
    End If
    Me!Datumswert.Left = Me!Datumswert.Width * (intPosition - 1)
End Sub
```

Listing 2: Die Datensätze sollen nun auf sieben Positionen aufgeteilt werden.

Auf diese Weise erhalten wir das Ergebnis aus Bild 7. Nun wollen wir als Nächstes alle sieben Datensätze einen Zeilenumbruch einfügen.

Jede Woche in einer Zeile

Wir haben mit der Einstellung der Eigenschaft **MoveLayout** auf den Wert **False** dafür

Mehrere Datensätze pro Zeile im Bericht

Mit Berichten kann man mehr anstellen, als dies bei Betrachtung der Entwurfsansicht und der Eigenschaften den Anschein macht. So können Sie mit Berichten auch durchaus mehrere Datensätze pro Zeile anzeigen, was durch reines Platzieren von Steuerelementen in den entsprechenden Bereichen allerdings nur in der mehrspaltigen Ansicht möglich ist. In diesem Artikel schauen wir uns die verschiedenen Möglichkeiten an.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1706_MehrereDatensaetzeProZeile.accdb**.

Mehrspaltige Berichte

In diesem Beispiel wollen wir eine Reihe von Adressblöcken in einer mehrspaltigen Ansicht anlegen. Dazu verwenden wir die Tabelle **tblKunden** der Beispieldatenbank als Datenherkunft für die folgenden Beispielberichte. Der Bericht zur Darstellung der Kundendaten soll **rptKundenadressen** heißen und sieht im Entwurf wie in Bild 1 aus. Nachdem wir die Felder aus der Feldliste in den Bericht gezogen haben, sind noch folgende Schritte nötig, damit dieser in mehreren Zeilen angezeigt wird:

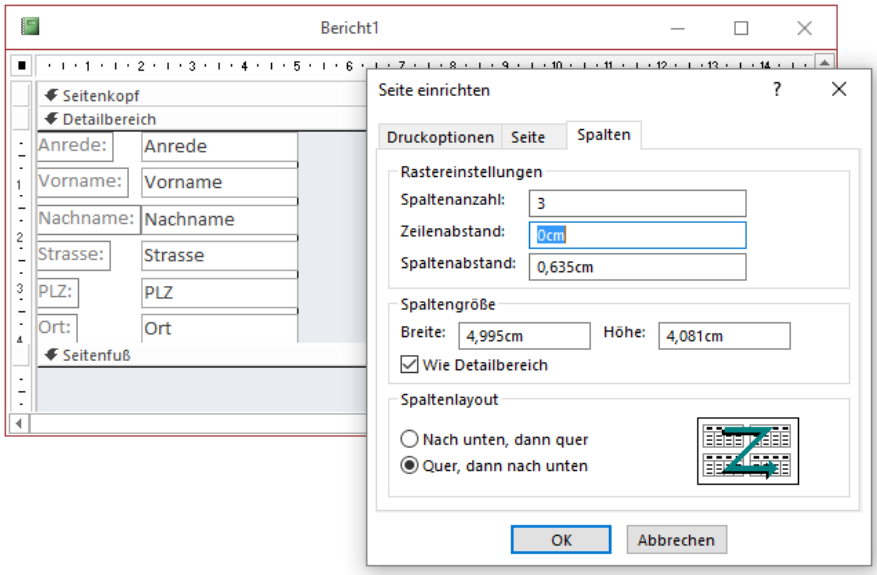


Bild 1: Einstellungen für einen mehrspaltigen Bericht

- Wir stellen die Breite des Detailbereichs auf eine Breite ein, die maximal der Spaltenbreite einer Spalte entspricht. Wenn Sie dies nicht tun und die Breite des Detailbereichs multipliziert mit der Anzahl der gleich noch auszuwählenden Spalten ist größer als die gesamte Breite des Berichts, erhalten Sie die Meldung aus Bild 2.
- Öffnen des Dialogs **Seite einrichten** über den entsprechenden Ribbon-Befehl. Hier wechseln Sie zur Registerseite **Spalten** und stellen dort die

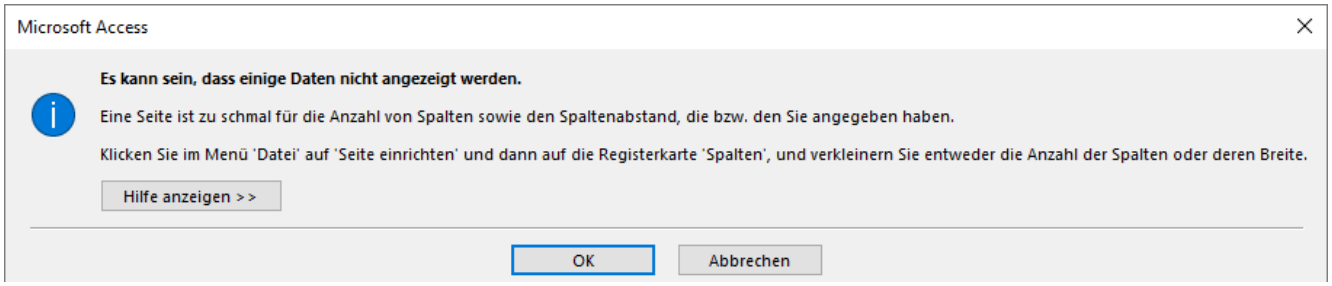


Bild 2: Meldung, wenn die Summe der Spaltenbreiten die Breite des Berichts überschreitet

Anzahl der Spalten mit der Eigenschaft **Spaltenanzahl** auf 3 ein. Unter **Spaltenlayout** behalten wir die Einstellung **Quer, dann nach unten** bei.

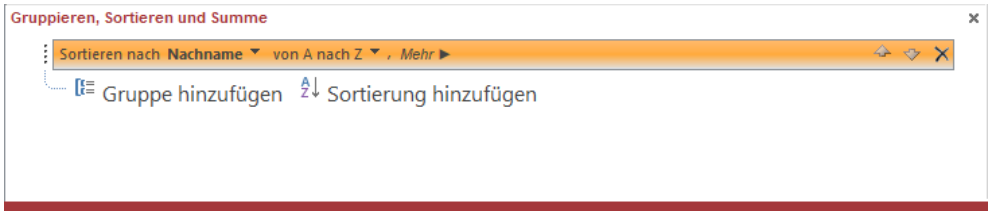


Bild 3: Einstellungen für einen mehrspaltigen Bericht

Damit der Bericht die Kunden nach dem Nachnamen sortiert ausgibt, haben wir diesem Bericht noch eine Sortierung nach dem Feld **Nachname** hinzufügt. Dies erledigen Sie durch einen Mausklick auf die Schaltfläche **Sortierung hinzufügen** unten im Access-Fenster. Hier wählen Sie dann die gewünschte Sortierung aus, in diesem Fall das Feld **Nachname** (siehe Bild 3).

Danach können wir einen ersten Versuch wagen und uns die Seitenansicht anschauen. Diese sieht wie in Bild 4 aus und zeigt zunächst abwechselnd weiß und grau hinterlegte Bereiche sowie eingerahmte Feldwerte. Dies ändern wir als Erstes:

- Wechseln Sie in die Entwurfsansicht und markieren Sie alle gebundenen Felder im Detailbereich. Stellen Sie dann die Eigenschaft **Rahmenart** auf **Transparent** ein.
- Anschließend markieren Sie den Detailbereich mit einem Mausklick auf den grauen Bereich mit der entsprechenden Beschriftung und stellen die Eigenschaft **Alternative Hintergrundfarbe** auf den gleichen Wert ein, der auch für die Eigenschaft **Hintergrundfarbe** eingestellt ist (siehe Bild 5).

Wenn Sie nun erneut in die Seitenansicht wechseln, sieht das Bild schon besser aus (siehe Bild 6). Die alternierenden Hintergrundfarben sind verschwunden und auch die Textrahmen sind ausgeblendet.

Bild 4: Mehrspaltiger Bericht in der Seitenansicht

Anfangsbuchstabe als Zeilenüberschrift

Nun wollen wir den Bericht noch etwas hübscher machen und die ausgegebenen Adressblöcke nach dem Anfangsbuchstaben gruppieren. Dazu passen wir zunächst die Datenherkunft des Berichts an. Die Abfrage sieht nun wie in Bild 7 aus. Wir haben alle Felder der Tabelle **tblKunden** zum Abfrageentwurf hinzugefügt und noch ein weiteres Feld namens **Anfangsbuchstabe**. Dieses enthält den Ausdruck **Links(Nachname;1)** als Wert. Es liefert jeweils den Anfangsbuchstaben des Nachnamens des aktuellen Datensatzes.

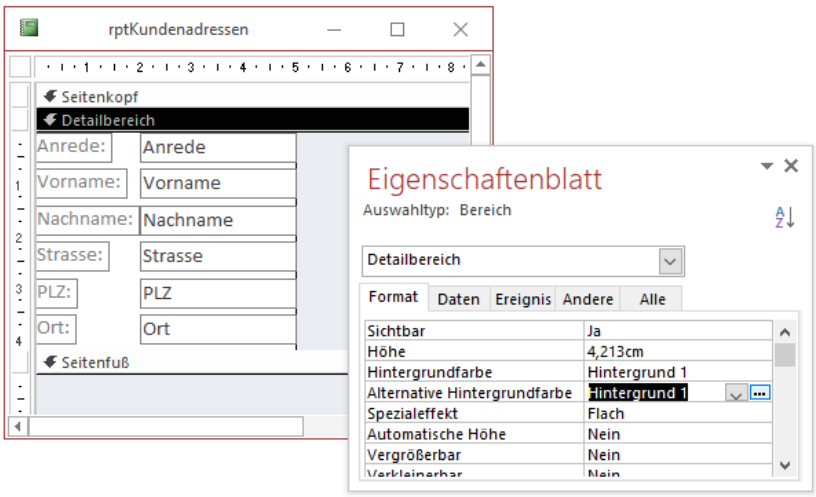


Bild 5: Anpassen der Hintergrundfarbe

Gruppierung hinzufügen

Dem Berichtsentwurf fügen wir nun neben der Sortierung im unteren Bereich noch eine Gruppierung hinzu, indem wir auf die Schaltfläche **Gruppierung hinzufügen** klicken. Die Gruppierung wird standardmäßig unter den bereits bestehenden Sortierungen und Gruppierungen angelegt, sodass wir diese noch mit der **Nach oben**-Schaltfläche um eine Position nach oben verschieben müssen. Die Gruppierung soll nach dem Inhalt des Feldes **Anfangsbuchstabe** erfolgen. Das Feld **Anfangsbuchstabe** fügen wir dem automatisch erstellten Gruppenkopfbereich für diese Gruppe hinzu (siehe Bild 8).

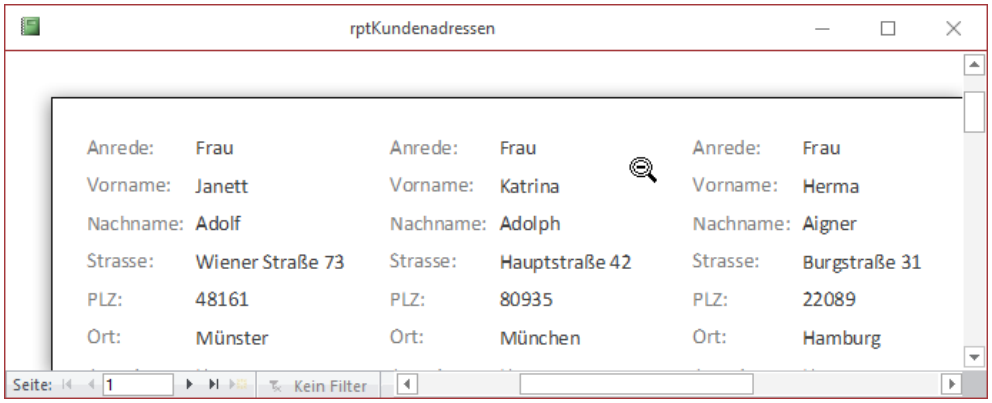


Bild 6: Bericht ohne Ränder und Textrahmen

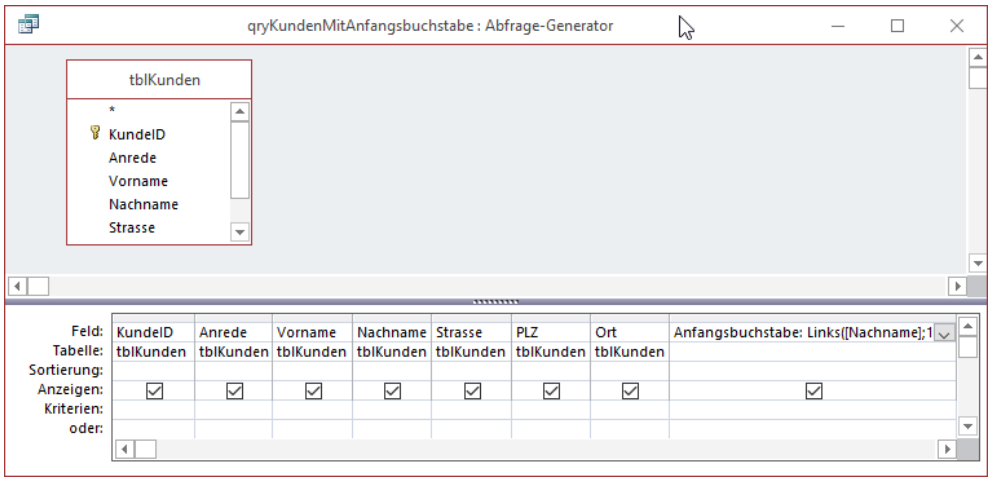


Bild 7: Datenherkunft für den Bericht mit Gruppierung nach Anfangsbuchstaben

Der Wechsel in die Seitenansicht bremst uns ein wenig: Die Gruppenkopfbereiche werden nämlich auf die gleiche Breite gesetzt wie die mehrspaltig angezeigten Adressblöcke. Außerdem befinden sich die Anfangsbuchstaben auch nicht über den dazu gehörenden Adressblöcken, sondern daneben (siehe Bild 9). Um es gleich festzuhalten: Es gibt keine Möglichkeit, unsere gewünschte Darstellung zu erhalten, ohne VBA einzusetzen.

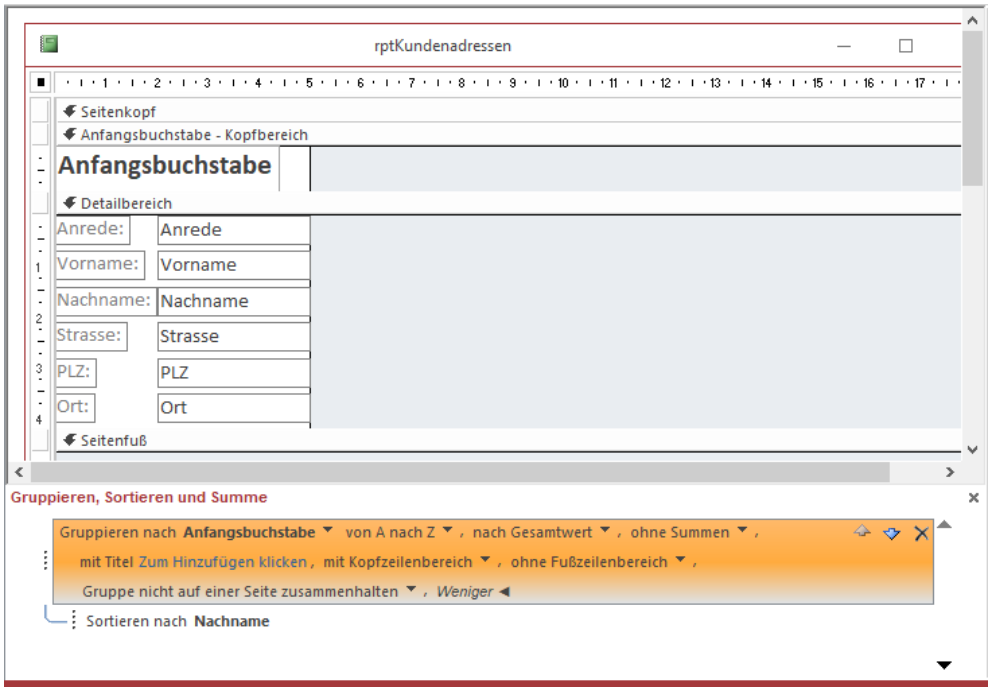


Bild 8: Einfügen einer neuen Gruppierung nach dem Anfangsbuchstaben

Alternative mit VBA

Zu der hier erreichten Darstellungsart gibt es neben den Einstellungen für die Spaltenanzahl noch eine weitere Möglichkeit. Diese verwendet VBA, um anzugeben, ob ein Datensatz noch in der aktuellen Zeile gedruckt oder in die folgende Zeile verschoben werden soll. Die Grundlagen hierzu finden Sie im Artikel **Zeilenvorschub in Berichten steuern**.

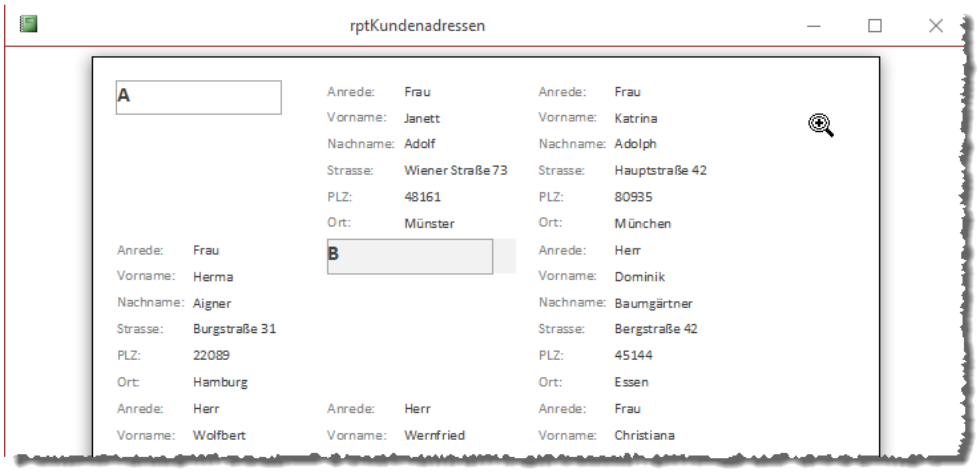


Bild 9: Die Gruppierung nach dem Anfangsbuchstaben ist etwas missglückt.

Von dort haben wir den Großteil der folgenden Prozeduren. Zunächst deklarieren wir zwei Variablen. Die erste nimmt die aktuelle Spalte auf, die zweite die Breite einer Spalte:

```
Dim intSpalte As Integer
Dim intWidth As Integer
```

Die Breite der Spalte ermitteln wir aus der Breite des Feldes **Anrede** plus **100** als Abstand zwischen zwei Spalten:

```
Private Sub Report_Load()
    intWidth = Me.Anrede.Width + 100
End Sub
```

Danach übernehmen wir die Prozedur aus dem oben genannten Artikel. Dort sorgen wir dafür, dass

Haupt- und Unterformular synchron

Manchmal zeigt das Hauptformular die gleichen Daten wie das Unterformular an – beispielsweise, wenn das Unterformular als Übersichtsliste die Daten in der Datenblattansicht liefert und ein Wechseln des Datensatzes im Unterformular zur Anzeige der Details des gleichen Datensatzes in im Hauptformular führen soll. Soll dies auch umgekehrt der Fall sein, also das beim Wechsel des Datensatzes im Hauptformular der entsprechende Datensatz im Unterformular markiert wird, wird es kompliziert. Mögliche Probleme und die Auflösung liefert dieser Beitrag.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1706_HauptUndUnterformularSynchron.accdb**.

Beispielkonstellation

Wir wollen die Daten einer Tabelle namens tblHersteller aus unserer Beispiellösung Handyverwaltung herausnehmen und an diesem Beispiel zeigen, wie Haupt- und Unterformular synchron gehalten werden können. Dazu legen wir zunächst ein neues Unterformular namens **sfmHersteller** an, das die Abfrage **qryHerstellerBezeichnungNachAlphabet** als Datenherkunft verwendet. Diese Abfrage liefert nur die beiden Felder **HerstellerID** und **Bezeichnung** der Tabelle **tblHersteller**, und zwar alphabetisch nach dem Inhalt des Feldes **Bezeichnung** sortiert.

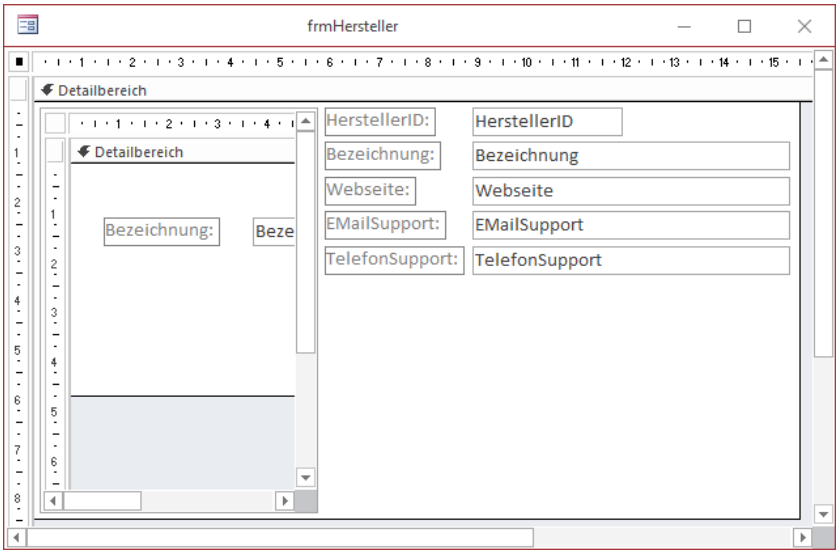


Bild 1: Haupt- und Unterformular zur Anzeige der Daten aus der gleichen Datenherkunft

Das Hauptformular **frmHersteller** verwendet eine ähnliche Abfrage namens **qryHerstellerNachBezeichnung**, welche allerdings alle Felder der Tabelle **tblHersteller** enthält. Auch hier gibt es eine Sortierung nach dem Feld **Bezeichnung**.

Das Unterformular platzieren wir wie in Bild 1 im Hauptformular. Nun wollen wir, dass das Hauptformular und das Unterformular immer den gleichen Datensatz anzeigen beziehungsweise dass im Unterformular immer der Datensatz markiert ist, der aktuell im Hauptformular angezeigt wird und umgekehrt.

Erster Versuch

Dabei stellen wir uns nun vor, dass Folgendes funktionieren könnte und probieren es aus. Für das Ereignis **Beim Anzeigen** des Hauptformulars legen wir die folgende Ereignisprozedur an:

```
Private Sub Form_Current()
    Me!sfmHersteller.Form.Recordset!HerstellerID _
                                    = Me!HerstellerID
End Sub
```

Diese Prozedur stellt also den Datensatzmarkierer für das Recordset im Unterformular auf den Datensatz

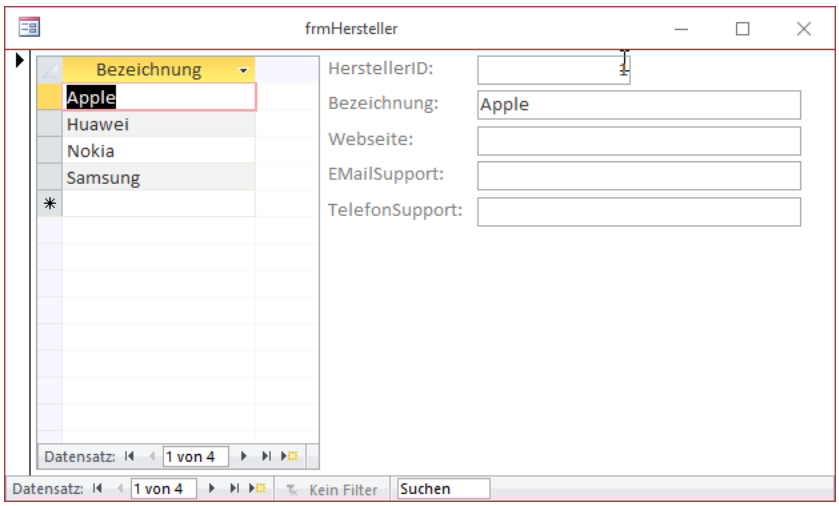


Bild 2: Beim Anzeigen erscheinen die Datensätze im Haupt- und Unterformular synchron.

ein, dessen Primärschlüsselfeld **HerstellerID** dem passenden Wert im Hauptformular entspricht.

Für das entsprechende Ereignis des Unterformulars verwenden wir eine ganz ähnliche Ereignisprozedur, welche die folgende Zeile enthält:

```
Private Sub Form_Current()
    Me.Parent.Recordset.FindFirst "HerstellerID = " _
        & Me!HerstellerID
End Sub
```

Diese versucht, den Datensatzzeiger im Recordset des Hauptformulars auf den Datensatz einzustellen, dessen **HerstellerID** im Unterformular ausgewählt wurde. Öffnen wir nun das Formular in der Formularansicht, erhalten wir die Ansicht aus Bild 2 – es klappt also wie gewünscht.

Das ist auch der Fall, wenn wir zwischen den Datensätzen im Haupt- oder im Unterformular navigieren. Es tritt erst ein Problem auf, wenn wir den Datensatzzeiger im Haupt- oder Unterformular auf einen neuen, leeren Datensatz verschieben.

Dann erhalten wir die Fehlermeldung aus Bild 3. Der Grund für diese Fehlermeldung ist, dass **Me!HerstellerID** zu diesem Zeitpunkt, also beim Anlegen eines neuen Datensatzes, den Wert **Null** enthält. Dadurch sieht der Ausdruck, der als Kriterium der **FindFirst**-Methode verwendet wird, wie folgt aus:

```
HerstellerID =
```

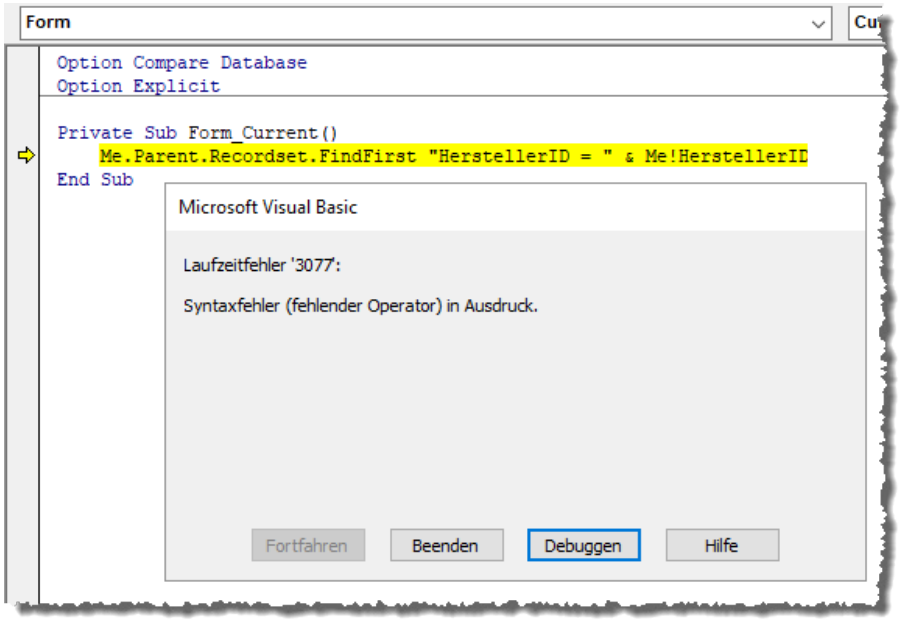


Bild 3: Fehler beim Anzeigen eines neuen, leeren Datensatzes

Diesen Ausdruck kann Access nicht auswerten, also wird die Fehlermeldung aus der Abbildung generiert.

Ändern wir die beiden Ereignisprozeduren also etwas ab, um die Fehlermeldung zu verhindern. Dazu fassen wir die Angabe des Wertes **Me!Hersteller** einfach mit der Funktion **Nz** ein und geben als zweiten Parameter den Wert **0** an, was dafür sorgt, dass die Funktion den Wert **0** liefert, wenn **HerstellerID** den Wert Null aufweist:

```
'Hauptformular
Private Sub Form_Current()
    Me!sfmHersteller.Form.Recordset.FindFirst "Herstel-
lerID = " & Nz(Me!HerstellerID, 0)
End Sub

'Unterformular
Private Sub Form_Current()
    Me.Parent.Recordset.FindFirst "HerstellerID = " &
Nz(Me!HerstellerID, 0)
End Sub
```

Dies führt nun immerhin dazu, dass beim Ansteuern eines neuen, leeren Datensatzes im Haupt- oder Unterformular kein Fehler mehr ausgelöst wird. Allerdings finden wir hier ein etwas merkwürdiges Verhalten vor, denn wenn Sie etwa mit der Tabulator-Taste im Unterformular vom letzten auf den neuen, leeren Datensatz wechseln, springt der Datensatzzeiger auf den ersten Datensatz im Unterformular. Das Gleiche geschieht auch manchmal, wenn Sie mit der Maus von einem anderen Datensatz auf den neuen, leerenDatensatz wechseln. Wenn Sie hingegen soeben auf den letzten Datensatz gewechselt sind und der Datensatzzeiger dadurch auf den ersten Datensatz gesprungen ist und dann mit der Maus auf den neuen, leeren Datensatz klicken, verbleibt der Datensatzzeiger auf diesen Datensatz.

Was geschieht im Hintergrund genau? Um das herauszufinden, haben wir den beiden Ereignisprozeduren jeweils eine **Debug.Print**-Anweisung hinzugefügt, welche einen Hinweis auf die jeweilige Prozedur im Haupt- oder Unterformular im Direktfenster ausgibt:

```
'Hauptformular
Private Sub Form_Current()
    Debug.Print "Hauptformular_Current"
    Me!sfmHersteller.Form.Recordset.FindFirst "Herstel-
lerID = " & Nz(Me!HerstellerID, 0)
End Sub

'Unterformular
Private Sub Form_Current()
    Debug.Print "Unterformular_Current"
    Me.Parent.Recordset.FindFirst "HerstellerID = " &
Nz(Me!HerstellerID, 0)
End Sub
```

Wenn wir das Formular nun öffnen, werden die Ereignisse in dieser Reihenfolge ausgelöst:

```
Unterformular_Current
Hauptformular_Current
```

Das Verschieben des Datensatzzeigers über das Ereignis im Unterformular sorgt also dafür, dass auch das Ereignis **Beim Anzeigen** im Hauptformular ausgelöst wird. Obwohl dieses wiederum den Datensatzzeiger im Unterformlar verschiebt, löst dies nicht nochmal das Ereignis **Beim Anzeigen** im Unterformular aus. Offensichtlich hat Access einen Automatismus eingebaut, der Zirkelbezüge zwischen Prozeduren schnell erkennt und nach einmaliger Ausführung abbricht. Dabei wird die Prozedur im Hauptformular genau nach dem Aufruf von **FindFirst** im Unterformular ausgelöst. Nach dem Abarbeiten der Prozedur im Hauptformular wird dann noch die letzte Zeile der Prozedur im Unterformular erledigt (siehe Bild 4).

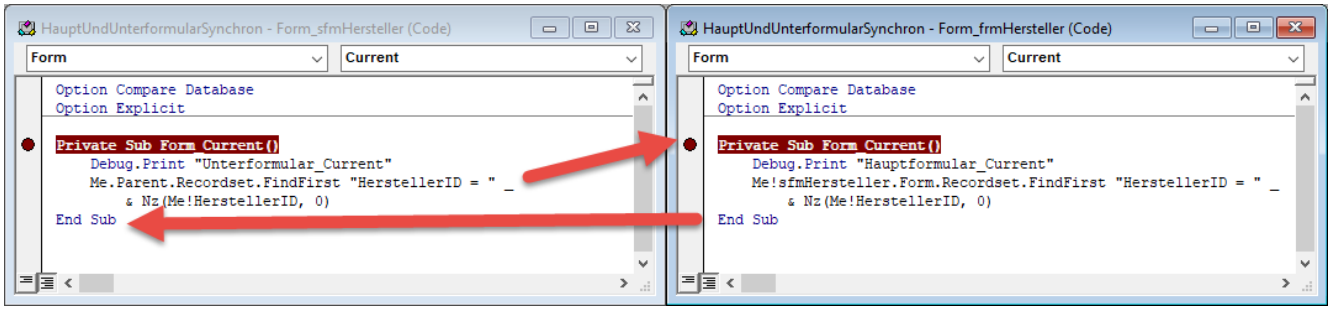


Bild 4: Ablauf der beiden Prozeduren beim Öffnen des Formulars

Wenn Sie mit der Tabulator-Taste von einem Datensatz zum nächsten wechseln, löst dies jeweils die gleichen Ereignisse aus. Wenn wir auf diese Weise allerdings auf dem neuen, leeren Datensatz landen, werden folgende Ereignisse ausgelöst:

```
Unterformular_Current
Hauptformular_Current
Unterformular_Current
```

Dabei löst **FindFirst** im Unterformular die Prozedur im Hauptformular aus und **FindFirst** im Hauptformular nochmals die entsprechende Prozedur im Unterformular. Erst danach bricht die Abarbeitung der Ereignisprozeduren ab.

Wenn der Datensatzzeiger hingegen nach dem erstmaligen Auswählen des neuen, leeren Datensatzes direkt auf den ersten verschoben wird und wir dann nochmal mit der Maus auf den neuen, leeren Datensatz klicken, wird nur noch die folgende Ereignisprozedur ausgelöst:

```
Unterformular_Current
```

Es wäre schön, wenn wir dieses Verhalten noch harmonisieren könnten und dem Benutzer ein reproduzierbares Verhalten liefern könnten.

Alle Ereignisse im Hauptformular

Der Einfachheit halber verschieben wir dazu das Ereignis **Form_Current** des Unterformulars in das Hauptformular. Mehr Informationen, wie das gelingt, finden Sie im Artikel **Alle Ereignisse im Hauptformular**. Das Ergebnis für unser Beispiel sieht dann wie in Listing 1 aus. Wir benötigen eine Variable, mit der wir das Unterformular referenzieren können und die Ereignisprozedur **Form_Load**, um dieses beim Öffnen des Hauptformulars zu referenzieren. Die übrigen beiden Ereignisprozeduren haben wir nur so angepasst, dass Sie sich auf die richtige Formularinstanz beziehungsweise das richtige Steuerelement beziehen. Wenn wir das Formular nun öffnen, wird nur noch **Form_Current** des Hauptformulars aufgerufen:

```
Hauptformular_Current
```

Das ist logisch, denn wenn die **Form_Load**-Ereignisprozedur im Hauptformular ausgelöst und somit das Unterformular der Variablen sfm zugeordnet wird, wäre die **Form_Current**-Prozedur des Unterformulars normalerweise schon ausgelöst worden.

```
Dim WithEvents sfm As Form

Private Sub Form_Current()
    Debug.Print "Hauptformular_Current"
    Me!sfmHersteller.Form.Recordset.FindFirst "HerstellerID = " & Nz(Me!HerstellerID, 0)
End Sub

Private Sub Form_Load()
    Set sfm = Me!sfmHersteller.Form
    sfm.OnCurrent = "[Event Procedure]"
End Sub

Private Sub sfm_Current()
    Debug.Print "Unterformular_Current"
    Me.Recordset.FindFirst "HerstellerID = " & Nz(sfm!HerstellerID, 0)
End Sub
```

Listing 1: Alle Ereignisprozeduren im Klassenmodul des Hauptformulars

Handyverwaltung IV: Mobilfunkgeräte verwalten

Im vierten Teil der Artikelreihe zum Thema »Handyverwaltung« setzten wir die Arbeit an den Formularen der Lösung fort. Dieser Teil kümmert sich um die Verwaltung der Mobilfunkgeräte. Dazu legen wir wieder ein Übersichtsformular und ein Detailformular an. Das Detailformular wird etwas anspruchsvoller, da wir darin auch die zum jeweiligen Mobilfunkgerät gehörende SIM-Karte verwalten wollen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1706_Handyverwaltung.accdb**.

Vorbereitungen

Den Teil des Datenmodells, den wir uns in diesem Teil der Artikelreihe ansehen, finden Sie in Bild 1. Diesmal geht es um die Mobilfunkgeräte, zu denen ja nicht nur Handys, sondern auch Tablets gehören. Nach aktuellem Stand muss man der Tabelle der Mobilfunkgerätearten (**tblMobilfunkgeraetearten**) ja auch noch Uhren hinzurechnen (Stichwort **AppleWatch**). Dies ist das erste Lookup-Feld, welches wir bei der Eingabe von Mobilfunkgeräten berücksichtigen müssen.

Die zweite Art von zusätzlichen Daten betrifft die SIM-Karten, die in den Geräten stecken. Hierbei ist zu berücksichtigen, dass es Geräte gibt, in denen keine SIM-Karte steckt, aber auch SIM-Karten, die in keinem Gerät stecken. Der Regelfall wird aber sein, dass eine SIM-Karte auch einem Gerät zugeordnet ist.

Dem tragen wir durch eine Tabelle Rechnung, die wie eine Verknüpfungstabelle zur Herstellung einer m:n-Beziehung angezeigt wird und die **tblMobilfunkgeraeteSIMKarten** benannt wird. Aber handelt es sich tatsächlich um eine m:n-Tabelle? Immerhin gibt es ja noch eine weitere Einschränkung: jede SIM-Karte kann nämlich nur in einem einzigen Gerät stecken, und jedes Gerät kann auch nur eine SIM-Karte aufnehmen. (Ups: Stimmt das wirklich, oder gibt es vielleicht sogar Geräte, welche mehr als eine SIM-Karte aufnehmen – etwa, um verschiedene Netze abzudecken? Wie auch immer: Wir werden diesen Fall außen vor lassen.)

Die Beziehungen zwischen der Tabelle **tblMobilfunkgeraeteSIMKarten** und den Tabellen **tblMobilfunkgeraete** und **tblSIMKarten** sehen im Beziehungen-Fenster wie normale 1:n-Beziehungen aus. Wenn wir allerdings doppelt auf den Beziehungspfeil klicken, erkennen wir beispielsweise im Dialog zum Bearbeiten der Beziehung aus Bild 2, dass hier in der Tat eine 1:1-Beziehung angelegt wurde.

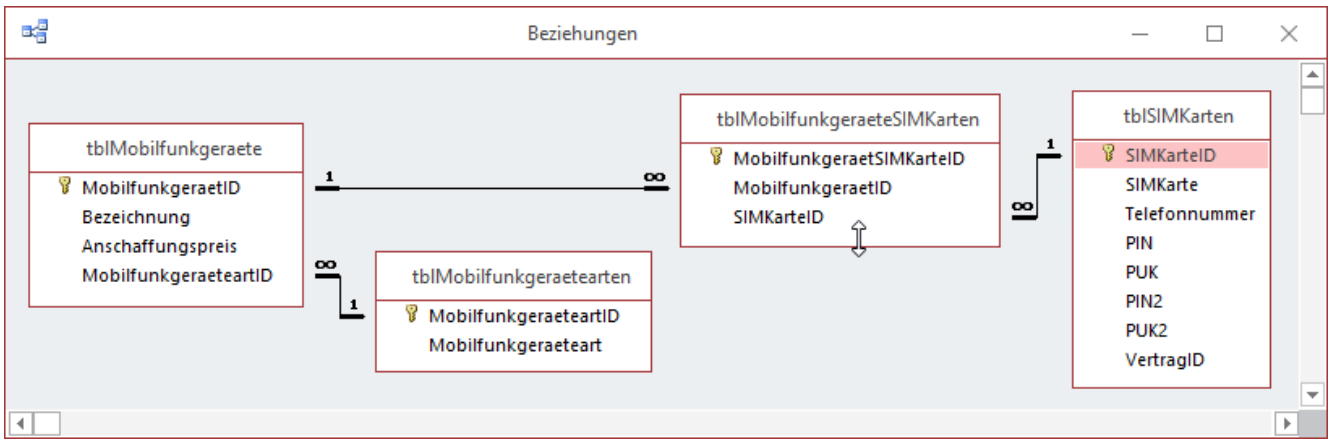


Bild 1: Der Teil des Datenmodells, um den wir uns in diesem Artikel kümmern

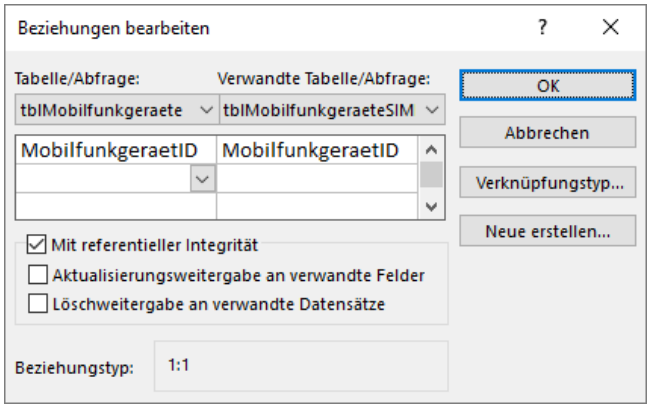


Bild 2: Die Beziehung wird als 1:1-Beziehung eingetragen.

Das ist auch kein Wunder, denn wir haben für die beiden Fremdschlüsselfelder der Verknüpfungstabelle, also für **MobilfunkgeraetID** und **SIMKarteID**, nicht etwa einen zusammengesetzten, eindeutigen Index angelegt, sondern jedes Fremdschlüsselfeld für sich mit einem eindeutigen Index versehen. Deshalb legt Access standardmäßig eine 1:1-Beziehung für diese Beziehungen an. Interessant ist jedoch, dass Access diese im Beziehungen-Fenster nicht als 1:1-Beziehung, sondern als 1:n-Beziehungen abbildet.

Erweiterung des Datenmodells

Nun, da wir uns dem Formular für die Mobilfunkgeräte zuwenden wollen, werfen wir einen genauen Blick auf die Tabelle **tblMobilfunkgeraete** und stellen fest, dass diese doch noch ein wenig mager ausgestattet ist. **MobilfunkgeraetID**, **Bezeichnung**, **Anschaffungspreis** und **MobilfunkgeraeteartID** sind wohl doch etwas wenig Informationen. Als Erstes fehlt uns der Hersteller des Geräts. Ist es ein iPhone? Ein Samsung? Hier stellt sich die allerdings die Frage, ob wir den Hersteller gleich mit den Datensätzen der Tabelle **tblMobilfunkgeraete** verknüpfen wollen. Immerhin gibt es ja auch noch die einzelnen Modelle und Baureihen der verschiedenen Geräte. Wollen wir also den Hersteller in der Tabelle speichern und auch noch die Angabe des Modells? Das macht eigentlich keinen Sinn, denn es gibt ja jedes Modell auch nur von einem Hersteller. Wenn wir dann den Hersteller zu den Modellen speichern und nochmals zu den Mobilfunkgeräten, hätten wir redundante Daten, die später zu Inkonsistenzen führen könnten. Dann wäre ein Mobilfunkgerät beispielsweise ein iPhone, aber ein unaufmerksamer Benutzer stellt vielleicht als Hersteller des Geräts **Samsung** ein.

Also gehen wir noch einen Schritt weiter und fügen der Tabelle **tblMobilfunkgeraete** zunächst ein Fremschlüsselfeld namens **ModellID** hinzu, welches seine Daten aus der Tabelle **tblModelle** bezieht. Die Tabelle **tblModelle** wiederum enthält beispielsweise Felder mit der Bezeichnung des Modells, der Angabe des Speicherplatzes des Modells und mehr.

Außerdem legen wir hier ein Fremdschlüsselfeld für den Hersteller namens **HerstellerID** an. Den Hersteller wollen wir aus einer Lookup-Tabelle auswählen, die wir zunächst hinzufügen. Die Tabelle heißt **tblHersteller** und enthält die Felder **HerstellerID**, **Bezeichnung**, **Webseite**, **EMailSupport** und **TelefonSupport** – diese können Sie natürlich nach Bedarf erweitern. Für das

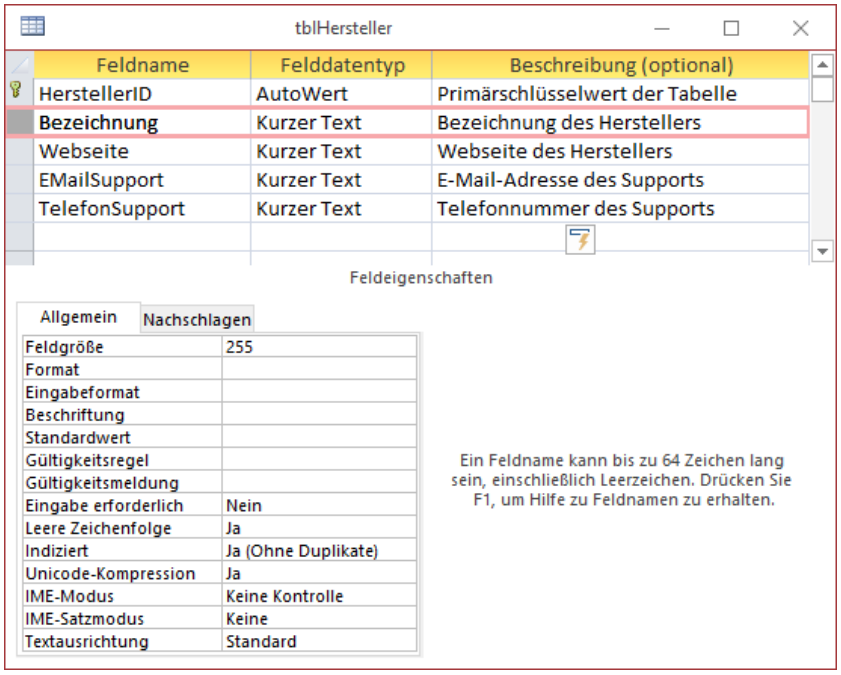


Bild 3: Die Beziehung wird als 1:1-Beziehung eingetragen.

Feld **Bezeichnung** legen wir einen eindeutigen Index fest (siehe Bild 3).

Wir können der Tabelle auch gleich einige Datensätze hinzufügen, damit wir etwas Material zum Testen haben (siehe Bild 4).

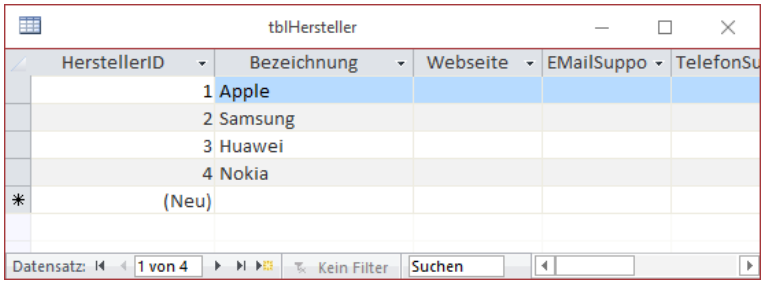


Bild 4: Datensätze der Tabelle **tblHersteller**

Mobilfunkgeräteart verschieben

Wenn wir schon dabei sind, einen genauen Blick auf das Datenmodell zu werfen, sollten wir auch noch schauen, ob das Fremdschlüsselfeld **MobilfunkgeraeteartID** wirklich in die Tabelle **tblMobilfunkgeraete** gehört. In der Tat sollten wir es wohl auch in die Tabelle **tblModelle** verschieben, denn sonst hätten wir das gleiche Problem wie mit den Herstellern – ein iPhone ist nun einmal ein Smartphone und kein Tablet. Genau das wäre aber bei fehlerhafter Eingabe in den aktuellen Stand der Tabelle **tblMobilfunkgeraete** eine mögliche Kombination. Um uns möglichst wenig Arbeit zu machen, nutzen wir die Zwischenablage, um das Feld zu verschieben: Markieren Sie das Feld **MobilfunkgeraeteartID** über den Datensatzmarkierer in der Entwurfsansicht der Tabelle **tblMobilfunkgeraete**, betätigen Sie die Tastenkombination **Strg + C**, wechseln Sie zum Entwurf der Tabelle **tblModelle** und fügen Sie die Zeile dort nach vorheriger Markierung der Zielzeile mit der Tastenkombination **Strg + V** ein. Dies kopiert das Feld mit allen Eigenschaften, also auch mit den Angaben zum Nachschlagefeld. Sie müssen lediglich im Beziehungen-Dialog die alte Beziehung zwischen der Tabelle **tblMobilfunkgeraete** und **tblMobilfunkgeraetearten** löschen, bevor Sie das Feld **MobilfunkgeraeteartID** aus der Tabelle **tblMobilfunkgeraete** entfernen können. Anschließend fügen Sie die passende Beziehung zwischen dem Feld **MobilfunkgeraeteartID** der Tabelle **tblModelle** und dem gleichnamigen Feld der Tabelle **tblMobilfunkgeraetearten** hinzu.

Der aktuelle Zustand dieses Teils des Datenmodells sieht nun wie in Bild 5 aus.

Hersteller verwalten

Für die Hersteller bauen wir uns ein Haupt- und ein Unterformular, wobei beide Formulare wieder die gleiche Datenherkunft nutzen – eine Abfrage, welche die Daten der Tabelle **tblHersteller** nach dem Feld **Bezeichnung** sortiert. In diesem Fall haben wir einige Feinheiten eingebaut, durch die der im Haupt- und im Unterformular angezeigte Datensatz synchron gehalten wird. Das heißt, wenn Sie im Unterformular zwischen den Datensätzen navigieren, wird jeweils der passende Datensatz im Hauptformular angezeigt und umgekehrt. Detailinformationen dazu finden Sie im Artikel **Haupt- und Unterformular synchron**. Das Formular sieht anschließend wie in Bild 6 aus.

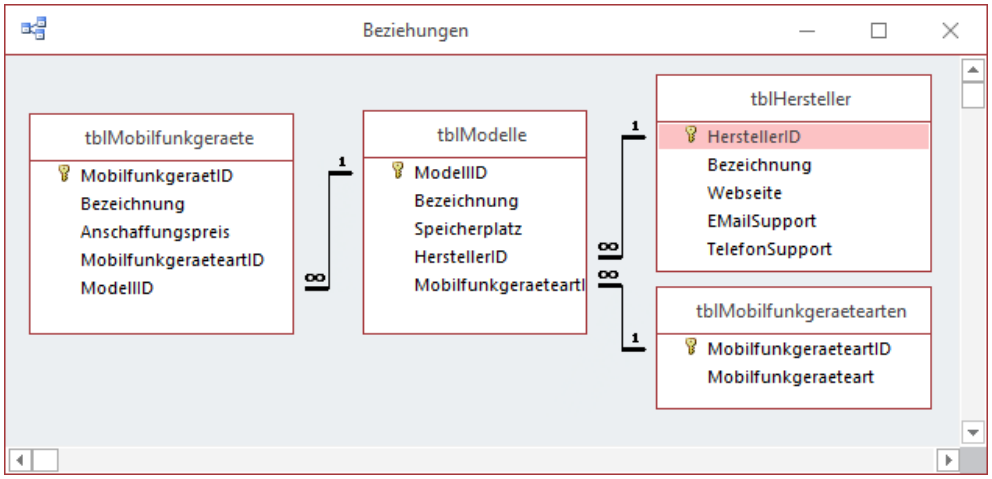


Bild 5: Datenmodell der Tabellen rund um die Mobilfunkgeräte

Formular zur Verwaltung der Mobilfunkgeräte

Für die Verwaltung der Mobilfunkgeräte wollen wir diesmal wieder ein Übersichtsformular erstellen und ein Detailformular zur Anzeige der Einzelheiten der Geräte. Das Übersichtsformular bauen wir als Haupt- und Unterformular auf.

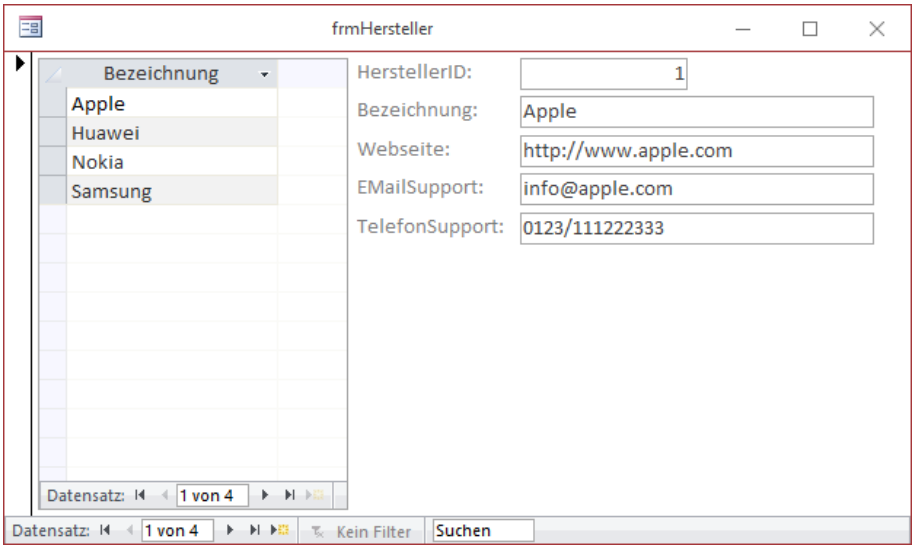


Bild 6: Verwaltung der Hersteller

Das Unterformular sfmMobilfunkgeraeteUebersicht

Das Unterformular **sfmMobilfunkgeraeteUebersicht** verwendet die Abfrage **qryMobilfunkgeraeteUebersicht** als Datenherkunft. Das Unterformular soll alle wichtigen Informationen über das Gerät anzeigen, also binden wir auch die verknüpften Tabellen wie **tblModelle**, **tblHersteller** und **tblMobilfunkgeraetearten** ein. Den Entwurf der Abfrage finden Sie in Bild 7. Da in diesem Unterformular keine neuen Modelle, Hersteller oder Mobilfunkgerätearten zugewiesen werden sollen, fügen wir nicht die Nachschlagefelder der jeweiligen Tabellen hinzu, sondern nur die in den Tabellen enthaltenen Felder. Da wir in den Tabellendefinitionen die Bezeichnungen der enthaltenen Objekte, also etwa in der Tabelle

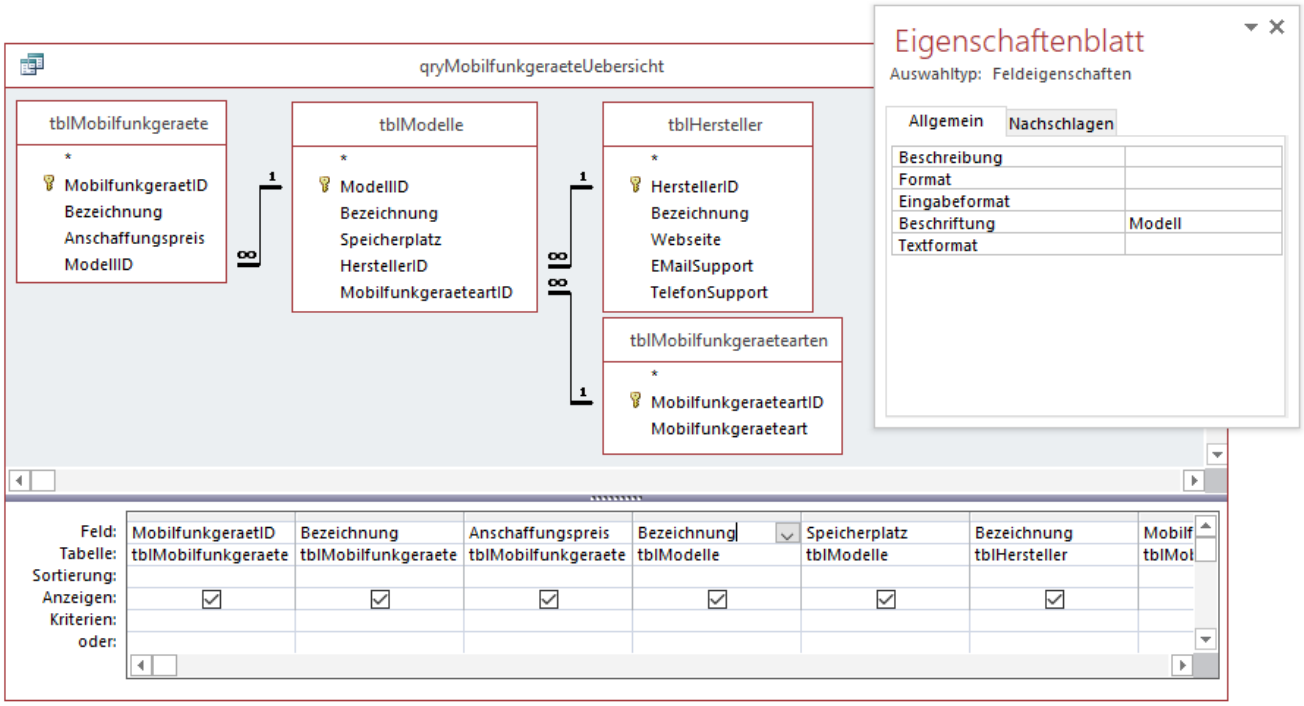


Bild 7: Datenherkunft des Unterformulars **sfmMobilfunkgeraeteUebersicht**

tblMobilfunkgeraete, **tblModelle** oder **tblHersteller** auch mit dem Feldnamen **Bezeichnung** versehen haben, fügen wir den Feldern im Abfrageentwurf noch eine entsprechende Beschriftung hinzu, die dann beim Hinzufügen der Felder in die jeweiligen Formulare berücksichtigt wird. Dies erledigen wir über die Eigenschaft **Beschriftung** im Eigenschaften-Fenster der entsprechenden Eigenschaft.

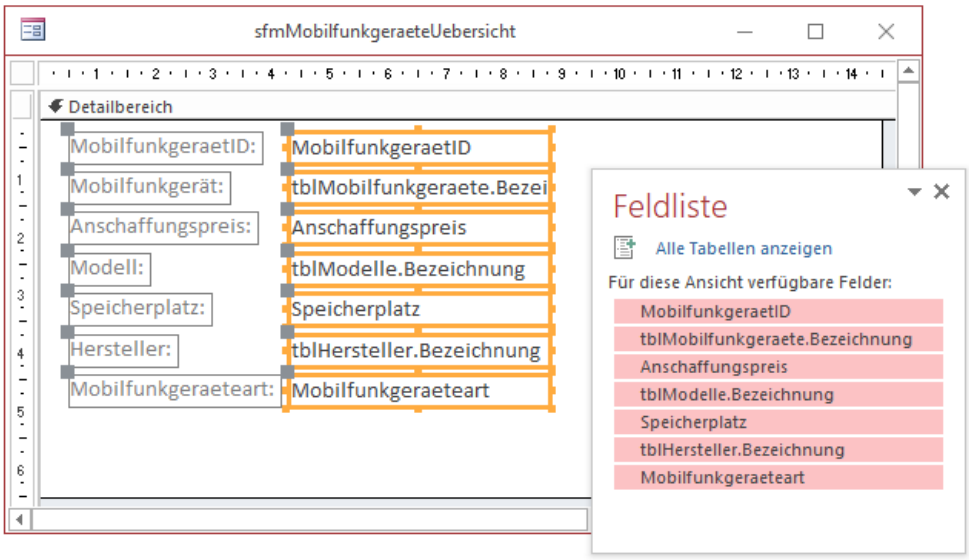


Bild 8: Hinzufügen der Felder zum Entwurf des Formulars **sfmMobilfunkgeraeteUebersicht**

Wenn wir diese Abfrage dem Formular **sfmMobilfunkgeraeteUebersicht** zuweisen und in der Entwurfsansicht die Feldliste einblenden, zeigt diese bei den mehrfach vorkommenden Feldern namens Bezeichnung jeweils zusätzlich den Tabellennamen an (siehe Bild 8). Wenn wir die Felder jedoch in den Detailbereich des Entwurfs ziehen, werden die Beschriftungen in den Bezeichnungsfeldern entsprechend der in der Eigenschaft **Beschriftung** der Felder der Abfrage angepasst. Über das Unterformular direkt sollen keine Datensätze bearbeitet, hinzugefügt oder gelöscht werden, daher stellen wir die Eigenschaften **Anfügen zulassen**, **Löschen zulassen** und **Bearbeitungen zulassen** auf den Wert **Nein** ein.

Wir brauchen nun nur noch die Eigenschaft **Standardansicht** auf den Wert **Datenblatt** einzustellen und können das Formular dann speichern und schließen.

Das Hauptformular frmMobilfunkgeraeteUebersicht

In das Hauptformular **frmMobilfunkgeraeteUebersicht** ziehen wir zunächst das Unterformular **sfmMobilfunkgeraeteUebersicht**. Dann stellen wir die Eigenschaften **Datensatzmarkierer**, **Navigationsschaltflächen**, **Trennlinien** und **Bildlaufleisten** des Formulars auf **Nein** und die Eigenschaft **Automatisch zentrieren** auf **Ja** ein. Außerdem soll das Unterformular-Steuerelement automatisch vergrößert werden, wenn der Benutzer das Hauptformular vergrößert. Dazu stellen wir die Eigenschaften **Horizontaler Anker** und **Vertikaler Anker** jeweils auf **Beide** ein. Nicht vergessen, die Eigenschaften für das Bezeichnungsfeld zum Unterformular, die automatisch mit geändert werden, wieder auf **Links** und **Oben** einzustellen!

Außerdem fügen wir dem Formular drei Schaltflächen namens **cmdDetailsAnzeigen**, **cmdNeuesGeraet** und **cmdGeraetLoeschen** hinzu. Der Entwurf des Hauptformulars sieht dann wie in Bild 9 aus.

Die Schaltfläche **cmdDetailsAnzeigen** ruft die Prozedur aus Listing 1 auf. Diese ermittelt den Primärschlüsselwert des aktuell im Unterformular markierten Datensatzes und ruft dann das Formular **frmMobilfunkgeraetDetails** auf, welches wir im Anschluss erstellen. Als Parameter übergeben wir mit der **DoCmd.OpenForm**-Methode unter anderem eine Bedingung, welche festlegt, welchen Datensatz wir im zu öffnenden Formular anzeigen wollen. Durch das Setzen des Parameters **WindowMode** auf den Wert **acDialog** wird der Code der aufrufenden Prozedur erst weiter ausgeführt, wenn das geöffnete Formular den Fokus

wieder abgibt, also geschlossen oder unsichtbar gemacht wurde.

Die Schaltfläche **cmdNeuesGeraet** soll beim Anklicken ebenfalls dieses Formular aufrufen, allerdings nicht zur Anzeige eines bestehenden Datensatzes, sondern zum Anlegen eines neuen. Auch hier verwenden wir die Methode **DoCmd.OpenForm**, übergeben hier jedoch für den Parameter **DataMode** den Wert **acFormAdd**. Damit zeigt das zu öffnende Formular gleich einen neuen, leeren Datensatz an (siehe Listing 2).

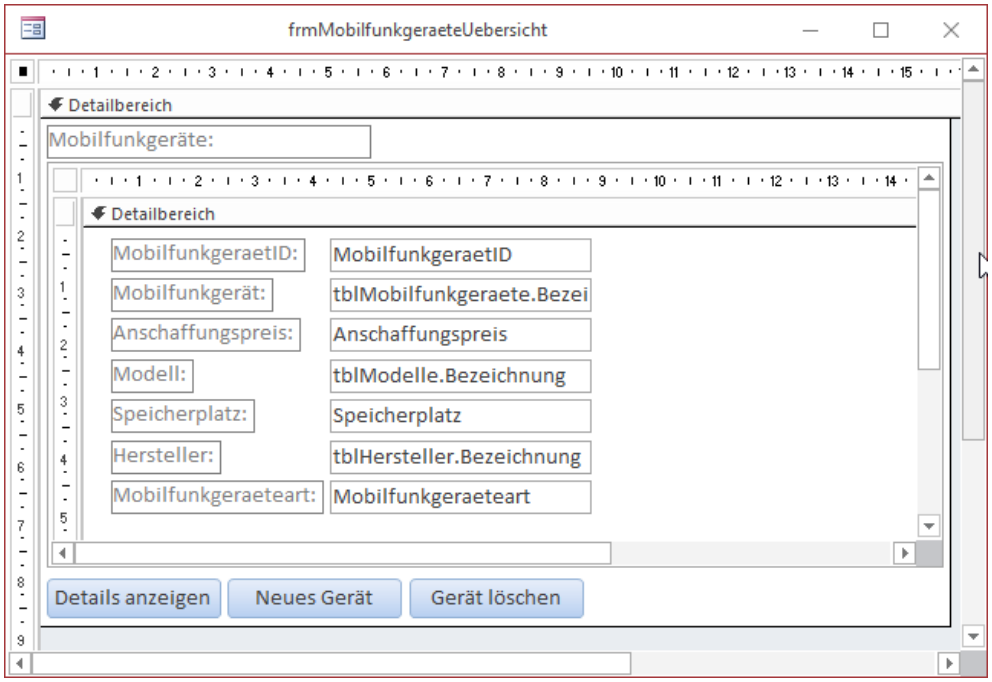


Bild 9: Entwurf des Hauptformulars **frmMobilfunkgeraeteUebersicht**

Die dritte Schaltfläche namens **cmdGeraetLoeschen** ruft die Prozedur aus Listing 3 auf. Diese ermittelt die Bezeichnung des zu löschenden Gerätes und schreibt sie in die Variable **strGeraet**. Damit stattet sie die Meldung einer **MsgBox**-Anweisung aus und fragt den Benutzer, ob er das aktuell markierte Gerät wirklich löschen will. Falls ja, ermittelt die Prozedur den Primärschlüsselwert des zu löschenden Geräts und führt eine entsprechende **DELETE**-Abfrage mit der **Execute**-Methode des aktuellen **Database**-Objekts aus. Anschließend aktualisiert sie die im Unterformular angezeigten Daten mit der **Requery**-Methode.

Steuerelemente aktivieren und deaktivieren

Die Prozedur SteuerelementeAktivieren, die in allen drei erwähnten Ereignisprozeduren aufgerufen wird, haben wir noch nicht beschrieben. Diese sorgt dafür, dass die Schaltflächen zum Anzeigen der Details

```
Private Sub cmdDetailsAnzeigen_Click()
    Dim lngMobilfunkgeraetID As Long
    lngMobilfunkgeraetID = Me!sfmMobilfunkgeraeteUebersicht.Form!MobilfunkgeraetID
    DoCmd.OpenForm "frmMobilfunkgeraetdetails", WindowMode:=acDialog, WhereCondition:="MobilfunkgeraetID = " & lngMobilfunkgeraetID
End Sub
```

Listing 1: Anzeigen der Details zu einem Mobilfunkgerät

```
Private Sub cmdNeuesGeraet_Click()
    DoCmd.OpenForm "frmMobilfunkgeraetdetails", WindowMode:=acDialog, DataMode:=acFormAdd
    Me!sfmMobilfunkgeraeteUebersicht.Form.Requery
    SteuerelementeAktivieren
End Sub
```

Listing 2: Anzeigen des Formulars **frmMobilfunkgeraetdetails** zum Anlegen eines neuen Geräts

Das Webbrowser-Steuerelement

Neben den vielen einfachen Steuerelement, die in Access verfügbar sind, findet sich auch eines, mit dem Sie die Inhalte von Internetseiten in Access-Formularen anzeigen können: das Webbrowser-Steuerelement. Dieses Steuerelement ist bis Access 2007 noch kein fester Bestandteil von Access gewesen und musste als ActiveX-Steuerelement nachgerüstet werden. Seit Access 2010 jedoch können Sie mit dem eingebauten Steuerelement arbeiten und können den einen oder anderen Vorteil nutzen – unter anderem die Möglichkeit, das Steuerelement an ein Feld als Datenquelle zu binden.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1801_WebbrowserSteuerelement.accdb**.

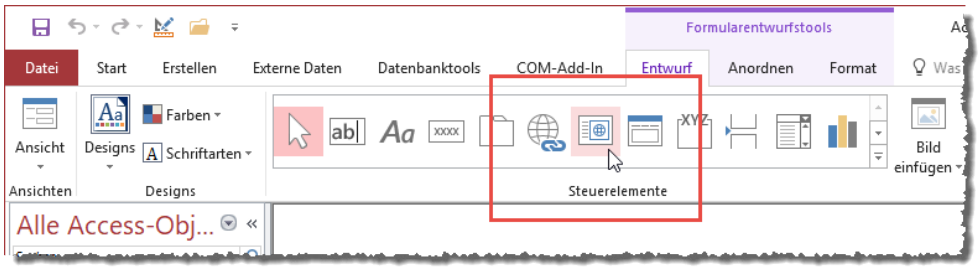


Bild 1: Auswahl des Webbrowser-Steuerelements über das Ribbon

Webbrowser-Steuerelement hinzufügen

Wenn Sie ein Formular mit dem Webbrowser-Steuerelement ausstatten wollen, wählen Sie in der Entwurfsansicht des Formulars im Ribbon den Eintrag **Entwurf|Steuerelemente|Webbrowsersteuerelement** aus (siehe Bild 1).

Klicken Sie auf diesen Eintrag, können Sie wie in Bild 2 einen Rahmen aufziehen, der die Größe des einzufügenden Webbrowser-Steuerelements markiert. Anschließend erscheint das Steuerelement als rechteckiger Rahmen mit dem Text **Ungebunden** als einzigem Inhalt.

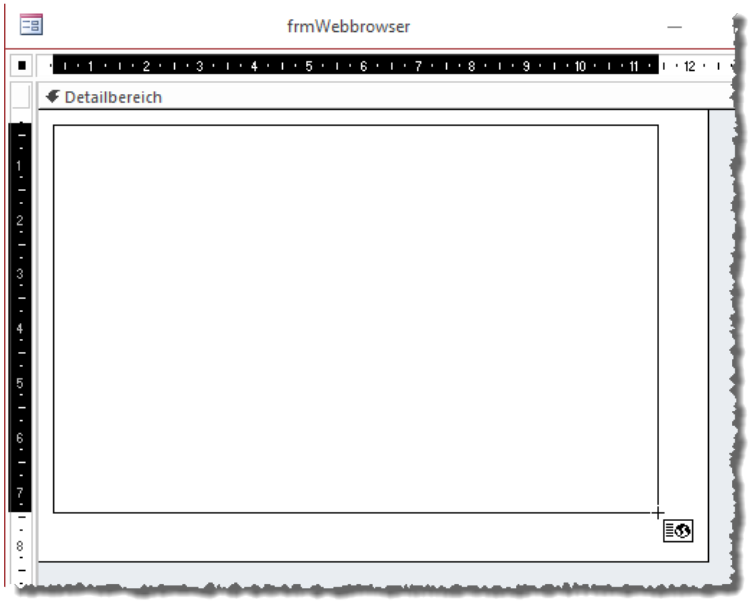


Bild 2: Einfügen des Webbrowser-Steuerelements

Eigenschaften des Webbrowser-Steuerelements

Schauen wir uns die Eigenschaften des neuen Steuerelements im Eigenschafts-Fenster an, finden wir keine besonderen neuen Eigenschaften, durch die sich das Steuerelement von anderen Steuerelementen wie etwa dem Textfeld abheben würde – allein die Seite mit den Ereigniseigenschaften liefert ein paar unbekannte Einträge (siehe Bild 3). Dazu gehören etwa die Eigenschaften **Bei Vor Navigieren**, **Bei Dokument vollständig**, **Bei Navigationsfehler** oder **Bei Fortschrittsänderung**. Auf der Registerseite **Daten** finden Sie die übliche Eigenschaft **Steuer-**

elementinhalt sowie die allein dem Webbrowser-Steuerelement vorbehaltene Eigenschaft **Transformation**. Auf der Seite **Format** fallen uns die beiden Eigenschaften **Bildlauf zum Anfang durchführen** und **Bildlauf nach links durchführen** auf – wir schauen uns gleich an, ob diese speziell für das Webbrowser-Steuerelement gedacht sind.

Eine Eigenschaft, die wir auf jeden Fall zu Beginn festlegen, heißt **Name** und nimmt den Wert **ctlWebbrowser** an.

VBA-Befehle des Webbrowser-Steuerelements

Fügen wir nun dem Formular ein Klassenmodul hinzu und legen die Ereignisprozedur **Beim Laden** für das Formular selbst an. Dann können wir über die Eingabe von **Me!ctlWebbrowser** und anschließendem Betätigen der Punkt-Taste per IntelliSense die vorhandenen Elemente für das Webbrowser-Steuerelement ansehen.

Noch einfacher gelingt das natürlich über den Objektkatalog, der wie in Bild 4 nach der Eingabe des

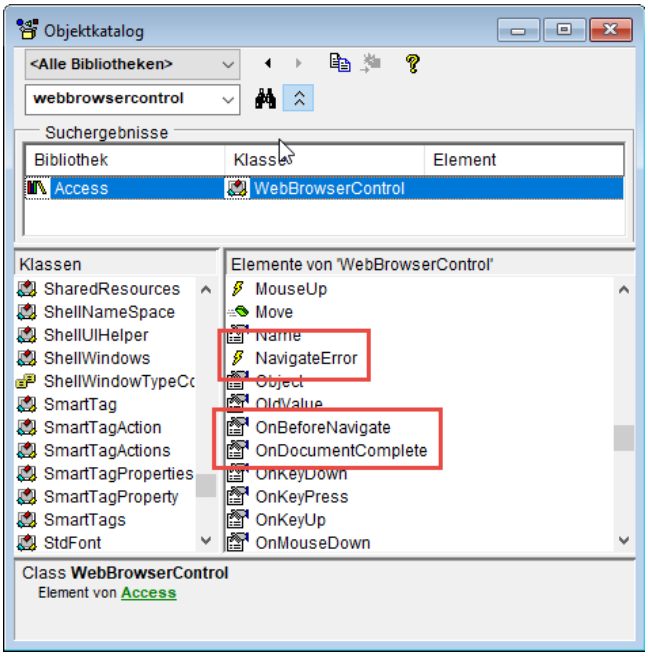


Bild 4: Objektkatalog mit den Elementen des Webbrowser-Steuerelements

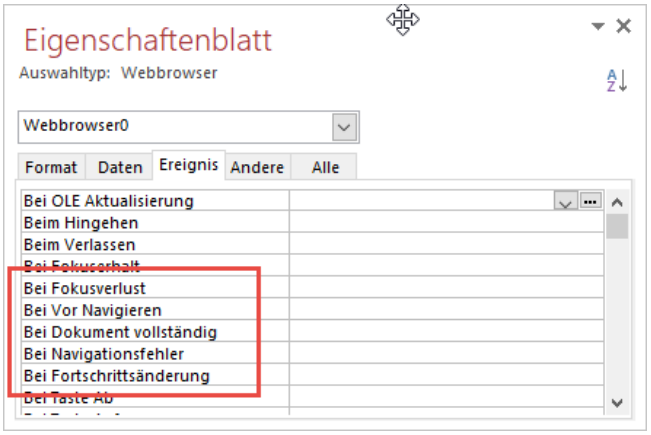


Bild 3: Ereigniseigenschaften des Steuerelements

Suchbegriffs **WebbrowserControl** die Eigenschaften, Methoden und Ereignisse dieses Steuerelements anzeigt. Hier finden wir die folgenden für das Webbrowser-Steuerelement spezifischen Elemente:

- **BeforeNavigate2**: Ereignis, das vor dem Laden der Seite ausgelöst wird. Dieses Ereignis und somit das Laden der Internetseite kann durch Setzen des Parameters **Cancel** auf den Wert **True** abgebrochen werden.
- **DocumentComplete**: Ereignis, das beim vollständigen Laden der Seite ausgelöst wird.
- **LocationURL**: Eigenschaft
- **NavigateError**: Ereignis, das beim Auftreten eines Fehlers ausgelöst wird.
- **OnBeforeNavigate**: Ereigniseigenschaft
- **OnDocumentComplete**: Ereigniseigenschaft
- **OnNavigateError**: Ereigniseigenschaft
- **OnProgressChange**: Ereigniseigenschaft
- **ProgressChange**: Ereignis, das beim Ändern des Fortschritts ausgelöst wird.
- **ReadyState**: Eigenschaft

```
Private Sub ctlWebbrowser_BeforeNavigate2(ByVal pDisp As Object, URL As Variant, flags As Variant, TargetFrameName As Variant, _
        PostData As Variant, Headers As Variant, Cancel As Boolean)
    Debug.Print "BeforeNavigate2"
End Sub
Private Sub ctlWebbrowser_DocumentComplete(ByVal pDisp As Object, URL As Variant)
    Debug.Print "DocumentComplete"
End Sub
Private Sub ctlWebbrowser_NavigateError(ByVal pDisp As Object, URL As Variant, TargetFrameName As Variant, _
        StatusCode As Variant, Cancel As Boolean)
    Debug.Print "NavigateError"
End Sub
Private Sub ctlWebbrowser_ProgressChange(ByVal Progress As Long, ByVal ProgressMax As Long)
    Debug.Print "ProgressChange"
End Sub
```

Listing 1: Ereignisse des Webbrowser-Steuerelements

Ereignisse erforschen

Um die Ereigniseigenschaften des Webbrowser-Steuerelements zu erforschen, legen wir diese einfach alle einmal an und fügen zunächst jeweils eine **Debug.Print**-Anweisung hinzu, welche den Namen des Ereignisses ausgibt (siehe Listing 1).

Erster Aufruf

Wenn wir nun von der Entwurfsansicht in die Formularansicht des Formulars wechselt, lädt das Webbrowser-Steuerelement natürlich noch keine Internetseite, da wir ihm ja nirgends mitgeteilt haben, welche Internetseite angezeigt werden soll (siehe Bild 5).

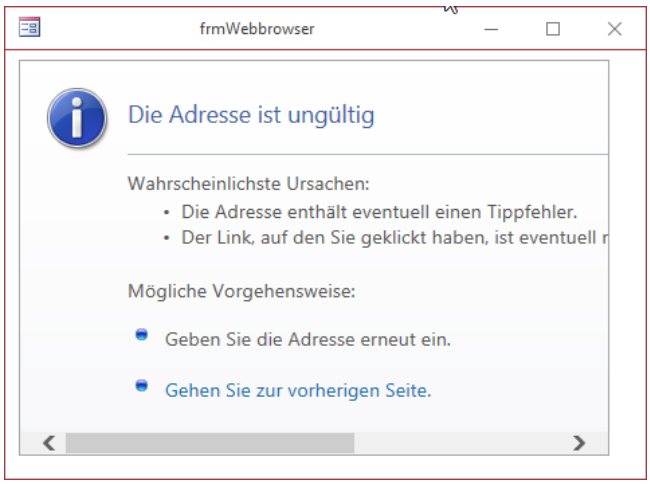


Bild 5: Beim ersten Öffnen zeigt das Webbrowser-Steuerelement logischerweise noch keine Seite an.

Wenn wir nun allerdings in den Direktbereich des VBA-Editors blicken, finden wir heraus, dass sich doch eine ganze Menge getan hat. Hier ist die Ausgabe im Direktbereich:

```
BeforeNavigate2
ProgressChange
ProgressChange
DocumentComplete
BeforeNavigate2
BeforeNavigate2
ProgressChange
ProgressChange
DocumentComplete
```

Mit diesen Ereignissen können Sie verschiedene Vorgänge steuern – zum Beispiel können Sie beim Auslösen des Ereignisses **DocumentComplete** bestimmte Steuerelemente aktivieren, die Sie für den Zugriff auf Elemente der Seite nutzen wollen.

Die Ereignisprozeduren liefern, wie Sie oben im Listing sehen, verschiedene Parameter. Der Parameter **URL** liefert jeweils die geladene Internetadresse. Das Ereignis **ProgressChange** liefert zwei Parameter.

Berichtsansicht

Neuere Access-Versionen (ab Access 2010) enthalten eine neue Ansicht für Berichte, nämlich die Berichtsansicht. Während die Seitenansicht die Ansicht liefert, die auch in gedruckter Form auf dem Papier oder in der PDF-Datei landen würde, liefert die Berichtsansicht eine Endlosansicht mit vielen zusätzlichen Funktionen. Dazu gehört etwa die Möglichkeit, Schaltflächen auf dem Bericht anzulegen und zu betätigen oder Daten dynamisch hinzuzufügen oder auf verschiedene Arten anzuzeigen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1801_Berichtsansicht.accdb**.

Was ist die Berichtsansicht?

Die Berichtsansicht bietet im Entwurf die gleichen Möglichkeiten wie die Seitenansicht – Sie können also die anzuzeigenden Felder nach Wunsch anordnen, Sortierungen und Gruppen definieren und so weiter. Zusätzlich können Sie aber auch noch Schaltflächen hinzufügen. Das können Sie auch in einem Bericht, den Sie in der Seitenansicht öffnen, allerdings lassen sich die Schaltflächen dort nicht betätigen.

Bevor Missverständnisse auftreten: Es handelt sich nicht um zwei verschiedene Berichtsarten – Sie können den gleichen Entwurf in der Seitenansicht und auch in der Berichtsansicht anzeigen. Die Frage ist allein, ob dies im Einzelfall aufgrund der Konstellation des jeweiligen Entwurfs sinnvoll ist. Nur, weil Sie Schaltflächen hinzufügen können, die in der Berichtsansicht auch betätigt werden können, wollen Sie solche Elemente ja noch längst nicht ausdrucken.

Öffnen eines Berichts in der Berichtansicht

Wenn Sie einen Bericht in der Berichtsansicht öffnen wollen, können Sie dies einfach durch einen Doppelklick auf den Eintrag des Berichts im Navigationsbereich erledigen. Wenn der Bericht bereits in der Entwurfsansicht geöffnet ist, finden Sie Kontextmenü des Berichts den Eintrag **Berichtsansicht** (siehe Bild 1).

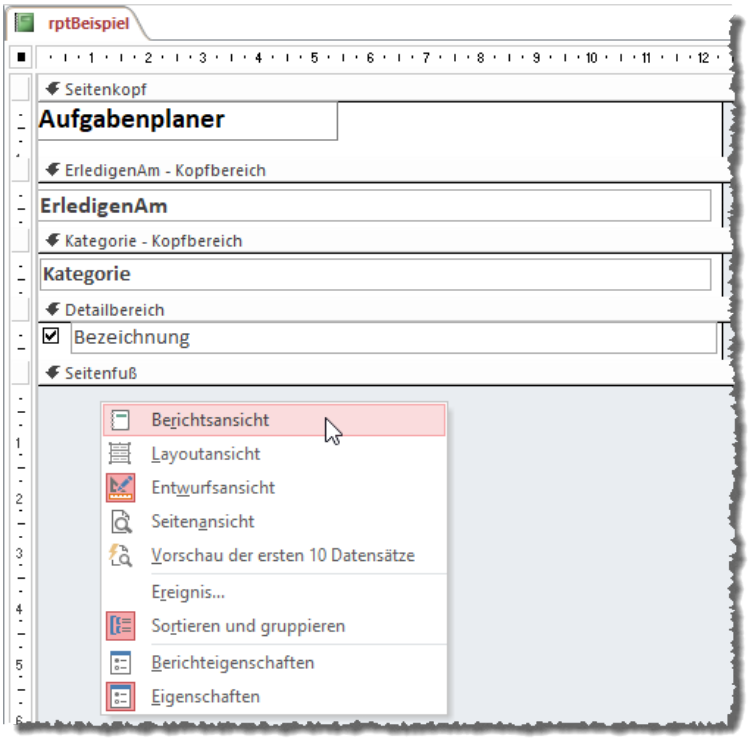


Bild 1: Anzeigen der Berichtsansicht vom Entwurf aus

Schließlich können Sie einen Bericht auch noch per VBA in der Berichtsansicht öffnen. Dazu setzen Sie die folgende Anweisung ab, beispielsweise im Direktbereich des VBA-Editors (**Strg + G**):

```
DoCmd.OpenReport "rptBeispiel", acViewReport
```

Schließlich können Sie Berichte auch noch innerhalb eines Unterformular-Steuerelements anzeigen, was

wir uns im Detail im Artikel **Berichte im Unterformular** ansehen.

Beispieltabellen

In unserem Beispiel zu diesem Artikel wollen wir uns die Anordnung von Aufgaben nach Datum und Kategorie sortiert ansehen. Im Artikel **Aufgabenplaner** bauen wir die Beispieldatenbank des vorliegenden Artikels noch zu einer Lösung aus.

Für die Beispiele erstellen wir zwei Tabellen. Die erste enthält schlicht die Kategorien und heißt dementsprechend **tblKategorien**. Sie verwendet die beiden Felder **KategorieID** und **Bezeichnung**.

Die zweite Tabelle **tblAufgaben** speichert die eigentlichen Aufgaben. Sie enthält die folgenden Felder:

- **AufgabeID**: Primärschlüsselfeld der Tabelle
- **Bezeichnung**: Bezeichnung der Aufgabe
- **Erledigt**: **Ja/Nein**-Feld, das angibt, ob die Aufgabe erledigt ist
- **ErledigenAm**: Datumsfeld, welches das planmäßige Erledigungsdatum enthält
- **KategorieID**: Fremdschlüsselfeld zur Auswahl eines Eintrags der Tabelle **tblKategorien**

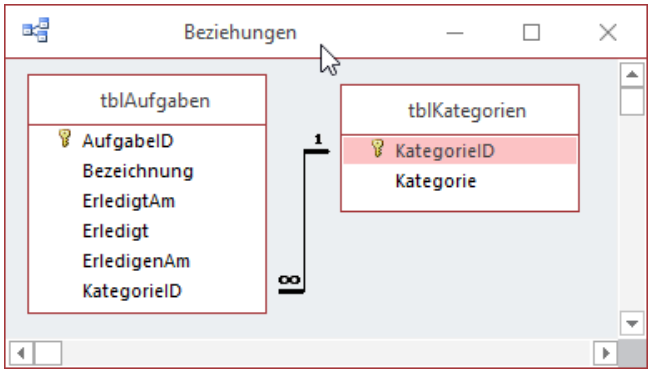


Bild 2: Datenmodell der Beispieldatenbank

Die beiden Tabellen sind wie in Bild 2 miteinander verknüpft.

Datenherkunft

Nachdem Sie einen neuen, leeren Bericht in der Entwurfsansicht geöffnet haben, fügen wir diesem über die Eigenschaft **Datensatzquelle** eine Datenherkunft hinzu, in diesem Fall eine neue Abfrage. Diese sieht wie in Bild 3 aus und führt die beiden Tabellen **tblAufgaben** und **tblKategorien** zusammen. Sie enthält alle Felder der Tabelle **tblAufgaben** sowie zusätzlich das Feld **Kategorie** der Tabelle **tblKategorien**.

Gruppierungen erstellen

Für den Bericht selbst legen wir nun zuerst eine Gruppierung über das Feld **ErledigenAm** an, indem wir im unteren Bereich namens **Gruppieren, Sortieren und Summe** auf **Gruppe hinzufügen** klicken und dann die Konfiguration wie in Bild 4 einstellen. Wichtig ist hier, dass Sie das Feld **ErledigenAm** für **Gruppieren nach** festlegen, den Eintrag **mit Kopfzeilenbereich** wählen und – ganz wichtig – den auf **nach Quartal** voreingestellten Filter auf **nach Tag** einstellen.

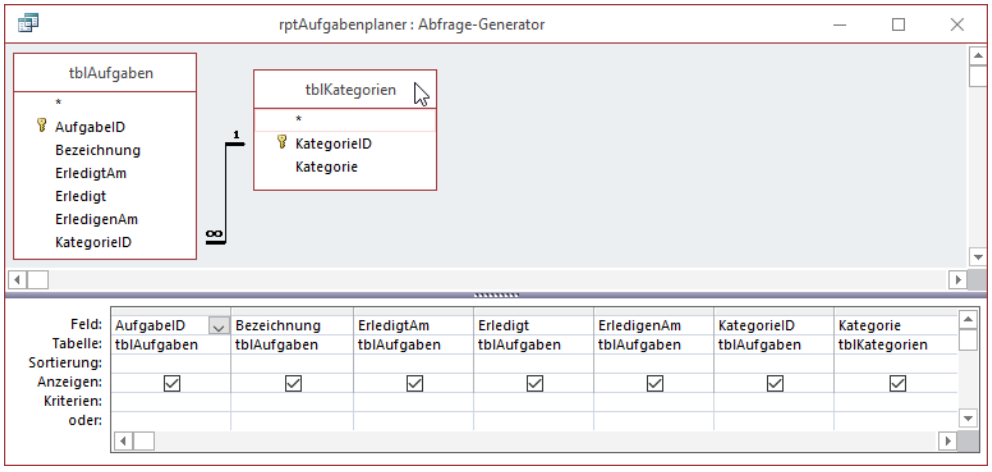


Bild 3: Datenherkunft des Beispielberichts

Danach fügen Sie eine weitere Gruppierung hinzu, diesmal nach dem

Feld **KategorieID**. Auch hier benötigen wir einen Kopfzeilenbereich. Die übrigen Einstellungen können Sie beibehalten.

Nun sollten im Bericht bereits die Kopfbereiche für die Felder **ErledigenAm** und **KategorieID** sichtbar sein. Fügen Sie dem Bereich **Seitenkopf** ein Bezeichnungsfeld mit der Überschrift **Aufgabenplaner** hinzu.

Dem Kopfbereich für das Feld **ErledigenAm** fügen Sie das Feld **ErledigenAm** hinzu, entfernen das Bezeichnungsfeld, das automatisch mit angelegt wird und schieben das Textfeld ganz nach links.

Dem Bereich **Kategorie - Kopfbereich** fügen Sie das Feld **Kategorie** hinzu. Auch hier entfernen Sie das Bezeichnungsfeld und verschieben das verbleibende Textfeld ganz nach links.

Fehlt noch der Detailbereich, welcher die eigentlichen Daten aufnimmt: Das Kontrollkästchen für das **Ja/Nein**-Feld **Erledigt** sowie das Feld **Bezeichnung** mit der Bezeichnung der Aufgabe. Auch hier entsorgen wir die Bezeichnungsfelder und arrangieren die Steuerelemente neu. Wenn wir nun in die Berichtsansicht wechseln, sieht der Bericht mit einigen Beispieldaten wie in Bild 5 aus.

Das Ergebnis ist schon recht gut, wenn wir auch für eine saubere Darstellung noch die Rahmen der Textfelder ausblenden und die Hintergrundfarbe einheitlich weiß gestalten sollten. Dies erledigen wir wieder in der Entwurfsansicht, wo wir die Eigenschaft **Rahmenart** gleich aller Steuerelemente auf **Transparent** einstellen.

Außerdem stellen wir die Eigenschaft **Alternative Hintergrundfarbe** für die drei Bereiche **ErledigenAm - Kopfbereich**, **KategorieID - Kopfbereich** und **Detailbereich** auf den gleichen Wert ein wie die Eigenschaft **Hintergrundfarbe** (hier **Hintergrund 1**).

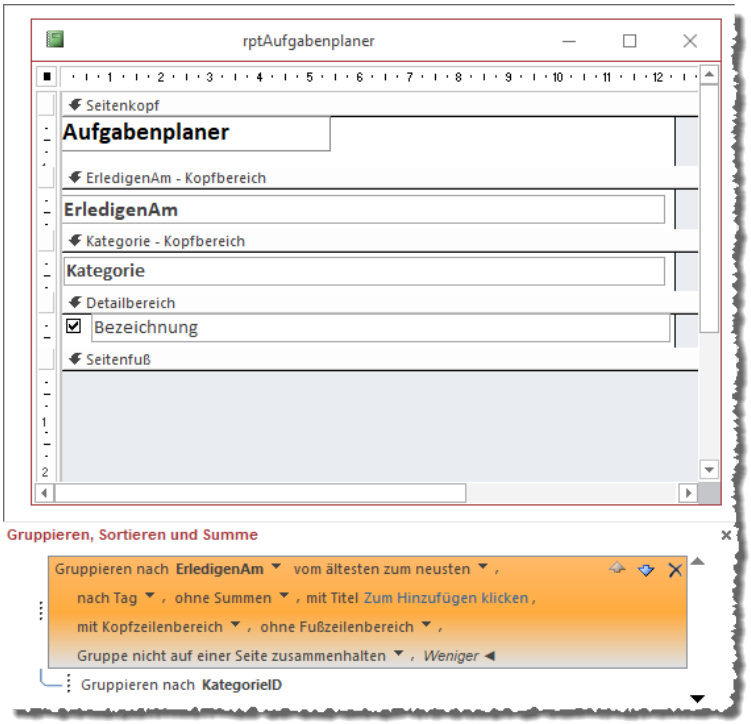


Bild 4: Entwurf des Berichts **rptAufgabenplaner**

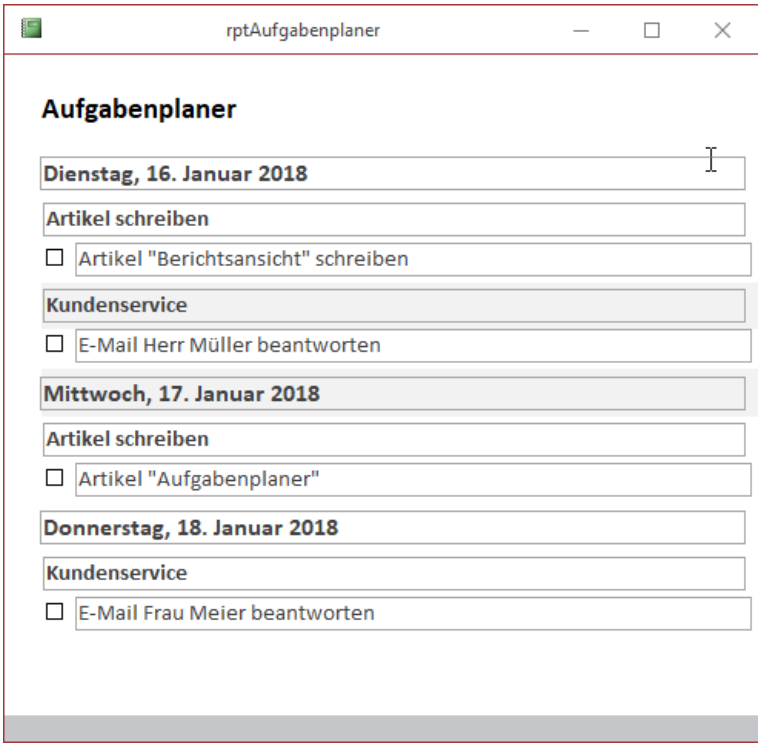


Bild 5: Die Berichtsansicht des ersten Entwurfs

Berichtsansicht im Unterformular

In den Artikeln **Berichtsansicht** und **Aufgabenplaner** haben wir uns mit der Berichtsansicht beschäftigt. Diese wird umso interessanter, da man sie über ein Unterformular-Steuerelement in einem Formular integrieren kann. Dies schauen wir uns in diesem Artikel genauer an. Wir bauen dabei die Lösung aus dem Artikel **Aufgabenplaner** weiter aus, indem wir den dort vorgestellten Bericht als Unterformular in ein Formular einbetten, mit dem wir weitere Funktionen zur Lösung hinzufügen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1801_BerichtsansichtImUnterformular.accdb**.

Berichte im Unterformular

In Access-Versionen älter als Access 2010 war es nicht möglich, Berichte im Unterformular anzuzeigen. Dies gelingt nun jedoch recht einfach – und zwar genau wie das Hinzufügen eines Unterformulars zu einem Formular.

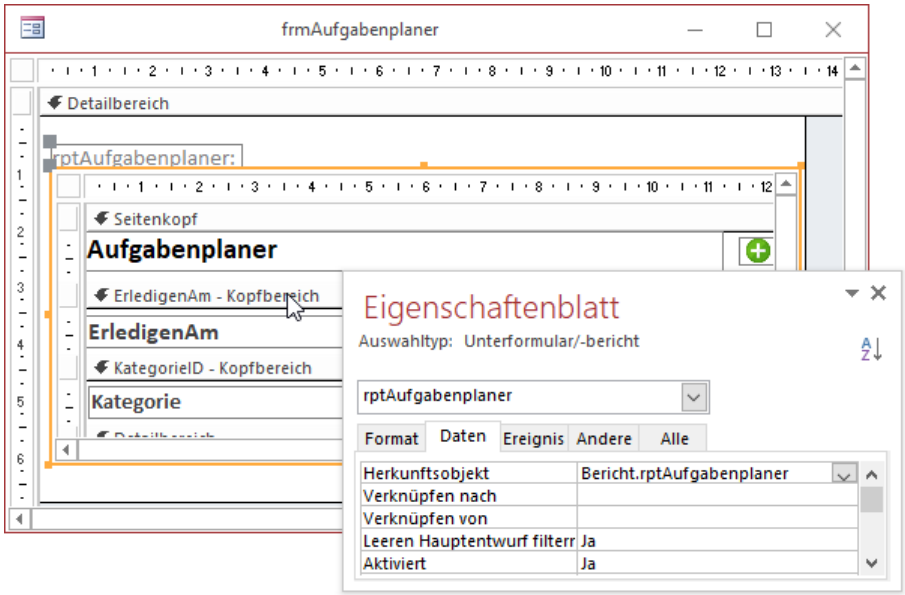


Bild 1: Hinzufügen eines Berichts zu einem Formular als Unterformular

In diesem Fall wollen wir direkt auf der Lösung aus dem Artikel Aufgabenplaner aufsetzen, in dem wir eine kleine Lösung basierend auf einem Bericht als Benutzeroberfläche gebaut haben. Diesen Bericht wollen wir nun in einem Unterformular zu einem Formular namens frmAufgabenplaner hinzufügen.

Dieses Formular bereiten wir vorher kurz vor: Da es nicht an eine Datenquelle gebunden sein soll und somit auch keine Daten anzeigt, stellen wir die Eigenschaften **Bildlaufleisten**, **Navigationsschaltflächen**, **Datensatzmarkierer** und **Trennlinien** auf **Nein** und die Eigenschaft **Automatisch zentrieren** auf **Ja** ein. Danach ziehen wir einfach den Bericht **rptAufgabenplaner** aus dem Navigationsbereich in den Entwurf des Formulars hinein.

Das Ergebnis sieht etwa wie in Bild 1 aus. Was ist hier geschehen? Access hat beim Fallenlassen des Berichts **rptAufgabenplaner** automatisch ein Unterformular-Steuerelement zum Formular hinzufügt und für die Eigenschaft **Herkunftsobjekt** dieses Steuerelements den Wert **Bericht.rptAufgabenplaner** eingestellt. Dadurch zeigt das Unterformular nun den Entwurf des Berichts **rptAufgabenplaner** an.

Kleine Korrekturen

Wie Sie sehen, zeigt dieses oben im Seitenkopfbereich die Überschrift **Aufgabenplaner** an, die wir in diesem Kontext nicht benötigen. Außerdem bietet der Seitenkopfbereich noch die Schaltfläche **cmdNeu** zum Hinzufügen neuer Aufgaben zur Tabelle

tblAufgaben. Wir wollen den Seitenkopf-Bereich nun komplett aus dem Bericht entfernen. Die Überschrift können wir ersatzlos löschen, die Schaltfläche fügen wir dem Hauptformular hinzu. Außerdem wollen wir, dass sich die Höhe des Unterberichts automatisch an die Höhe des Formulars anpasst, wozu wir die Eigenschaft **Vertikaler Anker** des Unterformular-Steuerelements auf den Wert **Beide** einstellen. Eine Anpassung des horizontalen Ankers ist in diesem Fall nicht nötig, da die Steuerelemente im Bericht selbst nicht angepasst werden können – es gibt im Bericht schlicht keine passenden Eigenschaften. In der Entwurfsansicht sieht das Formular nun wie in Bild 2 aus.

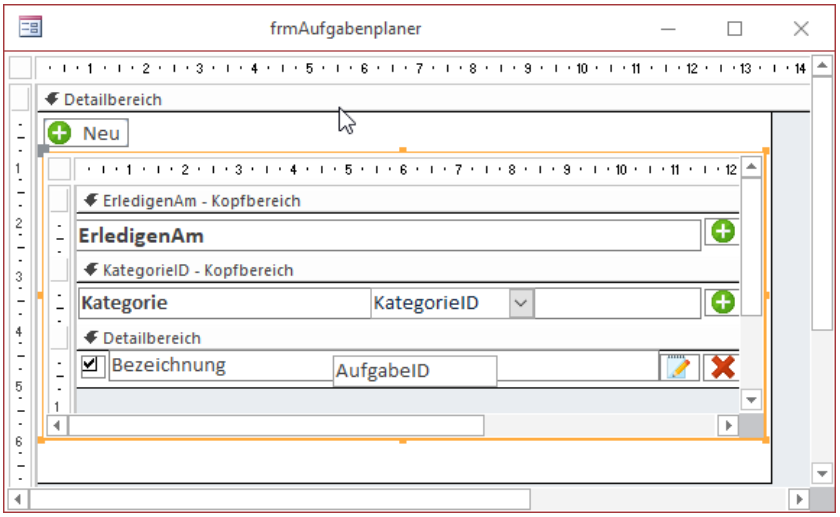


Bild 2: Ausgerichteter Unterbericht

Den Code der Schaltfläche **cmdNeu** müssen wir etwas anpassen, da wir nach dem Anlegen der neuen Aufgabe ja nicht mehr das Objekt aktualisieren wollen, in dem die Prozedur ausgeführt wird, sondern den im Unterformular-Steuerelement befindlichen Bericht. Dies sieht dann wie folgt aus:

```
Private Sub cmdNeu_Click()
    DoCmd.OpenForm "frmAufgabe", WindowMode:=acDialog, _
                                 DataMode:=acFormAdd
    Me!rptAufgabenplaner.Report.Requery
End Sub
```

Die **Requery**-Methode wird also für das **Report**-Objekt im Unterformular-Steuerelement **rptAufgabenplaner** ausgelöst. Das das Unterformular-Steuerelement mit **rptAufgabenplaner** benannt wurde, liegt schlicht daran, dass die Unterformular-Steuerelemente für Objekte, die aus dem Navigationsbereich in ein Formular gezogen werden, automatisch den Namen des gezogenen Objekts erhalten.

Interessanterweise gibt es aber für im Unterformular-Steuerelement befindliche Berichte eine eigene Eigenschaft zum Referenzieren, nämlich **Report**.

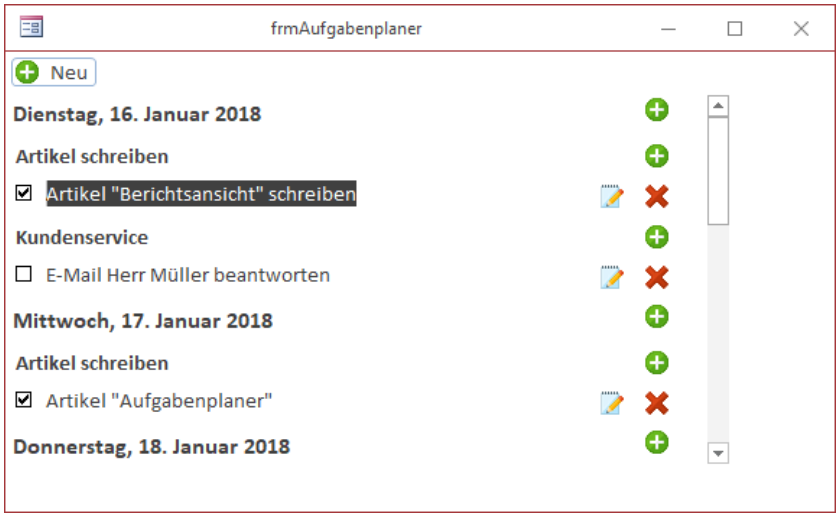


Bild 3: Der neu gestaltete Bericht im Hauptformular mit **Neu**-Schaltfläche

Aus optischen Gründen haben wir auch noch die Eigenschaft Rahmenart des Unterformular-Steuerelements auf Transparent eingestellt, sodass das Formular in der Formularansicht wie in Bild 3 aussieht.

Benutzerdefinierter Filter

Warum nun den Bericht in der Berichtsansicht im Hauptformular unterbringen? Ganz einfach: Weil wir benutzerdefinierte Filter festlegen

wollen, mit den wir beispielsweise nach der Bezeichnung der Aufgabe, der Katgorie oder dem Datum filtern können. Das gelingt zwar auch schon über die Benutzeroberfläche mit Bordmitteln, aber diese sind nicht direkt erreichbar. Wenn wir wissen, wonach der Benutzer filtern will, können wir die notwendigen Steuerelemente auch direkt sichtbar im Formular anlegen.

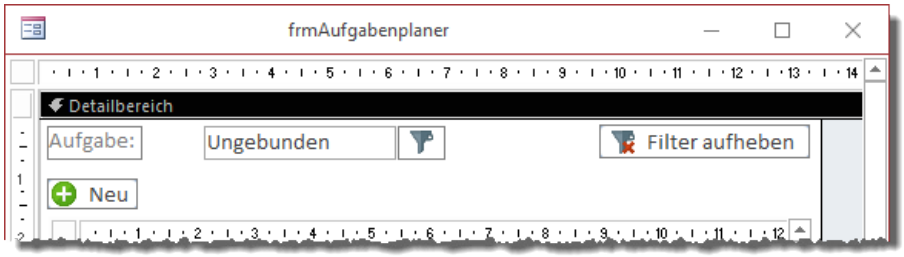


Bild 4: Steuerelemente zum Filtern und zum Aufheben des Filters

Beginnen wir mit einem Filter, der die im Bericht im Unterformular befindlichen Aufgaben nach der Bezeichnung der Aufgabe filtert. Dazu fügen wir dem Hauptformular ein Textfeld namens **txtFilter** hinzu sowie eine Schaltfläche namens **cmdFiltern**, mit der wir den Filter anwenden können. Außerdem fügen wir noch eine Schaltfläche hinzu, mir der wir den angewendeten Filter wieder aufheben können. Der Entwurf des Formulars sieht dann wie in Bild 4 aus.

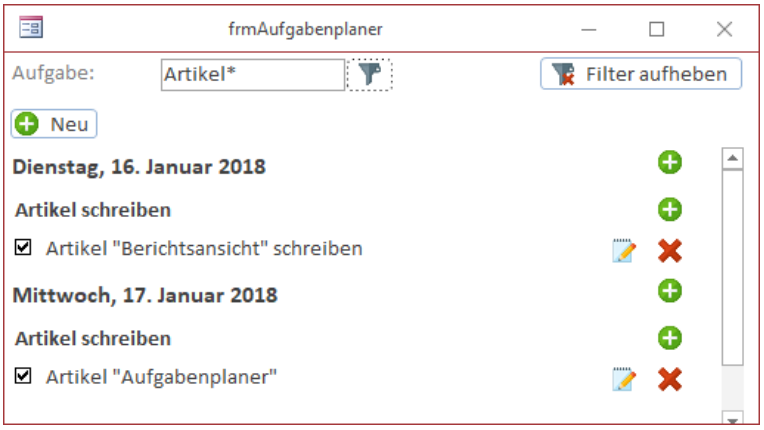


Bild 5: Filtern nach der Bezeichnung einer Aufgabe

Um den Filter anzuwenden, fügen die der Schaltfläche **cmdFiltern** die folgende Prozedur hinzu:

```
Private Sub cmdFiltern_Click()
    Dim strFilter As String
    strFilter = _
            "Bezeichnung LIKE '" & Me!txtFilter & "'"
    With Me!rptAufgabenplaner.Report
        .Filter = strFilter
        .FilterOn = True
    End With
End Sub
```

Diese ermittelt den Inhalt des Textfeldes **txtFilter** und stellt eine Bedingung zusammen, die beispielsweise **Bezeichnung LIKE 'Artikel*'** lautet. Dieser wird dann an die Eigenschaft **Filter** des Berichts im Unterformular übergeben. Außerdem stellen wir die Eigenschaft **FilterOn** des gleichen Elements auf **True** ein. Damit werden nun nur noch die der Bedingung entsprechenden Elemente im Bericht im Unterformular angezeigt (siehe Bild 5).

Praktischerweise bleiben nun auch die Datumsangaben und die Kategorien erhalten.

Um den Filtern wieder aufzuheben, fügen wir eine weitere Schaltfläche namens **cmdKeinFilter** hinzu. Diese löst die folgende Prozedur aus:

```
Private Sub cmdKeinFilter_Click()
    Me!txtFilter = ""
    With Me!rptAufgabenplaner.Report
        .Filter = ""
        .Requery
    End With
End Sub
```

Wenn Sie bereits einmal unter VBA mit Filtern gearbeitet haben, sollte das kein Problem sein. Allerdings gibt es einen kleinen Haken: Allein das Einstellen der Eigenschaft **Filter** auf eine leere Zeichenfolge oder

Aufgabenplaner

Im Artikel **Berichtsansicht** haben Sie eine neue Ansicht für Berichte kennen gelernt, die ganz neue Möglichkeiten eröffnet. Sie können damit hierarchische Daten anzeigen, ohne das TreeView-Steuerelement nutzen zu müssen und in einer Kombination aus Bericht und Formular arbeiten. Das wollen wir gleich einmal in einer Lösung ausprobieren, mit der Sie Ihre täglichen Aufgaben verwalten können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1801_Aufgabenplaner.accdb**.

Vorarbeiten

Die Vorarbeiten zu diesem Artikel haben im Artikel Berichtsansicht stattgefunden. Dort haben wir bereits ein kleines Datenmodell mit den beiden Tabellen tblAufgaben und tblKategorien erstellt und ein Grundgerüst eines Berichts, der im Entwurf wie in Bild 1 aussieht.

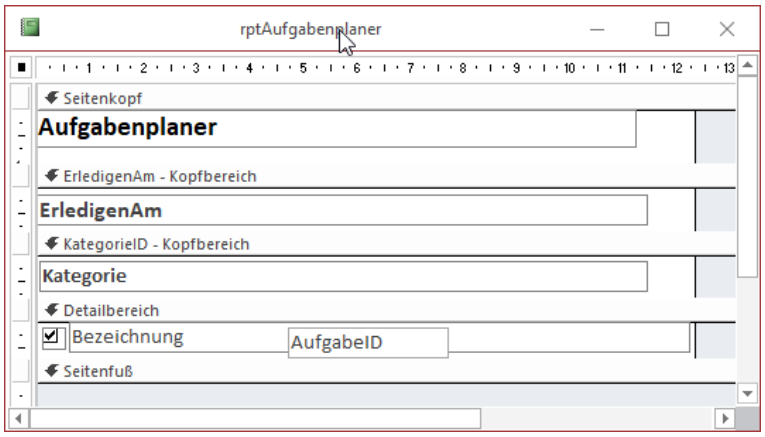


Bild 1: Ausgangspunkt dieser Lösung – der Bericht **rptAufgabenplaner**

Kontrollkästchen nutzen

Wenn Sie nun auf das Kontrollkästchen klicken, erfolgt nicht die erwartete Änderung des Zustands des Kontrollkästchens. Im Gegenteil – ein Warnsignal macht deutlich, dass die gewünschte Funktion nicht verfügbar ist. Also schauen wir uns an, ob wir das Kontrollkästchen auf andere Weise für unsere Zwecke einsetzen können.

Dazu hinterlegen wir für die Ereigniseigenschaft **Beim Klicken** des Kontrollkästchens eine Ereignisprozedur. Vorher stellen wir die Eigenschaft **Name** dieses Steuerelements jedoch noch von **Erledigt** auf **chkErledigt** um. Wählen Sie dann den Wert **[Ereignisprozedur]** für die Eigenschaft **Beim Klicken** aus und klicken Sie auf die Schaltfläche mit den drei Punkten.

Wir gehen nun einmal naiv an die Aufgabe heran und hinterlegen die folgende Prozedur für das Ereignis:

```
Private Sub chkErledigt_Click()
    If Me!Erledigt = False Then
        Me!Erledigt = True
    Else
        Me!Erledigt = False
    End If
End Sub
```

Hier wollen wir schlicht prüfen, welchen Wert das Feld **Erledigt** aktuell hat und den Wert in Abhängigkeit davon auf **True** oder **False** ändern. Wenn wir nun in die Berichtsansicht wechseln und auf das Kontrollkästchen eines der Einträge klicken, erhalten wir die Fehlermeldung aus Bild 2.

Offensichtlich weisen die gefundenen Felder eines Berichts auch in der Berichtsansicht keine Möglichkeit auf, Werte zuzuweisen. Doch damit geben wir uns noch nicht geschlagen. Es gibt ja noch die Möglichkeit, die Daten in der zugrunde liegenden Tabelle zu ändern – und wer weiß, vielleicht können wir den Bericht ja nach dieser Aktion aktualisieren, sodass er doch noch den geänderten Wert im Kontrollkästchen anzeigt?

Tabellenwert ändern und Bericht aktualisieren

Die Prozedur **chkErledigt_Click** aktualisieren wir nun wie in Listing 1.

Die neue Version der Prozedur ermittelt zunächst den Primärschlüsselwert des aktuellen Datensatzes, also den Wert des Feldes **AufgabeID**.

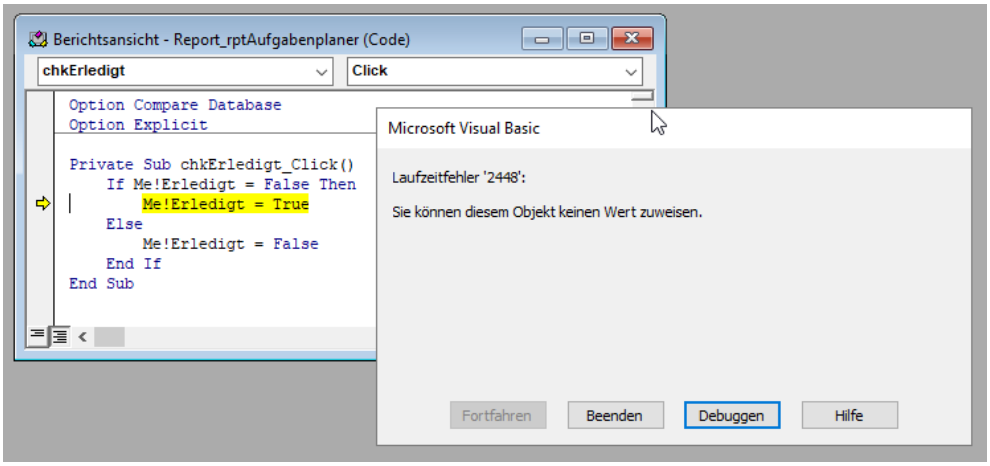


Bild 2: Fehler beim Zuweisen eines Wertes an ein Feld im Bericht

Dazu muss dieses Feld zunächst einmal zum Entwurf des Berichts hinzugefügt werden, am Besten in den Detailbereich. Da wir die Anzeige dieses Wertes in einem Textfeld für die Benutzung des Berichts nicht benötigen, stellen wir die Eigenschaft **Sichtbar** des Textfeldes auf **Nein** ein (siehe Bild 3).

Danach können wir das Feld nutzen, um den Wert des Primärschlüssels des aktuellen Datensatzes zu ermitteln und in der Variablen **lngAufgabeID** zu speichern. Auch den Wert des Feldes **Erledigt** fragen wir hier ab. Hat dieses den Wert **False**, tragen wir den Wert **True** in die Variable **bolErledigt** ein und umgekehrt. Dieser Variable soll den einzustellenden Wert für das Feld **Erledigt** aufnehmen.

Danach fügen wir der Variablen **db** einen Verweis auf das aktuelle **Database**-Objekt hinzu und stellen in der Variablen **strSQL** die auszuführende SQL-Abfrage zusammen. Wenn wir den Wert des Feldes **Erledigt** für den ersten Datensatz mit dem Primärschlüsselwert **1** auf **True** einstellen wollen, sieht der Inhalt von strSQL etwa so aus:

```
Private Sub chkErledigt_Click()
    Dim db As DAO.Database
    Dim lngAufgabeID As Long
    Dim bolErledigt As Boolean
    Dim strSQL As String
    lngAufgabeID = Me!AufgabeID
    If Me!Erledigt = False Then
        bolErledigt = True
    Else
        bolErledigt = False
    End If
    Set db = CurrentDb
    strSQL = "UPDATE tblAufgaben SET Erledigt = " & CInt(bolErledigt) & " WHERE AufgabeID = " & lngAufgabeID
    db.Execute strSQL, dbFailOnError
    Me.Requery
    Me!chkErledigt.SetFocus
End Sub
```

Listing 1: Prozedur zum Einstellen des Wertes des Feldes **Erledigt**

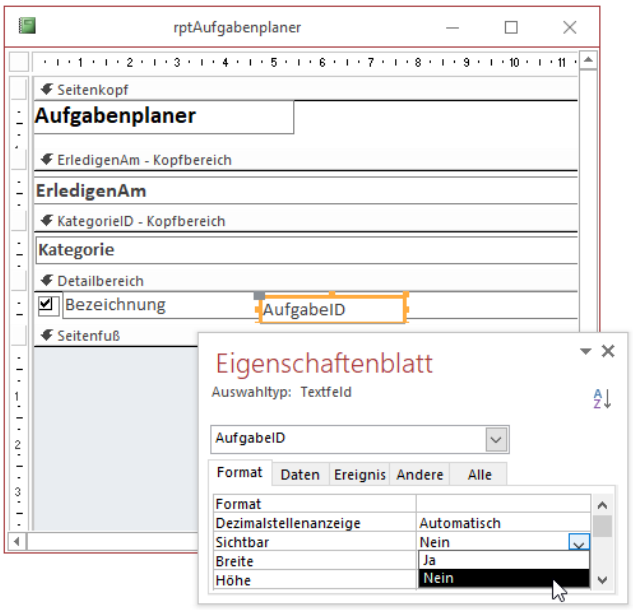


Bild 3: Hinzufügen des Feldes **AufgabeID** als unsichtbares Textfeld

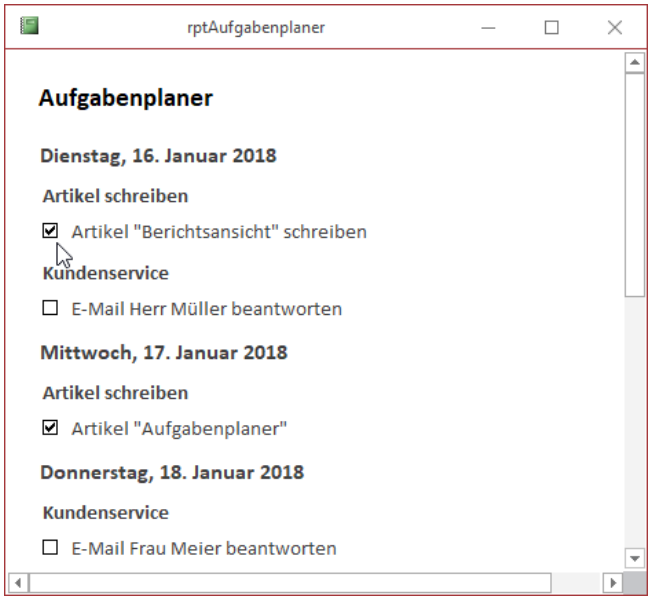


Bild 4: Nun werden die Kontrollkästchen korrekt aktualisiert.

```
UPDATE tblAufgaben SET Erledigt = -1 WHERE AufgabeID = 1
```

Diese Abfrage führen wir dann mit der **Execute**-Methode des **Database**-Objekts aus. Anschließend aktualisieren wir den Inhalt des Berichts in der Berichtsansicht mit der **Requery**-Methode, was tatsächlich zum gewünschten Ergebnis führt (siehe Bild 4).

Die Anweisung zum Einstellen des Fokus auf das Kontrollkästchen verhindert, dass das Textfeld mit der Bezeichnung der Aufgabe im Anschluss den Fokus erhält.

Aufgabenbezeichnung bearbeiten

Wie Sie sich vorstellen können, ist auch das Bearbeiten des Inhalts eines Textfeldes in der Berichtsansicht nicht ohne Weiteres möglich. Aber auch hier gibt es Workarounds.

Der einfachste ist, den Text einer Bezeichnung per Doppelklick in einer **InputBox** anzuzeigen und mit dieser dann den neuen Wert für die Bezeichnung abzufragen. Dazu hinterlegen wir für das Ereignis **Beim Doppelklicken** des Textfeldes mit dem Feld **Bezeichnung**, dessen Eigenschaft **Name** wir zuvor auf **txtBezeichnung** ändern, die Ereignisprozedur aus Listing 2. Diese arbeitet weitgehend wie die aus dem ersten Beispiel zum Ändern des Wertes des Feldes **Erledigt**.

Neue Aufgabe anlegen

Wenn Sie eine neue Aufgabe anlegen wollen, wird es schon interessanter. Hier gibt es verschiedene Möglichkeiten:

- Für das geplante Datum der Aufgabe ist noch kein Eintrag im Bericht vorhanden. Dann muss eine komplett neue Aufgabe angelegt werden – inklusive Bezeichnung, geplantem Erledigungsdatum und Kategorie.

- Das Datum der Aufgabe ist bereits im Bericht vorhanden, aber es liegt für dieses Datum noch keine Aufgabe in der gewünschten Kategorie vor. Dann können wir in einem Formular zum Anlegen der neuen Aufgabe zumindest schon einmal das Datum vorbelegen.

- Es ist auch bereits eine Aufgabe unterhalb des gewünschten Datums vorhanden, welche der betreffenden Kategorie angehört. Dann braucht man nur die Bezeichnung der Aufgabe einzugeben.

```
Private Sub txtBezeichnung_DblClick(Cancel As Integer)
    Dim db As DAO.Database
    Dim strBezeichnung As String
    Dim strSQL As String
    Dim lngAufgabeID As Long
    strBezeichnung = Me!txtBezeichnung
    lngAufgabeID = Me!AufgabeID
    strBezeichnung = InputBox("Neue Bezeichnung eingeben:", "Bezeichnung ändern", strBezeichnung)
    If Len(strBezeichnung) > 0 Then
        Set db = CurrentDb
        strSQL = "UPDATE tblAufgaben SET Bezeichnung = '" & strBezeichnung & "' WHERE AufgabeID = " & lngAufgabeID
        db.Execute strSQL, dbFailOnError
        Me.Requery
    End If
End Sub
```

Listing 2: Prozedur zum Einstellen des Wertes des Feldes **Bezeichnung**

Letzteres könnten wir mit der **InputBox** aus dem vorherigen Beispiel erledigen. Die ersten beiden Schritte erfordern jedoch den Einsatz eines Formulars, mit dem die drei Eigenschaften festgelegt beziehungsweise mit Standardwerten belegt werden können.

Formular zum Anlegen neuer Aufgaben

Das Formular zum Anlegen neuer Aufgaben soll **frmAufgabe** heißen und an die Tabelle **tblAufgaben** gebunden sein.

Die Steuerelemente sind schnell hinzugefügt, sodass der Entwurf – inklusive zweier Schaltflächen namens **cmdOK** und **cmdAbbrechen** – wie in Bild 5 aussieht.

Da wir mit dem Formular immer nur einen Datensatz bearbeiten möchten, stellen wir die Eigenschaften **Datensatzmarkierer**, **Navigationsschaltflächen**, **Trennlinien** und **Bildlaufleisten** jeweils auf den Wert **Nein** ein. Damit das Formular beim Öffnen zentriert angezeigt wird, erhält die Eigenschaft **Automatisch zentrieren** den Wert **Ja**.

Interessanter wird allerdings das Öffnen vom Bericht aus sowie das Voreinstellen der Standardwerte für das Feld **Datum** und gegebenenfalls zusätzlich für das Feld **KategorieID**.

Einen einzigen Wert könnten wir beim Aufruf mit der **DoCmd.OpenForm**-Methode leicht mit dem Parameter **OpenArgs** übergeben. Wenn je nach Anforderung jedoch ein zweiter Parameter übergeben werden soll, wird es spannend.

Aufruf des Formulars

Schauen wir uns jedoch erst einmal an, wie wir das Formular überhaupt vom Bericht **rptAufgabenplaner** aus aufrufen. Dazu fügen wir dem Bericht drei Schaltflächen hinzu, die wir in den Bereichen **Seitenkopf**,

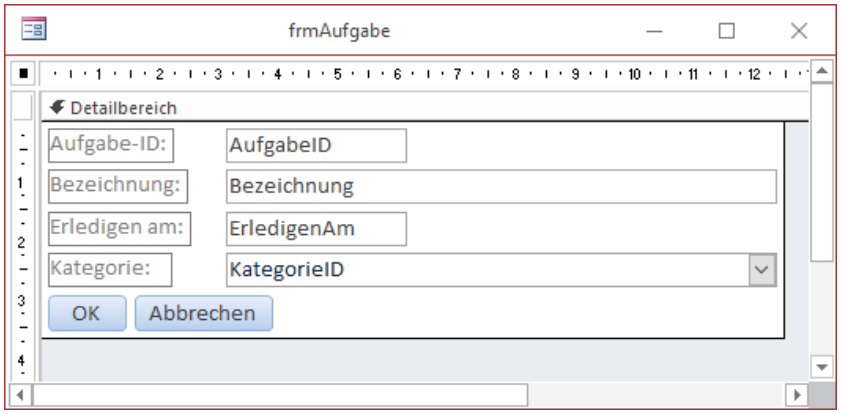


Bild 5: Entwurf des Formulars zur Eingabe einer Aufgabe

Handyverwaltung: Personen, Verträge, Handys

Bevor wir die Anwendung zur Verwaltung von Handy, Verträgen, Personen und mehr mit einem Ribbon ausstatten können, müssen wir noch die Details einer Person um ein paar Dinge ergänzen. So zeigt unser Detailformular bisher nur die Personendaten an, aber nicht, welche Verträge und welche Handys beziehungsweise Mobilfunkgeräte dieser Person zugeordnet sind. Dies holen wir im vorliegenden Artikel nach – und ergänzen auch noch die Formulare zur Anzeige der Mobilfunkgeräte und Verträge um die Zuordnung zur jeweiligen Person.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1801_Handyverwaltung.accdb**.

Datenmodell

Das Datenmodell ist bereits für die Ergänzungen dieses Artikels vorbereitet. Die Personen können wir den Verträgen über die Tabelle **tblPersonenVertraege** zuordnen, die Personen den Mobilfunkgeräten über die Tabelle **tblPersonenMobilfunkgeräte**. Die beteiligten Tabellen finden Sie in der Übersicht aus Bild 1.

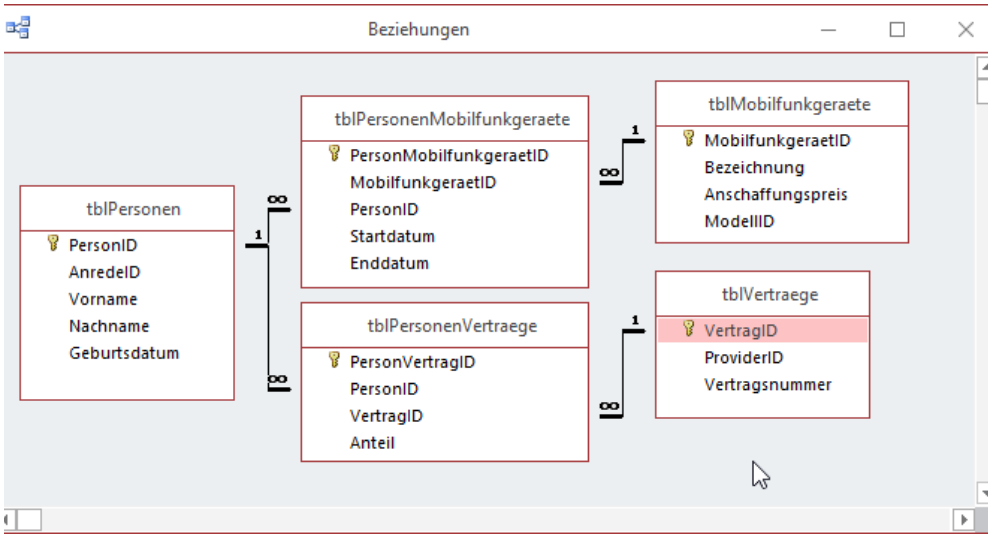


Bild 1: Die an der Zuordnung von Handys und Verträgen zu den Personen beteiligten Tabellen

Warum benötigen wir m:n-Beziehungen und keine einfachen Lookupfelder in den Tabellen **tblMobilfunkgeraete** und **tblVertraege**? Bei den Mobilfunkgeräten gehen wir davon aus, dass die Geräte durchaus »vererbt« werden können, beispielsweise von Familienmitglied zu Familienmitglied. Deshalb speichern wir mit Datum, wann welches Gerät von welcher Person genutzt wurde. Andersherum kann es auch sein, dass ein Familienmitglied gleich mehrere Geräte verwendet – beispielsweise ein Smartphone und ein Tablet oder zusätzlich noch eine Smartwatch. Bei den Verträgen wollen wir es ermöglichen, die Kosten aufzuteilen zwischen mehreren Personen – oder auch einer Person die Kosten von mehreren Verträgen aufzudrücken.

Zuordnung von Mobilfunkgeräten zu Personen

Um die Mobilfunkgeräte einer Person zuzuordnen, gibt es zwei Möglichkeiten: Wir führen im Formular **frmPersonendetails** in einem Unterformular eine Liste der Geräte dieser Person. Oder wir fügen zum Formular **frmMobilfunkgeraetDetails** entsprechende Steuerelemente hinzu, um die Zuordnung zu einer Person festzulegen.

Wir beginnen mit dem Formular **frmPersonendetails**. Zunächst legen wir eine Abfrage an, die wir als Datenherkunft für das noch zu erstellende Unterformular **sfmPersonenMobilfunkgeraete** anlegen. Dieses enthält die Tabelle **tblPersonenMobilfunkgeraete** als Datenquelle und liefert alle Felder, wobei die Abfrage die Datensätze absteigend nach dem Feld **Startdatum** sortiert (siehe Bild 2).

Diese Abfrage weisen wir nach dem Speichern dem neuen Unterformular **sfmPersonen-Mobilfunkgeraete** als Datenherkunft zu.

Dem Detailbereich des Formularentwurfs weisen wir nun die drei Felder **MobilfunkgeraetID**, **Startdatum** und **Enddatum** zu (siehe Bild 3). Außerdem stellen wir für die Eigenschaft **Standardansicht** den Wert **Datenblatt** fest.

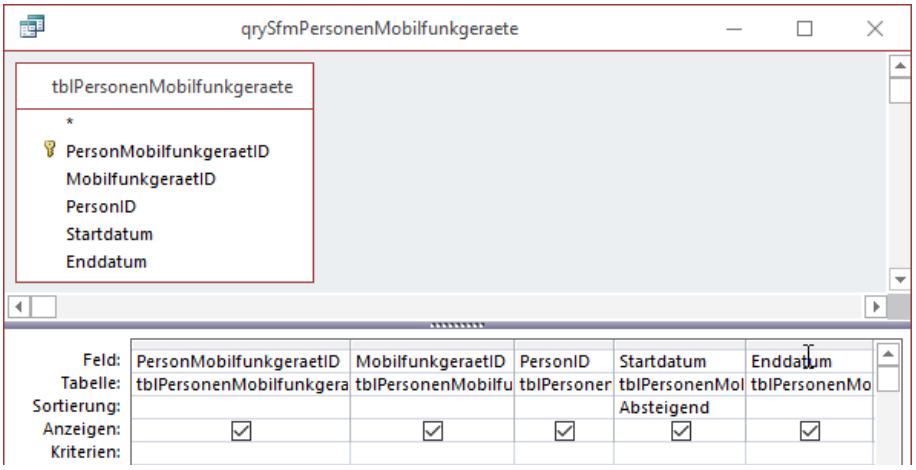


Bild 2: Abfrage als Datenherkunft des Unterformulars **sfmPersonenMobilfunkgeraete**

Danach speichern und schließen wir das Unterformular **sfmPersonenMobilfunkgeraete** und öffnen das Formular **frmPersonendetails** in der Entwurfsansicht. Die **OK**-Schaltfläche ziehen wir ein wenig nach unten und fügen per Drag and Drop aus dem Navigationsbereich das Unterformular **sfmPersonenMobilfunkgeraete** zum Entwurf des Formulars hinzu.

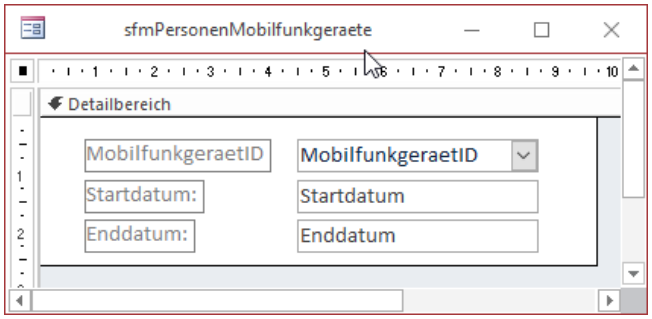


Bild 3: Das Unterformular **sfmPersonenMobilfunkgeraete**

Das Ergebnis sieht wie in Bild 4 aus. Das Eigenschaftsfenster sollte für das Unterformular-Steuerelement gleich den Wert **PersonID** in den beiden Eigenschaften **Verknüpfen von** und **Verknüpfen nach** eingetragen haben.

Dadurch werden gleich beim Wechsel in die Formularansicht automatisch nur die Einträge der Tabelle **tblPersonenMobilfunkgeraete** angezeigt, die auch der jeweiligen Person zugewiesen sind. Wenn Sie noch keine Werte von Hand in die Tabelle **tblPersonenMobilfunkgeraete** eingetragen haben, dürfte das Unterformular jedoch noch leer sein.

Wenn Sie allerdings schon ein paar Datensätze in die Tabelle

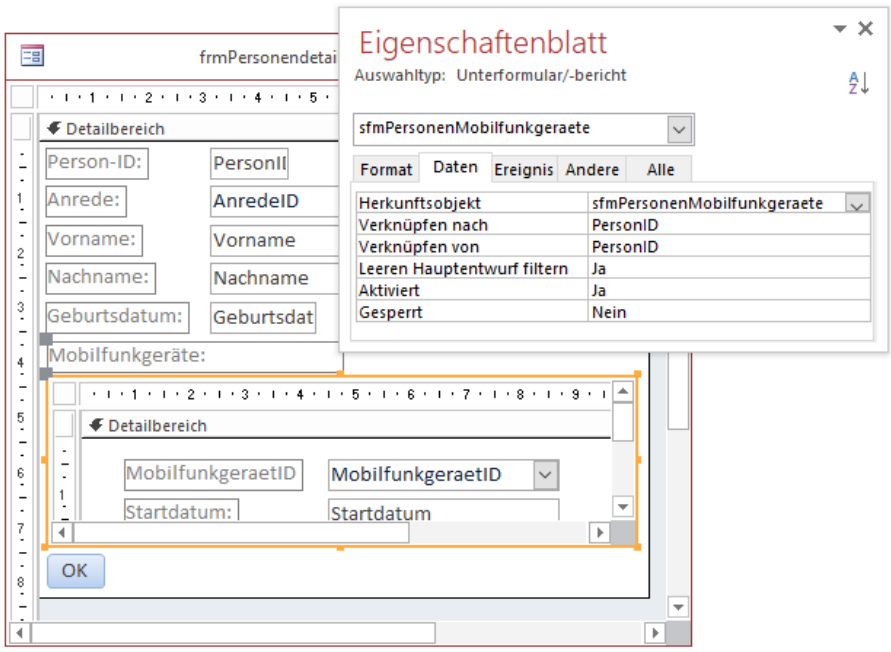


Bild 4: Einbau des Unterformulars in das Hauptformular

tblMobilfunkgeraete eingegeben haben, lässt sich im Formular **frmPersonendetails** zumindest schon einmal ein Datensatz über das Kombinationsfeld auswählen (siehe Bild 5).

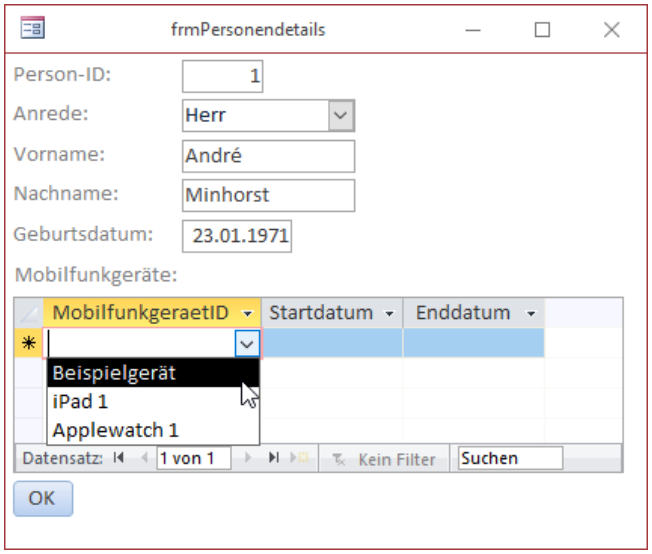


Bild 5: Das Formular **frmPersonendetails** mit den zugeordneten Mobilfunkgeräten

Die Auswahl können wir nun noch mit anderen Funktionen kombinieren. So könnten wir beispielsweise dafür sorgen, dass das Feld **Startdatum**, wenn es bei der Auswahl des Mobilfunkgeräts noch leer ist, mit dem aktuellen Datum gefüllt wird. Das erledigen wir, indem wir für das Ereignis **Nach Aktualisierung** des Kombinationsfeldes zur Auswahl des Mobilfunkgerätes eine entsprechende Ereignisprozedur hinterlegen. Dazu ändern wir zunächst den Namen des Kombinationsfeldes im Unterformular auf **cboMobilfunkgeraetID** (wir wollen zumindest alle Elemente, für die wir Ereignisprozeduren anlegen, mit einem Präfix versehen, welches den Steuerelementtyp erkennen lässt). Anschließend legen wir die folgende Ereignisprozedur an:

```
Private Sub cboMobilfunkgeraetID_AfterUpdate()
    If IsNull(Me!txtStartdatum) Then
        Me!txtStartdatum = Date
    End If
End Sub
```

Die Prozedur prüft, ob das Feld zum Speichern des Startdatums, welches wir ebenfalls mit einem entsprechenden Präfix versehen, leer ist. Falls ja, trägt es das aktuelle Datum als Startdatum ein.

Mobilfunkgeräte übernehmen

Den Aufwand mit der Verknüpfungstabelle haben wir durchgeführt, damit wir es auch speichern können, wenn ein Gerät einen gewissen Zeitraum der einen Person zugeordnet ist und danach einer anderen Person zugeteilt wird. Diesen Vorgang müssen wir nun noch in unserer Lösung berücksichtigen.

Wenn beispielsweise der Person mit der ID **1** das Gerät namens Beispielgerät zugeordnet ist und wir dieses Gerät nun der Person mit der ID **2** zuordnen wollen (siehe Bild 6), müssen wir auch hier eingreifen: Sonst haben nämlich bald beide Personen das gleiche Gerät als aktives Gerät in ihrer Liste stehen. Was tun? Wir müssen beim Auswählen des Gerätes auch noch prüfen, ob es bereits einer anderen Person zugeordnet ist. In

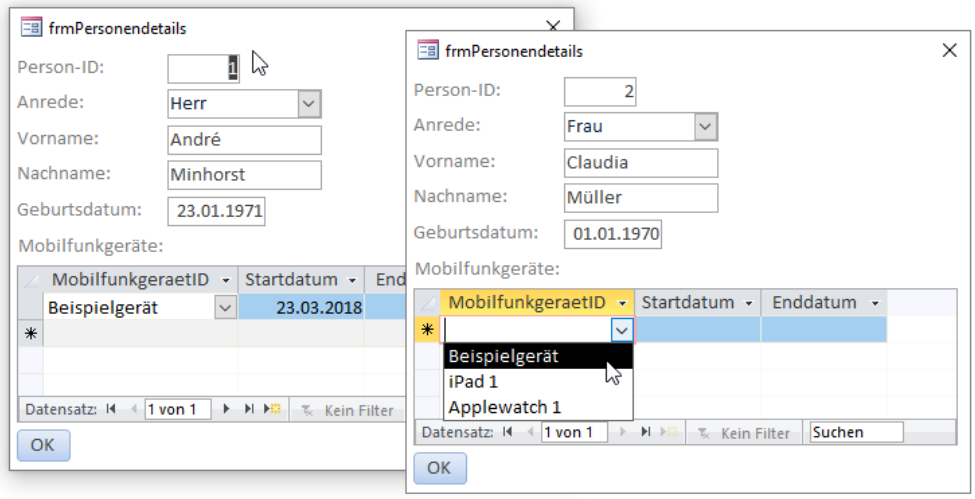


Bild 6: Mobilfunkgerät von einem zum anderen Benutzer

```
Private Sub cboMobilfunkgeraetID_BeforeUpdate(Cancel As Integer)
    Dim lngPersonID As Long
    Dim strPerson As String
    Dim db As DAO.Database
    Dim strSQL As String
    Dim intResult As VbMsgBoxResult
    lngPersonID = Nz(DLookup("PersonID", "tblPersonenMobilfunkgeraete", "MobilfunkgeraetID = " & Me!cboMobilfunkgeraetID _
        & " AND Enddatum IS NULL"), 0)
    If Not lngPersonID = 0 Then
        strPerson = DLookup("Vorname & ' ' & Nachname", "tblPersonen", "PersonID = " & lngPersonID)
        intResult = MsgBox("Das Gerät ist aktuell der Person '" & strPerson & "' zugeordnet. Übernehmen?", vbYesNo)
        If intResult = vbYes Then
            Set db = CurrentDb
            strSQL = "UPDATE tblPersonenMobilfunkgeraete SET Enddatum = " & SQLDatum(Date) & " WHERE PersonID = " _
                & lngPersonID & " AND Enddatum IS NULL"
            db.Execute strSQL
        Else
            Cancel = True
        End If
    End If
End Sub
```

Listing 1: Übernehmen eines Mobilfunkgerätes durch einen anderen Nutzer

diesem Falle benötigen wir weiteren Code, der nicht nur die Prüfung durchführt, sondern auch noch ein Meldungsfenster anzeigt, mit dem wir den Benutzer fragen, ob dieses Mobilfunkgerät den Besitzer wechseln soll. In diesem Fall nutzen wir dann nicht das Ereignis **Nach Aktualisierung**, sondern **Vor Aktualisierung**. Wenn sich der Benutzer nämlich entscheidet, das vergebene Gerät nicht einer anderen Person zuzuordnen, können wir dieser Ereignisprozedur über den Parameter **Cancel** nämlich mitteilen, dass die Auswahl des Eintrags über das Kombinationsfeld abgebrochen wird.

Die Prozedur aus Listing 1 erledigt genau dies: Sie versucht zunächst, den Primärschlüsselwert eines Datensatzes der Tabelle **tblPersonenMobilfunkgeraete** zu ermitteln, der bereits mit dem auszuwählenden Mobilfunkgerät verknüpft ist und bei dem das Feld **Enddatum** zusätzlich noch leer ist.

Findet die Prozedur einen solchen Eintrag, landet dieser in der Variablen **lngPersonID**, welche sonst den beim Initialisieren der Variablen vergebenen Wert **0** behält. Die folgende **If...Then**-Bedingung prüft, ob **lngPersonID** einen anderen Wert als **0** aufweist. Falls ja, ermittelt sie mit einer weiteren **DLookup**-Funktion den vollen Namen dieser Person.

Im ersten Parameter der **DLookup**-Funktion geben wir daher ausnahmsweise mal nicht nur den Namen eines einzigen Feldes an, sondern stellen einen Ausdruck aus zwei Feldnamen, getrennt durch ein Leerzeichen, zusammen:

```
"Vorname & ' ' & Nachname"
```

Danach erscheint eine Meldungsbox, welche den Benutzer darauf hinweist, dass das Mobilfunkgerät aktuell mit dem ermittelten Benutzer verknüpft ist und ihn fragt, ob es an den neuen Benutzer übertragen werden soll. Klickt der Benutzer hier auf die Schaltfläche **Ja**, stellt die Prozedur eine SQL-Anweisung zusammen, welche den Wert des Feldes **Enddatum** für alle Datensätze der Tabelle **tblPersonenMobil-**

funkgeraete, deren Feld **Enddatum** noch leer ist und deren Feld **PersonID** mit dem Wert aus der Variablen **lngPersonID** übereinstimmt, auf das aktuelle Datum einstellt.

Wenn wir danach einen Blick in die Tabelle **tblPersonenMobilfunkgeraete** werfen, erhalten wir die beiden Datensätze aus Bild 7.

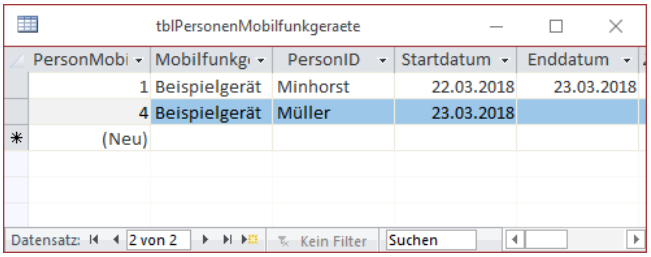


Bild 7: Mobilfunkgerät, das von einer Person zur nächsten gewandert ist

Prüfung beim Öffnen der Anwendung

Das Problem bei Daten dieser Art ist, dass sich schnell Inkonsistenzen einschleichen können. Das heißt, dass ein Benutzer beispielsweise direkt die Daten der Tabelle **tblPersonenMobilfunkgeraete** manipulieren könnte und dadurch ein Gerät gleichzeitig mehreren Personen zugewiesen ist. Wenn Sie den SQL Server oder ein anderes RDBMS-System als Backend nutzen würden, könnten Sie einen Trigger definieren, der bei Änderungen am Datensatz ausgelöst wird und diese verhindert, wenn die Änderung die Integrität der Daten gefährdet.

Über diese Möglichkeiten verfügen wir unter Access theoretisch auch – wenn Sie die Datenmakros verwenden wollen, die seit Access 2010 bereitgestellt werden. In Bild 8 zeigen wir, wie Sie ein Datenmakro für unseren Anwendungsfall programmieren können. Für dieses legen wir zunächst das Datenmakro-Pendant zur **DLookup**-Funktion fest und prüfen, ob es in der Tabelle **tblPersonenMobilfunkgeraete** bereits einen Datensatz gibt, der den gewählten Wert für das Feld **MobilfunkgeraetID** besitzt, dessen Feld **Enddatum** noch den Wert **Null** aufweist und der nicht mit dem aktuellen Datensatz übereinstimmt. Der komplette Ausdruck sieht so aus:

```
[t1].[MobilfunkgeraetID]=[tblPersonenMobilfunkgeraete].
[MobilfunkgeraetID] Und [t1].[Enddatum] Ist Null Und
[t1].[PersonMobilfunkgeraetID]<>[tblPersonenMobilfunkge
raete].[PersonMobilfunkgeraetID]
```

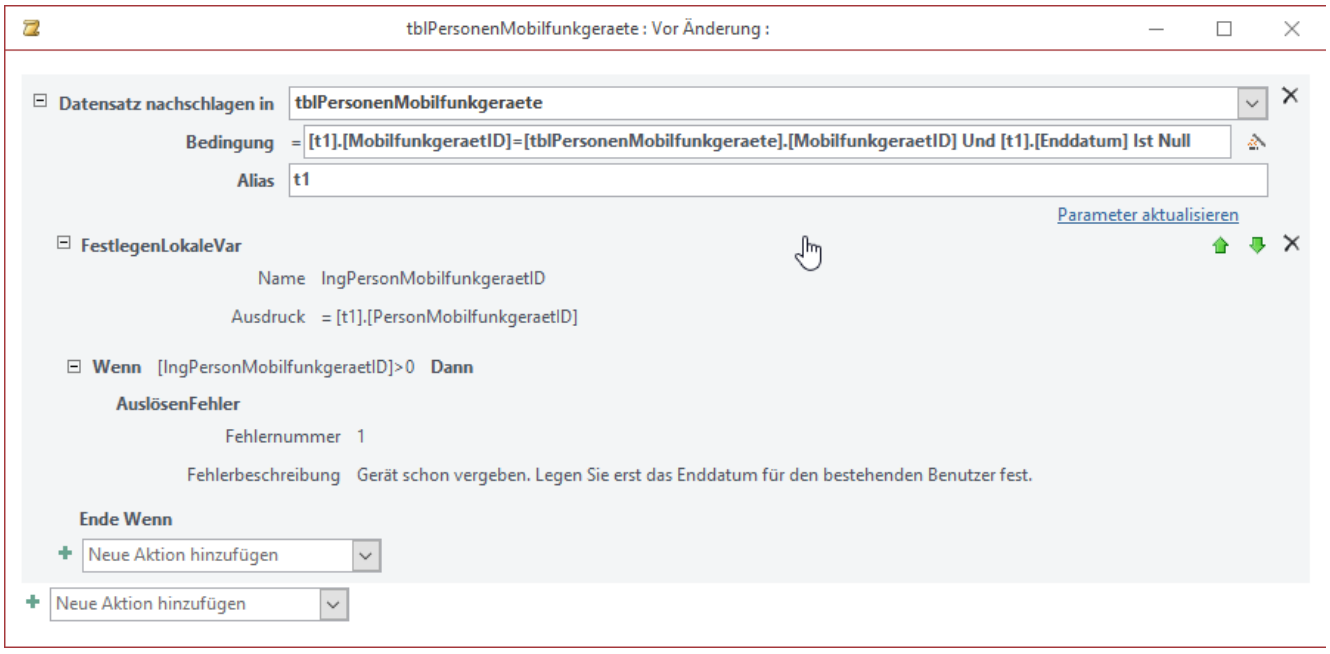


Bild 8: Datenmakro, das Änderungen direkt an der Tabelle verhindert.

Ribbon für die Handyverwaltung

Unsere Anwendung »Handyverwaltung« haben wir nun mit allen notwendigen Tabellen, Abfragen und Formularen ausgestattet. Es fehlen noch Berichte, um die enthaltenen Daten auszugeben und natürlich das Sahnehäubchen auf die Benutzeroberfläche: ein Ribbon, mit dem sich die verschiedenen Formulare und Funktionen aufrufen lassen. Dieses werden wir im vorliegenden Artikel vorstellen.

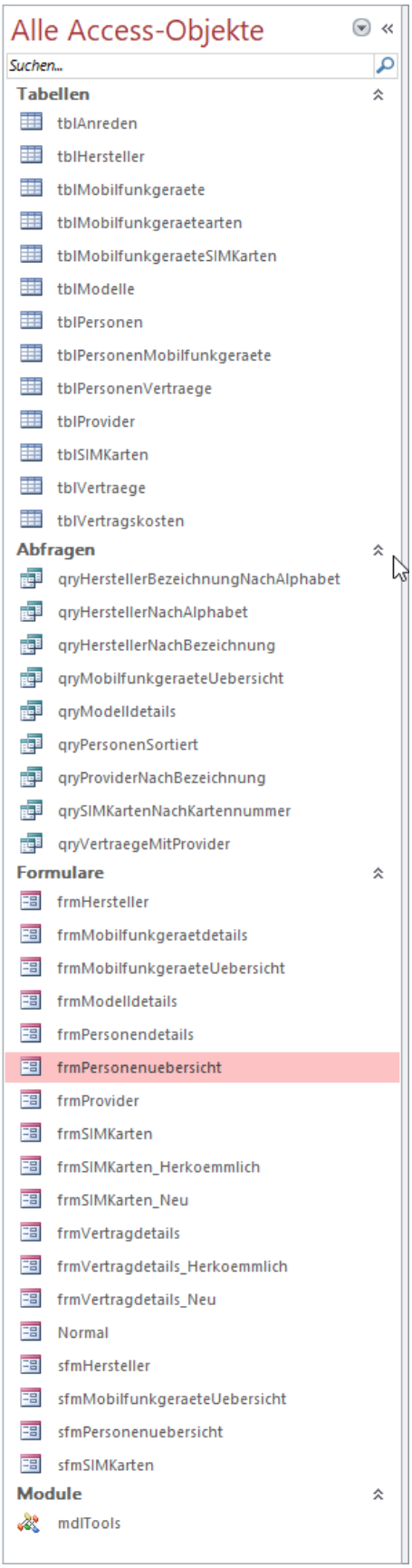


Bild 1: Access-Objekte

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1802_Handyverwaltung.accdb**.

Benötigte Ribbon-Befehle

Was genau wollen wir mit dem Ribbon steuern? Als Erstes wollen wir dem Benutzer nicht zumuten, dass er die Formulare und (noch zu erstellenden) Berichte der Anwendung per Doppelklick auf die Einträge im Navigationsbereich öffnen muss. Die Anzahl der Elemente ist zwar noch überschaubar (siehe Bild 1), aber dem Benutzer wollen wir doch eine optisch ansprechendere Möglichkeit bieten, die Funktionen der Anwendung anzusteuern. Außerdem sind die meisten Elemente im Navigationsbereich für den Benutzer irrelevant beziehungsweise sollen überhaupt nicht geöffnet werden können – zum Beispiel die Tabellen, Abfragen und Module. Auch die Unterformulare sollte er tunlichst nicht öffnen, außer als Teil des entsprechenden Hauptformulars. Und viele der Formulare erfordern für eine einwandfreie Funktion auch die Angabe von Öffnungsparametern, Filtern und anderer Einstellungen, sodass ein Öffnen per Doppelklick aus dem Navigationsbereich heraus keinen Sinn macht.

Welche Elemente aber wollen wir nun im Ribbon unterbringen und wie ordnen wir diese an? Wir haben ja verschiedene Möglichkeiten für die Erstellung einer Hierarchie, die normalerweise über die Ebenen **tab**-Element, **group**-Element und untergeordnete Steuerelemente wie etwa das **button**-Element organisiert werden.

Um zu ermitteln, welche Formular und Funktionen wir vom Ribbon aus aufrufen wollen, hilft ein Blick in das Datenmodell (siehe Bild 2). Hier können wir zunächst die Verknüpfungstabellen zum Definieren der m:n-Beziehungen herauslassen. Es bleiben dann die folgenden Entitäten:

- Personen
- Mobilfunkgeräte
- SIM-Karten

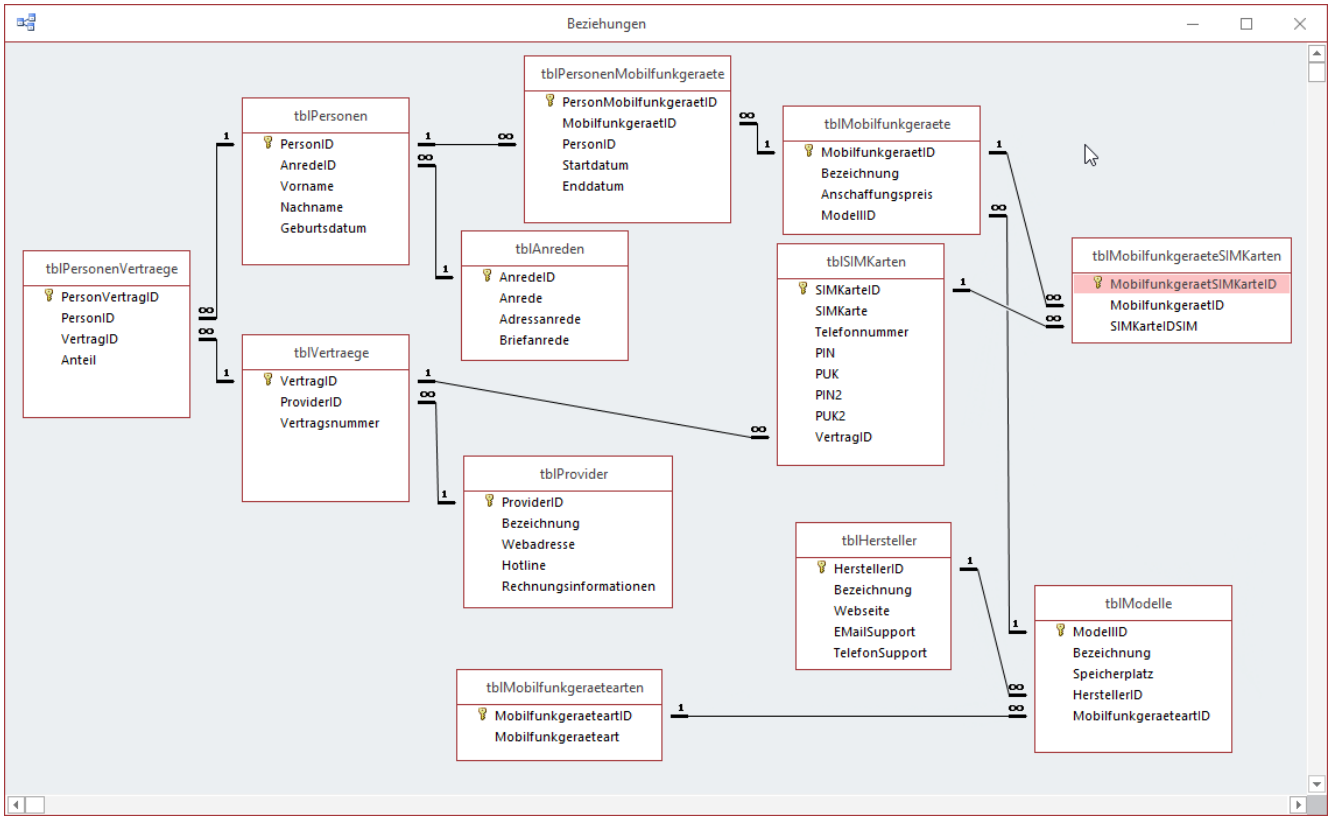


Bild 2: Datenmodell

- Verträge
- Provider
- Hersteller
- Modelle

Für diese Elemente haben wir jeweils teilweise Übersichtsformular, teilweise Detailformulare, aber auch kombinierte Formulare. Normalerweise würde man dies vielleicht konsistenter in immer der gleichen Art von Darstellung für Übersichten und Details programmieren, aber wir wollten in dieser Beispiellösung ja auch die verschiedenen Möglichkeiten darstellen.

Im Prinzip gibt es hier keine Stammdaten, die nicht immer wieder mal geändert werden, daher wollen wir alle Entitäten gleich behandeln und diese in einem einzigen tab-Element darstellen. Für jede Gruppe von Elementen erstellen wir jedoch jeweils zwei Einträge, nämlich einen zur Anzeige der Übersicht und einen zum Anlegen eines neuen Eintrags. Bei den Entitäten, wo wir die Übersicht und die Detailansicht in eigenen Formularen anzeigen, ist das kein Problem.

Bei den Entitäten, die in einem einzigen Formular erscheinen, prinzipiell auch nicht – hier wird das gleiche Formular eben einmal zur Anzeige der Übersicht mit dem ersten Datensatz in der Detailansicht angezeigt und einmal mit einem neuen, leeren Datensatz.

Wir haben noch zwei Tabellen ausgelassen: **tblAnreden** und **tblMobilfunkgeraetearten**. Während die Anreden mit den Einträgen **Herr** und **Frau** vollständig sind und nicht mehr bearbeitet werden müssen, gibt es für die Mobilfunkgerätearten schlicht keine eigene Eingabemaske. Sollten Mobilfunkgerätearten hinzukommen, kann der Benutzer diese direkt über das entsprechende Kombinationsfeld **cboMobilfunkgeraetearten** im Formular **frmModelldetails** eingeben.

Ribbon-Tabelle hinzufügen

Um das Ribbon zu definieren, benötigen wir als Erstes eine geeignete Tabelle, um die Ribbon-Definition zu speichern. Diese nennen wir **USysRibbons**, damit Access diese Tabelle beim Starten automatisch erkennt und die darin enthaltenen Ribbon-Definitionen verwenden kann. Unter anderem tauchen diese dann in den Access-Optionen auf, wo Sie die beim Öffnen der Anwendung anzuzeigende Ribbon-Definition festlegen können. Legen Sie also eine neue Tabelle an und fügen Sie die folgenden Felder hinzu:

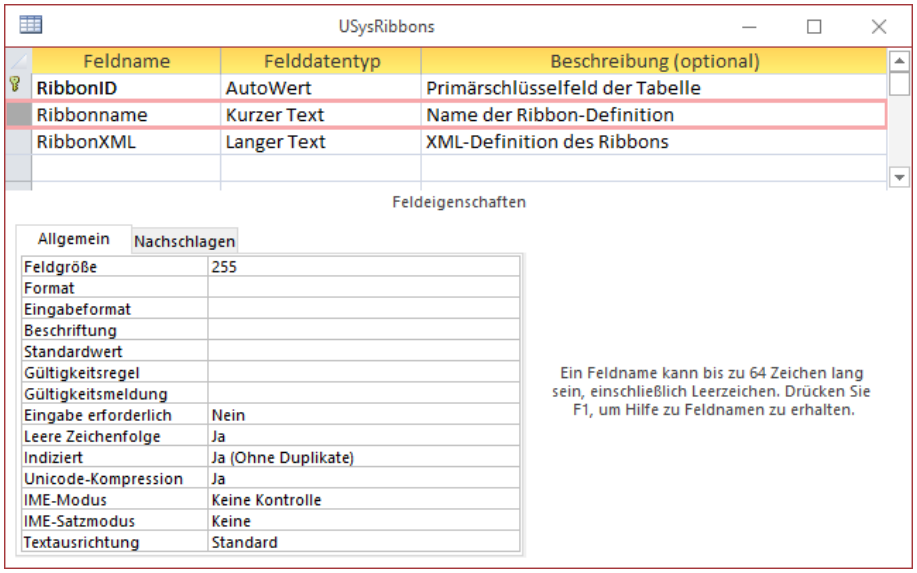


Bild 3: Tabelle zum Speichern der Ribbon-Definitionen

- **RibbonID**: Primärschlüsselfeld der Tabelle
- **Ribbonname**: Name der Ribbon-Definition
- **RibbonXML**: XML-Definition des Ribbons

Für das Feld **Ribbonname** legen wir vorsichtshalber einen eindeutigen Index fest, damit nicht versehentlich zwei Mal der gleiche Name angegeben wird (siehe Bild 3).

Wenn Sie die Tabelle gespeichert und geschlossen haben, wundern Sie sich möglicherweise, dass die Tabelle nicht im Navigationsbereich erscheint. Das ist der Fall, weil die Tabelle durch die Bezeichnung automatisch zur Systemtabelle wird.

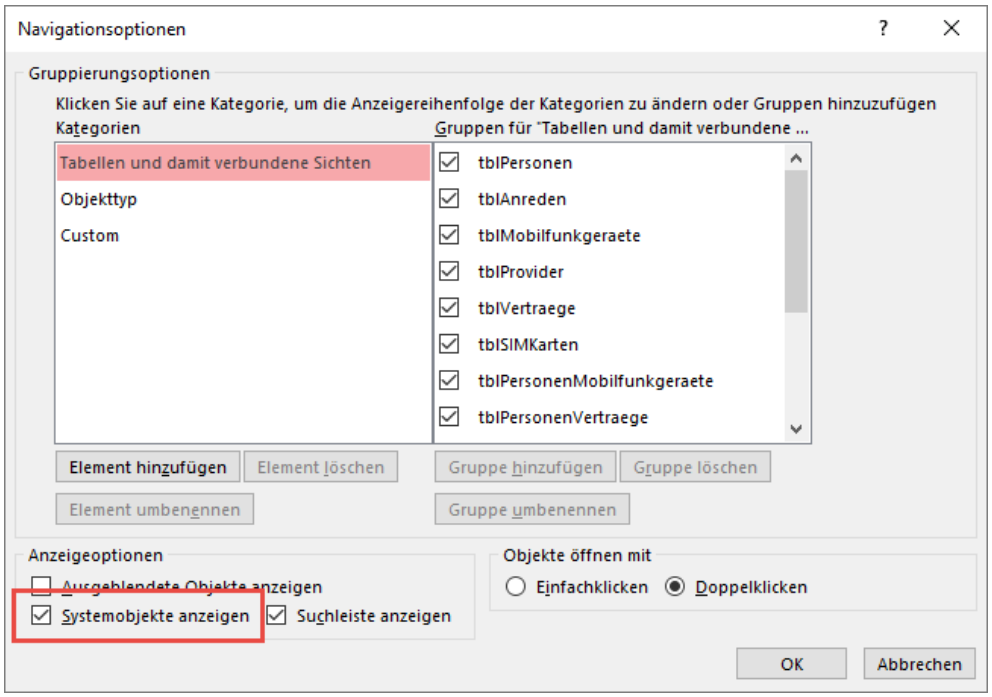


Bild 4: Aktivieren der Anzeige der Systemobjekte im Navigationsbereich

Wenn Sie diese Tabelle im Navigationsbereich sehen wollen, müssen Sie mit der rechten Maustaste auf die Titelleiste des Navigationsbereichs klicken und den Eintrag **Navigationsoptionen...** aus dem Kontextmenü auswählen. Es erscheint der Dialog aus Bild 4, wo Sie die Option **Systemobjekte anzeigen** aktivieren. Anschließend

erscheint neben den anderen Systemtabellen wie **MSysObjects** und so weiter auch die Tabelle **USysRibbons** im Navigationsbereich.

Ribbon-Definition erstellen

Damit können wir uns an die Arbeit machen und die Ribbon-Definition erstellen. Wir benötigen zunächst ein umschließendes customUI-Element, dass wir mit dem Attribut **loadImage** ausstatten und diesem den Wert **loadImage** zuweisen. Damit legen wir fest, dass wann immer ein **image**-Attribut für Element angegeben ist, die für **loadImage** angegebene Callback-Funktion aufgerufen wird, damit es das dem Element image zugewiesene Bild einliest und für das Element im Ribbon anzeigt:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui" loadImage="loadImage">
```

Danach folgt das ribbon-Element, für das wir ein Attribut definieren, nämlich **startFromScratch**. Wenn wir dieses auf true einstellen, wird das Ribbon beim Starten der Anwendung komplett neu aufgebaut und die eingebauten Elemente werden dabei nicht berücksichtigt:

```
<ribbon startFromScratch="true">
```

Darin befindet sich das **tabs**-Element, welches ein oder mehrere **tab**-Elemente enthalten kann – je nachdem, wieviele Registerseiten das Ribbon enthalten soll:

```
<tabs>
```

Unser einziges **tab**-Element soll für das Attribut **id** den Wert **tabHandyverwaltung** und für **label** den Wert **Handyverwaltung** aufnehmen, der dann als Beschriftung angezeigt wird:

```
<tab id="tabHandyverwaltung" label="Handyverwaltung">
```

Innerhalb des **tab**-Elements folgen dann die **group**-Elemente, die dann wiederum die tatsächlichen Steuerelemente enthalten. Wir schauen uns dies am Beispiel der Gruppe mit der Beschriftung **Personen** an:

```
<group id="grpPersonen" label="Personen">
```

Auch hier reichen und die beiden Attribute **id** und **label** aus.

Anders sieht es bei den **button**-Elementen aus, von denen wir jeder Gruppe zwei hinzufügen. Die erste soll jeweils die Übersicht anzeigen, die zweite ein Formular mit einem neuen Datensatz. Für jedes **button**-Element hinterlegen wir mit dem Attribut image den Namen des Bildes, dass die Schaltfläche im Ribbon anzeigen soll. Diese Bilder speichern wir in der Tabelle **MSysResources**, die wir weiter unten beschreiben. Das Attribut **label** nimmt die Beschriftung des **button**-Elements auf, hier immer **Übersicht** oder **Neu**. Für **id** hinterlegen wir eine eindeutige Bezeichnung. Daher kommen wir mit **btnUebersicht** und **btnNeu** nicht aus, sondern müssen noch den Entitätstyp hinzunehmen – also etwa **btnPersonenUebersicht** und **btnPersonNeu**. Für das Attribut **onAction** hinterlegen wir den Namen der VBA-Prozedur, die beim Anklicken der Schaltfläche aufgerufen werden soll.

Hier tragen wir immer **onAction** ein, da wir nur eine solche Prozedur verwenden, die dann das aufrufende Steuerelement auswertet und dementsprechende Anweisungen ausführt. Schließlich gibt das Attribut **size** mit dem Wert **large** an, dass die Schaltflächen groß und mit 32x32 Pixel großen Icons angezeigt werden sollen. Hier die beiden Definitionen für die Schaltflächen der Gruppe **Personen**:

```
<button image="users4" label="Übersicht" id="btnPersonenUebersicht" onAction="onAction" size="large"/>
<button image="user_add" label="Neu" id="btnPersonNeu" onAction="onAction" size="large"/>
```

Die Gruppe endet schließlich mit dem schließenden **group**-Element:

```
</group>
```

Außerdem folgenden noch die weiteren schließenden Elemente:

```
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Die gesamte XML-Definition des Ribbons sieht schließlich wie in Listing 1 aus. Diesen Code kopieren oder schreiben Sie in das Feld **RibbonXML** der Tabelle **USysRibbons**. Tragen Sie außerdem für den gleichen Datensatz der Tabelle den Wert **Main** in das Feld **Ribbonname** ein. Die Tabelle sieht danach wie in Bild 5 aus.

Bevor wir die Ribbon-Definition nun als die beim Start der Anwendung anzuzeigende Definition auswählen, müssen wir noch einige weitere Schritte unternehmen:

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui" loadImage="loadImage">
  <ribbon startFromScratch="true">
    <tabs>
      <tab id="tabHandyverwaltung" label="Handyverwaltung">
        <group id="grpPersonen" label="Personen">
          <button image="users4" label="Übersicht" id="btnPersonenUebersicht" onAction="onAction" size="large"/>
          <button image="user_add" label="Neu" id="btnPersonNeu" onAction="onAction" size="large"/>
        </group>
        <group id="grpVertraege" label="Verträge">
          <button image="document_text" label="Übersicht" id="btnVertraegeUebersicht" onAction="onAction" size="large"/>
          <button image="document_text_plus" label="Neu" id="btnVertragNeu" onAction="onAction" size="large"/>
        </group>
        <group id="grpProvider" label="Provider">
          <button image="antenna" label="Übersicht" id="btnProviderUebersicht" onAction="onAction" size="large"/>
          <button image="antenna_plus" label="Neu" id="btnProviderNeu" onAction="onAction" size="large"/>
        </group>
        <group id="grpMobilfunkgeraete" label="Mobilfunkgeräte">
          <button image="smartphone" label="Übersicht" id="btnMobilfunkgeraeteUebersicht" onAction="onAction" size="large"/>
          <button image="smartphone_plus" label="Neu" id="btnMobilfunkgeraetNeu" onAction="onAction" size="large"/>
        </group>
        <group id="grpSIMKarten" label="SIM-Karten">
          <button image="sim_card" label="Übersicht" id="btnSIMKartenUebersicht" onAction="onAction" size="large"/>
          <button image="sim_card_plus" label="Neu" id="btnSIMKarteNeu" onAction="onAction" size="large"/>
        </group>
        <group id="grpHersteller" label="Hersteller">
          <button image="factory" label="Übersicht" id="btnHerstellerUebersicht" onAction="onAction" size="large"/>
          <button image="factory_plus" label="Neu" id="btnHerstellerNeu" onAction="onAction" size="large"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Listing 1: Code der Ribbon-Definition

Daten im Ribbon anzeigen

Im Artikel **Ribbon für die Handyverwaltung** haben wir ein Ribbon für eine bestehende Anwendung angelegt. Dort gibt es einige Übersichtsformulare, die man per Ribbon aufrufen kann. Aber warum erst die Übersicht anzeigen, wenn man auch direkt über das Ribbon auf die Details eines Datensatzes zugreifen kann? Zum Beispiel, indem man die Elemente per Kombinationsfeld oder auf andere Art und Weise darstellt? Dieser Artikel stellt die Möglichkeiten des Ribbons zur Anzeige von Daten aus den Tabellen der Datenbank vor.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1802_DatenImRibbon.accdb**.

Listenelemente

Das Ribbon bietet fünf verschiedene Elemente, mit denen Sie Listen anzeigen können:

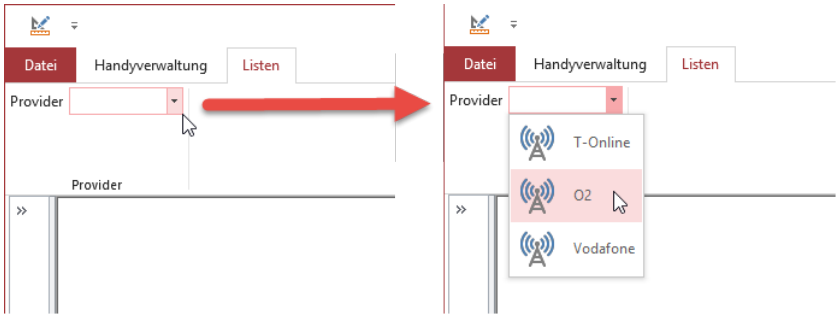


Bild 1: Beispiel einer **comboBox**

- **comboBox**: Entspricht etwa einem Kombinationsfeld unter Access, allerdings ohne die Möglichkeit der Eingabe von Daten.
- **dropDown**: wie **comboBox**, allerdings mit mehr Möglichkeiten
- **dynamicMenu**: eine weitere Alternative, deren Besonderheit darin besteht, seinen Inhalt im XML-Format entgegenzunehmen
- **gallery**: Kann Elemente in mehreren Spalten und Zeilen anzeigen und außerdem Abbildungen darstellen.
- **menu**: Nimmt normale Menüeintrage wir früher unter Access 2003 und älter auf, bietet aber zusätzlich die Möglichkeit zur Angabe einer Beschreibung.

Das comboBox-Element

Das **comboBox**-Element können Sie statisch per XML-Code füllen oder auch dynamisch über entsprechende Callback-Funktionen. Ein Beispiel für ein statisches **comboBox**-Element definieren wir wie folgt:

```
<comboBox label="Provider" id="cboProvider">
  <item id="itm1" label="T-Online" image="antenna"/>
  <item id="itm2" label="O2" image="antenna"/>
  <item id="itm3" label="Vodafone" image="antenna"/>
</comboBox>
```

Das Ergebnis finden Sie in Bild 1. Damit die Bilder angezeigt werden, benötigen Sie weiteren Code, den wir im Artikel **Ribbon für die Handyverwaltung** beschreiben.

comboBox dynamisch füllen

Das können wir auch dynamisch hinbekommen. Dazu fügen wir einem neuen **comboBox**-Element namens **cboProviderDynamisch** einfach vier weitere Attribute hinzu, nämlich **getItemCount**, **getItemLabel** und **getItemID**. Die Definition des Elements sieht danach wie folgt aus:

```
<comboBox label="Provider dynamisch"
  id="cboProviderDynamisch"
  getItemCount="getItemCount"
  getItemLabel="getItemLabel"
  getItemID="getItemID"/>
```

```
Sub getItemCount(control As IRibbonControl, ByRef count)
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim i As Integer
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM tblProvider", dbOpenDynaset)
    Do While Not rst.EOF
        ReDim Preserve strProvider(2, i + 1) As String
        strProvider(0, i) = rst!ProviderID
        strProvider(1, i) = rst!Bezeichnung
        i = i + 1
        rst.MoveNext
    Loop
    count = i
    rst.Close
    Set rst = Nothing
    Set db = Nothing
End Sub
```

Listing 1: Ermitteln der Anzahl der einzulesenden Datensätze und Speichern der Datensätze in einem Array

Für die drei Callback-Attribute hinterlegen wir entsprechende Prozeduren. Dabei ist eine kleine Vorbemerkung nötig. Die Callback-Prozeduren werden nämlich nach einem bestimmten Schema aufgerufen. Die Callback-Prozedur für das Attribut **getItemCount** wird einmal zu Beginn aufgerufen. Diese erwartet als Rückgabewert vor allem einen Wert für den Parameter **count**, der die Anzahl der anzuzeigenden Einträge enthalten sollte. Die anderen beiden Callbacks für die Attribute **getItemID** und **getItemLabel** werden entsprechend der Anzahl der mit count ermittelten Einträge aufgerufen. Dabei übergibt der zweite Parameter jeweils den Wert für den Index des aktuellen Durchlaufs an. Das heißt, das wir optimalerweise in der Prozedur **getItemCount** ein Array anlegen, das wir mit den beim Zählen und dabei Durchlaufen der Datensätze mit den relevanten Daten, also dem Index und der Beschriftung füllen. Damit wir dieses zweidimensionale Array in der ersten Prozedur füllen und anschließend in den beiden Prozeduren **getItemID** und **getLabelID** auslesen können, deklarieren wir es im Kopf unseres Moduls namens **mdlRibbon**, in dem sich auch die übrigen Ribbon-spezifischen Prozeduren befinden:

```
Dim strProvider() As String
```

Diese nutzen wir dann in der Prozedur **getItemCount**, die wir wie in Listing 1 aufbauen. Sie ermittelt die anzuzeigenden Daten aus der Tabelle **tblProvider** über ein Recordset, das sie anschließend in einer **Do While**-Schleife durchläuft. Dabei erweitert sie zunächst die Anzahl der Einträge im Array mit der **Redim**-Anweisung, wobei wir die Anzahl der ersten Dimension auf **2** und die der zweiten Dimension auf die aktuelle Anzahl der Datensätze plus eins einstellen und die bereits eingetragenen Elemente durch das **Preserve**-Schlüsselwort beibehalten. Dann schreibt die Prozedur innerhalb der Schleife den Wert des Feldes **ProviderID** in die erste Spalte des Arrays und den des Feldes **Bezeichnung** in die zweite Spalte. Außerdem erhöht sie die Zählervariable **i** um eins.

Danach trägt sie den Wert von **i** für den Rückgabeparameter **count** ein.

Nun haben wir beispielsweise den Wert 3 zurückgegeben. Das bedeutet, dass die beiden Prozeduren **getItemID** und **getItemLabel** jeweils drei Mal aufgerufen werden. Die Prozedur **getItemID** sieht dabei beispielsweise wie folgt aus:

```
Sub getItemID(control As IRibbonControl, _
        index As Integer, ByRef id)
    id = strProvider(0, index)
End Sub
```

Die Prozedur erhält mit **control** einen Verweis auf das aufrufende Ribbon-Element und mit **index** die

Nummer des aktuell auszulesenden Elements. Außerdem hat sie einen Rückgabeparameter namens **id**. Die einzige Zeile dieser Prozedur trägt den Wert des Array für die erste Spalte und die mit **index** übergebene Spalte in die Variable **id** für den Rückgabewert ein.

Die Prozedur **getItemLabel** arbeitet auf identische Weise. Der Unterschied ist, dass der Rückgabeparameter **label** heißt und das wir diesem den Wert aus der zweiten Spalte und den angegebenen Index aus dem Array zuweisen:

```
Sub getItemLabel(control As IRibbonControl, _
        index As Integer, ByRef label)
    label = strProvider(1, index)
End Sub
```

Das Ergebnis sieht wie in Bild 2 aus, also prinzipiell genauso wie beim ersten Beispiel. Die Beispielbilder haben wir weggelassen, da es ja auch keine in der Tabelle **tblProvider** gespeicherten Bilder für diese Einträge gibt.

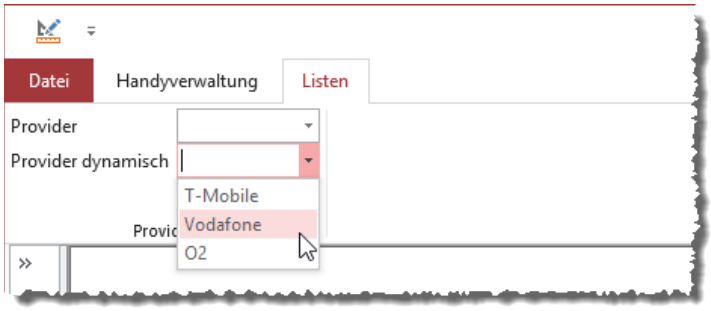


Bild 2: Dynamisch gefülltes **comboBox**-Element

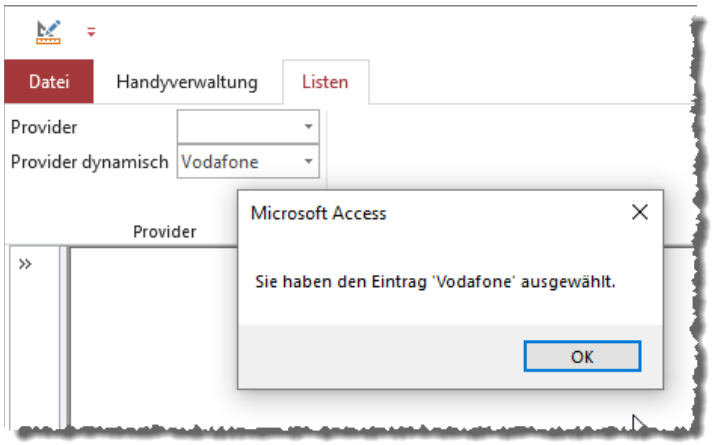


Bild 3: Auswertung des gewählten Eintrags

Beim Auswählen eines Eintrags

Wenn der Benutzer nun einen der Einträge auswählt, wollen wir diese Auswahl natürlich auch auswerten können. Dazu nutzen wir im Falle des **comboBox**-Elements das Attribut **onChange**, das wir wie folgt zum Element hinzufügen:

```
<comboBox label="Provider dynamisch"
  id="cboProviderDynamisch"
  onChange="onChange"
  getItemCount="getItemCount"
  getItemLabel="getItemLabel"
  getItemID="getItemID"/>
```

Wenn der Benutzer nun einen der Einträge auswählt, soll ein Meldungsfenster die Bezeichnung des gewählten Eintrags ausgeben, was wie in Bild 3 aussieht.

Dazu legen wir die folgende Prozedur an:

```
Sub onChange(control As IRibbonControl, text As String)
    MsgBox "Sie haben den Eintrag '" _
        & text & "' ausgewählt."
End Sub
```

Das ist ja sehr praktisch, dass der Parameter **text** direkt die Bezeichnung des gewählten Elements liefert. Wenn man allerdings nicht wie im Beispiel einfach nur die Bezeichnung ausgeben, sondern etwas mit dem zugrunde liegenden Datensatz erledigen möchte, wird es kompliziert – man müsste dann per **DLookup** erst den Primärschlüsselwert zu diesem Eintrag ermitteln und könnte erst dann weiterarbeiten. Allerdings gibt es auch keinen Ausweg – es sei denn, Sie setzen auf das zweite in diesem Artikel vorgestellte Steuerelement, das **dropDown**-Element!

Das dropDown-Element

Wenn man wie wir ein **dropDown**-Element genauso ausstattet wie das eingangs beschriebene

comboBox-Element und beide genau gleich aussehen, fragt man sich natürlich, wozu Microsoft überhaupt zwei solcher Steuerelemente vorgesehen hat (siehe Bild 4). Hier sehen Sie den Code, der für das bis auf die Bilder, die wir schlicht eingespart haben, identische **dropDown**-Element nötig ist:

```
<dropDown label="Provider" id="drpProvider">
  <item id="itm1" label="T-Online"/>
  <item id="itm2" label="O2"/>
  <item id="itm3" label="Vodafone"/>
</dropDown>
```

Die Unterschiede werden erst dann offensichtlich, wenn wir uns die Attribute und hier speziell die Callback-Attribute ansehen.

Ein voll ausgestattes **dropDown**-Element sieht beispielsweise wie folgt aus:

```
<dropDown label="Provider dynamisch"
  id="drpProviderDynamisch"
  onAction="onActionDropDown"
  getItemCount="getItemCount"
  getItemLabel="getItemLabel"
  getItemID="getItemID"
  getSelectedItemID="getSelectedItemID"/>
```

Hier sehen Sie schon, dass es kein **onChange**-Attribut gibt, das beim Auswählen eines Eintrags ausgewählt wird, sondern stattdessen ein **onAction**-Attribut. Außerdem finden wir hier zusätzlich ein Attribut namens **getSelectedItemID**.

Wir können die für das **comboBox**-Element verwendeten Prozeduren **getItemCount**, **getItemLabel** und **getItemID** beibehalten beziehungsweise mitnutzen. Neu hinzugefügt haben wir die Callback-Prozedur **getSelectedItemID**, die wie folgt aussieht:

```
Sub getSelectedItemID(control As IRibbonControl, _
        ByRef index)
    index = strProvider(0, 1)
End Sub
```

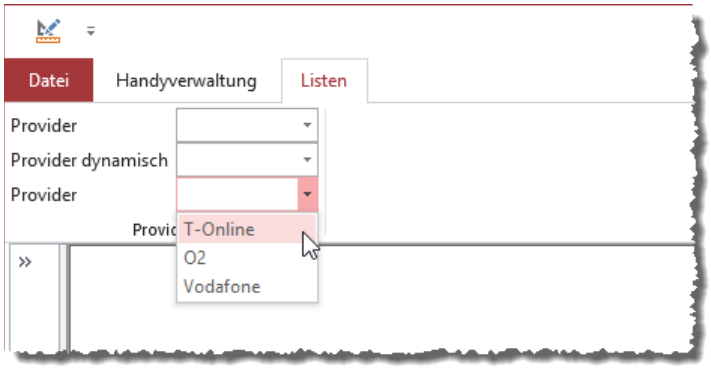


Bild 4: Eineiiger Zwilling des **comboBox**-Elements – aber nur optisch: das **dropDown**-Element

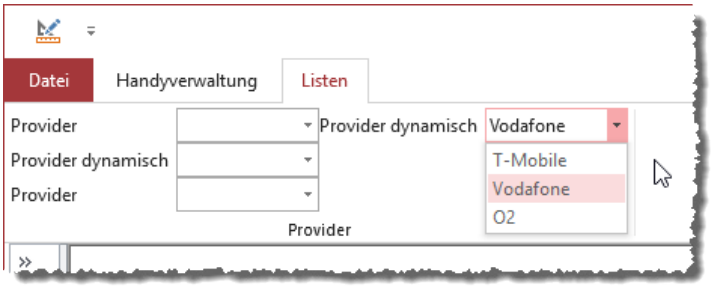


Bild 5: Voreinstellung eines Eintrags im **dropDown**-Element

Diese Prozedur weist als Rückgabeparameter den Parameter **index** auf, mit dem Sie den Wert des ausgewählten Eintrags festlegen können. Wir stellen diesen hier auf den Wert der ersten Spalte des ersten Eintrags des Arrays **strProvider** ein. Dies sorgt dafür, dass der entsprechende Eintrag direkt beim Anzeigen des **dropDown**-Elements vorausgewählt wird (siehe Bild 5).

Es gibt noch eine zweites, ähnliches Callback-Attribut namens **getSelectedItemIndex**. Damit können Sie den auszuwählenden Index zurückgeben.

Ausgewählten Eintrag synchronisieren

Interessant wird es natürlich, wenn Sie über das **dropDown**-Element einen Eintrag auswählen, dessen Detailseite angezeigt werden soll. Wenn der Benutzer dann einen anderen Eintrag in der Detailseite anzeigt, soll auch der aktuell im **dropDown**-Element ausgewählte Eintrag aktualisiert werden. Dies erreichen wir allerdings nur, wenn wir ein paar weitere Elemente

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui" onLoad="OnLoad_Main" loadImage="loadImage">
  <ribbon startFromScratch="true">
    <tabs>
      ...
      <tab id="tabListen" label="Listen">
        <group id="grpProviderliste" label="Provider">
          ...
          <dropDown label="Provider dynamisch" id="drpProviderDynamisch" onAction="onActionDropDown" getItemCount="getItemCount"
            getItemLabel="getItemLabel" getItemID="getItemID" getSelectedItemID="getSelectedItemID"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Listing 2: Ribbon-Definition mit **onLoad**-Methode

zum Ribbon und zum Code hinzufügen. Das erste ist das Callback-Attribut **onLoad**, welches Sie wie in Listing 2 mit dem Wert **onLoad_Main** versehen. Für dieses legen wir die folgende Callback-Prozedur fest:

```
Sub onLoad_Main(ribbon As IRibbonUI)
    Set objRibbon_Main = ribbon
End Sub
```

Diese Prozedur hat nur eine Aufgabe: Sie erhält über den Parameter **ribbon** einen Verweis auf die auslösende Ribbon-Definition und speichert diesen in der Variablen **objRibbon_Main**. Diese deklarieren wir im allgemeinen Teil des Moduls **mdlRibbon** wie folgt mit dem Datentyp **IRibbonUI**:

```
Public objRibbon_Main As IRibbonUI
```

Dies sorgt dafür, dass gleich nach dem Anzeigen der Elemente der Ribbon-Definition (dies kann ja auch nur eine einzige Schaltfläche sein, die Sie einer bestehenden Ribbon-Gruppe hinzugefügt haben) ein Verweis darauf in der Variablen **objRibbon_Main** landet.

Was haben wir nun davon? Dieses Objekt bietet uns nun die zum Beispiel Möglichkeit, eine der beiden Methoden **Invalidate** oder **InvalidateControl** aufzurufen. Die erste sorgt dafür, dass alle Elemente, welche Ihre Werte über Callback-Attribute erhalten, beim nächsten Erscheinen neu mit den entsprechenden Eigenschaftswerten gefüllt werden. Die zweite erwartet den Namen des betroffenen Ribbon-Elements als Parameter und aktualisiert nur die Callback-Attribut dieses einen Elements.

Wir machen nun folgendes: Den Wert des Primärschlüsselfeldes des Eintrags der Tabelle **tblProvider**, welchen das **dropDown**-Element **drpProviderDynamisch** anzeigen soll, speichern wir in einer Variablen, die wir wie folgt deklarieren:

```
Dim lngAktuellerProviderID As Long
```

Den Wert dieser Variablen übergeben wir nun in der Callback-Prozedur, die durch das Callback-Attribut **getSelectedItemID** ausgelöst wird. Sollte **lngAktuellerProviderID** zu diesem Zeitpunkt noch den Wert **0** aufweisen, lesen wir per **DLookup** den ersten Wert der Tabelle **tblProvider** in diese Variable ein:

```
Sub getSelectedItemID(control As IRibbonControl, _
        ByRef index)
    If lngAktuellerProviderID = 0 Then
        lngAktuellerProviderID = DLookup("ProviderID", _
            "tblProvider")
    End If
```

Autowert festlegen

Immer wieder erhalte ich die Frage, ob und wie man den Autowert anpassen kann – beispielsweise, um diesen bei einem bestimmten Nummernkreis beginnen zu lassen, sagen wir bei 1200001. Zusätzlich sollen die Werte dann natürlich auch noch durchnummeriert sein. Also schauen wir uns in diesem Artikel einmal an, welche Möglichkeiten wir haben und ob dies überhaupt sinnvoll ist.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1802_AutowertFestlegen.accdb**.

Standard-Autowert

Wenn Sie einer Tabelle ein Autowert-Feld hinzufügen, können Sie zwei Einstellungen für die Eigenschaft **Neue Werte** vornehmen: **Inkrement** und **Zufall**. Dort findet sich nicht etwa eine Einstellung, mit der Sie den Startwert oder das Inkrement selbst festlegen können (siehe Bild 1).

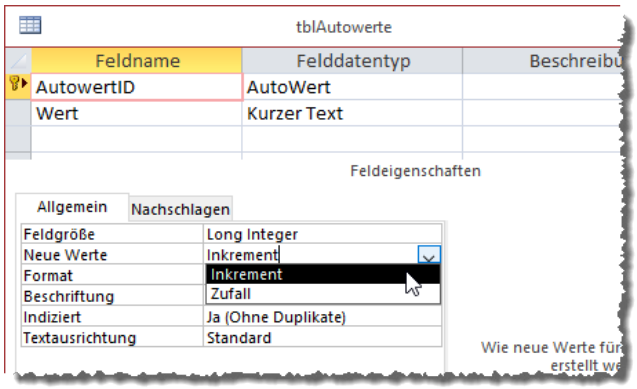


Bild 1: Zwei Varianten für den Autowert

Die Einstellung **Zufall** ist für unsere Zwecke gar nicht geeignet, also begnügen wir uns vorerst mit **Inkrement**. Damit erhalten wir, wie nicht anders zu erwarten, ein Autowert-Feld, dass mit dem Wert **1** initialisiert wird (siehe Bild 2).

Autowert per Code hochsetzen

Die erste Möglichkeit, den Autowert hochzusetzen, ist per Code. Sie erstellen einfach so viele Datensätze, bis die gewünschte Zahl erreicht ist:

```
Public Sub AutowertHochsetzen()
    Dim db As DAO.Database
    Dim l As Long
    Set db = CurrentDb
    For l = 1 To 100000
        db.Execute "INSERT INTO tblAutowerte(Wert) 
                    VALUES('')", dbFailOnError
    Next l
    db.Execute "DELETE FROM tblAutowerte", dbFailOnError
End Sub
```

Das dauerte bei 100.000 Datensätzen auf dem Testsystem allerdings schon knapp eine Minute. Das Ergebnis überzeugt allerdings. Man muss nur schauen, dass man nicht zuviele Datensätze hinzufügt – dann war die Arbeit umsonst (siehe Bild 3).

Autowert direkt setzen

Es geht allerdings auch noch einfacher – und vor allem schneller. Dabei setzen Sie den Autowert einfach

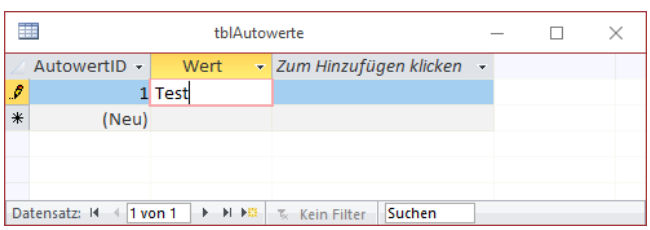


Bild 2: Normales Verhalten des Autowertfeldes

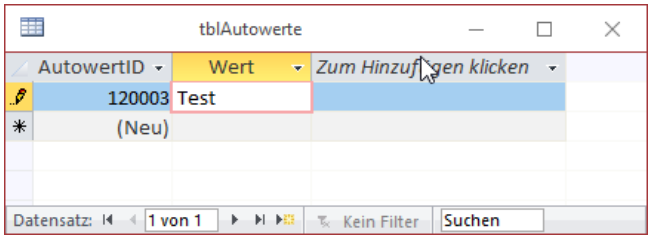


Bild 3: Hochsetzen des Autowerts per Code

Access und Excel: Makros aufzeichnen

Excel ist nicht nur Quelle für Daten, die dann in Access-Tabellen landen, sondern es lässt sich dank der gemeinsamen Programmiersprache VBA hervorragend programmieren. So können Sie von Access aus Excel-Dateien erstellen und mit Daten füllen oder auch vorhandene Excel-Dateien öffnen und bearbeiten oder Daten daraus auslesen. Wie aber findet man beispielsweise heraus, wie die verschiedenen Objekte unter Excel heißen und wie man auf die verschiedenen Bereiche wie Tabellen oder Zellen zugreift? All dies zeigt die vorliegende Beitragsreihe.

Makros unter Excel

Bevor wir richtig einsteigen, noch etwas zur Begriffsklärung: Wenn Sie zuvor entweder nur mit Excel oder nur mit Access gearbeitet haben, kommen Sie möglicherweise bei der Verwendung des Begriffs Makro unter den beiden Programmen durcheinander:

- Unter Access ist das Makro ein eigener Objekttyp, der genau wie Tabellen, Abfragen, Formulare oder Berichte sowie Module angelegt werden kann. Hier bilden Makros eine vereinfachte Möglichkeit, die Benutzeroberfläche ohne den Einsatz von VBA zu automatisieren.

- Unter Excel hingegen ist Makro eine Bezeichnung für eine VBA-Prozedur. Allerdings gibt es hier im Gegensatz zu Access die Möglichkeit, Aktionen des Benutzers auf der Benutzeroberfläche von

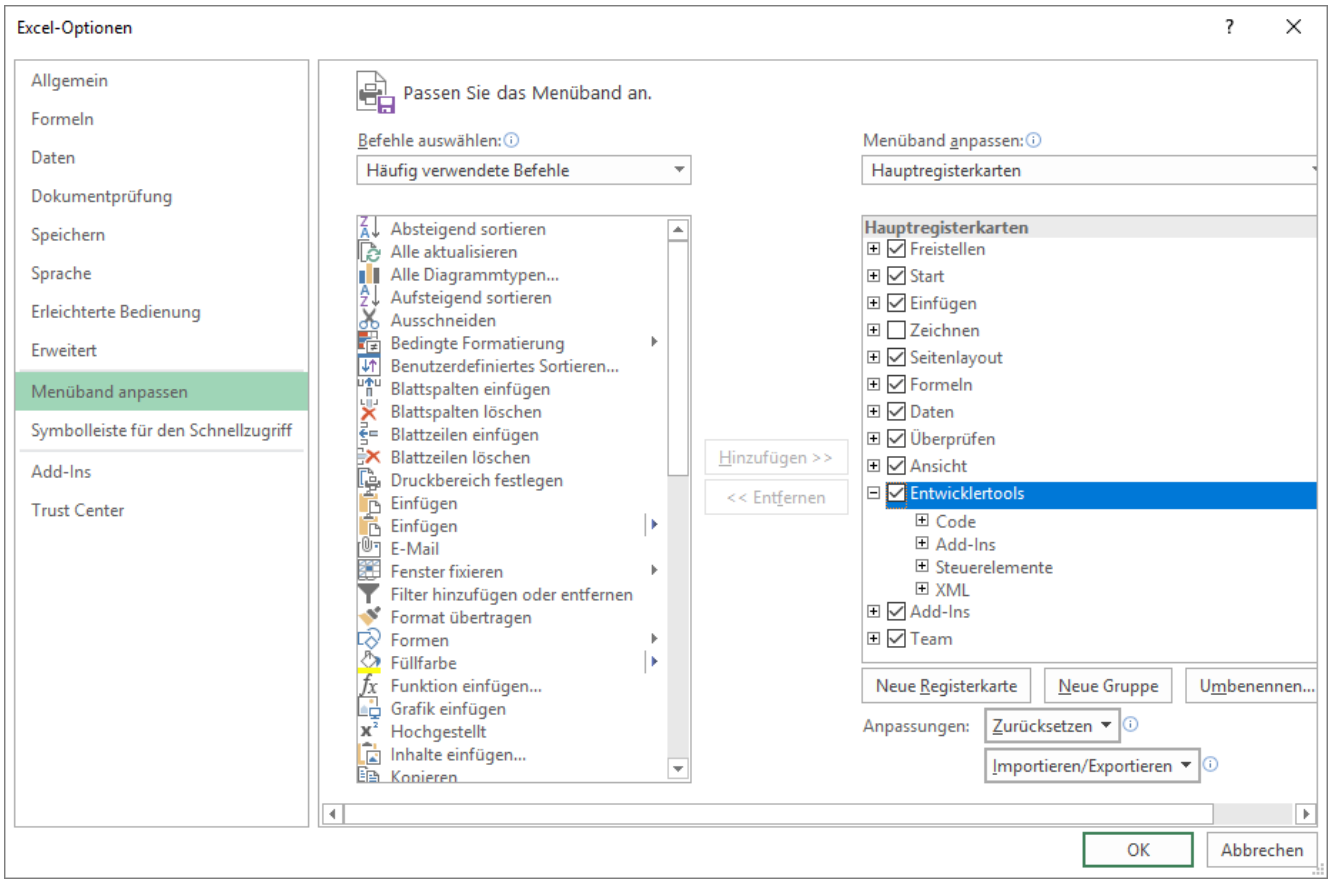


Bild 1: Ribbon-Tab **Entwicklertools** einblenden

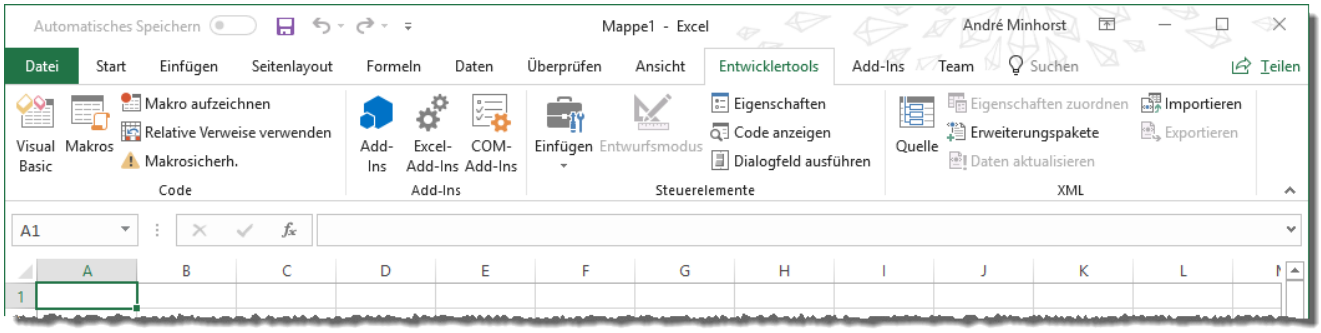


Bild 2: Das Tab **Entwicklertools**

Excel in Form von VBA-Prozeduren aufzuzeichnen – hier spricht man auch davon, ein Makro aufzuzeichnen. Diese Möglichkeit gibt es übrigens auch unter Word.

Wenn wir also nachfolgend von Makro sprechen, dann geht es um die unter Excel aufgezeichneten VBA-Prozeduren.

Makro aufzeichnen

Wenn wir einmal dabei sind, zeigen wir Ihnen gleich, wie Sie unter Excel ein Makro aufzeichnen können. Zunächst einmal ist der notwendige Ribbon-Befehl gar nicht verfügbar, denn Sie müssen diesen zuerst einblenden. Das ist aber kein Problem.

Unter Excel 2016 geht dies beispielsweise wie folgt:

- Wählen Sie den Ribbon-Befehl **Datei|Optionen** aus.
- Aktivieren Sie im Bereich **Menüband anpassen** rechts unter Hauptregisterkarten den Eintrag **Entwicklertools** (siehe Bild 1).

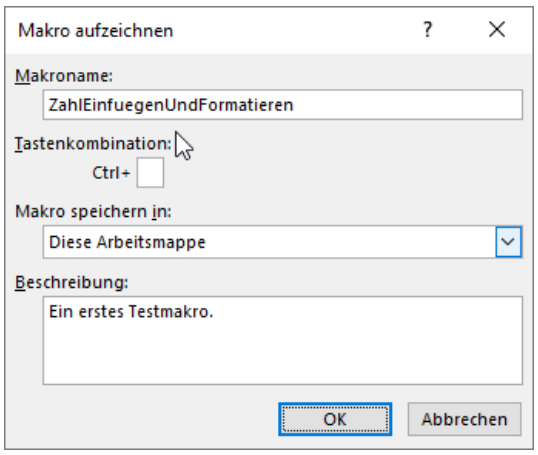


Bild 3: Dialog zum Aufzeichnen eines Makros

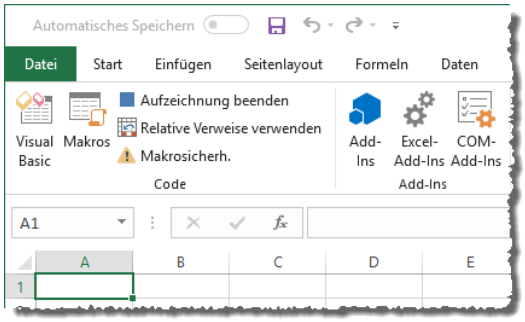


Bild 4: Die Schaltfläche **Makro aufzeichnen** verwandelt sich in **Aufzeichung beenden**.

- Danach finden Sie ein neues Ribbon-Tab mit der Beschriftung Entwicklertools vor (siehe Bild 2).

Makro aufnehmen

Nun wollen wir uns anschauen, wie wir eine einfache Aktion in Excel aufzeichnen und was wir dem so automatisch erstellten Code entnehmen können. Der Ribbon-Befehl zum Starten der Aufnahme eines Excel-Makros lautet **Entwicklertools|Code|Makro aufzeichnen**. Klicken Sie diesen an, erscheint der Dialog aus Bild 3. Hier haben Sie folgende Möglichkeiten:

- **Makroname**: Erwartet eine Bezeichnung. Wenn Sie nur schnell etwas testen wollen, behalten Sie den vorgegebenen Namen bei.
- **Tastenkombination**: Damit können Sie eine Tastenkombination festlegen, die das Excel-Makro startet.
- **Makro speichern in**: **Diese Arbeitsmappe** speichert das Excel-Makro in der aktuell geöffneten Datei. **Neue Arbeitsmappe** legt das Excel-Makro in einer neuen Arbeitsmappe ab. **Persönliche Ma-**

Lookup-Daten löschen

Wenn Sie mit Lookup-Daten arbeiten und dem Benutzer erlauben, Daten in die Lookup-Tabelle wie etwa eine Tabelle zum Verwalten von Kategorien einzugeben, sollten Sie auch eine Löschen-Funktion für diese Daten bereitstellen. Benutzer sind nämlich schnell übereifrig und legen ähnliche und somit unter Umständen redundante Daten an. Wenn dann fleißig Datensätzen den redundanten Lookupwerten zugeordnet wurden, ist guter Rat teuer: Wie die Dubletten entfernen, und was geschieht mit den bereits verknüpfte Daten? Dieser Artikel bringt Lichts ins Dunkel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1802_LookupLoeschen.accdb**.

Ausgangslage

Wir beschäftigen uns in diesem Artikel mit den beiden Tabellen **tblArtikel** und **tblKategorien** (siehe Bild 1). Zu jedem Artikel kann der Benutzer eine Kategorie auswählen. Um das flexibel zu gestalten, soll der Benutzer eigene Kategorien eingeben können – und zwar, indem er diese einfach in das Kombinationsfeld zur Auswahl der Kategorien einträgt und eine kurze Bestätigung nach Eingabe eines neuen Eintrags bestätigt. Dabei geschieht es dann, dass der Benutzer eine neue Kategorie anlegt – wie etwa **Bier** –, aber übersieht, dass es ja bereits eine Kategorie namens **Getränke** gibt, in die der neue Artikel **Pilsener Quellbräu** aufgenommen werden kann. In diesem Fall soll der Benutzer später seinen Fehler korrigieren können, indem er die falsch angelegte Kategorie wieder löscht. Sollten jedoch bereits Datensätze der Tabelle **tblArtikel** mit den zu löschenden

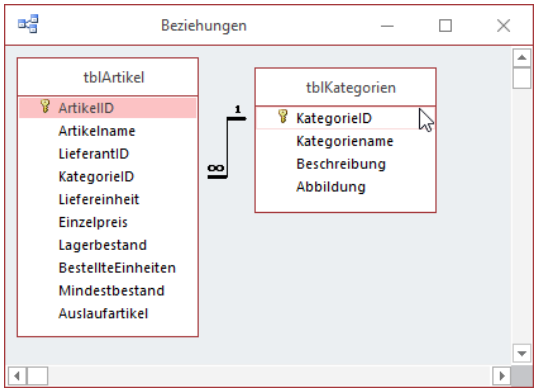


Bild 1: Tabellen der Beispieldatenbank

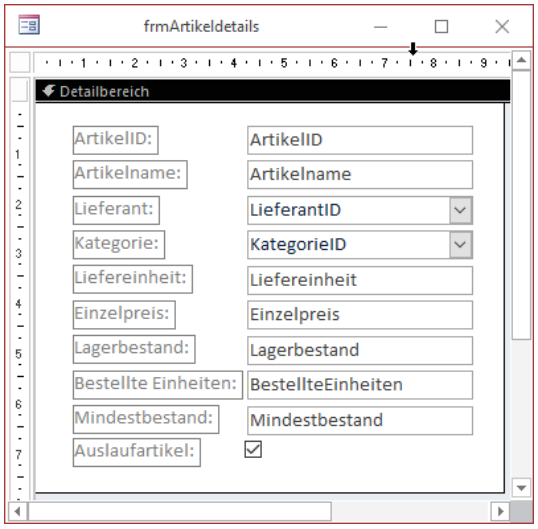


Bild 2: Entwurf des Formulars

Datensätzen aus der Tabelle **tblKategorien** angelegt sein, gelingt dies nicht so einfach: Wir haben nämlich referenzielle Integrität für die Beziehung zwischen den beiden Tabellen definiert, und zwar ohne Löschweitergabe. Das heißt, dass wir einen Datensatz aus der Tabelle **tblKategorien**, der mit mindestens einem Datensatz der Tabelle **tblArtikel** verknüpft ist, nicht einfach löschen können. Oder andersherum formuliert: Wir können nur leere Kategorien löschen.

Damit hat der Benutzer eine Aufgabe vor der Brust: Er muss nämlich alle Datensätze der Tabelle **tblArtikel** ausfindig machen, die mit der zu löschenden Kategorie verknüpft sind und diese mit einer anderen Kategorie verknüpfen und kann erst dann die überflüssige Kategorie löschen. Dies wollen wir im folgenden etwas einfacher gestalten.

Beispielformular

Ausgangspunkt dafür ist das Formular **tblArtikeldetails**, das im Entwurf wie in Bild 2 aussieht und über

die Eigenschaft **Datenherkunft** an die Tabelle **tblArtikel** gebunden wird. Da das Fremdschlüsselfeld **KategorieID** gleich im Tabellenentwurf als Nachschlagefeld ausgelegt wurde, erscheint es nach dem Hineinziehen der Felder aus der Feldliste in den Detailbereich gleich als Kombinationsfeld (übrigens genau wie das Feld **LieferantID**).

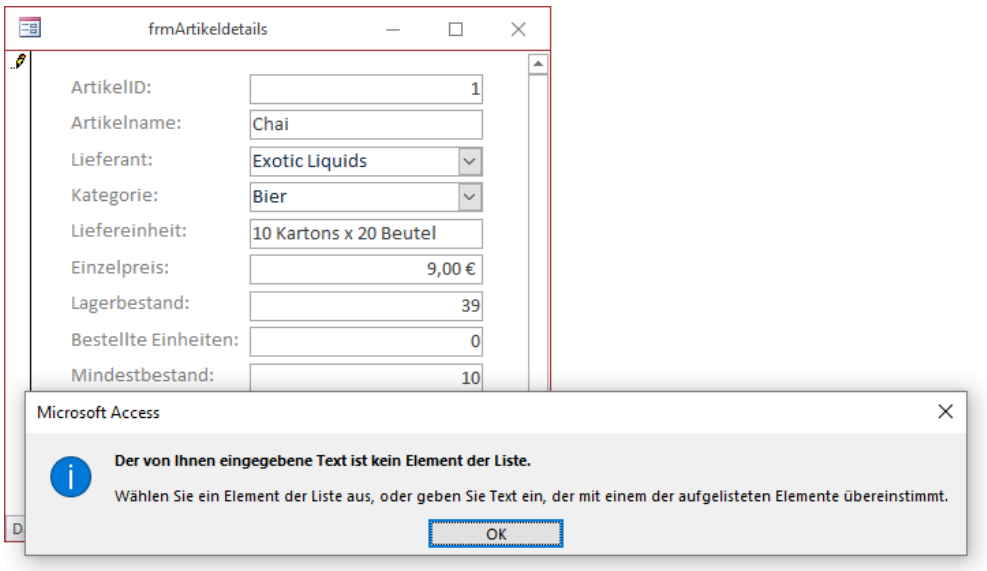


Bild 3: Im Standardzustand bewirkt das Einfügen einer neuen Kategorie diesen Fehler.

Wenn wir dann allerdings in die Formularansicht wechseln, erhalten wir die Fehlermeldung aus Bild 3. Kein Wunder: Der Eintrag ist in der Liste nicht bekannt.

Neuen Eintrag hinzufügen

Das ändern wir durch die Ereignisprozedur **Bei nicht in Liste** des Kombinationsfeldes, das wir zuvor in **cboKategorieID** umbenennen. Die Prozedur sieht wie in Listing 1 aus und liefert mit dem Parameter **NewData** den Wert des vom Benutzer neu hinzugefügten Eintrags. Die Prozedur fragt den Benutzer, ob die angegebene Kategorie neu in die Liste der Kategorien aufgenommen werden soll. Falls ja, führt sie eine **INSERT INTO**-Abfrage aus, welche einen entsprechenden neuen Datensatz zur Tabelle **tblKategorien** hinzufügt. In diesem Fall stellen wir den Wert des Rückgabeparameters **Response** auf **acDataErrAdded** ein, was Access mitteilt, dass ein neuer Eintrag hinzugefügt wurde.

Danach haben wir also eine neue Kategorie, die gegebenenfalls überflüssig ist, weil sie schon durch eine andere Kategorie abgedeckt wird, und der Benutzer fügt fleißig neue Artikel hinzu und weist sie dieser Kategorie zu. Nun benötigen wir Elemente für die Benutzeroberfläche, mit denen wir die Einträge der Tabelle **tblKategorien** verwalten können und dies es auch erlauben, den Artikeln mit einer zu löschenden Kategorie auf einfache Weise die richtige Kategorie zuzuweisen. Wir aber erledigen wir das auf effiziente Weise?

```
Private Sub cboKategorieID_NotInList(NewData As String, Response As Integer)
    Dim intResult As VbMsgBoxResult
    Dim db As DAO.Database
    intResult = MsgBox("Möchten Sie die Kategorie '" & NewData & "' hinzufügen?", vbYesNo, "Neue Kategorie")
    If intResult = vbYes Then
        Set db = CurrentDb
        db.Execute "INSERT INTO tblKategorien(Kategoriename) VALUES('" & NewData & "')", dbFailOnError
        Response = acDataErrAdded
    End If
End Sub
```

Listing 1: Hinzufügen einer neuen Kategorie zur Tabelle **tblKategorien**

Die übersichtlichste Methode ist die Anzeige eines neuen Formulars, dass die zu löschende Kategorie samt der verknüpfte Datensätze der Tabelle **tblArtikel** anzeigt. Außerdem sollte dieses Formular noch die Möglichkeit bieten, vor dem Löschen eine alternative Kategorie für die betroffenen Artikel auszuwählen – und das gegebenenfalls noch nach vorheriger Selektion der Artikel, um diese auch mehreren verschiedenen Kategorien zuordnen zu können.

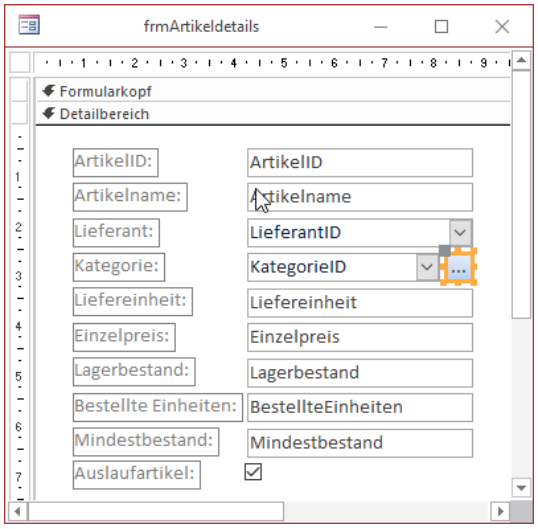


Bild 4: Schaltfläche zum Bearbeiten der Kategorien

in Bild 5 aus. Das obere Kombinationsfeld soll die Kategorie anzeigen, die im Formular **frmArtikeldetails** ausgewählt war, als der Benutzer auf die Schaltfläche mit den drei Punkten geklickt hat.

Dazu weisen wir dem Kombinationsfeld als Datensatzherkunft zunächst eine Abfrage auf Basis der Tabelle **tblKategorien** zu, die wie folgt lautet:

```
SELECT KategorieID, Kategoriename FROM tblKategorien ORDER BY Kategoriename;
```

Aufruf des Formulars zum Bearbeiten der Kategorien

Damit der Benutzer dieses Formular aufrufen kann, fügen wir dem Formular **frmArtikeldetails** eine Schaltfläche neben dem Kombinationsfeld **cboKategorieID** hinzu (siehe Bild 4). Die Schaltfläche heißt **cmdKategorieBearbeiten** und soll die folgende Ereignisprozedur auslösen:

```
Private Sub cmdKategorieBearbeiten_Click()
    DoCmd.OpenForm _
        "frmKategorienZuweisen", _
        WindowMode:=acDialog, _
        OpenArgs:=Me!cboKategorieID
End Sub
```

Die Prozedur öffnet das Formular **frmKategorienZuweisen** als modalen Dialog und übergibt die ID der aktuell im Kombinationsfeld **cboKategorieID** ausgewählten Kategorie als Öffnungsargument an das geöffnete Formular.

Formular zum Zuordnen und Löschen von Kategorien

Das Formular, mit dem wir überflüssige Kategorien loswerden wollen, heißt **frmKategorienZuweisen** und sieht im Entwurf wie

Außerdem stellen wir die Eigenschaft **Spaltenanzahl** auf den Wert **2** und **Spaltenbreiten** auf **0cm** ein, damit nur die zweite Spalte mit der Bezeichnung der Kategorie im Kombinationsfeld angezeigt wird.

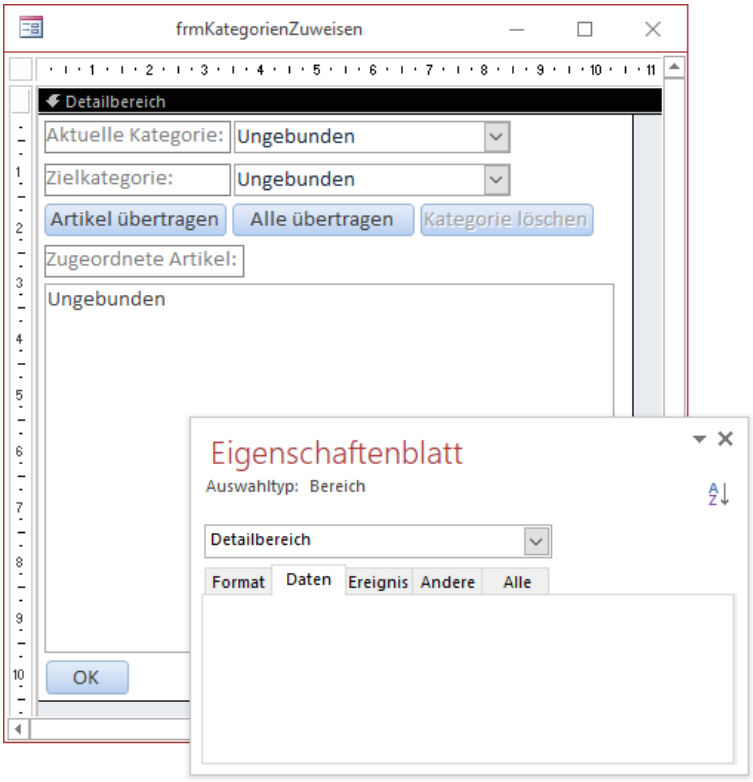


Bild 5: Formular zum Zuweisen und Löschen von Kategorien

Damit die Kategorie, deren **KategorieID** als Öffnungsargument des Formulars übergeben wurde, im Kombinationsfeld erscheint, bestücken wir die durch das Ereignis **Beim Laden** des Formulars ausgelöste Ereignisprozedur wie folgt:

```
Private Sub Form_Load()
    Me!cboAktuelleKategorieID = Me.OpenArgs
End Sub
```

Da das Formular nicht an eine Datenherkunft gebunden wird, stellen wir die Eigenschaften **Navigationsleiste**, **Datensatzmarkierer**, **Trennlinien** und **Bildlaufleisten** auf den Wert **Nein** und **Automatisch zentrieren** auf **Ja** ein. Wenn Sie das Formular nun für eine der Kategorien öffnen, sollte dies wie in Bild 6 aussehen.

Das zweite Kombinationsfeld statten wir fast mit der gleichen Datensatzherkunft wie das erste Kombinationsfeld aus – mit einem Unterschied: Die im ersten Kombinationsfeld angezeigte Kategorie soll dort nicht erhalten sein. Es macht ja auch keinen Sinn, die Artikel einer Kategorie der gleichen Kategorie zuzuweisen.

Da wir aber zur Entwurfszeit noch nicht wissen, welcher Datensatz im ersten Kombinationsfeld ausgewählt wird, stellen wir die Datensatzherkunft des zweiten Kombinationsfeldes auch erst beim Laden des Formulars ein. Dazu erweitern wir die Prozedur **Form_Load** wie in Listing 2.

Die erste Anweisung kennen Sie bereits, die zweite stellt eine SQL-Anweisung zusammen, die alle Datensätze der Tabelle **tblKategorien** außer der mit **Me.OpenArgs** gelieferten Kategorie enthält. Für die

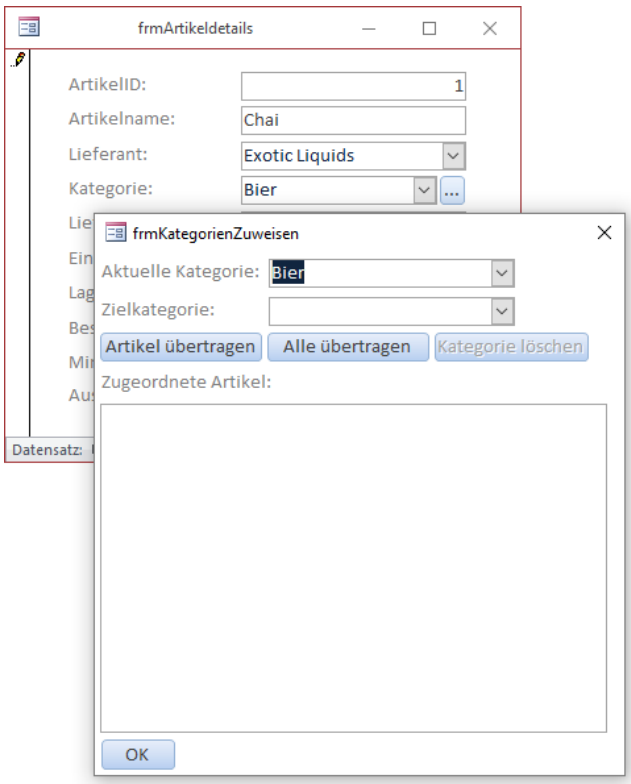


Bild 6: Formular mit der zu bearbeitenden Kategorie

Kategorie mit der ID 9 sieht die Abfrag beispielsweise so aus:

```
SELECT KategorieID, Kategoriename
FROM tblKategorien
WHERE KategorieID NOT IN (9)
ORDER BY Kategoriename
```

Die dritte Anweisung stellt das Kombinationsfeld **cboZielkategorieID** auf den ersten Eintrag der Datensatzherkunft ein. Auch für das Kombinationsfeld **cboZielkategorieID** stellen wie die Eigenschaften **Spaltenanzahl** auf **2** und **Spaltenbreiten** auf **0cm** ein.

```
Private Sub Form_Load()
    Me!cboAktuelleKategorieID = Me.OpenArgs
    Me!cboZielkategorieID.RowSource = "SELECT KategorieID, Kategoriename FROM tblKategorien WHERE KategorieID NOT IN (" _
        & Me.OpenArgs & ") ORDER BY Kategoriename"
    Me!cboZielkategorieID = Me.cboZielkategorieID.ItemData(0)
End Sub
```

Listing 2: Aktionen beim Laden des Formulars **frmKategorienZuweisen**

Füllen des Listenfeldes

Nun wollen wir noch das Listenfeld namens **lstZugeordneteArtikel** mit den Datensätzen der Tabelle **tblArtikel** bestücken, die aktuell mit der im ersten Kombinationsfeld **cboAktuelleKategorieID** angezeigten Kategorie verknüpft sind. Das erledigen wir initial ebenfalls in der Prozedur **Form_Load**. Das Listenfeld soll nur die Artikelnamen anzeigen, also stellen wir die Eigenschaft **Spaltenanzahl** auf **2** und die Eigenschaft **Spaltenbreiten** auf **0cm** ein. Der Prozedur **Form_Load** fügen wir am Ende die folgende Zeile hinzu:

```
Me!lstZugeordneteArtikel.RowSource = _
        "SELECT ArtikelID, Artikelname FROM tblArtikel _
              WHERE KategorieID = " & Me.OpenArgs & " _
                                  ORDER BY Artikelname"
```

Wenn wir das Formular nun über die Schaltfläche **cmdKategorieBearbeiten** des Formulars **frmArtikeldetails** öffnen, sieht dieses wie in Bild 7 aus.

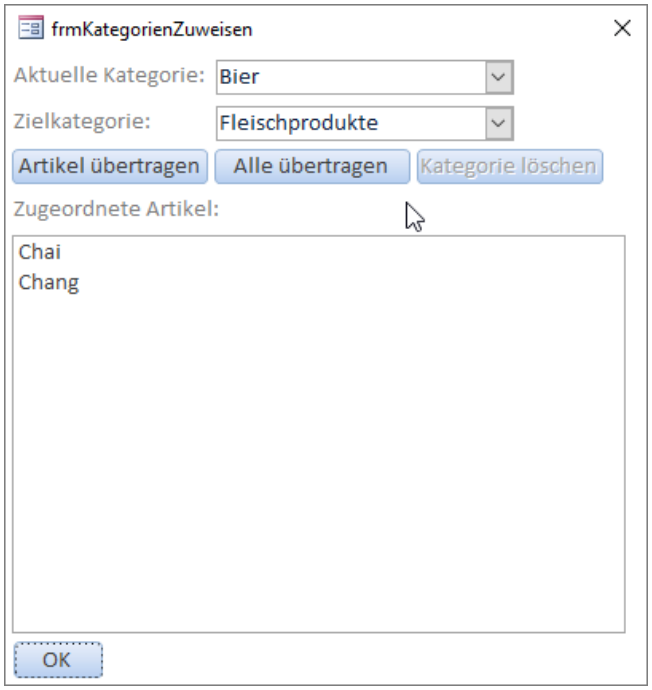


Bild 7: Alle notwendigen Daten sind nun verfügbar.

Daten aktualisieren

Nun könnte es sein, dass der Benutzer noch andere Kategorien als die aktuell im oberen Kombinationsfeld dargestellte entfernen möchte. Dann müssen wir auch das zweite Kombinationsfeld und das Listenfeld an die neue Auswahl anpassen. Also legen wir eine Prozedur an, die durch das Ereignis **Nach Aktualisierung** des Kombinationsfeldes **cboAktuelleKategorieID** ausgelöst wird. Nach dem Aufruf dieser Prozedur sollte das Formular dann etwa wie in Bild 8 aussehen.

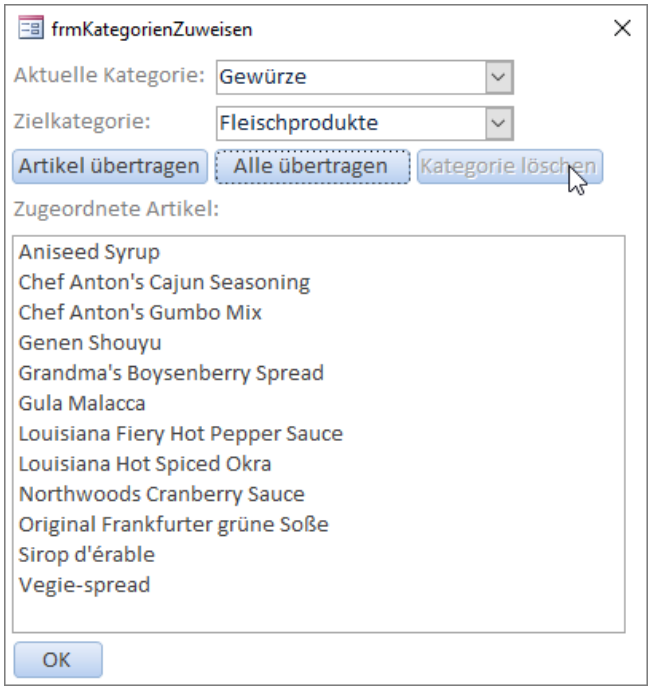


Bild 8: Formular nach dem Wechsel zu einer anderen Kategorie

Die Prozedur ermittelt zunächst den Primärschlüsselwert der im ersten Kombinationsfeld angezeigten Kategorie und speichert diese in der Variablen **lngKategorie**. Dann führt sie drei Anweisungen aus, die den hinteren drei Anweisungen der Ereignisprozedur **Form_Load** sehr ähneln.

Der einzige Unterschied ist, dass hier kein Bezug auf die mit dem Öffnungsargument übermittelte Kategorie mehr genommen wird, sondern auf die durch den Benutzer im Kombinationsfeld **cboAktuelleKategorieID** ausgewählte Kategorie. Dementsprechend zeigt das zweite Kombinationsfeld nun alle Datensätze der Tabelle **tblKategorien** an, die nicht im Kombinationsfeld **cboAktuelleKategorieID** ausgewählt sind und das Listenfeld liefert nun alle Artikel, die zu der aktuell

```
Private Sub cboAktuelleKategorieID_AfterUpdate()
    Dim lngAktuelleKategorieID As Long
    lngAktuelleKategorieID = Me!cboAktuelleKategorieID
    Me!cboZielkategorieID.RowSource = "SELECT KategorieID, Kategoriename FROM tblKategorien WHERE KategorieID NOT IN (" _
        & lngAktuelleKategorieID & ") ORDER BY Kategoriename"
    Me!cboZielkategorieID = Me!cboZielkategorieID.ItemData(0)
    Me!lstZugeordneteArtikel.RowSource = "SELECT ArtikelID, Artikelname FROM tblArtikel WHERE KategorieID = " _
        & lngAktuelleKategorieID & " ORDER BY Artikelname"
End Sub
```

Listing 3: Aktualisierung des ersten Kombinationsfeldes

```
Private Sub SteuerelementeAktualisieren(lngAktuelleKategorieID As Long)
    Me!cboZielkategorieID.RowSource = "SELECT KategorieID, Kategoriename FROM tblKategorien WHERE KategorieID NOT IN (" _
        & lngAktuelleKategorieID & ") ORDER BY Kategoriename"
    Me!cboZielkategorieID = Me!cboZielkategorieID.ItemData(0)
    Me!lstZugeordneteArtikel.RowSource = "SELECT ArtikelID, Artikelname FROM tblArtikel WHERE KategorieID = " _
        & lngAktuelleKategorieID & " ORDER BY Artikelname"
End Sub
```

Listing 4: Ausgelagerte Anweisungen zur Aktualisierung der Steuerelemente

im Kombinationsfeld **cboAktuelleKategorieID** angezeigten Kategorie gehören (siehe Listing 3).

Bevor wir weitergehen, prüfen wir, ob wir nicht schon dieser Stelle eine Vereinfachung des Codes vornehmen können. Das ist in der Tat der Fall: Die drei hinteren Zeilen der Prozeduren **Form_Load** und **cboAktuelleKategorie_AfterUpdate** unterscheiden sich nur durch einen Parameter.

Also vereinfachen wir diesen Code, indem wir die drei Zeilen in eine neue Prozedur packen, diese mit einem Parameter versehen und einen Aufruf dieser Prozedur in die ursprünglichen Routinen einbauen. Die Prozedur **SteuerelementeAktualisieren** sieht nun wie in Listing 4 aus. Damit können wir die Prozedur **Form_Load** wie folgt anpassen:

```
Private Sub Form_Load()
    Me!cboAktuelleKategorieID = Me.OpenArgs
    SteuerelementeAktualisieren Me.OpenArgs
End Sub
```

Und auch die Prozedur **cboAktuelleKategorieID_AfterUpdate** können wir erheblich schlanker machen:

```
Private Sub cboAktuelleKategorieID_AfterUpdate()
    Dim lngAktuelleKategorieID As Long
    lngAktuelleKategorieID = Me!cboAktuelleKategorieID
    SteuerelementeAktualisieren lngAktuelleKategorieID
End Sub
```

Auswahl der zu übertragenden Artikel

Die der Kategorie im Kombinationsfeld **cboAktuelleKategorieID** zugewiesenen Artikel sollen vielleicht nicht alle in die gleiche neue Kategorie übertragen werden, sondern in verschiedene Kategorien. Deshalb wollen wir dem Benutzer die Mehrfachauswahl im Listenfeld ermöglichen und stellen dazu die Eigenschaft **Mehrfachauswahl** auf **Erweitert** ein.

Dies sorgt dafür, dass er die Einträge im Listenfeld nun etwa wie in der Dateiliste im Windows Explorer selektieren kann – also bei gedrückter Umschalttaste Bereiche auswählen oder bei gedrückter **Strg**-Taste einzelne Einträge aus- oder abwählen (siehe Bild 9).

Damit haben wir alle Vorbereitungen getroffen, um endlich die Funktion für die Schaltflächen mit der Beschriftung **Artikel übertragen** und **Alle übertragen** zu definieren.

Mehrere Datensätze im Register-Steuerelement

Wenn Sie etwa eine Kundendatenbank nutzen und während eines Tages Aktionen rund um die Pflege mehrerer Kunden ausführen, möchten Sie diese vielleicht nicht direkt nach dem Bearbeiten wieder schließen, sondern gegebenenfalls noch geöffnet lassen, weil noch zusätzliche Schritte nötig sind. Dieser Artikel zeigt, wie Sie Kunden in einem Hauptformular aus einer Liste mit Suchfunktion auswählen und die einzelnen Kundendatensätze dann in der Detailansicht in den Unterformularen eines Registersteuerelements anzeigen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1803_MehrereDatensaetzeImRegister.accdb**.

Listenfeld mit Schnellsuche

Der oberen Teil unserer kleinen Beispiellösung soll ein Listenfeld enthalten, dass die Datensätze der Tabelle tblKunden liefert. Darüber legen wir ein Textfeld an, mit dem wir schnell nach den Namen der Kunden suchen und die Datensätze im Listenfeld dann filtern können – direkt bei Eingabe der einzelnen Buchstaben.

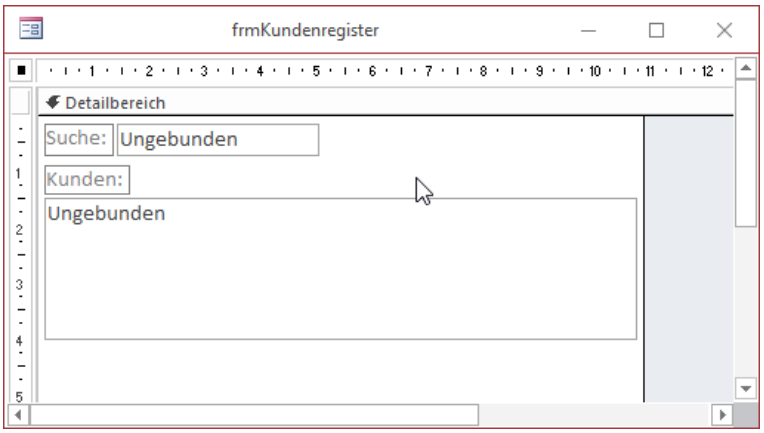


Bild 1: Suchfeld und Listenfeld zur Ausgabe der Suchergebnisse

Dazu fügen wir dem Formular ganz oben ein Textfeld namens **txtSuche** hinzu. Darunter legen wir ein Listenfeld namens **lstKunden** an (siehe Bild 1). Damit die Größe des Listenfeldes beim Ändern der Größe des Formulars angepasst wird, stellen wir seine Eigenschaften **Horizontaler Anker** und **Vertikaler Anker** auf **Beide** ein. Für das Formular legen wir für die Eigenschaften **Datensatzmarkierer**, **Navigationsschaltflächen**, **Bildlaufleisten** und **Trennlinien** den Wert **Nein** und für **Automatisch zentrieren** den Wert **Ja** fest.

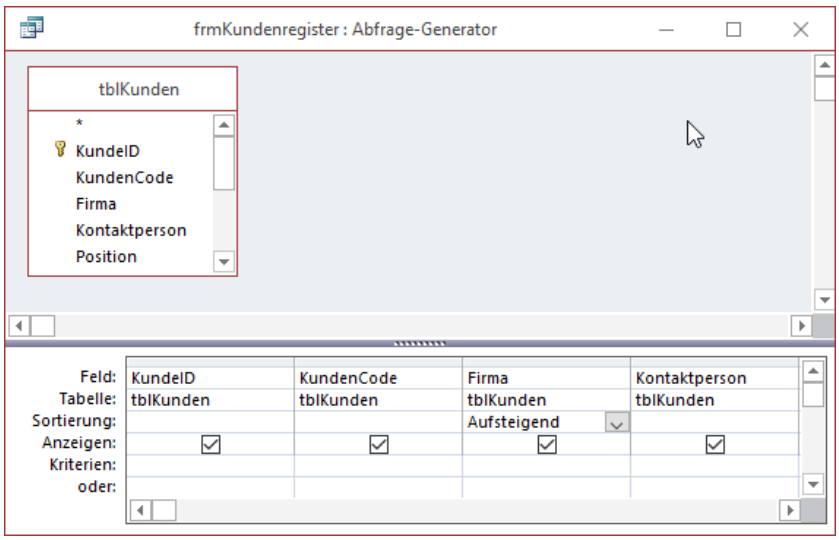


Bild 2: Datensatzherkunft des Listenfeldes

Die Datensatzherkunft des Listenfeldes stellen wir auf die Abfrage aus Bild 2 ein. Diese liefert alle Datensätze der Tabelle **tblKunden**, allerdings nur die Felder **KundeID**, **KundenCode**, **Firma** und **Kontaktperson**. Die Sortierung erfolgt aufsteigend nach dem Feld **Firma**. Damit das Listenfeld nur den Inhalt ab den zweiten Feld der Datensatzherkunft anzeigt, stellen wir

```
Private Sub txtSuche_Change()
    Dim strSuche As String
    Dim strSQL As String
    Dim strKriterium As String
    strSuche = Me!txtSuche.Text
    If Len(strSuche) > 0 Then
        strKriterium = "WHERE Kundencode LIKE '*" & strSuche & "*' OR Firma LIKE '*" & strSuche _
            & "*' OR Kontaktperson LIKE '*" & strSuche & "*'"
    End If
    strSQL = "SELECT KundeID, KundenCode, Firma, Kontaktperson FROM tblKunden " & strKriterium & "ORDER BY Firma"
    Me!lstKunden.RowSource = strSQL
End Sub
```

Listing 1: Prozedur zum Durchsuchen des Listenfeldes

die Eigenschaft **Spaltenanzahl** auf **4** und die Eigenschaft **Spaltenbreiten** auf **0cm;2cm;5cm;5cm** ein.

Dann hinterlegen wir für die Ereigniseigenschaft **Bei Änderung** des Textfeldes **txtSuche** die Ereignisprozedur aus Listing 1. Die Prozedur trägt zunächst den aktuell im Textfeld **txtSuche** angezeigten Text in die Variable **strSuche** ein. Dann prüft sie, ob die Variable einen Suchbegriff mit einer Länge von mindestens einem Zeichen enthält. Ist das der Fall, setzt die Prozedur in der Variablen **strKriterium** einen Ausdruck zusammen, der die drei Felder **Kundencode**, **Firma** und **Kontaktperson** mit dem Ausdruck aus **strSuche** vergleicht und die einzelnen Kriterien mit dem **OR**-Operator verbindet. Dieses Kriterium tragen wir dann zusammen mit einem SQL-Ausdruck, der etwa dem Inhalt der ursprünglich als Datensatzherkunft für das Listenfeld verwendeten Abfrage entspricht, für die Eigenschaft **RowSource** des Listenfeldes **lstKunden** ein.

Wenn der Benutzer beispielsweise nur den Buchstaben **A** in das Textfeld einträgt, erhält das Listenfeld die folgende Datensatzherkunft:

```
SELECT KundeID, KundenCode, Firma, Kontaktperson
FROM tblKunden
WHERE Kundencode LIKE '*A*' OR Firma LIKE '*A*'
OR Kontaktperson LIKE '*A*'ORDER BY Firma
```

Das Zwischenergebnis sieht nun etwa wie in Bild 3 aus. Hier haben wir den Suchbegriff **Restauran** eingegeben, was alle Einträge liefert, die diesen Ausdruck in einem der drei Felder **Kundencode**, **Firma** oder **Kontaktperson** enthält.

Bild 3: Beispiel für die Suche nach bestimmten Kunden

Registersteuerelement hinzufügen

Nun wollen wir das Registersteuerelement hinzufügen, das die per Doppelklick ausgewählten Kunden auf jeweils einer Registerseite anzeigt. Dazu legen wir zuerst das Registersteuerelement wie in Bild 4 an und geben ihm den Namen **regKunden**.

Nun verhält es sich unter Access so, dass wir zur Laufzeit keine Steuerelemente zu einem Formular hinzu-

fügen können. Dies ist nur in der Entwurfsansicht möglich.

Es ist allerdings durchaus möglich, zur Laufzeit Registerseiten hinzuzufügen oder zu entfernen. Leider hilft uns das nicht weiter, wenn wir jedem Register ein Unterformular hinzufügen wollen, welches das Detailformular mit den Kundendaten enthält. Oder doch? Wie wir dies nutzen können, schauen wir uns später noch an.

Gewünschte Anzahl der Register

Da wir Unterformulare nur zur Entwurfszeit zum Formular hinzufügen können, müssen wir uns überlegen, wieviele Kunden das Registeuersteuerelement gleichzeitig anzeigen können soll. Wir wollen uns auf die Zahl zehn beschränken. In mehr als zehn Kundendatensätzen wollen Sie vermutlich nicht gleichzeitig arbeiten. Also fügen wir dem Registersteuerelement manuell so viele Registerseiten hinzu, dass wir zehn Seiten erhalten. Dies erledigen Sie über den Kontextmenü-Eintrag **Seiten einfügen** des Registersteuerelements in der Entwurfsansicht des Formulars (siehe Bild 5).

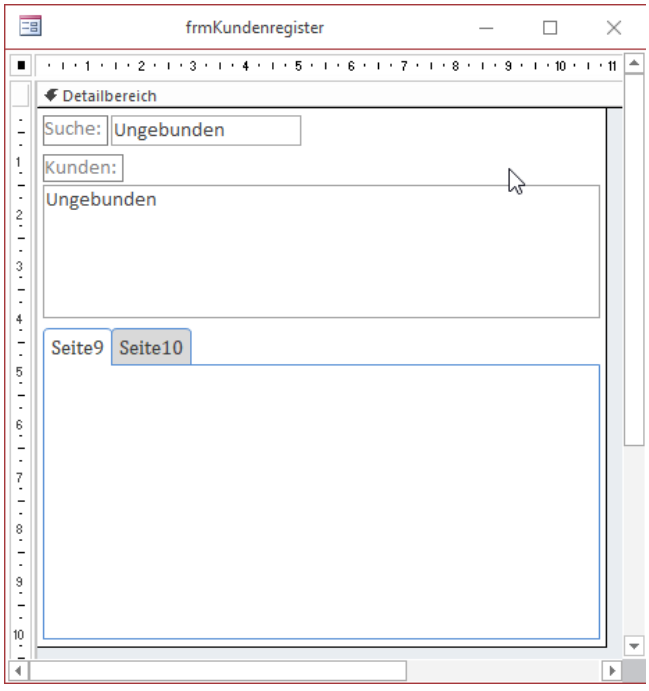


Bild 4: Das Formular mit dem Registersteuerelement für die Anzeige der Kunden

Hier wird schnell deutlich, dass sich das Registersteuerelement den für die Seitenüberschriften benötigen Platz schnell nimmt und das Steuerelement sowie das Formular verbreitert. Das können wir ändern, indem wir die Eigenschaft **Mehrere Zeilen** des Registersteuerelements auf **Ja** einstellen. Allerdings wirkt sich diese Änderung offensichtlich nicht aus, wenn wir in der Entwurfsansicht arbeiten – die zweizeilige Anzeige wird nicht aktiviert, stattdessen wird das Steuerelement immer breiter, je mehr Registerseiten wir hinzufügen. Also beschränken wir die Breite des Registersteuerelement zunächst dadurch, dass wir die Beschriftungen der Registerseiten, die wir später ohnehin an den Kundendatensatz anpassen, auf einfache Zahlen von **1** bis **10** einstellen. Außerdem stellen wir die Eigenschaft **Name** gleich auf Werte von **pge01** bis **pge10** ein.

Unterformulare hinzufügen

Danach fügen wir jeder Registerseite ein Unterformular-Steuerelement hinzu. Hier ist es besonders wichtig, dass alle Unterformular-Steuerelemente die gleiche Position haben, damit diese nachher beim Hin- und Herschalten zwischen den Seiten

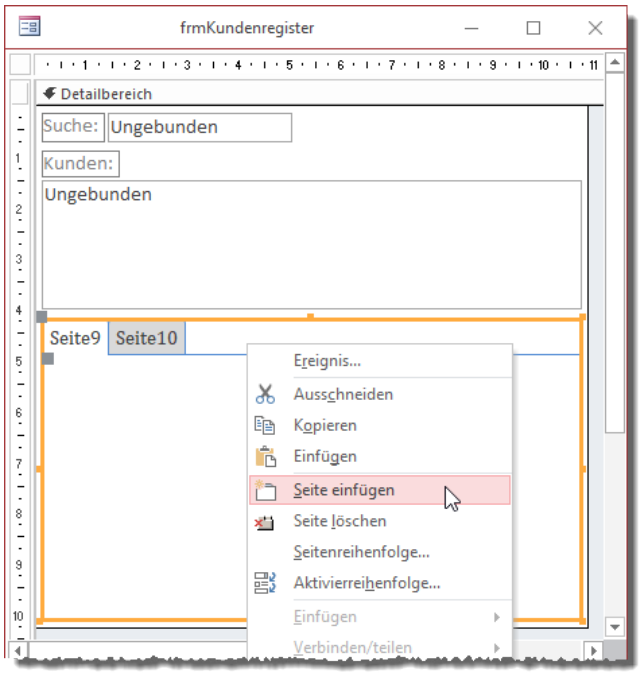


Bild 5: Manuelles Hinzufügen von Registersteuerseiten

nicht springen. Also legen wir das erste Unterformular-Steuerelement wie gewünscht an und merken uns die Werte der vier Eigenschaften **Oben**, **Links**, **Höhe** und **Breite** des ersten Unterformulars. Damit dieses aber überhaupt erst den vollständigen Platz der Registerseite einnimmt, stellen wir die Werte der vier genannten Eigenschaften auf die entsprechenden Werte der Registerseite ein.

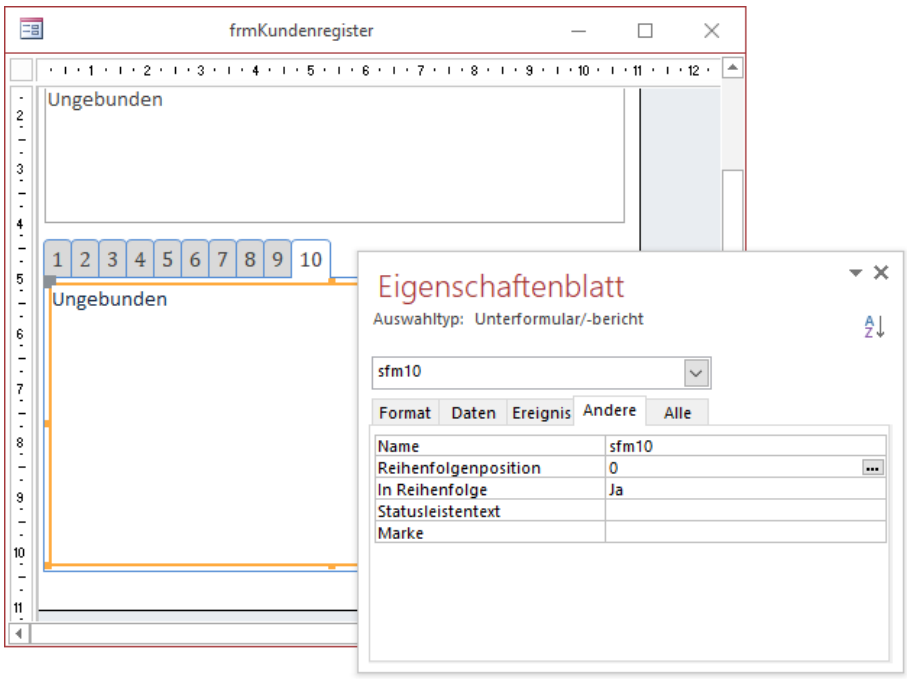


Bild 6: Die Registerseiten sind vollständig mit Unterformularen gefüllt.

Wenn wir dann das Unterformular-Steuerelement der ersten Registerseite kopieren und in die zweite Registerseite einfügen, landet dieses leider wieder an einer etwas anderen Stelle etwas weiter unten und etwas weiter links. Dadurch wird auch die Registerseite etwas größer und breiter. Diesmal brauchen wir allerdings nur die Eigenschaften **Oben** und **Links** einzustellen, damit das Formular die gleiche Position wie das auf der vorherigen Registerseite einnimmt. Am schnellsten kommen wir zum Ziel, wenn wir das Unterformular-Steuerelement mit angepasster Höhe und Breite von den ersten in die folgenden Registerseiten einfügen und dann per Copy and Paste auch die Werte der Eigenschaften **Oben** und **Links** übertragen. Außerdem stellen wir noch die Namen der Unterformular-Steuerelemente auf **sfm01** bis **sfm10** ein.

Zusätzlich stellen wir noch die Eigenschaft **Rahmenart** der Unterformular-Steuerelemente auf den Wert **Transparent** ein.

Das Ergebnis sieht dann zunächst wie in Bild 6 aus.

Registerseiten ausblenden

Nun zeigt das Registersteuerelement nicht gleich beim Öffnen des Formulars zehn Kunden-Datensätze an, sondern erst einmal keinen. Deshalb blenden wir die zehn Seiten auch zunächst einmal aus, indem wir ihre Eigenschaft **Sichtbar** auf den Wert **Nein** einstellen. Das erledigen wir allerdings nicht zu Fuß, sondern fügen der Prozedur, die durch das Ereignis **Beim Laden** des Formulars ausgelöst wird, die folgenden Zeilen hinzu:

```
Private Sub Form_Load()
    Dim i As Integer
    Me!txtSuche.SetFocus
    For i = 1 To 10
        Me!regKunden.Pages(i - 1).Visible = False
    Next i
End Sub
```

Dadurch ist der Bereich, in dem das Formular normalerweise mindestens eine Registerseite anzeigen sollte, nach dem Öffnen des Formulars zunächst leer.

Kunden anzeigen

Das wollen ändern, nachdem der Benutzer per Doppelklick auf einen der Einträge im Listenfeld den ersten anzuzeigenden Kunden ausgewählt hat. Zuvor benötigen wir jedoch noch das Unterformular, das wir

den Unterformular-Steuerelementen auf den einzelnen Registerseiten zuordnen wollen. Dieses Formular statten wir mit der Tabelle **tblKunden** als Datenherkunft aus und fügen ihm alle Felder dieser Tabelle hinzu. Das Unterformular sieht dann wie in Bild 7 aus.

Bild 7: Unterformular mit den Kundendetails

Kundendetails im Register einblenden

Nun wollen wir testweise erst einmal die Daten des aktuell doppelt im Listenfeld angeklickten Kunden im Unterformular auf der ersten Seite des Registersteuerelements anzeigen. Dazu hinterlegen wir für die Ereigniseigenschaft **Beim Doppelklicken** des Listenfeldes die Ereignisprozedur aus Listing 2.

Diese stellt zunächst mit der Eigenschaft **SourceObject** das auf der ersten Registerseite mit dem Unterformular-Steuerelement anzuzeigende Formular auf **sfmKundendetails** ein. Die Eigenschaft **RecordSource** des **Form**-Objekts des Unterformular-Steuerelements füllen wir mit einer Abfrage, welche den aktuell im Listenfeld markierten Kunden liefert.

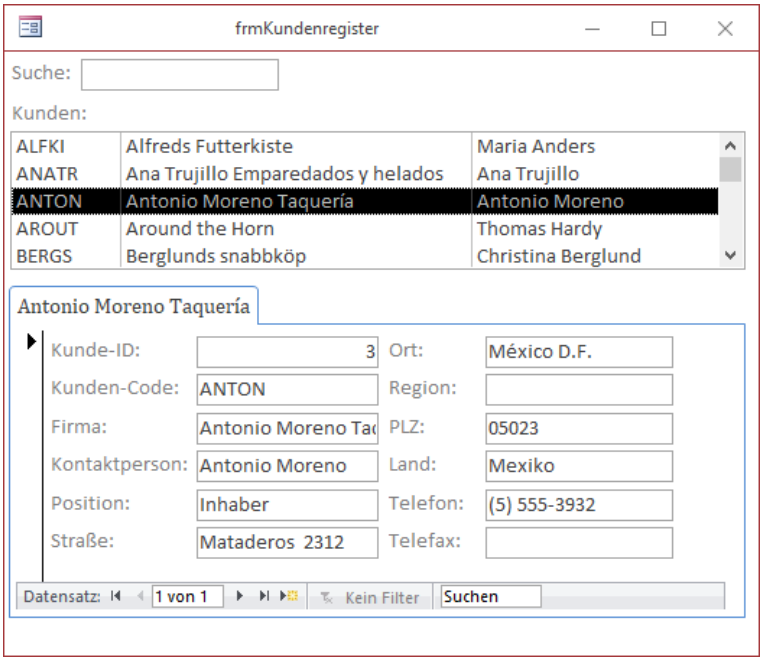


Bild 8: Anzeige eines ersten Datensatzes auf der ersten Registerseite

Dann machen wir die erste Registerseite durch Einstellen der Eigenschaft **Visible** für **Pages(0)** auf den Wert **True** sichtbar. Gleiches erledigen wir für die Eigenschaft **Visible** des Unterformular-Steuerelements.

Schließlich legen wir als Überschrift der ersten Registerseite den Wert der zweiten Spalte des Listenfeldes fest (**Column(2)**). Dazu weisen wir diesen Wert der Eigenschaft Caption der Seite **Pages(0)** des Registersteuerelements zu.

```
Private Sub lstKunden_DblClick(Cancel As Integer)
    Me!sfm01.SourceObject = "sfmKundendetails"
    Me!sfm01.Form.RecordSource = "SELECT * FROM tblKunden WHERE KundeID = " & Me!lstKunden
    Me!regKunden.Pages(0).Visible = True
    Me!sfm01.Visible = True
    Me!regKunden.Pages(0).Caption = Me.lstKunden.Column(2)
End Sub
```

Listing 2: Anzeige eines ersten Datensatzes im Register-Steuerelement

Details aus dem Unterformular öffnen

Ein Unterformular kann einige Daten anzeigen, aber oft ist die Breite des Unterformulars begrenzt und der Benutzer muss scrollen, um die nicht sichtbaren Spalten zu lesen. Das ist nicht unbedingt komfortabel. Manchmal soll dann doch ein Detailformular die Daten zu einem Datensatz in übersichtlicher Form anzeigen. Wie aber öffnen wir das Detailformular zu einem Datensatz vom Unterformular aus? Und wie aktualisieren wir das Unterformular, wenn wir den geöffneten Datensatz im Detailformular geändert haben? All dies zeigt der vorliegende Artikel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1803_OeffnenAusUnterformular.accdb**.

Haupt- und Unterformular erstellen

Das Unterformular soll den Namen **sfmArtikel** erhalten und alle Datensätze der Tabelle **tblArtikel** anzeigen. Dazu stellen Sie die Eigenschaft **Datenherkunft** des Unterformulars auf die Tabelle **tblArtikel** ein. Wechseln Sie dann zur Anzeige der Feldliste und ziehen Sie alle Einträge dieser Liste in den Detailbereich der Entwurfsansicht des Formulars (siehe Bild 1). Die Anordnung spielt in diesem Fall kenie besondere Rolle, da wir durch Einstellen der Eigenschaft **Standardansicht** auf den Wert **Datenblatt** ohnehin dafür sorgen, dass die Datensätze im Unterformular in der Datenblatt-Ansicht erscheinen.

Schließen Sie dann das Formular und öffnen Sie ein neues, leeres Formular in der Entwurfsansicht. Dieses speichern wir unter dem Namen **frmArtikel**. Fügen Sie dann

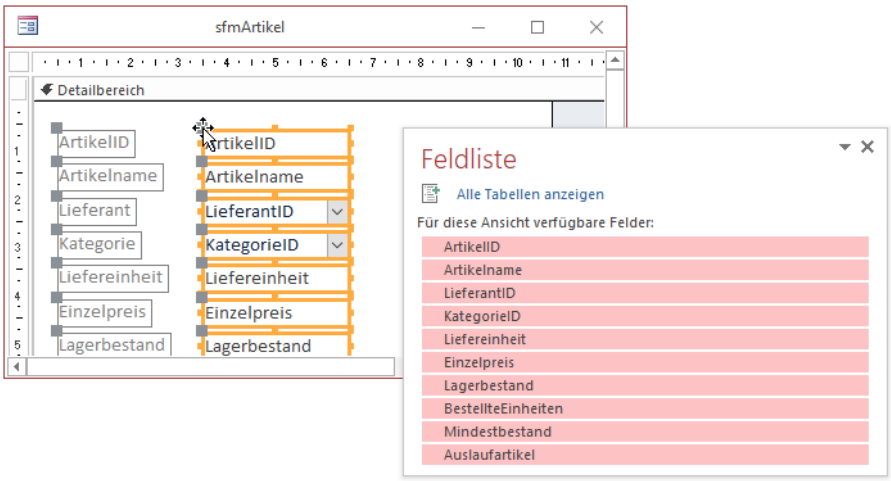


Bild 1: Formular-Entwurf des Unterformulars **sfmArtikel**

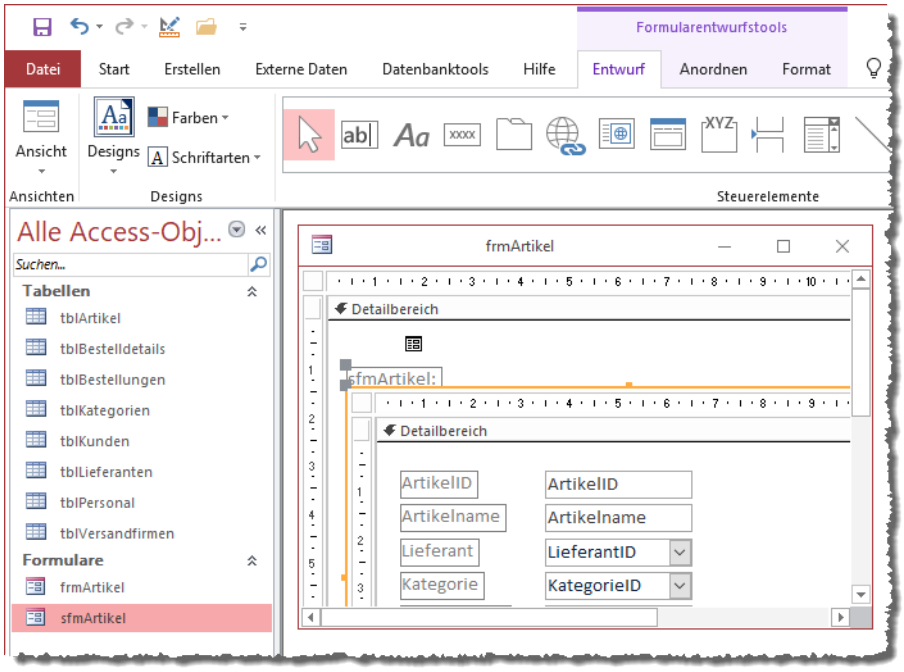


Bild 2: Hinzufügen des Unterformulars zum Hauptformular

das Unterformular hinzu, in dem Sie dieses aus dem Navigationsbereich in den Detailbereich des Formularentwurfs ziehen (siehe Bild 2).

Damit das Unterformular beim Vergrößern des Hauptformulars mitwächst, stellen wir seine beiden Eigenschaften **Horizontaler Anker** und **Vertikaler Anker** auf **Beide** ein.

Detailformular erstellen

Das Detailformular erstellen Sie ebenfalls als neues, leeres Formular, das Sie unter dem Namen **frmArtikeldetails** speichern. Fügen Sie der Datenherkunft ebenfalls die Tabelle **tblArtikel** hinzu und ziehen Sie alle Felder der Datenquelle aus der Feldliste in den Formularentwurf. Außerdem legen wir im Detailformular noch eine **OK**-Schaltfläche namens **frmOK** an (siehe Bild 3).

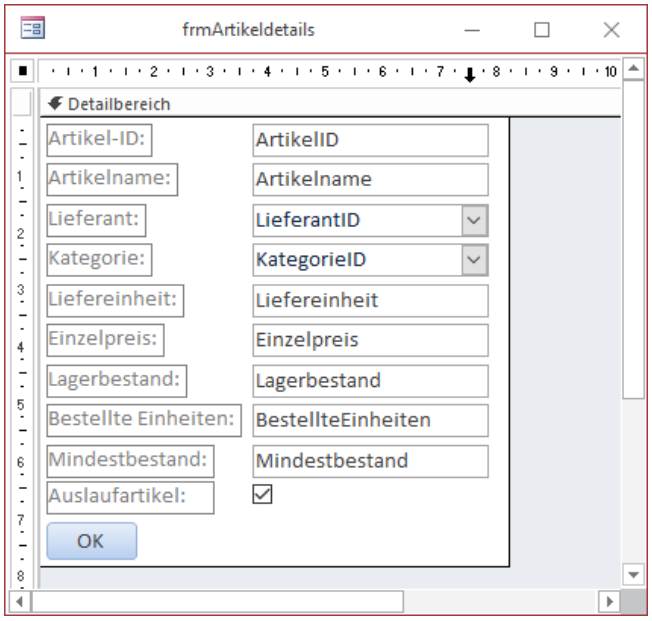


Bild 3: Entwurf des Detailformulars

Details per Schaltfläche anzeigen

Schließen Sie das Detailformular und öffnen Sie erneut das Formular mit dem Unterformular. Über dem Formular fügen wir nun eine Schaltfläche zum Anzeigen der Details des jeweils markierten Datensatzes im Unterformular ein. Sie erhält den Text **Details anzeigen** und den Namen **cmdDetailsAnzeigen**.

Außerdem stellen Sie den Wert der Eigenschaft **Beim Klicken** auf **[Ereignisprozedur]** ein und legen durch einen Klick auf die Schaltfläche mit den drei Punkten (**...**) rechts von der Eigenschaft die Ereignisprozedur **cmdDetailsAnzeigen_Click** an. Um einfach nur das Detailformular mit dem aktuell im Unterformular markierten Datensatz zu öffnen, ergänzen Sie die Prozedur wie folgt:

```
Private Sub cmdDetailsAnzeigen_Click()
    Dim lngArtikelID As Long
    lngArtikelID = _
                Me!sfmArtikel.Form!ArtikelID
    DoCmd.OpenForm "frmArtikeldetails", _
            WhereCondition:="ArtikelID = " _
                        & lngArtikelID
End Sub
```

Die Prozedur ermittelt den Wert des Primärschlüsselfeldes des aktuell im Unterformular ausgewählten Datensatzes. Das Unterformular-Steuerelement referenzieren wir mit **Me!sfmArtikel**, das darin enthaltene Unterformular mit **Me!sfmArtikel.Form** und das Feld schließlich mit **Me!sfmArtikel.Form!ArtikelID**. Diesen Wert speichern wir in der Variablen **lngArtikel**. Diese nutzen wir in der folgenden **DoCmd.OpenForm**-An-

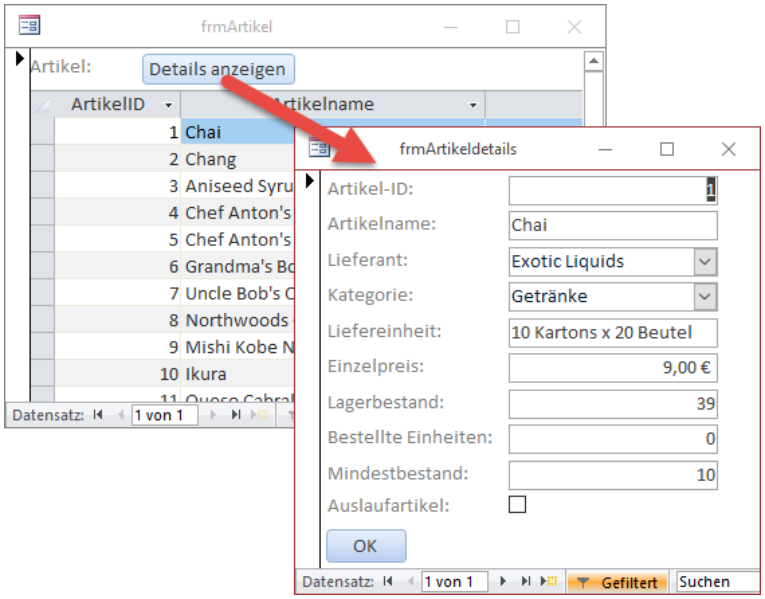


Bild 4: Aufruf des Detailformulars

Tabellerische Berichtslayouts

Wenn Sie die Felder der Datensatzquelle eines Berichts aus der Feldliste in den Bericht ziehen, landen diese dort immer in der gleichen Anordnung – nämlich untereinander und mit Beschriftungsfeldern versehen. Zum Glück bietet Access seit einiger Zeit Funktionen, mit denen Sie das Layout dieser Elemente im Berichtsentwurf schnell hin- und herschalten können. Dieser Artikel zeigt, wie das funktioniert.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1803_Berichtslayouts.accdb**.

Steuerelemente zum Bericht hinzufügen

Wir gehen davon aus, dass Sie die Daten einer einfachen Tabelle wie etwa der Tabelle **tblKunden** in einem Bericht ausgeben möchten. Daher legen wir einen neuen Bericht namens **rptKunden** an (über den Ribbonbefehl **Erstellen|Berichte|Berichtsentwurf**) und weisen diesem als Datensatzquelle die Tabelle **tblKunden** zu.

Dann zeigen wir durch einen Klick auf den Ribboneintrag **Entwurf|Tools|Vorhandene Felder hinzufügen** die Feldliste an und ziehen die dort noch zu markierenden Felder per Drag and Drop in den Detailbereich des Berichtentwurfs (siehe Bild 1).

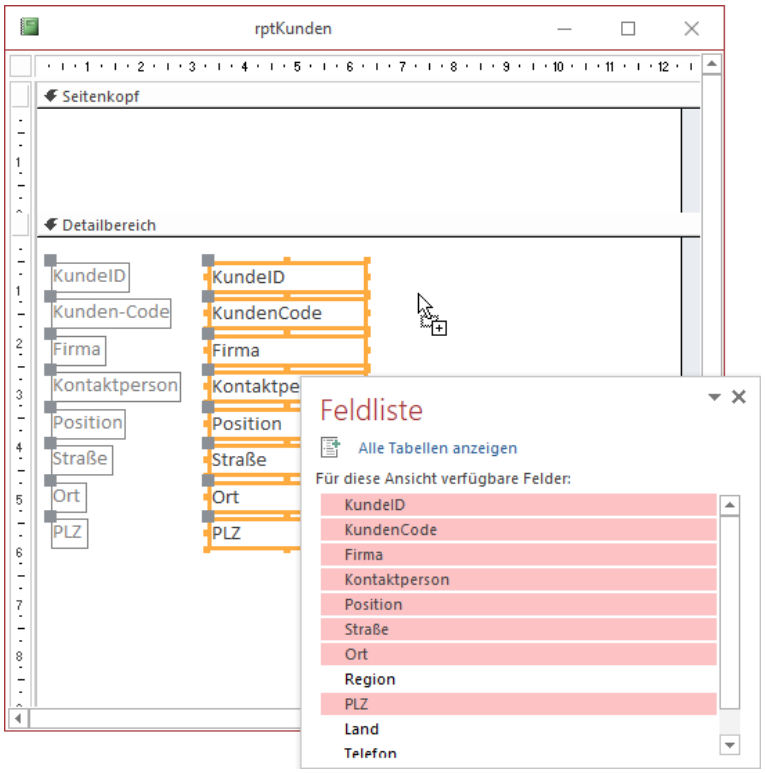


Bild 1: Einfügen der Felder in den Detailbereich des Berichts

Hier erhalten wir nun die Standardanordnung – die Felder landen untereinander im Bericht und erhalten auf der linken Seite passende Bezeichnungsfelder.

Nun möchten wir allerdings eine tabellarische Anordnung erzeugen, das heißt, wir wollen beispielsweise alle Bezeichnungsfelder am unteren Rand des Bereichs **Seitenkopf** anordnen und die Textfelder nebeneinander im Detailbereich – und zwar so, dass die Textfelder genau unter dem jeweiligen Bezeichnungsfeld landet und diese auch jeweils genau gleich breit sind. Wer schon mit Access in älteren Versionen gearbeitet hat, weiß, dass man dies früher tatsächlich von Hand erledigen musste.

Anorden mit Layout

Seit einigen Access-Versionen gibt es jedoch die Möglichkeit, dies auf einfachere Weise zu realisieren. Dazu markieren Sie zunächst alle betroffenen Steuerelemente, wobei es ausreicht, die gebundenen Steuerelemente zu berücksichtigen. Dann wählen wir im Ribbon das Tab **Anordnen** aus, wo wir in der Gruppe **Tabelle** die beiden Einträge **Gestapelt** und **Tabelle** finden (siehe Bild 2).

Hier wollen wir den Eintrag **Tabelle** anklicken, der das Resultat aus Bild 3 verursacht. Es wurde als Erstes ein Rahmen um alle Elemente gezogen, den Sie an dem Symbol oben links auf dem Formular bewegen können – momentan allerdings nur nach links und nach rechts, da die Elemente die aktuellen Berichtsbereiche nicht verlassen können. Alle Bezeichnungsfelder wurden in den Seitenkopf-Bereich verschoben, während die Textfelder im Detailbereich verblieben sind. Außerdem wurden die Elemente wie gewünscht in tabellarischer Form nebeneinander angeordnet statt wie zuvor übereinander. Schon jetzt hat uns dieser einfache Mausklick also eine Menge manueller Arbeit abgenommen.

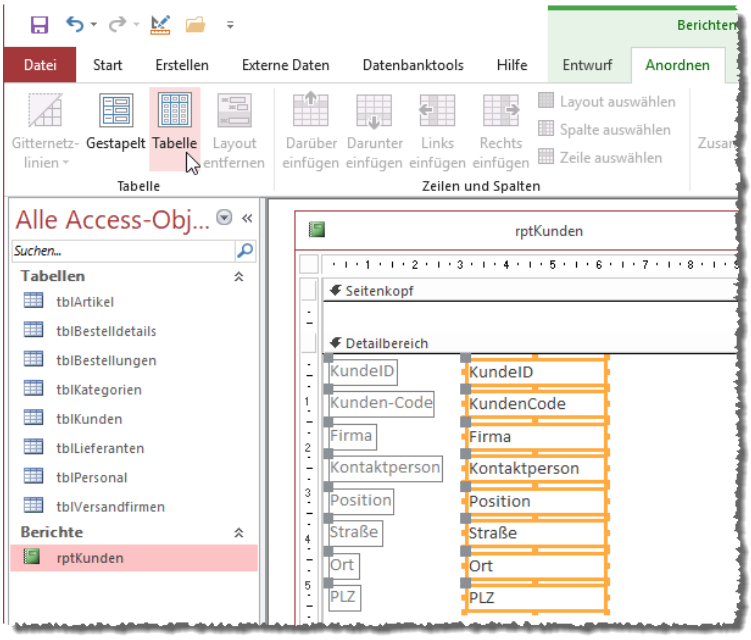


Bild 2: Markieren aller Felder und Anwenden eines Layouts

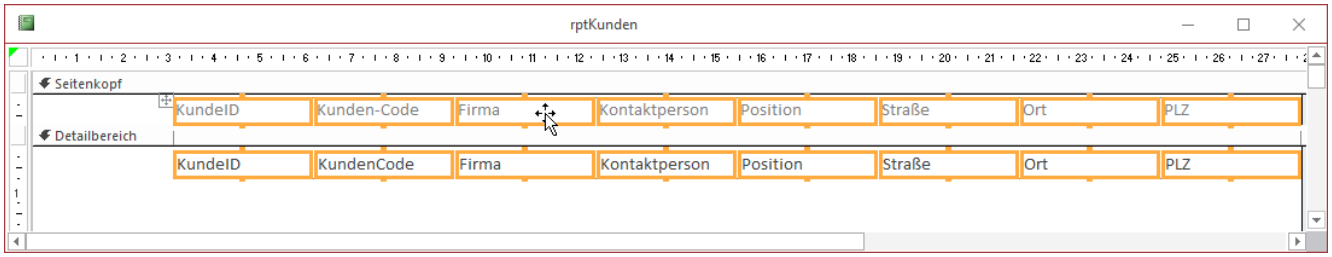


Bild 3: Anwendung des Layouts **Tabelle**

Welche Möglichkeiten ergeben sich daraus noch? Zunächst einmal können wir die Breiten von Bezeichnungsfeld und Textfeld anpassen und das jeweils andere Element wird, da es an die gleiche Spalte im Raster gebunden ist, automatisch mit angepasst.

Ein Doppelklick auf den rechten oder linken viereckigen Griff eines der Bezeichnungsfelder oder Textfelder passt die Breite automatisch an den Inhalt an. Das hilft hier natürlich nur bedingt weiter, da wir ja den Inhalt der Textfelder in der Entwurfsansicht nicht kennen – wir kommen jedoch später auf dieses Feature zurück.

Wenn die Überschriften zu hoch landen

Wenn der Seitenkopf des Berichts beim Anlegen des tabellarischen Layouts eine zu große Höhe hatte, legt Access die Spaltenüberschriften ganz oben in diesem Bereich an. Das sieht dann beispielsweise wie in Bild 4 aus. Wie können wir dieses Layout retten und die Spaltenüberschriften weiter unten platzieren, nämlich

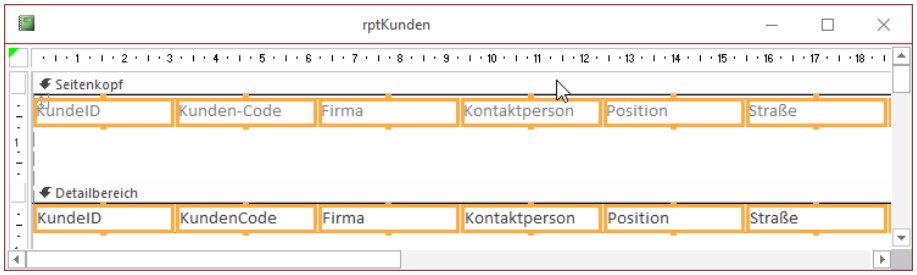


Bild 4: Missglücktes Layout

direkt am unteren Rand des Seitenkopf-Bereichs? Dazu sind nur wenige Handgriffe nötig.

Als Erstes sorgen Sie dafür, dass nur noch die obere Reihe mit den Bezeichnungsfelder markiert ist. Dazu ziehen Sie einfach mit der Maus einen Rahmen auf, der alle Bezeichnungsfelder erfasst (siehe Bild 5).

Dann greifen Sie einen der viereckigen Griffe unten an den Bezeichnungsfeldern und ziehen diesen nach unten, bis der untere Rand etwa mit dem unteren Rand des Seitenkopfes abschließt (siehe Bild 6).

Das Zwischenergebnis sieht nun wie in Bild 7 aus. Die Bezeichnungsfelder haben nun eine Höhe, die etwa der Höhe des Seitenkopf-Bereiches entspricht. Das ist natürlich nicht das Ziel – wir wollen ja die Höhe beibehalten, aber die Bezeichnungsfelder sollen am unteren Rand des Seitenkopf-Bereiches landen.

Dazu ist der letzte Schritt nötig, bei dem wir einen der oberen viereckigen Griffe mit der Maus anklicken und nach unten ziehen – soweit nach unten, bis die Bezeichnungsfelder die gewünschte Höhe erreichen. Das Ergebnis sieht dann wie in Bild 8 aus.

Von Bereich zu Bereich

Was tun, wenn Sie die Spaltenüberschriften gar nicht im Seitenkopf-Bereich haben möchten, sondern in einem Gruppenkopf-Bereich? Dazu fügen wir als Erstes eine Gruppierung hinzu, beispielsweise nach dem Anfangsbuchstaben des Kunden-Codes. Dazu fügen wir über den Bereich unter dem Berichtsentwurf eine neue Gruppierung hinzu und stellen für die Eigenschaft **Gruppieren nach** das Feld **Kunden-Code** ein und für die Eigenschaft mit dem Wert **nach Gesamtwert** in **nach dem ersten Zeichen** um. Die Option **mit Kopfzeilenbereich** dürfte bereits aktiviert sein. Es erscheint nun auch der gewünschte Gruppenkopf-Bereich, dem wir ein Textfeld mit dem Steuerelementinhalt **=Left(KundenCode;1)** hinzufügen – so zeigt dieses den ersten Buchstaben des Kundencodes für die jeweilige Gruppierung an. Wir wollen nun möglichst einfach die Spaltenüberschriften, die sich aktuell im Seitenkopf-Bereich befinden,

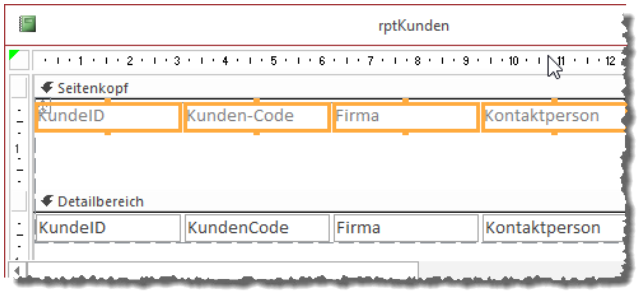


Bild 5: Markieren der Bezeichnungsfelder

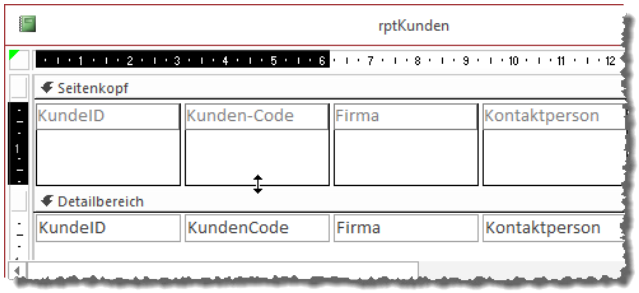


Bild 6: Vergrößern durch Ziehen nach unten

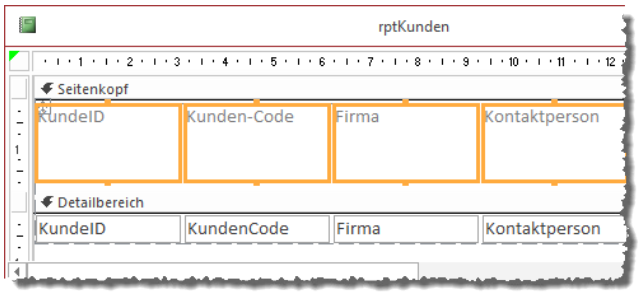


Bild 7: Zwischenergebnis: Vergrößerte Bezeichnungsfelder

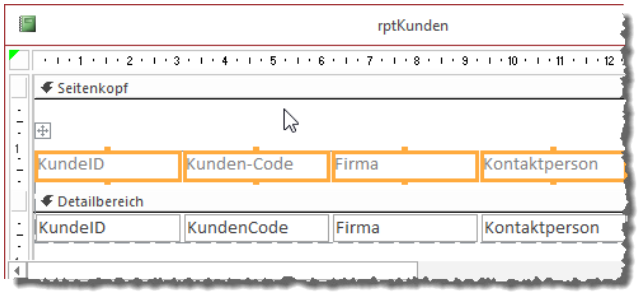


Bild 8: Herstellen der alten Höhe

in den neu angelegten Gruppenkopf-Bereich überführen (siehe Bild 9).

Das versuchen wir zu erreichen, indem wir zunächst alle Bezeichnungsfelder markieren – und zwar durch

Datenaustausch mit Excel per VBA

Die Methoden für das Importieren, Exportieren oder Verknüpfen von Daten zwischen Access und Excel haben Sie in den Artikeln »Access und Excel: Import und Export« und »Access und Excel: Verknüpfungen« bereits vor einiger Zeit kennengelernt. Hier knüpfen wir nun an und zeigen Ihnen, wie Sie per VBA die in diesen Artikeln gezeigten Methoden nutzen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1803_ExceldatenPerVBA.accdb**.

Assistenten per VBA

Bevor wir die volle Automatisierung betrachten, schauen wir uns an, wie Sie die Assistenten, die Sie in den oben genannten Artikeln kennengelernt haben, per VBA starten. Wozu das? Nun: Gegebenenfalls möchten Sie dem Benutzer die Möglichkeit geben, in einer von Ihnen gefertigten Anwendung auf die Daten aus einer Excel-Anwendung zuzugreifen und ihm dabei größtmögliche Freiheit lassen – und dabei dennoch die üblichen Elemente der Benutzeroberfläche dafür bereit zu stellen, also die eingebauten Ribbon-Befehle. Eine professionelle Access-Anwendung sollte nämlich ein von Ihnen erstelltes und auf die Anwendung zugeschnittenes Ribbon enthalten, mit dem Benutzer nur die für diese Anwendung vorgesehenen Funktionen starten darf.

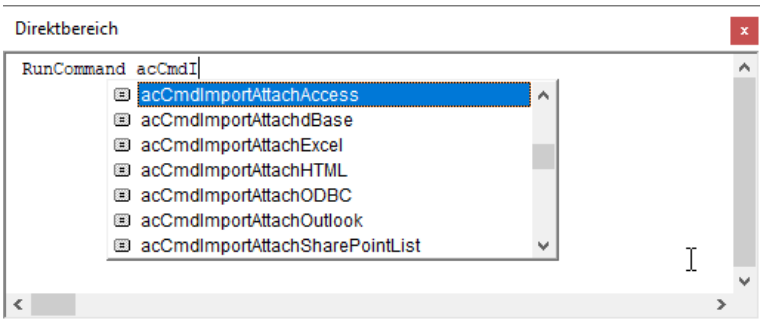


Bild 1: Auswahl der Befehle zum Importieren aus verschiedenen Quellen

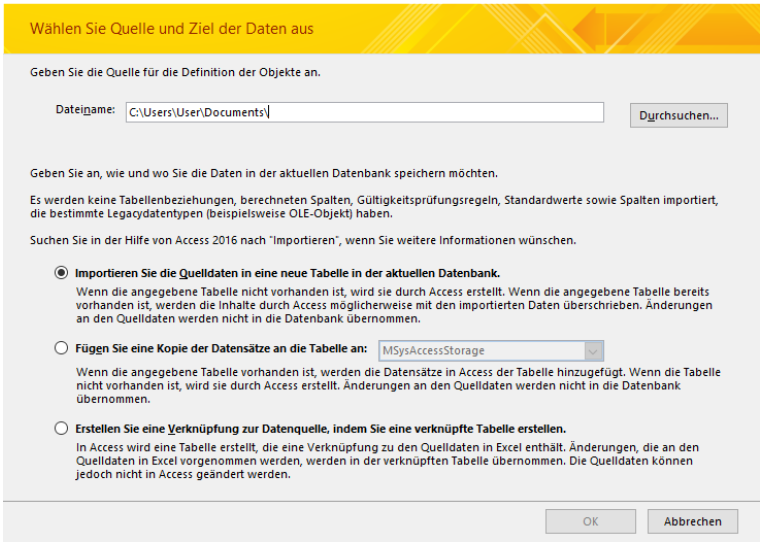


Bild 2: Assistent zum Importieren oder Verknüpfen von Excel-Daten

Die Befehle, die üblicherweise über die Benutzeroberfläche gestartet werden, finden Sie unter VBA meist als Aufrufe der Methode **RunCommand** mit verschiedenen Parametern. Dazu geben Sie beispielsweise die Anweisung RunCommand ins Direktfenster ein und tippen dann die Anfangsbuchstaben der gewünschten Aktion, in Bild 1 zum Beispiel **acCmdI...** – und erhalten dann alle Methoden, die mit dem Import von Daten zu tun haben. Uns interessiert in diesem Fall beispielsweise die Methode mit dem Parameter **acCmdImportAttachExcel**:

```
RunCommand acCmdImportAttachExcel
```

Dieser zeigt nach dem Aufruf etwa den Dialog aus Bild 2 an, mit dem Sie Daten importieren oder verknüpfen können.

Wenn Sie hingegen Daten exportieren wollen, rufen Sie diesen Befehl mit dem Parameter **acCmdExportExcel** auf:

```
RunCommand acCmdExportExcel
```

Wenn Sie diesen Befehl aufrufen, kann es allerdings sein, dass Sie die Meldung aus Bild 3 erhalten. Warum das? Das geschieht doch nie, wenn Sie den entsprechenden Ribbon-Befehl aufrufen?

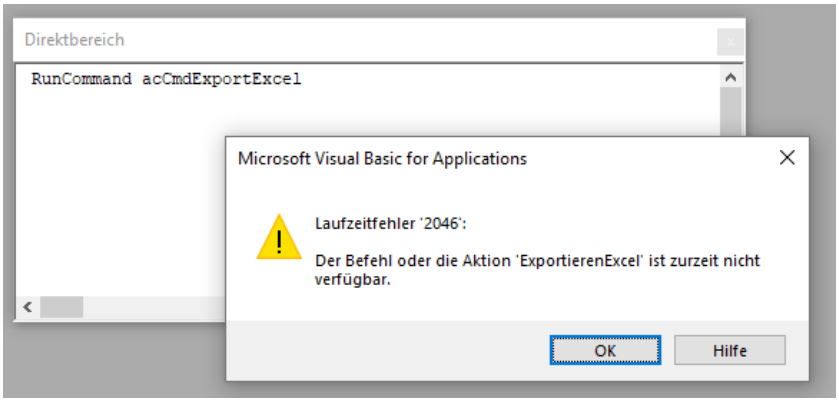


Bild 3: Fehlerhafter Aufruf des Export-Assistenten

Die Lösung liegt nahe: Werfen Sie beim Auftreten dieses Fehlers einen Blick auf das Ribbon-Tab **Externe Daten**, finden Sie in der Gruppe **Exportieren** lediglich deaktivierte Einträge vor (siehe Bild 4). In diesem Fall ist schlicht und einfach gerade kein Element im Navigationsbereich markiert, dass Sie exportieren könnten. Zur Sicherheit können Sie die Fehlerbehandlung in einer Prozedur zum Aufruf dieser Methode durch eine benutzerdefinierte Fehlerbehandlung ersetzen, die prüft, ob das Öffnen des Assistenten einen Fehler auslöst und gegebenenfalls eine entsprechende Meldung anzeigen:

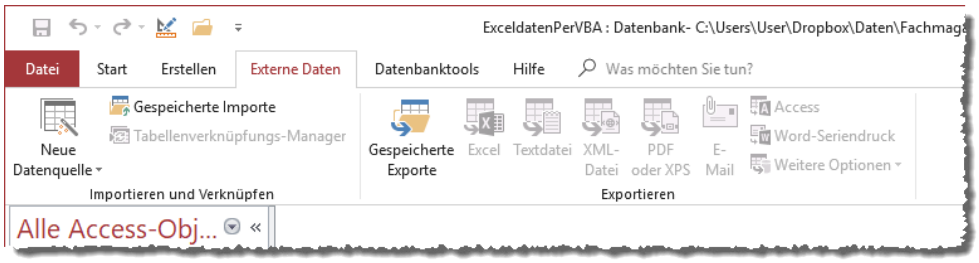


Bild 4: Die Export-Befehle sind deaktiviert.

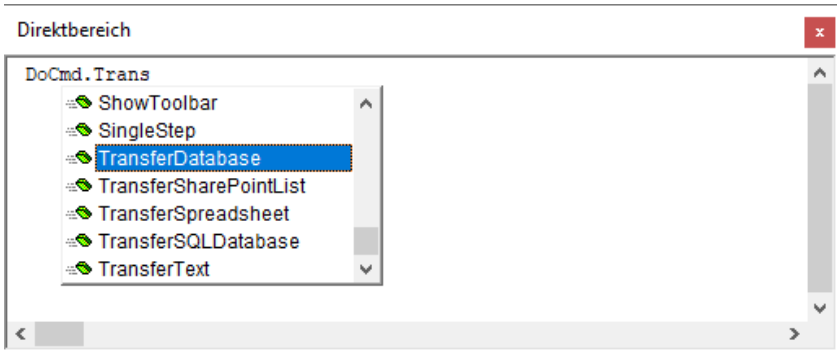


Bild 5: Die **Transfer...**-Methoden des **DoCmd**-Objekts

```
Public Sub ExcelExportieren()
    On Error Resume Next
    RunCommand acCmdExportExcel
    If Err.Number = 2046 Then
        MsgBox "Sie müssen eine Tabelle oder Abfrage ⤵
                markieren, bevor Sie den Befehl zum ⤵
                        Exportieren aufrufen."
    End If
    On Error GoTo 0
End Sub
```

Export mit Parametern

Nun wollen Sie dem Benutzer aber vermutlich möglichst viel Arbeit abnehmen und mögliche Fehlerquellen bei der Verwendung von Assistenten abnehmen. Daher gibt es noch tiefergehende Möglichkeiten, Daten zu importieren, verknüpfen oder exportieren. Diese bietet die Methode **TransferSpreadsheet** des **DoCmd**-Objekts an. Neben der Anweisung **RunCommand** ist das **DoCmd**-Objekt mit seinen verschiedenen Methoden die zweite Möglichkeit, die Befehle, die Sie sonst über die Benutzeroberfläche starten, auszuführen. TransferSpreadsheet ist dabei nur eine von verschiedenen Möglichkeiten, Daten mit anderen Datenquellen auszutauschen (siehe Bild 5).

Bild 6: Die Parameterliste der **TransferSpreadsheet**-Methode

Für den Austausch mit tabellenartig aufgebauten Dateien wie es etwa bei den Excel-Tabellen der Fall ist, nutzen wir die Methode **TransferSpreadsheet**. Diese bietet, wie wir Bild 6 entnehmen können, eine Reihe von Parametern, die wir im Folgenden erläutern:

- **TransferType**: Einer der Werte **acExport**, **acImport** oder **acLink**. **acExport** exportiert aus der angegebenen Tabelle, **acImport** importiert in die angegeben Tabelle und **acLink** erstellt eine Verknüpfung zu der angegebenen Datei.

- **SpreadsheetType**: Gibt das Format der Quell-/Zieldatei an beziehungsweise die Excel-Version, welche die Datei lesen kann/erstellt hat. Mögliche Werte: **acSpreadsheetTypeExcel3**, **acSpreadsheetTypeExcel4**, **acSpreadsheetTypeExcel5**, **acSpreadsheetTypeExcel7**, **acSpreadsheetTypeExcel8**, **acSpreadsheetTypeExcel9**, **acSpreadsheetTypeExcel12** und **acSpreadsheetTypeExcel12XML**.

- **TableName**: Name der Tabelle, aus der die Daten stammen (bei **acExport**) oder in welche die Daten geschrieben (**acImport**) beziehungsweise mit der die Daten verknüpft werden sollen (**acLink**).

- **FileName**: Name der Quell- beziehungsweise Zieldaten, je nach Import oder Export, inklusive Verzeichnis.

- **HasFieldNames**: Gibt an, ob die Quell-/Zieldatei Spaltenüberschriften in der ersten Zeile enthält oder diese beim Exportieren angelegt werden sollen.

- **Range**: Gibt beim Import den Bereich der Zellen an, die importiert werden sollen. Diesen geben Sie entweder durch die per Doppelpunkt verknüpfen Namen der Zellen links oben und unten rechts an (zum Beispiel **A1:Z26**) oder durch Angabe eines benannten Bereiches. Diese müssen Sie zuvor in der Excel-Tabelle definieren – dazu später mehr.

Beispiel für den Export nach Excel

Für ein Beispiel des Exports nach Excel haben wir eine Abfrage namens **qryArtikelMitKategorieUndLieferant** vorbereitet, welche die Daten der Tabellen **tblArtikel**, **tblKategorien** und **tblLieferanten** zusammenfasst (siehe Bild 7). Diese wollen wir nun nach Excel exportieren.

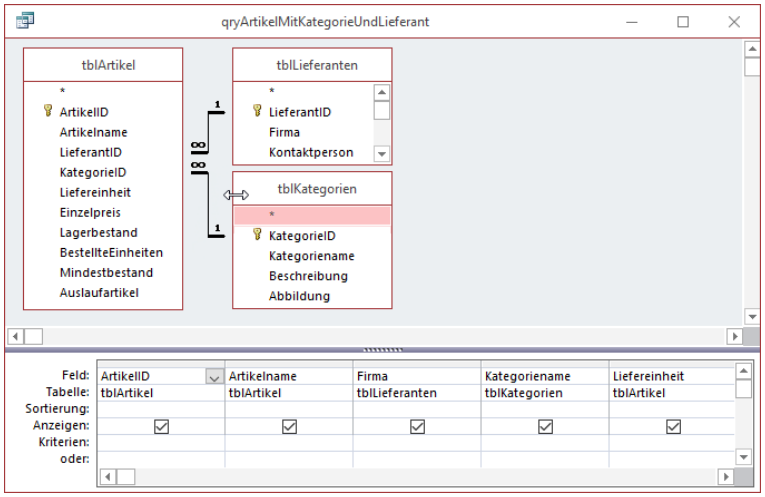


Bild 7: Abfrage mit Artikeln, Kategorie- und Lieferantenbezeichnungen

Excel fernsteuern, Teil 1: Initialisierung

Von Access aus können Sie andere Anwendungen per VBA fernsteuern, wenn diese Anwendung dies unterstützt. Das ist grundsätzlich bei den übrigen Office-Anwendungen wie Excel, Outlook, Word oder PowerPoint der Fall. In dieser Artikelserie schauen wir uns an, wie Sie eine Excel-Anwendung fernsteuern können und wie Sie Dokumente anlegen, Daten in eine Excel-Tabelle schreiben oder die Daten auslesen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **AccessUndExcelFernsteuern_1803.accdb**.

Early Binding und Late Binding

Sie können eine Anwendung wie etwas Microsoft Excel auf verschiedene Arten referenzieren, wenn Sie diese fernsteuern wollen. Die unterschiedlichen Herangehensweisen heißen Early Binding und Late Binding. Diese beiden haben wir bereits in einem früheren Artikel namens **Early Binding und Late Binding** erläutert. Im vorliegenden Beitrag wollen wir die Technik Early Binding nutzen, da dies eine komfortablere Programmierung ermöglicht.

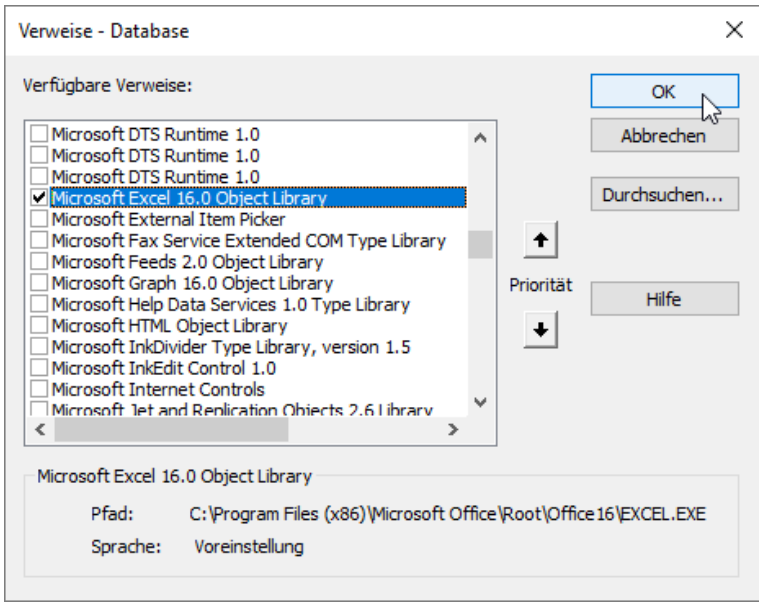


Bild 1: Verweis auf die Excel-Bibliothek

Dazu fügen wir dem VBA-Projekt der aktuellen Access-Datenbank einen Verweis auf die Bibliothek **Microsoft Excel x.0 Object Library** hinzu, wobei das **x** für die auf Ihrem Rechner installierte Version von Microsoft Excel handelt. Um dies zu erledigen, öffnen Sie den VBA-Editor – beispielsweise mit der Tastenkombination **Alt + F11** –, und rufen dann den Menüeintrag **Extras|Verweise** auf. Hier suchen Sie den gewünschten Eintrag aus und markieren diesen über das entsprechende Kontrollkästchen (siehe Bild 1).

Wenn Sie nun mit der Taste **F2** oder dem Menüeintrag **Ansicht|Objektkatalog** den Objektkatalog öffnen, finden Sie dort im oberen Kombinationsfeld den Eintrag **Excel** vor.

Wählen Sie diesen aus, erscheinen unten alle Elemente der Excel-Bibliothek (siehe Bild 2). Dies ist

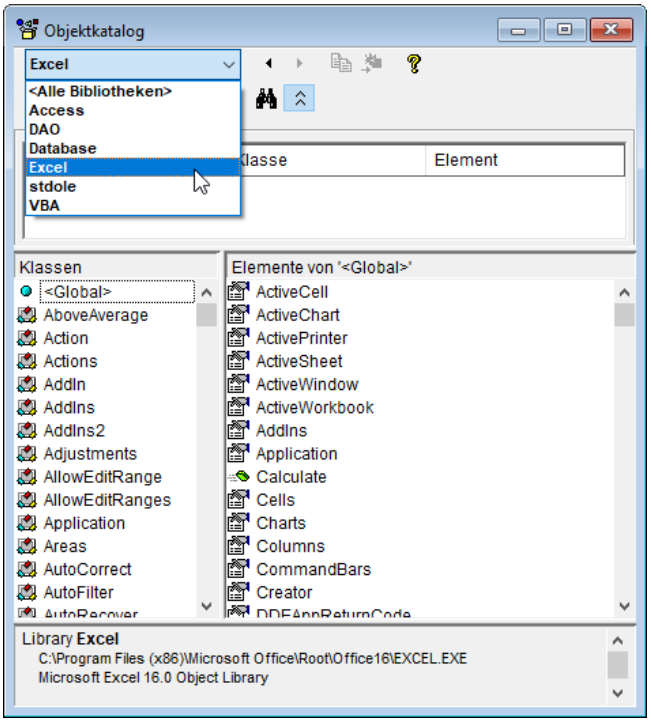


Bild 2: Die Excel-Elemente im Objektkatalog

eine gute Anlaufstelle, wenn Sie einmal nach bestimmten Klassen, Methoden, Eigenschaften oder Ereignissen suchen wollen.

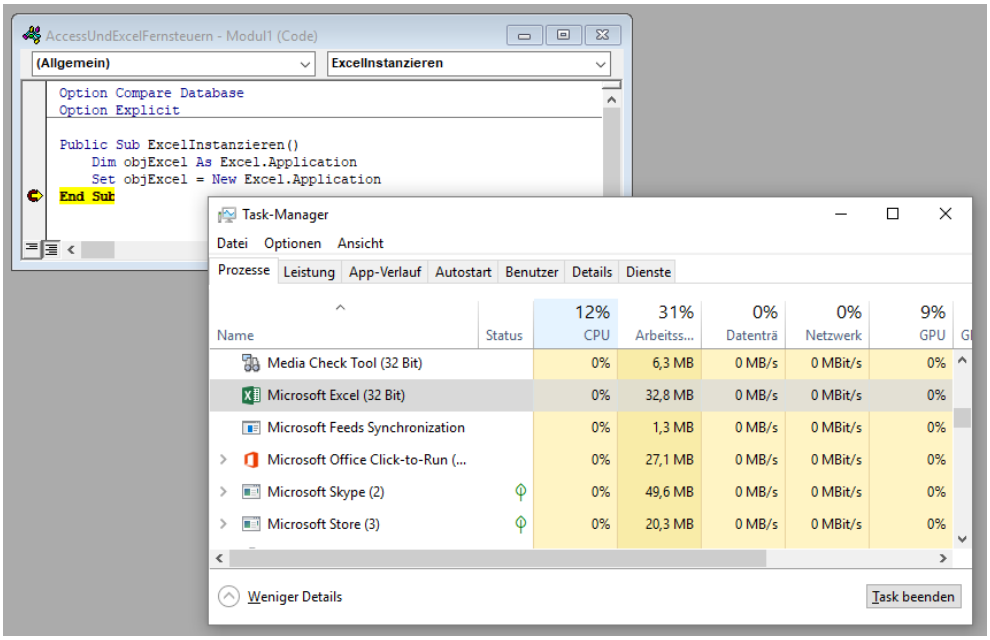


Bild 3: Die frisch gestartete Excel-Instanz im Tast-Manager

Excel-Instanz erstellen

Bevor wir auf die Daten einer Excel-Datei zugreifen können, um diese entweder einzulesen oder auch zu ändern, benötigen wir Zugriff auf eine Excel-Instanz. Das kann eine laufende Instanz sein, also eine solche, die der Benutzer bereits geöffnet hat, oder wir erstellen einfach per VBA-Code eine neue Instanz.

In beiden Fällen wollen Sie einen Verweis auf die Excel-Instanz in einer Objektvariablen speichern, um später über diese auf die Excel-Instanz zuzugreifen. Diese deklarieren wir wie folgt, zunächst in einer einfachen Prozedur, die wir in einem Standardmodul anlegen:

```
Public Sub ExcelInstanzieren()
    Dim objExcel As Excel.Application
End Sub
```

Neue Instanz erstellen

Nun wollen wir eine neue Excel-Instanz erstellen und den Verweis darauf in der Variablen **objExcel** speichern. Das geht so:

```
Public Sub ExcelInstanzieren()
    Dim objExcel As Excel.Application
    Set objExcel = New Excel.Application
End Sub
```

Dies erstellt eine neue Excel-Instanz. Diese verschwindet aber genau so schnell wieder, wie sie erstellt wurde, denn ihre Variable ist ja nur lokal in der Prozedur deklariert. Wenn die Prozedur beendet und somit die Variable zurückgesetzt wird, verschwindet auch die Excel-Instanz wieder (die Sie an dieser Stelle übrigens noch nicht einmal sehen).

Ob das wirklich geschieht, können Sie folgendermaßen testen: Setzen Sie einen Haltepunkt auf die Zeile **End Sub**. Wenn Sie die Prozedur nun mit der Taste **F5** starten, bleibt die Prozedur an dieserm Haltepunkt stehen. An dieser Stelle öffnen Sie nun den Task-Manager (einfach Task-Manager in die Suchen-Box von Windows eingeben und die Eingabetaste betätigen) und schauen, ob es dort einen Eintrag namens **Microsoft Excel (32 Bit)** gibt (gegebenenfalls auch **Microsoft Excel (64 Bit)**). Dieser Eintrag sollte etwa unter Windows 10 im Bereich **Hintergrundprozesse** zu finden sein (siehe Bild 3).

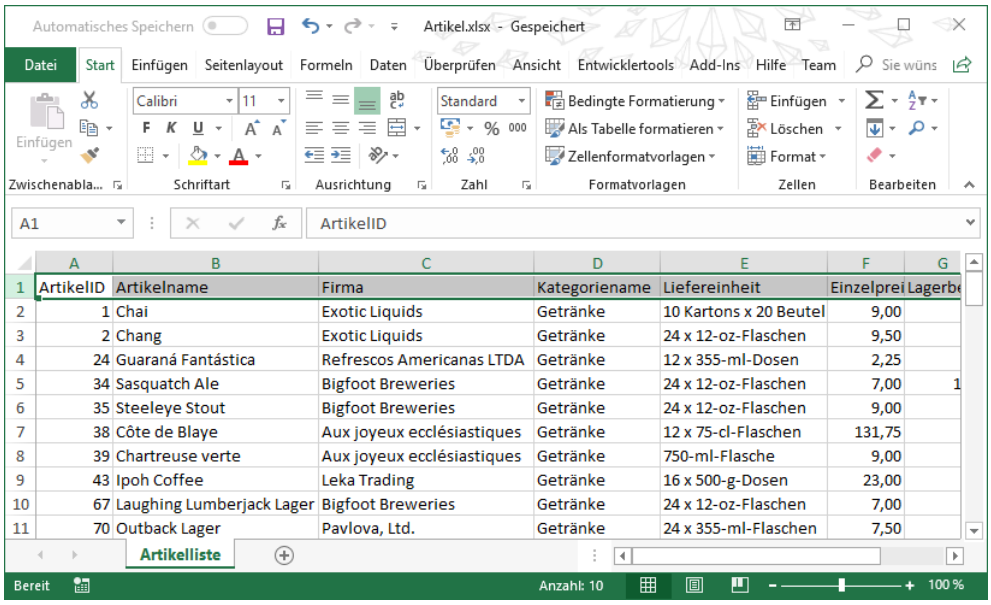
Lassen Sie die Routine nun weiterlaufen, verschwindet auch diese Instanz schnell wieder aus dem Task Manager. Sie können auch mehrere Instanzen gleichzeitig starten. Dazu benötigen Sie dann auch mehrere Objektvariablen. Der folgende Code liefert beispielsweise gleich zwei Einträge für Excel im Task-Manager:

Exceldaten per SQL und DAO

Im der Artikelreihe »Excel fernsteuern« haben Sie erfahren, wie Sie per Automation auf eine Excel-Datei zugreifen und dabei Daten aus der Tabelle lesen, schreiben oder bearbeiten können. Die Daten Zelle für Zelle zu durchlaufen ist dabei in vielen Fällen unumgänglich, vor allem, wenn Sie dabei auch noch individuelle Formatierungen unterbringen wollen. Wenn Sie jedoch schnell Daten von Excel nach Access oder umgekehrt bewegen wollen, ohne eine Excel-Instanz zu erzeugen und keine besonderen Anforderungen an die Formatierung haben, gibt es eine praktische Alternative – und zwar eine, bei der Sie sogar von Ihren vorhandenen DAO-Kenntnissen profitieren können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1803_ExcelDatenPerSQLUndDAO.accdb**.



	ArtikelID	Artikelname	Firma	Kategoriename	Liefereinheit	Einzelprei	Lagerbe
2	1	Chai	Exotic Liquids	Getränke	10 Kartons x 20 Beutel	9,00	
3	2	Chang	Exotic Liquids	Getränke	24 x 12-oz-Flaschen	9,50	
4	24	Guaraná Fantástica	Refrescos Americanas LTDA	Getränke	12 x 355-ml-Dosen	2,25	
5	34	Sasquatch Ale	Bigfoot Breweries	Getränke	24 x 12-oz-Flaschen	7,00	1
6	35	Steeleye Stout	Bigfoot Breweries	Getränke	24 x 12-oz-Flaschen	9,00	
7	38	Côte de Blaye	Aux joyeux ecclésiastiques	Getränke	12 x 75-cl-Flaschen	131,75	
8	39	Chartreuse verte	Aux joyeux ecclésiastiques	Getränke	750-ml-Flasche	9,00	
9	43	Ipoh Coffee	Leka Trading	Getränke	16 x 500-g-Dosen	23,00	
10	67	Laughing Lumberjack Lager	Bigfoot Breweries	Getränke	24 x 12-oz-Flaschen	7,00	
11	70	Outback Lager	Pavlova, Ltd.	Getränke	24 x 355-ml-Flaschen	7,50	

Bild 1: Formular-Entwurf unseres Beispielformulars

Excel-Zugriff per OpenRecordset und SELECT

Wenn Sie per DAO auf die Daten einer Datenbank zugreifen wollen, um diese entweder per VBA zu durchlaufen oder diese zu löschen, anzulegen oder zu bearbeiten, beginnen Sie in der Regel mit dem Öffnen eines Recordset-Objekts auf Basis der Tabelle oder Abfrage, in der sich die Daten befinden, die durchlaufen oder bearbeitet werden sollen. Die Syntax für die dazu verwendete **OpenRecordset**-Methode des **Database**-Objekts ist dabei hinlänglich bekannt – Sie benötigen zwei Objektvariablen namens **db** und **rst** und dann geht es los:

```
Dim db As DAO.Database
Dim rst As DAO.Recordset
Set db = CurrentDb
Set rst = db.OpenRecordset("SELECT * _
                FROM tblArtikel", dbOpenDynaset)
'... etwas mit den Datensätzen machen
```

Benötigte Excel-Informationen

Aber wie können wir das nun für den Zugriff auf die Daten einer Excel-Tabelle wie etwa der aus Bild 1 nutzen? Dazu benötigen wir nur einige wenige Informationen über die Excel-Daten, auf die wir zugreifen wollen:

- den Dateinamen inklusive Verzeichnis,
- den Namen der Tabelle und
- den Bereich, auf den wir zugreifen wollen.

```
Public Sub ExcelPerRecordset()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim strDatei As String
    Dim strTabelle As String
    Dim strBereich As String
    Dim strSQL As String
    Set db = CurrentDb
    strDatei = CurrentProject.Path & "\Artikel.xlsx"
    strTabelle = "Artikelliste"
    strBereich = "A:J"
    strSQL = "SELECT * FROM [" & strTabelle & "$" & strBereich & "] IN '" & strDatei & "'[Excel 8.0;HDR=Yes;IMEX=0;]"
    Debug.Print strSQL
    Set rst = db.OpenRecordset(strSQL)
    Debug.Print rst.Fields(0).Name, rst.Fields(1).Name
    Do While Not rst.EOF
        Debug.Print rst.Fields(0), rst.Fields(1)
        rst.MoveNext
    Loop
End Sub
```

Listing 1: Diese Prozedur erstellt ein Recordset auf Basis einer Excel-Tabelle und gibt die enthaltenen Daten aus.

Den Rest bekommen wir durch geschickte Formulierung der **SELECT**-Abfrage hin. Diese ist gar nicht so viel komplizierter aufgebaut wie eine herkömmliche SELECT-Anweisung – wenn man weiß, wie es geht. Ein Beispiel ist das folgende:

```
SELECT * FROM [Artikelliste$A:J] IN 'C:\...\Artikel.
xlsx'[Excel 8.0;HDR=No;IMEX=0;]
```

Diese Abfrage erwartet hinter dem **FROM**-Schlüsselwort den Namen der Tabelle in der Excel-Datei (hier **Artikelliste**)und davon durch ein Dollar-Zeichen (**$**) getrennt den betroffenen Bereich, hier **A:J**. Dieser Ausdruck muss in eckigen Klammern zusammengefasst werden. Danach müssen wir allerdings noch angeben, in welcher Datei sich die Daten befinden. Diese geben wir mit dem **IN**-Schlüsselwort an.

Dieses erwartet zwei Bestandteile. Als Erstes den Dateinamen in Hochkommata (**'C:\...\Artikel.xlsx'**) und als zweites in eckigen Klammern Informationen darüber, in welchem Format die gelieferten Daten vorliegen (**[Excel 8.0;HDR=No;IMEX=0;]**).

Mit **HDR** geben Sie an, ob beim Zugriff vom Vorhandensein von Spaltenüberschriften in der ersten Zeile ausgegangen werden soll. In unseren Tests machte es allerdings keinen Unterschied – wir erhielten sowohl mit **Yes** als auch mit **No** nur die gewünschten Daten zurück und nicht die Spaltenüberschriften.

Die Prozedur aus Listing 1 zeigt, wie Sie die verschiedenen Parameter wie Dateiname, Tabellenname und Bereich einzeln in Variablen erfassen und diese dann zu einer **SELECT**-Anweisung zusammenstellen.

Diese wird dann als Parameter der **OpenRecordset**-Methode verwendet. Die folgenden Anweisungen durchlaufen in einer **Do While**-Schleife alle Datensätze des angegbenen Bereichs und geben die Daten im Direktfenster aus. Das Ergebnis sehen Sie schließlich in Bild 2.

Daten mit DAO bearbeiten

Wenn wir schon mit einem Recordset auf die Daten der Excel-Tabelle zugreifen, können wir dann nicht auch gleich die DAO-Methoden **AddNew**, **Edit** oder

Update verwenden, um die enthaltenen Daten zu bearbeiten? Wir schauen uns an, ob dies funktioniert. Dazu verwenden wir eine ähnlich aufgebaute Prozedur wie die aus dem vorherigen Beispiel. Hier ergänzen wir jedoch den Ausdruck für die Variable **strSQL** ein wenig, indem wir dieser hinten noch die folgende **WHERE**-Klausel anhängen:

```
... WHERE ArtikelID = 1
```

Die Bedingung müssen Sie bei solchen Abfragen auf Basis einer Excel-Tabelle noch hinter der Angabe **IN**-Klausel mit der Quelldatei einfügen. Danach erstellen wir wieder ein Recordset auf Basis der Abfrage aus **strSQL** und referenzieren es mit der Variablen **rst**. Deren Methode **Edit** nutzen wir, um den aktuellen Datensatz zum Bearbeiten vorzubereiten. Danach können wir beispielsweise dem Feld **Artikelname** einen neuen Wert zuweisen und die Änderung dann durch Aufrufen der **Update**-Methode in der zugrunde liegenden Datenquelle speichern (siehe Listing 2).

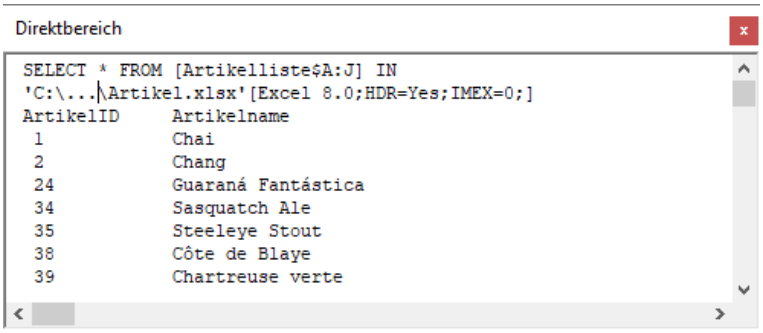


Bild 2: Ausgabe der Datensätze einer Excel-Datei per Schleife

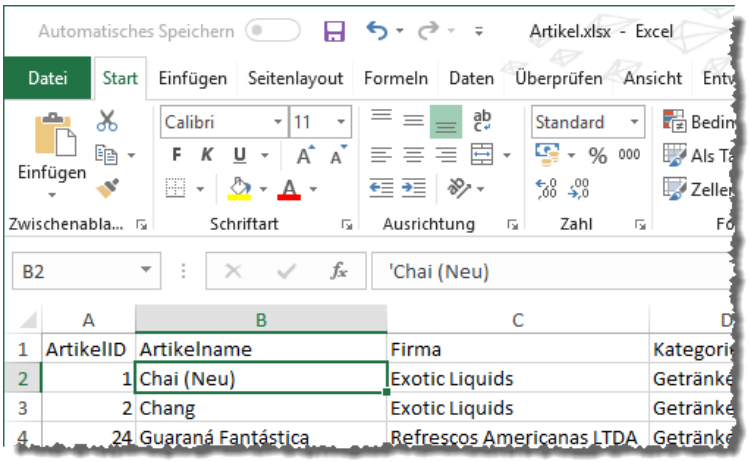


Bild 3: Geänderter Zellwert in einer Excel-Tabelle

Nach dem Ausführen dieser Prozedur finden Sie den bearbeiteten Datensatz in der Excel-Tabelle mit dem neuen Artikelnamen vor (siehe Bild 3).

Daten mit AddNew anfügen

Interessant ist auch das Hinzufügen von neuen Datensätzen zur Excel-Tabelle. Auch dies sollten wir mit den Befehlen von DAO bewerkstelligen können. Diesmal benötigen wir jedoch nicht die **Edit**-, sondern die **AddNew**-Methode. Das Resultat sieht dann wie in Listing 3 aus. Hier definieren wir die SQL-Anweisung in strSQL so, dass das Recordset keine Datensätze zurückliefert. Dazu stellen wir die Bedingung auf **1=2** ein – eine Bedingung, die für keinen Datensatz erfüllt ist.

```
Public Sub ExcelPerRecordsetBearbeiten()
    '... wie vorheriges Beispiel
    strSQL = "SELECT * FROM [" & strTabelle & "$" & strBereich & "] IN '" & strDatei & "'[Excel 8.0;HDR=Yes;IMEX=0;] WHERE ArtikelID = 1"
    Debug.Print strSQL
    Set rst = db.OpenRecordset(strSQL)
    rst.Edit
    rst!Artikelname = "Chai (Neu)"
    rst.Update
End Sub
```

Listing 2: Editieren eines Datensatzes per **Edit**/**Update**-Methode

Datensätze markieren mit bedingter Formatierung

Wenn Sie gern mit der Datenblattansicht arbeiten, ist Ihnen vermutlich schon aufgefallen, dass es keine Möglichkeit, mehrere Datensätze gleichzeitig zu markieren – außer, die Datensätze hängen zusammen. Mit der bedingten Formatierung und ein paar kleinen Tricks gelingt es uns allerdings doch, Datensätze etwa per Doppelklick zu selektieren und wieder abzuwählen. Dieser Artikel zeigt, wie es funktioniert.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1803_DatensaetzeMarkierenMitBedingterFormatierung.accdb**.

Vorbereitung

Als Beispiel wollen wir ein Unterformular nutzen, dass die Daten der Tabelle **tblKunden** in der Datenblattansicht anzeigt. Im Hauptformular wollen wir dann Schaltflächen unterbringen, mit denen wir entweder alle Einträge des Unterformulars selektieren können oder die Selektierung komplett aufheben.

Dazu legen wir zunächst das Unterformular namens **sfmKunden** an. Dieses statten wir über die Eigenschaft Datenherkunft mit der Tabelle **tblKunden** als Datenquelle aus. Ziehen Sie dann alle Felder der Datenherkunft aus der Feldliste in den Detailbereich des Formularentwurfs. Außerdem stellen wir die Eigenschaft **Standardansicht** auf **Datenblatt** ein (siehe Bild 1). Schließen Sie das Formular dann, damit wir es im nächsten Schritt als Unterformular zum Hauptformular hinzufügen können.

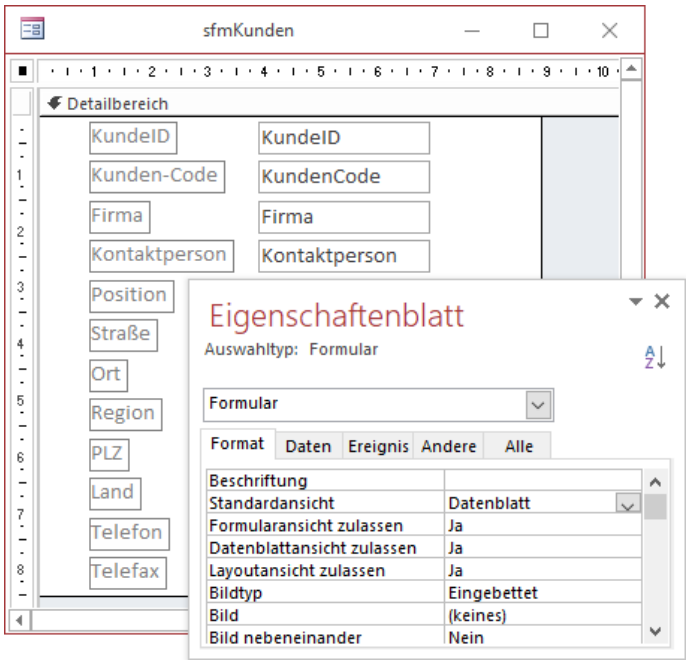


Bild 1: Entwurf des Unterformulars

Anschließend legen wir das Hauptformular namens **frmKunden** an und öffnen dieses in der Entwurfsansicht. Diesem fügen wir das Formular **sfmKunden** als Unterformular hinzu, indem wir es aus dem Navigationsbereich in den Detailbereich des Formularentwurfs ziehen. Außerdem stellen wir für das Hauptformu-

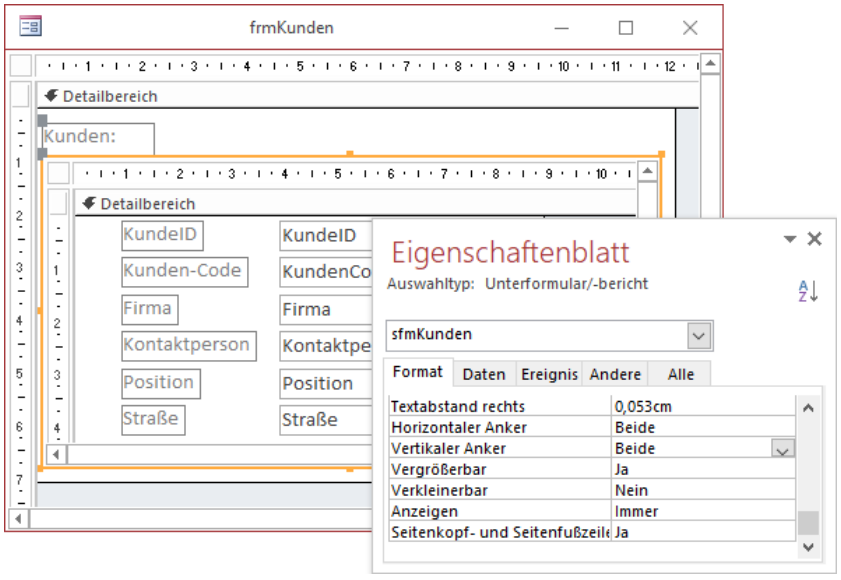


Bild 2: Entwurf des Hauptformulars

lar die Eigenschaften **Datensatzmarkierer**, **Navigationsschaltflächen**, **Trennlinien** und **Bildlaufleisten** auf den Wert **Nein** und **Automatisch zentrieren** auf **Ja** ein. Damit die Größe des Unterformulars beim Ändern der Größe des Hauptformulars automatisch angepasst wird, legen wir außerdem für die beiden Eigenschaften **Horizontaler Anker** und **Vertikaler Anker** noch den Wert **Beide** fest (siehe Bild 2).

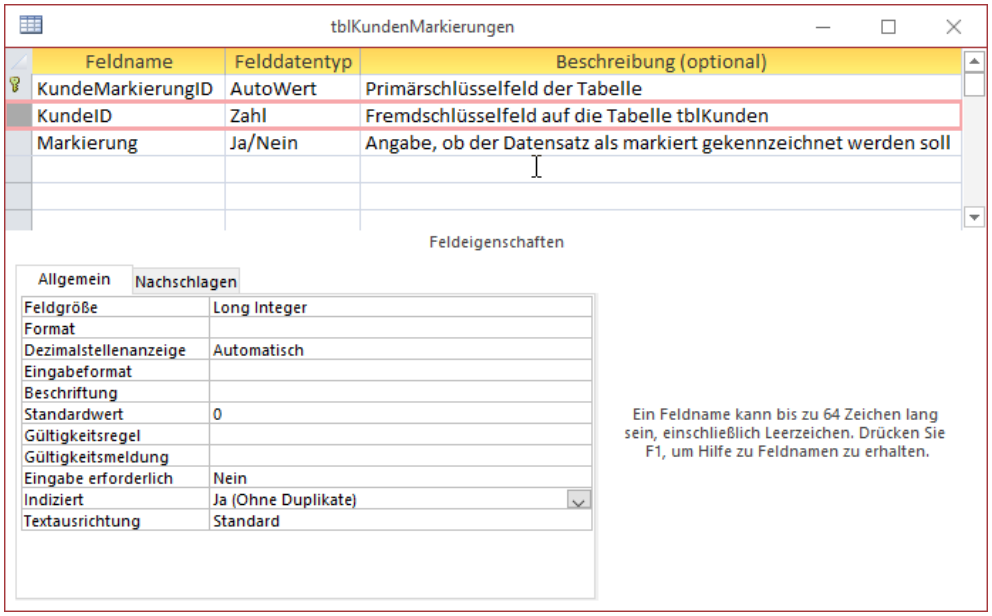


Bild 3: Entwurf der Zusatztabelle **tblKundenMarkierungen**

Feld für das temporäre Speichern als markierter Datensatz hinzufügen

Nun haben wir zwar schon das Formular aufgebaut, aber wenn wir eine bedingte Formatierung definieren wollen, benötigen wir dazu bestimmte Bedingungen, die erfüllt sein müssen. Wie aber wollen wir eine Bedingung definieren, die besagt, dass ein Datensatz gerade markiert ist oder nicht?

Dazu benötigen wir ein eigenes Feld, dass wir entweder direkt der Tabelle hinzufügen oder einer weiteren Tabelle, die wir dann per 1:1-Beziehung mit der eigentlichen Tabelle verknüpfen. Da es sein kann, dass Sie diese Technik an Tabellen anwenden wollen, die aus einer Datenbank stammen, deren Entwurf Sie nicht ändern können, wählen wir die zweite Variante. Also legen wir zunächst eine neue Tabelle namens **tblKundenMarkierungen** an.

Diese erhält drei Felder. Das erste ist das Primärschlüsselfeld dieser Tabelle, das zweite ein Fremdschlüsselfeld, welches auf den Primärschlüssel des Feldes **tblKunden** verweist und das dritte das **Ja/Nein**-Feld, mit dem wir festlegen, ob der mit dem diesem Datensatz verknüpfte Datensatz der Tabelle **tblKunden** als ausgewählt gekennzeichnet werden soll.

Wichtig für die Herstellung einer 1:1-Beziehung zwischen den Tabellen **tblKunden** und **tblKundenBeziehungen** ist, dass beide an der Beziehung beteiligten Felder einen eindeutigen Index haben. Beim Primärschlüsselfeld **KundeID** der Tabelle **tblKunden** ist das per Definitionem der Fall, beim Fremdschlüsselfeld **KundeID** der Tabelle **tblKundenMarkierungen** müssen wir nachhelfen. Dazu stellen wir die Eigenschaft **Indiziert** für dieses Feld im Entwurf der Tabelle auf den Wert **Ja (Ohne Duplikate)** ein (siehe Bild 3).

Beziehung hinzufügen

Danach wollen wir die Beziehung zwischen den beiden Tabellen **tblKunden** und **tblKundenMarkierungen** hinzufügen. Dazu öffnen wir über den Ribbonbefehl **Datenbanktools|Beziehungen|Beziehungen** das Beziehungen-Fenster. Diesem fügen wir die Tabellen **tblKunden** und **tblKundenMarkierungen** hinzu, sofern diese noch nicht sichtbar sind.

Dann erstellen wir die Beziehung zwischen den beiden Tabellen. Dabei ist es wichtig, dass Sie vom Feld **KundeID** der Tabelle **tblKunden** ausgehend auf das gleichnamige Feld der Tabelle **tblKundenMarkierungen** ziehen – andersherum erhalten Sie beim Speichern der Beziehung eine Fehlermeldung, wenn Sie

für die Beziehung referenzielle Integrität festgelegt haben.

Der Grund ist einfach: In diesem Fall müsste es für jeden Eintrag im Feld **KundeID** der Tabelle **tblKunden** nämlich einen passenden Eintrag im entsprechenden Feld der Tabelle **tblKundenMarkierungen** geben. Und das ist gerade dann nicht der Fall, wenn die Tabelle soeben erst angelegt wurde.

Also ziehen wir die Beziehung andersherum auf. Im nun erscheinenden Dialog **Beziehungen bearbeiten** erscheint dann im unteren Bereich auch gleich **1:1** als Beziehungstyp.

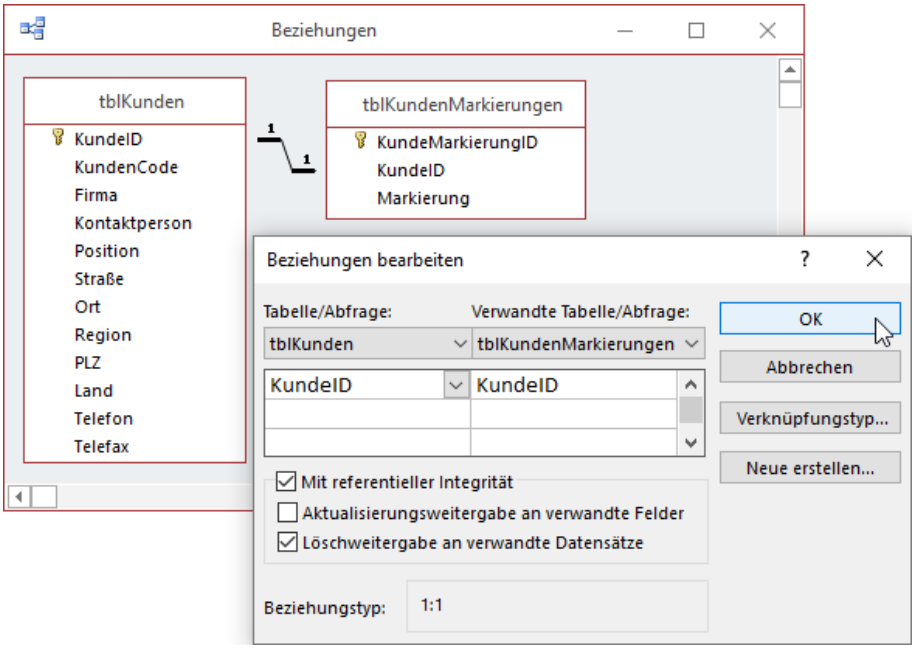


Bild 4: Beziehung zwischen den Tabellen **tblKunden** und **tblKundenMarkierungen**

Hier aktivieren Sie noch die Option **Mit referenzieller Integrität**. Sie sollten auch noch die Option **Löschweitergabe an verwandte Datensätze** aktivieren, damit die Datensätze der Tabelle **tblKundenMarkierungen**, die mit einem zu löschenden Datensatz der Tabelle **tblKunden** in Beziehung stehen, auch direkt mit gelöscht werden (siehe Bild 4).

Abfrage zum Zusammenführen der Daten erstellen

Wenn wir wollen, dass das Feld **Markiert** der Tabelle **tblKundenMarkierungen** bei den noch zu definierenden bedingten Formatierungen im Unterformular berücksichtigt wird, müssen wir dieses über die Datenherkunft verfügbar machen. Dazu erstellen wir eine neue Abfrage namens **qryKundenMarkierungen**. Dieser fügen wir die beiden Tabellen **tblKunden** und

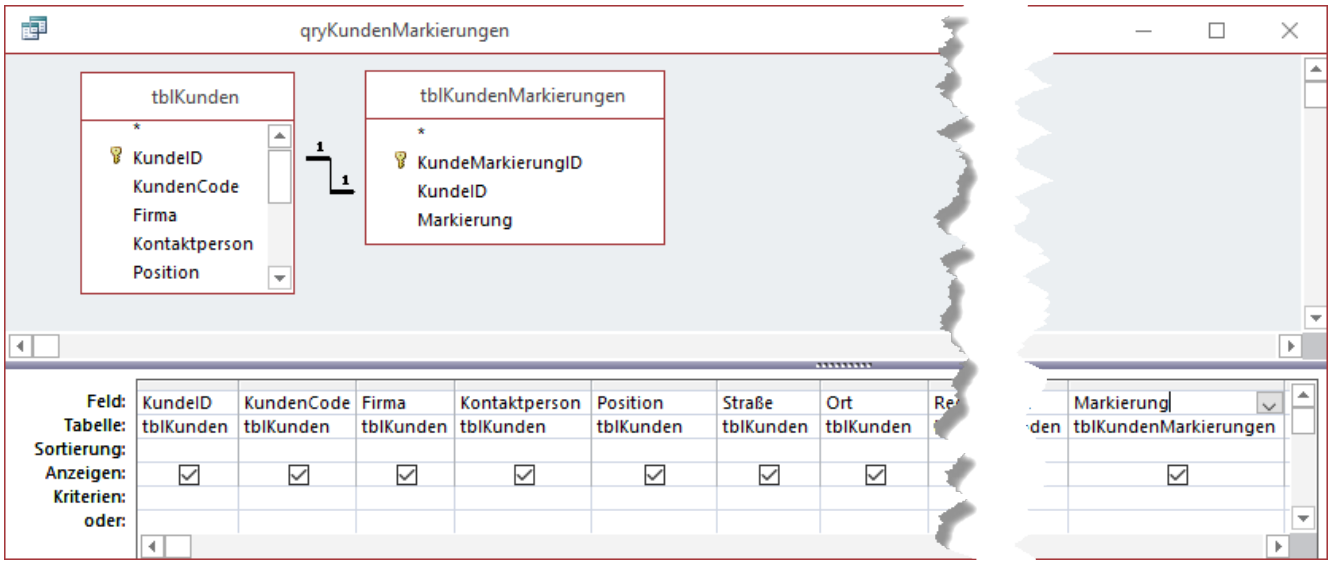


Bild 5: Abfrage mit den beiden Tabellen **tblKunden** und **tblKundenMarkierungen**

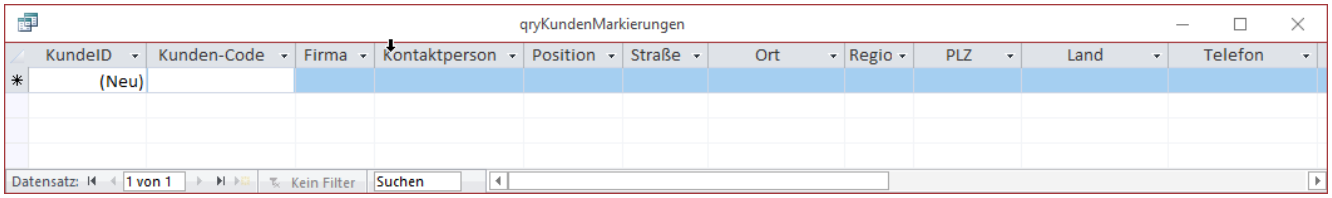


Bild 6: Die Abfrage **qryKundenMarkierungen** liefert keine Datensätze zurück.

tblKundenMarkierungen hinzu. Aus der Tabelle **tblKunden** übernehmen wir alle Felder, aus der Tabelle **tblKundenMarkierungen** fügen wir noch das Feld **Markierung** zum Entwurfsraster der Abfrage hinzu (siehe Bild 5).

Wenn wir nun in die Datenblattansicht der Abfrage wechseln, liefert diese allerdings keine Datensätze zurück (siehe Bild 6). Das ist auch kein Wunder, denn aktuell ist die Abfrage ja auch so definiert, dass Sie alle Datensätze zurückliefert, für dies es übereinstimmende Werte im Feld **KundeID** der beiden beteiligten Tabellen **tblKunden** und **tblKundenMarkierungen** gibt.

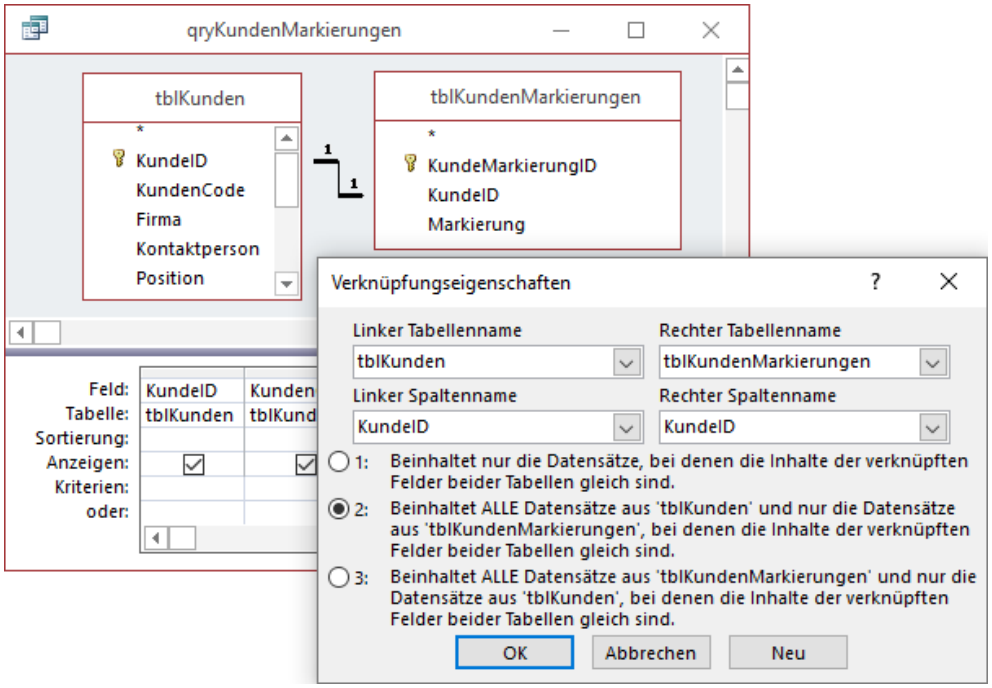


Bild 7: Finetuning der Verknüpfungseigenschaften

Diese kleine Nachlässigkeit wollen wir noch korrigieren. Dazu wechseln wir nochmals zum Entwurf der Abfrage und klicken doppelt auf den Beziehungspfeil zwischen den beiden Tabellen. Hier wählen wir nun die Option wie in Bild 7 aus. Dies sorgt dafür, dass alle Datensätze der Tabelle **tblKunden** im Abfrageergebnis erscheinen – selbst wenn die Tabelle **tblKundenMarkierungen** keinen einzigen Datensatz enthält.

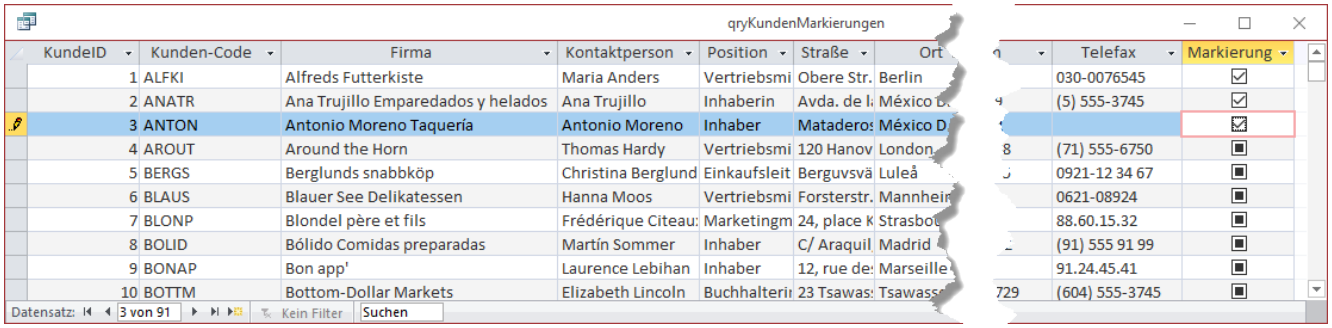


Bild 8: Setzen des Feldes **Markierung** auf den Wert **Ja**

Ein Wechsel zurück zur Datenblattansicht liefert dann auch alle Datensätze der Tabelle **tblKunden** zurück – inklusive des Feldes Markierung der Tabelle **tblKundenMarkierung**. Dieses zeigt den undefinierten Status für Kontrollkästchen an, da dieses Feld ja für diese Konstellation der verknüpften Tabellen noch gar keine Daten aufweist. Die Tabelle **tblKundenMarkierungen** ist zu diesem Zeitpunkt noch leer. Nun aktivieren wir aber einmal das Feld **Markierung** für einige Datensätze der Abfrage (siehe Bild 8).

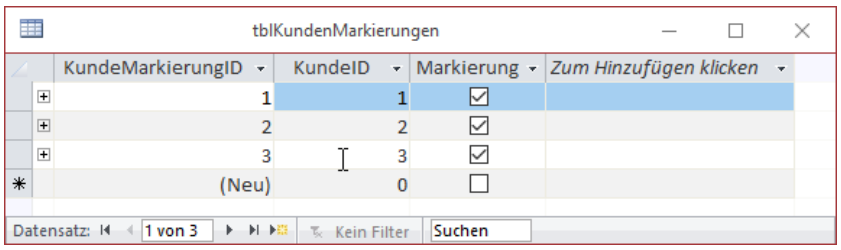


Bild 9: Neue Einträge in der Tabelle **tblKundenMarkierungen**

Wenn wir danach die Tabelle **tblKundenMarkierungen** öffnen, finden wir für jeden der drei zuvor in der Abfrage **qryKundenMarkierungen** angehakten Datensätze einen neuen Eintrag vor (siehe Bild 9).

Datenherkunft des Unterformulars austauschen

Nun öffnen wir erneut das Unterformular **sfmKunden** in der Entwurfsansicht und tauschen in der Eigenschaft Datensatzquelle die Tabelle **tblKunden** gegen die soeben erstellte Abfrage **qryKundenMarkierungen** aus. Wenn wir das Unterformular dann speichern und schließen und das Hauptformular in der Formularansicht öffnen, ändert sich erst einmal nichts – das Unterformular zeigt die gleichen Datensätze wie zuvor an.

Wir haben aber nun über das Feld **Markierung** die Möglichkeit, den Markiert-Status mit den angezeigten Datensätzen zu speichern.

Bedingte Formatierung definieren

Da wir ja weiter oben bereits ein paar Datensätze mit dem Status Markiert versehen haben, wollen wir nun zunächst die bedingte Formatierung einstellen und so die Markierung auch sichtbar machen.

Dazu nehmen wir uns zunächst das Feld **KundeID** vor. Fügen Sie die Einfügemarke in der Formularansicht in dieses Feld ein. Wenn Sie den Ribbon-Tab **Datenblatt** markiert haben, finden Sie rechts den Befehl **Formatierung|Bedingte Formatierung** vor (siehe Bild 10). Leider wird dieser deaktiviert, sobald sie mehr als ein Feld markieren – Sie können also in dieser Ansicht immer nur die bedingte Formatierung für jeweils ein Feld festlegen.

Da wir gegen Fließbandarbeit sind, schauen wir uns nach Alternativen um, die wir auch schnell finden. Wir schließen das Hauptformular und öffnen das Unterformular **sfmKunden** in der Entwurfsansicht. Wenn wir hier zur Ribbon-Gruppe **Format|Steuerelementformatierung** wechseln, finden wir ebenfalls den Eintrag **Bedingte Formatierung** vor. Markieren wir nun alle Textfelder des Formulars, bleibt diese Schaltfläche aktiviert – in der Entwurfsansicht können

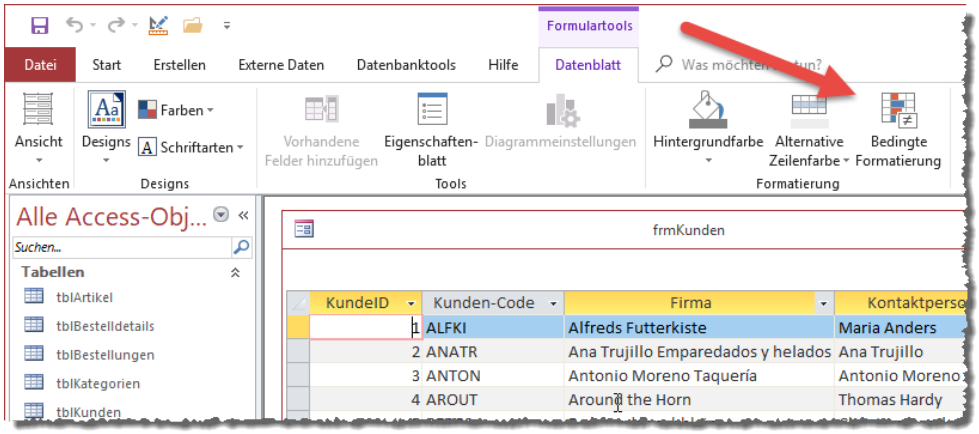


Bild 10: Der Befehl **Bedingte Formatierung** im Ribbon

Markierte Datensätze verarbeiten

Wenn Sie gern mit der Datenblattansicht arbeiten, ist Ihnen vermutlich schon aufgefallen, dass es keine Möglichkeit, mehrere Datensätze gleichzeitig zu markieren – außer, die Datensätze hängen zusammen. Mit der bedingten Formatierung und ein paar kleinen Tricks gelingt es uns allerdings doch, Datensätze etwa per Doppelklick zu selektieren und wieder abzuwählen. Dieser Artikel zeigt, wie es funktioniert.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1803_MarkierteDatensaetzeVerarbeiten.accdb**.

Vorbereitung

Im Artikel Datensätze markieren mit bedingter Formatierung haben wir gezeigt, wie Sie durch geschickten Einsatz der bedingten Formatierung komplette Datensätze in der Datenblattansicht etwa eines Unterformulars markieren können – auch wenn die Datensätze nicht zusammenhängen (siehe Bild 1). Neben der Möglichkeit, einen Datensatz per Doppelklick auf eines der Steuerelemente Einträge aus- oder abzuwählen, können Sie auch per Klick auf eine der beiden Schaltflächen **Alle auswählen** oder **Alle abwählen** die kompletten Datensätze aus- oder abwählen.

Ausgewählte Einträge ermitteln

Während Sie, wenn Sie die gleiche Vorgehensweise etwa mit einem Listenfeld abbilden, direkt über die Eigenschaften des Listenfeldes ermitteln müssen, welcher Eintrag markiert ist und welcher nicht, haben wir hier den Vorteil, dass wir den Zustand eines jeden Eintrags in der Tabelle **tblKundenMarkierungen** gespeichert haben. Das heißt, dass wir, wenn wir etwa alle markierten Einträge weiterverarbeiten wollen, direkt auf die zugrunde liegende Tabelle zugreifen können und nicht etwa auf das Datenblatt zugreifen müssen. Die Tabelle **tblKundenMarkierungen** sieht etwa wie in Bild 2 aus. Sie enthält grundsätzlich nur Einträge, die bereits einmal markiert waren. Einige davon können auch durch erneutes Doppelklicken abgewählt worden sein. Wenn wir nun alle markierten Datensätze ermitteln wollen, brauchen wir also nur die Tabelle **tblKundenMarkierungen** nach Datensätzen zu durchforsten, deren Feld **Markierung** den Wert **Ja** enthält.

In der Regel werden wir aber direkt auf die kompletten Daten zugreifen wollen, sodass wir doch die Abfrage **qryKundenMarkierungen** hinzuziehen werden, welche alle Felder der Tabelle **tblKunden** liefert. Diese wandeln wir nun in Form der Abfrage

Bild 1: Markierung von Datensätzen in der Datenblattansicht

qryMarkierteKunden etwas ab, sodass diese im Entwurf wie in Bild 3 aussieht. Die Abfrage ist wie die zuerst genannte Abfrage aufgebaut, enthält aber für das Feld **Markierung** noch den Vergleichswert **Wahr** als Kriterium.

Damit liefert diese Abfrage nur noch diejenigen Datensätze der Tabelle **tblKunden**, für die erstens ein verknüpfter Datensatz in der Tabelle **tblKunden-Markierungen** vorliegt und zweitens das Feld **Markierung** dieser Tabelle den Wert **True** enthält. In der Datenblattansicht liefert dieser Abfrage nun etwa das Ergebnis aus Bild 4.

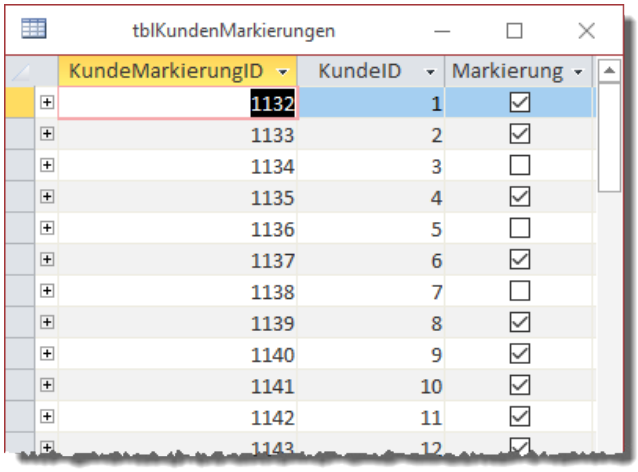


Bild 2: Speicherung der markierten Datensätze

Markierte Datensätze ausgeben

Diese können wir nun beispielsweise nutzen, um in einem Bericht eine Auflistung aller aktuell aufgelisteten Datensätze auszugeben. Dazu fügen wir der Beispieldatenbank einen neuen, leeren Bericht namens **rptSelektierteKunden** hinzu. Für die Eigenschaft **Datensatzquelle** legen wir die zuvor erstellte Abfrage **qryMarkierteDatensaetze** fest. Damit können wir nun die gewünschten Felder aus der Feldliste in den Detailbereich des Berichtsentwurfs ziehen. Hier werden diese nun untereinander angezeigt, wir wün-

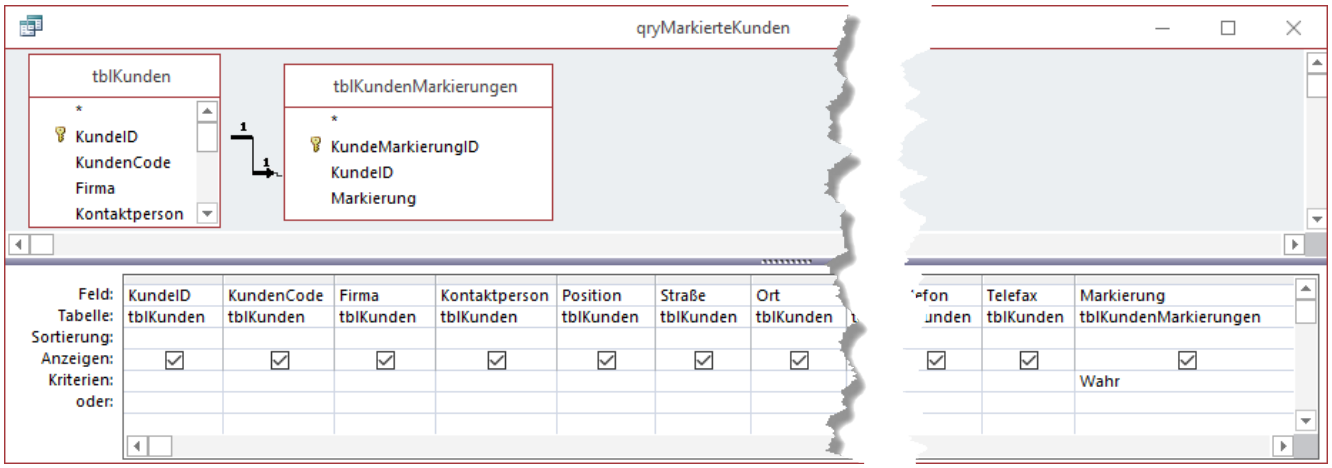


Bild 3: Abfrage, die nur die markierten Kunden liefert

Bild 4: Abfrage, die nur die markierten Kunden liefert, in der Datenblattansicht

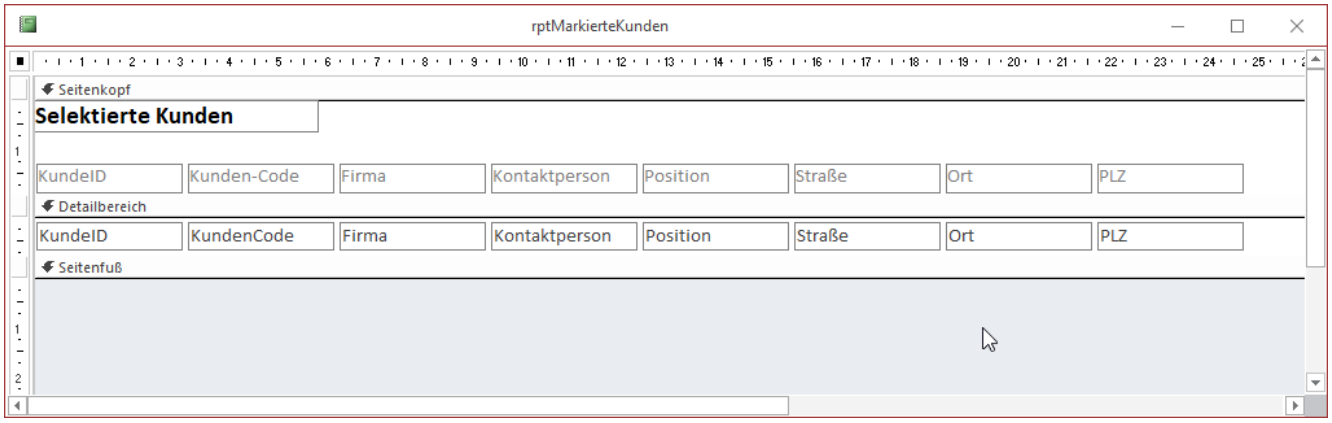


Bild 5: Entwurfsansicht des Berichts zur Ausgabe der markierten Kunden

schen uns allerdings eine Darstellung in Tabellenform. Das können wir mit den Layout-Möglichkeiten des Berichtsentwurfs relativ schnell bewerkstelligen. Wie das genau geht, erfahren Sie im Beitrag **Tabellerische Berichtslayouts**. Das Ergebnis sieht nach ein paar Schönheitskorrekturen wie in Bild 5 aus.

Wenn wir nun in die Seitenansicht wechseln, erhalten wir den Bericht aus Bild 6. Der Bericht zeigt tatsächlich nur die ausgewählten Datensätze aus dem Formular an.

Selektierte Kunden im Bericht hervorheben

Wir wollen nun noch einen weiteren Bericht programmieren, der diesmal zwar alle Kunden anzeigt, aber alle markierten Kunden seinerseits mit einer Markierung versieht. Diesen Bericht wollen wir **rptMarkierteKunden** nennen. Er ist genauso aufgebaut wie der vorherige Bericht, wir können diesen also einfach kopieren. Lediglich die Datensatzquelle müssen wir anpassen. Diese soll die Abfrage **qryKundenMarkierungen** enthalten.

Damit der Bericht die Zeilen mit den zuvor im Formular markierten Kunden ebenfalls markiert, arbeiten wir wieder mit der bedingten Formatierung. Eines vorweg: Im Bericht müssen wir nicht mit den Tricks arbeiten, die im Formular notwendig waren, um das Flackern beim Aus- oder Abwählen eines der Einträge nötig waren.

Hier brauchen wir uns nur darum zu kümmern, dass die entsprechenden Datensätze markiert hervorgehoben werden.

Bild 6: Bericht nur mit den markierten Datensatzen

Excel fernsteuern, Teil 2: Workbooks und -sheets

Von Access aus können Sie andere Anwendungen per VBA fernsteuern, wenn diese Anwendung dies unterstützt. Das ist grundsätzlich bei den übrigen Office-Anwendungen wie Excel, Outlook, Word oder PowerPoint der Fall. In dieser Artikelserie schauen wir uns an, wie Sie eine Excel-Anwendung fernsteuern können und wie Sie Dokumente anlegen, Daten in eine Excel-Tabelle schreiben oder die Daten auslesen. Im vorliegenden zweiten Teil geht es um den Umgang mit Excel-Dateien, den sogenannten Workbooks, und den darin enthaltenen Tabellen, in der Excel-Welt auch Worksheets genannt.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **AccessUndExcelFernsteuern_1803.accdb**.

Begriffsklärung

Normalerweise spricht man ja bei allem, was man in physischer Form im Dateisystem speichern kann, von Dateien oder Verzeichnissen. Unter Excel heißen diese Dateien genauer Workbooks. Jedes Workbook hat im unteren Bereich einige Registerreiter, über die Sie die einzelnen Tabellen eines Workbooks ansteuern können. Wir werden in dieser Artikelreihe die Begriffe **Workbook** und **Worksheet** verwenden.

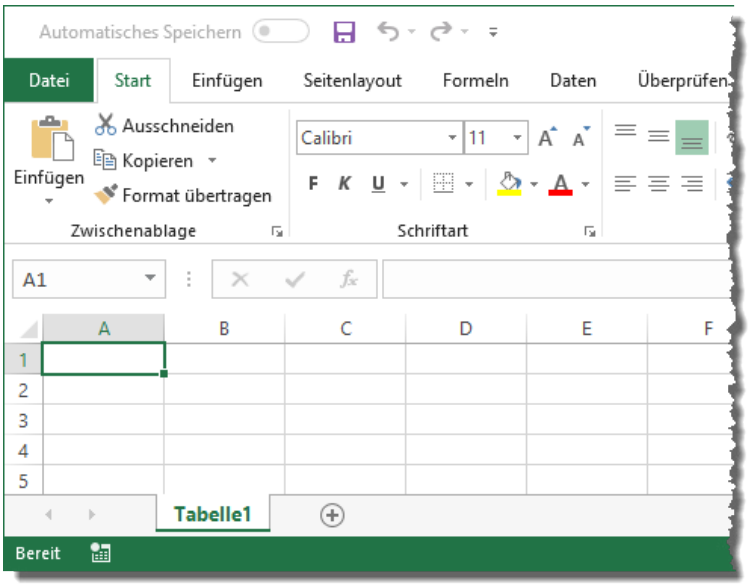


Bild 1: Excel mit einem neuen Workbook

Eine neues Workbook erstellen

Wenn Sie, wie im ersten Teil dieser Artikelreihe (**Excel fernsteuern, Teil 1: Initialisierung**) erläutert, eine neue Excel-Instanz erstellt haben, können Sie damit noch recht wenig anfangen. Sie benötigen ein Workbook, um etwas zu tun. Dieses können Sie entweder neu erstellen oder Sie öffnen ein bestehendes Workbook. Wir wollen uns zunächst ansehen, wir wir ein neues Workbook erstellen.
Wir gehen davon aus, dass wir im gleichen oder in einem anderen Modul eine Objektvariable für Excel deklariert haben:

```
Public objExcel As Excel.Application
```

in einem neuen Standardmodul namens **mdlWorkbooks** legen wir dann die folgende Prozedur an:

```
Public Sub WorkbookErstellen()
    Dim objWorkbook As Excel.Workbook
    Set objExcel = New Excel.Application
    objExcel.Visible = True
    Set objWorkbook = objExcel.Workbooks.Add
End Sub
```

Diese deklariert zunächst eine Objektvariable für das neue Workbook. Danach erzeugen wir die Excel-Instanz und blenden diese durch Einstellen der Variablen **Visible** auf den Wert **True** ein. Schließlich folgt der entscheidende Schritt: Das Erstellen des neuen Workbooks durch die Methode **Add** der **Workbooks**-Auflistung des **Excel.Application**-Objekts. Dieses erscheint nun wie in Bild 1 mit einem ersten Worksheet

namens **Tabelle1**, zu erkennen an der Registerkarte im unteren Bereich.

Workbook schließen

Bevor wir uns weitere Möglichkeiten ansehen, schließen wir das Workbook erst einmal wieder per VBA-Code. Das erledigen wir mit der **Close**-Methode des Workbooks aus **objWorkbook**:

```
objWorkbook.Close
```

Dies schließt das Workbook wieder, bevor wie es beispielsweise in Form einer Datei auf der Festplatte gespeichert haben.

Workbook speichern

Wenn Sie ein Workbook erstellen, wollen Sie es in den meisten Fällen vermutlich auch speichern. Das können Sie beispielsweise erledigen, wenn Sie das Workbook schließen. Das Schließen erfolgt über die **Close**-Methode. Diese bietet ein paar Parameter an, mit denen Sie festlegen können, ob eventuelle Änderungen gespeichert werden sollen:

- **SaveChanges**: Der Wert **True** gibt an, dass das Workbook beim Schließen gespeichert werden soll, wenn ungespeicherte Änderungen vorliegen.
- **Filename**: Gibt den Dateinamen an, unter dem das Workbook gespeichert werden soll.

Ein Beispiel für den Aufruf der **Close**-Methode sieht nun wie folgt aus:

```
objWorkbook.Close True, CurrentProject.Path ↵
                                    & "\Beispiel.xlsx"
```

Fügen wir diesen Befehl in die obigen Methode ein, in der wir eine neue Excel-Instanz erstellt und ein Workbook erzeugt haben, finden wir nach der Ausführung allerdings keine neue Datei mit der Dateiendung **.xlsx** im Verzeichnis der Datenbank vor. Der Grund ist einfach: Wir haben zwar ein Workbook hinzufügt, dieses aber vor dem Schließen nicht geändert. Deshalb wird es beim Schließen auch nicht gespeichert. Dies ändern wir, indem wir ein wenig vorgreifen und die Zelle **A1** mit einem Beispieltext füllen, bevor wir die Datei schließen:

```
Public Sub WorkbookSchliessenSpeichern()
    Dim objWorkbook As Excel.Workbook
    Set objExcel = New Excel.Application
    objExcel.Visible = True
    Set objWorkbook = objExcel.Workbooks.Add
    objWorkbook.Sheets(1).Range("A1") = "Beispieltext"
    objWorkbook.Close True, CurrentProject.Path ↵
                                    & "\Beispiel.xlsx"
    objExcel.Quit
    Set objExcel = Nothing
End Sub
```

Damit landet nun eine Datei namens **Beispiel.xlsx** in unserem Datenbankverzeichnis. Sollten Sie diese Prozedur erneut aufrufen, fragt Access, ob die vorhandene Datei gleichen Namens überschrieben werden soll.

Es gibt noch weitere Befehle zum Speichern des Workbooks, die ohne das Schließen des Workbooks aufgerufen werden können. Vor allem jedoch speichern diese das Workbook auch dann, wenn Sie noch keine Änderungen vorgenommen haben:

- **Save**: Speichert das Workbook unter dem Namen, unter dem es bereits zuvor gespeichert wurde oder unter dem Namen **Mappe1.xlsx**.
- **SaveAs**: Speichert das Workbook unter dem als Parameter angegebenen Namen. Diese Anweisung bietet allerdings noch viele weitere Parameter, etwa zum Festlegen des Dateiformats.

Für den Parameter zur Angabe des Dateiformats verwenden Sie etwa die folgenden Werte:

- **xlExcel8**: Excel-Workbook im Format von Excel 97-2003
- **xlExcel12**: Excel-Workbook (binär) im Format von Excel 2007-2013

- **xlOpenXMLWorkbook**: Excel-Workbook (XML) im Format von Excel 2007-2016

- **xlOpenXMLWorkbookMacroEnabled**: Excel-Workbook (XML) mit Makros im Format von Excel 2007-2016

Zu beachten ist hier, dass Sie jeweils die passende Dateiendung zur Excel-Version angeben sollten, also **.xls** für Dateien bis Excel 2003 und **.xlsx** für Dateien ab Excel 2007.

Workbook ohne Umweg über das Excel-Objekt erstellen

Die obige Methode geht davon aus, dass Sie ein Objekt des Typs **Excel.Application** erzeugen und damit dann ein **Workbook**-Objekt anlegen. Das ist immer dann sinnvoll, wenn Sie das Excel-Objekt für mehr als das pure Anlegen benötigen – etwa, um noch weitere Workbooks zu erzeugen.

Wenn Sie nur das Workbook erzeugen möchten, aber keine Verwendung für die Objektvariable des Typs **Excel.Application** haben, können Sie die folgende Variante nutzen:

```
Public Sub WorkbookCreate()
    Dim objWorkbook As Excel.Workbook
    Set objWorkbook = CreateObject("Excel.Sheet")
    objWorkbook.SaveAs CurrentProject.Path _
                                & "\CreateObject.xlsx"
    Set objWorkbook = Nothing
End Sub
```

Hier erstellen wir ein neues Objekt des Typs **Excel.Workbook**, interessanterweise aber mit dem Parameter **Excel.Sheet** für die **CreateObject**-Methode. Die Methode funktioniert jedoch einwandfrei und ist gefühlt auch etwas schneller als der Weg über das vorherige Erstellen der Excel-Instanz.

Es gibt allerdings eine Excel-Instanz, nur dass wir diese nicht per Objektvariable referenziert haben. Das wird deutlich, wenn wir versuchen, auch das wie oben erstellte Workbook anzuzeigen. Die **Workbook**-Klasse selbst liefert keine Eigenschaft namens **Vislble**, mit der wir die Excel-Instanz einblenden könnten. Wir müssen also wohl diese Eigenschaft des übergeordneten Objekts des Typs **Excel.Application** einstellen. Das übergeordnete Objekt referenzieren wir über die Eigenschaft **Parent** des **Workbook**-Objekts:

```
Public Sub WorkbookCreate()
    Dim objWorkbook As Excel.Workbook
    Set objWorkbook = CreateObject("Excel.Sheet")
    objWorkbook.SaveAs CurrentProject.Path _
                                & "\CreateObject.xlsx"
    objWorkbook.Parent.Visible = True
    Stop
    Set objWorkbook = Nothing
End Sub
```

Die **Stop**-Anweisung haben wir eingebaut, damit Sie sich von der Existens der Excel-Instanz überzeugen können.

Bestehende Excel-Datei öffnen

Nun, da wir ein paar Excel-Dateien auf die Festplatte gebannt haben, wollen wir diese per VBA erneut öffnen und gegebenenfalls auch anzeigen. Um bei Vorhandensein einer Objektvariablen mit einer Excel-Instanz ein Workbook zu öffnen, nutzen wir die **Open**-Methode der **Workbooks**-Auflistung:

```
Public Sub WorkbookOeffnenInstanz()
    Dim objWorkbook As Excel.Workbook
    Set objExcel = New Excel.Application
    objExcel.Visible = True
    Set objWorkbook = objExcel.Workbooks.Open(_
                CurrentProject.Path & "\Beispiel.xlsx")
End Sub
```

Dies zeigt das unter dem Dateinamen **Beispiel.xlsx** gespeicherte Workbook an.

Geöffnetes Workbook referenzieren

Wenn Sie bereits eine Excel-Instanz mit einem Workbook geöffnet haben und dieses per VBA referenzieren möchten, um darauf zuzugreifen oder Änderungen vorzunehmen, können Sie die Referenzierung mit

der **GetObject**-Methode durchführen. Dazu übergeben Sie dieser einfach den Pfad zu der Excel-Datei:

```
Public Sub WorkbookOeffnen-
CreateObject()
    Dim objWorkbook As Ex-
cel.Workbook
    Set objWorkbook =
GetObject(CurrentProject.
Path ⁊

& "\Beispiel.xlsx")
    objWorkbook.Parent.
Visible = True
    objWorkbook.Sheets(1).Range("A2") = "Neuer Text"
End Sub
```

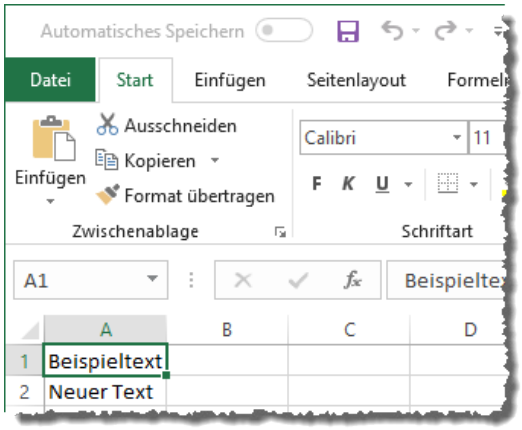


Bild 2: Referenzieren eines geöffneten Workbooks

In diesem Fall fügen wir noch einen Wert zur Zelle **A2** hinzu (siehe Bild 2).

Schnell auf Workbooks zugreifen

Im ersten Teil der Artikelreihe haben wir bereits die Funktion GetExcel vorgestellt, mit der Sie schnell eine neue Excel-Instanz hervorzaubern und nutzen können. Genau genommen verkürzen Sie damit gar nicht die Zeit, bis die Excel-Instanz zur Verfügung steht, aber wir haben den dafür notwendigen Code ausgelagert und Sie brauchen diesen nicht immer neu zu schreiben. Das wollen wir nun auch für ein Excel-Workbook machen, und zwar zum Erstellen eines neuen Workbooks und auch zum Öffnen bestehender Workbooks. Beides sollte möglichst durch eine einzige Funktion abgedeckt werden, der wir den Pfad der zu öffnenden beziehungsweise der zu erstellenden Workbook-Datei übergeben. Diese Funktion fügen wir wieder dem Standardmodul **mdlExcelobjekt** hinzu, das wir auch bereits im ersten Teil angelegt haben:

```
Public Function GetWorkbook(strWorkbook As String) ⁊
                                        As Excel.Workbook
    Dim objWorkbook As Excel.Workbook
    If Len(Dir(strWorkbook)) = 0 Then
        Set objWorkbook = GetExcel.Workbooks.Add()
        objWorkbook.SaveAs
strWorkbook
    Else
        Set objWorkbook = ⁊
                GetExcel.Work-
books.Open(strWorkbook)
    End If
    Set GetWorkbook = objWorkbook
End Function
```

Die Funktion erwartet den Pfad der zu erstellenden oder zu öffnenden Workbook-Datei als Parameter. Sie prüft dann im ersten Schritt mit der **Dir**-Funktion, ob es bereits eine Datei mit diesem Pfad gibt. Ist das nicht der Fall, verwendet sie die **Add**-Methode der **Workbooks**-Auflistung, um die Datei zu erstellen und speichert diese direkt im Anschluss mit der **SaveAs**-Methode unter dem angegebenen Dateinamen. Ist die Datei hingegen bereits vorhanden, öffnet die Funktion diese mit der **Open**-Methode unter Angabe des Pfades. In beiden Fällen landet ein Verweis auf das geöffnete Excel-Workbook in der Variablen **objWorkbook**, die dann als Funktionswert von **GetWorkbook** zurückgegeben wird.

Ein Aufruf dieser Funktion sieht dann etwa wie folgt aus:

```
Set objWorkbook = GetWorkbook("c:\Workbook.xlsx")
```

Zu beachten ist hier, dass das Workbook geöffnet bleibt, wenn Sie es nicht explizit wieder schließen. Auch die entsprechende Excel-Instanz bleibt im Hintergrund geöffnet, auch wenn Sie die Datenbankanwendung schließen und somit auch die Objektvariable leeren.

Probleme mit GetExcel

Auch mit **GetExcel** könnte es Probleme geben, wenn Sie mit dieser Funktion eine Excel-Instanz erzeugen, diese dann im Task-Manager beenden und anschließend nochmals auf die mit **GetExcel** unter der Variablen **m_Excel** gespeicherten Instanz zugreifen wollen.

Diese Instanzvariable wird nämlich nicht geleert, wenn die Instanz selbst von außerhalb beendet wird. Wenn dies nicht zu Testzwecken geschieht, sollte eine solche über diese Funktion erzeugte Excel-Instanz allerdings auch nicht über den Task-Manager beendet werden. Wir können diesen Fehler allerdings auch verhindern, indem wir die Funktion **GetExcel** wie folgt erweitern:

```
Public Function GetExcel() As Excel.Application
    Dim strTest As String
    If m_Excel Is Nothing Then
        Set m_Excel = New Excel.Application
    Else
        On Error Resume Next
        strTest = m_Excel.Name
        If Err.Number = 462 Then
            Set m_Excel = New Excel.Application
        End If
        On Error GoTo 0
    End If
    Set GetExcel = m_Excel
End Function
```

Wir versuchen hier, wenn **m_Excel** gefüllt ist, also eine Instanz existieren sollte, im **Else**-Zweig der **If... Then**-Bedingung auf den Namen der Excel-Instanz zuzugegreifen. Wenn die Instanz zwischenzeitlich von außerhalb beendet wurde, löst dies den Fehler **462** aus (**Der Remote-Server-Computer existiert nicht oder ist nicht verfügbar.**). In diesem Fall erstellen wir die Instanz einfach neu und weisen diese wiederum der Variablen **m_Excel** zu, die dann als Rückgabewert der Funktion zurückgeliefert wird.

Mehrere Workbooks

Wenn Sie ein Workbook in einer Excel-Instanz öffnen und dann von dieser Instanz aus über den Öffnen-Befehl ein weiteres Workbook, sieht es zwar so aus, als ob zwei Instanzen von Excel geöffnet wurden (siehe Bild 3). In der Tat handelt es sich hierbei allerdings nur um eine einzige Instanz, die zwei Workbooks geöffnet hat. Das soll nicht darüber hinwegtäuschen, dass auch zwei Instanzen mit jeweils einem geöffneten Workbook genauso aussehen wie diese eine Instanz mit zwei Workbooks! Unter VBA erzeugen Sie eine Instanz mit zwei Workbooks etwa wie folgt:

```
Public Sub ZweiWorkbooks()
    Dim objExcel As Excel.Application
    Dim objWorkbook1 As Excel.Workbook
    Dim objWorkbook2 As Excel.Workbook
    Dim objWorkbook As Excel.Workbook
    Dim i As Integer
    Set objExcel = GetExcel
    Set objWorkbook1 = objExcel.Workbooks.Add
    Set objWorkbook2 = objExcel.Workbooks.Add
    objExcel.Visible = True
    objWorkbook1.Sheets(1).Range("A1") = "Workbook 1"
    objWorkbook2.Sheets(1).Range("A1") = "Workbook 2"
    Debug.Print "Anzahl Workbooks: " _
                & objExcel.Workbooks.Count
    For i = 1 To objExcel.Workbooks.Count
        Debug.Print objExcel.Workbooks.Item(i).Name
    Next i
    For Each objWorkbook In objExcel.Workbooks
        Debug.Print objWorkbook.Name
    Next objWorkbook
End Sub
```

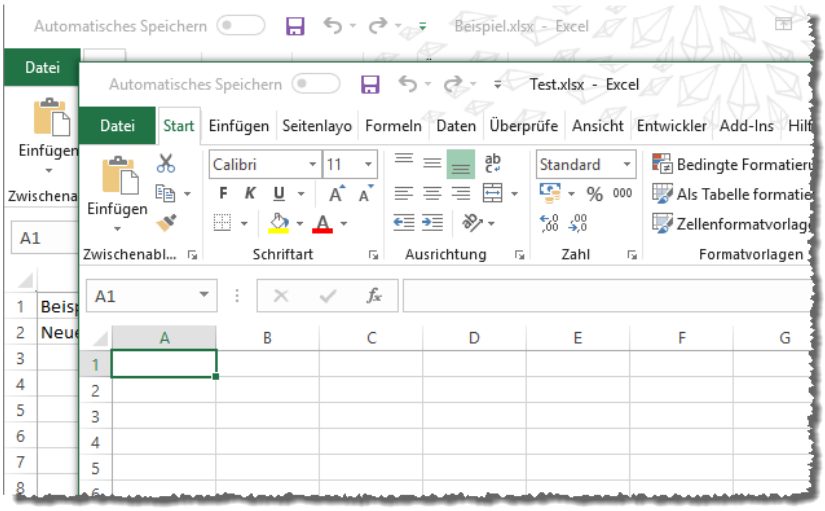


Bild 3: Mehrere Workbooks in einer Instanz

Das Navigationssteuerelement

Das ursprünglich in der Version Access 2010 für den Einsatz in Webdatenbanken konzipierte Navigationssteuerelement kann auch in Desktop-Datenbanken sinnvoll eingesetzt werden. Das ist insbesondere deshalb hervorzuheben, da die Webdatenbanken wohl keine große Verbreitung erfahren haben. Dennoch ist dieses Steuerelement nützlich, wenn Sie es etwa zum Einbinden und Anzeigen verschiedener Unterformulare oder auch des gleichen Unterformulars mit verschiedene Datensätzen einsetzen möchten.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1804_Navigationssteuerelement.accdb**.

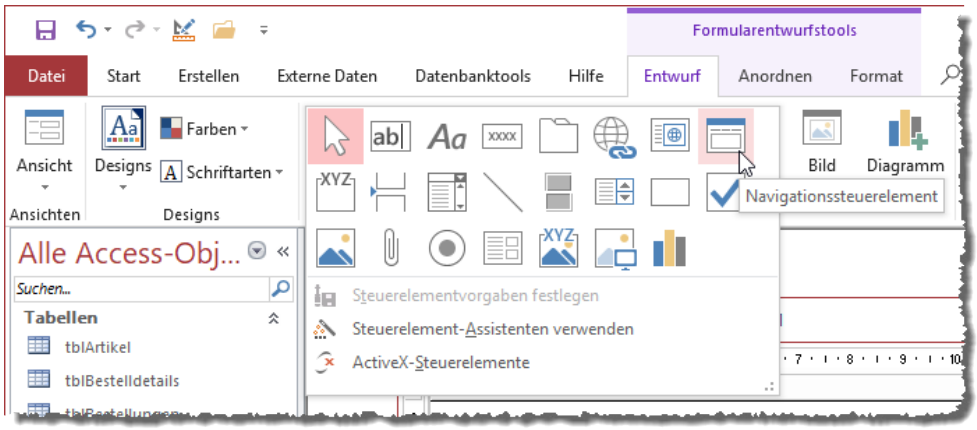


Bild 1: Einfügen eines Navigationssteuerelements über das Ribbon

Webdatenbanken und das Navigationssteuerelement
Webdatenbanken unterschieden sich vor allem von Desktop-Datenbanken auf Basis von Microsoft Access, indem sie es nicht erlauben, mehrere Fenster innerhalb eines Hauptfensters anzuzeigen wie dies etwa bei der Desktop-Variante der Fall ist.

Das ist wohl der Hauptgrund, warum Microsoft dieses Steuerelement entwickelt hat: Sie können damit leicht mehrere Unterformulare einbinden und diese über die Navigationsleiste des Steuerelements ansteuern.

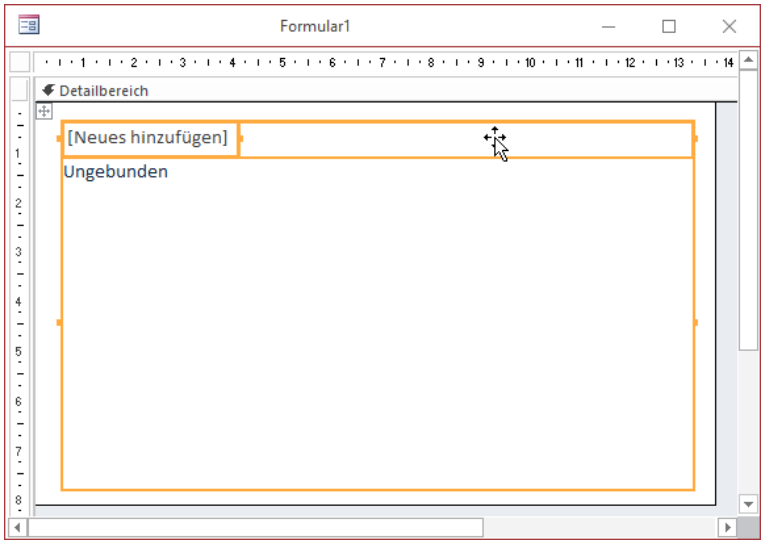


Bild 2: Ein frisch angelegtes Navigationssteuerelement

Schauen wir uns also an, wie wir von diesem Steuerelement auch in der Desktop-Version von Access profitieren können.

Navigationssteuerelement zu einem Formular hinzufügen

Für unser Beispiel erstellen wir zunächst eine neue, leere Beispieldatenbank, die wir mit den Tabellen der Südsturm-Datenbank angereichert haben, um Material zum Experimentieren zu haben.

Dann fügen wir der Beispieldatenbank ein Formular hinzu. Nun kommt

das Navigationssteuerelement. Dieses fügen wir über den Ribbon-Eintrag **Entwurf|Steuerelemente|Navigationssteuerelement** zum Entwurf des Formulars hinzu (siehe Bild 1).

Das so hinzugefügte Steuerelement sieht wie in Bild 2 aus. Es weist offensichtlich drei verschiedene Arten von Bereichen auf, die wir mit der Maus einzeln anklicken können.

Der erste Bereich ist das einzelne Navigationbutton-Element, das wir durch einen Klick auf einen der Navigationspunkte markieren. Um was für einen Steuerelementtyp es sich handelt, zeigt uns die folgende Anzeige im Kombinationsfeld des Eigenschaftenblattes (siehe Bild 3).

Klicken wir hingegen auf den freien Bereich rechts neben dem ersten, zunächst als Platzhalter erscheinenden Navigationsbutton, markieren wir das komplette Navigationssteuerelement (siehe Bild 4).

Fehlt noch der untere Bereich. Ein Mausklick darauf liefert uns im Eigenschaftenblatt die Bezeichnung **Navigationsunterformular** (siehe Bild 5).

Dabei handelt es sich also offensichtlich um den Bereich, in dem wir nach einem Klick auf eine der Navigationsschaltflächen das Unterformular anzeigen – beziehungsweise in dem beim Anzeigen des Steuerelements das für die aktive Navigationsschaltfläche festgelegte Element angezeigt wird.

Interessant ist auch, dass wir beim Markieren des Elements auch ein Gitterraster angezeigt bekommen sowie das Fadenkreuz links über dem Steuerelement, mit der wir das ganze Steuerelement verschieben können. Das bekommt man sonst nur beim Arbeiten mit den Layouts für Formulare und Berichte zu sehen.

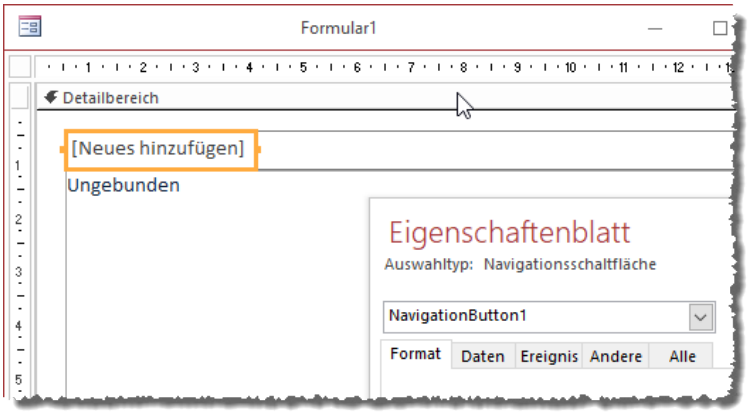


Bild 3: Navigationsbutton-Element

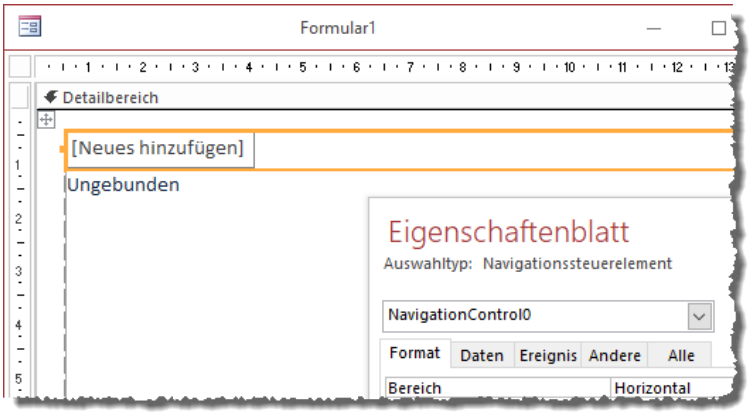


Bild 4: **NavigationControl**-Element

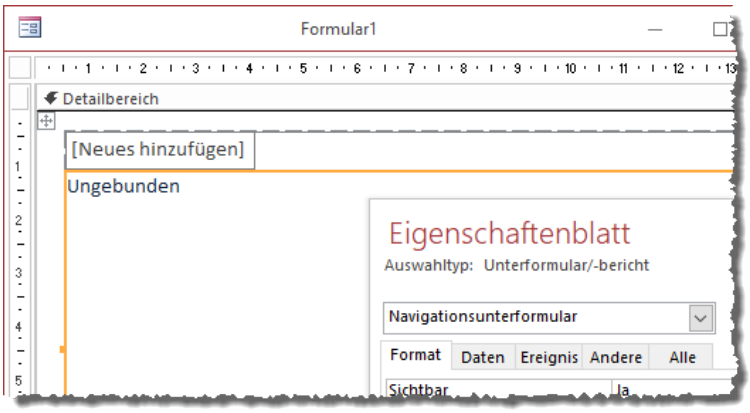


Bild 5: Navigationsunterformular

Formularansicht

Bevor wir erstmalig in die Formularansicht des Steuerelements wechseln, wollen wir noch die Eigenschaften **Navigationsschaltflächen**, **Datensatzmarkierer**, **Trennlinien** und **Bildlaufleisten** auf den Wert **Nein**

und **Automatisch zentrieren** auf **Ja** einstellen, damit die hier unnötigen Elemente entfernt werden.

Wechseln wir dann in die Formularansicht, sehen wir lediglich den Rahmen, der den Bereich für das Unterformular markiert (siehe Bild 6). Das zeigt, dass die vorhandene Navigationsschaltfläche mit der Beschriftung **[Neues hinzufügen]** nur ein Platzhalter für neue Navigationsschaltflächen ist. Wir werden also ein paar Element hinzufügen müssen, damit hier etwas geschieht.

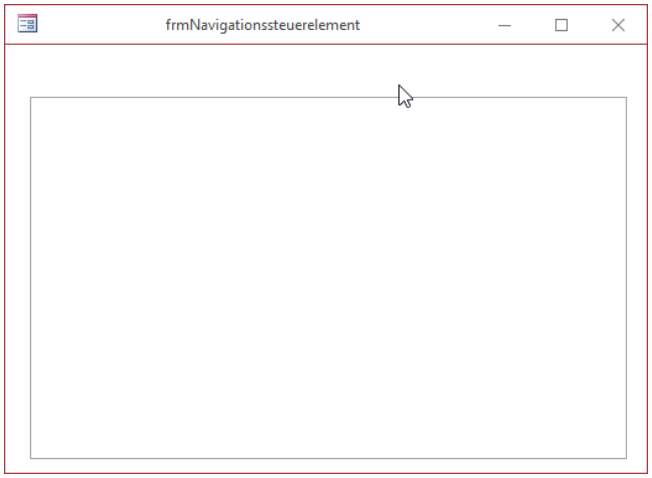


Bild 6: Ein leeres Navigationssteuerelement in der Formularansicht

Verankern

Wir wollen das Navigationssteuerelement auch gleich verankern, damit es sich mit dem Formular vergrößert oder verkleinert, wenn wir dessen Größe verändern. Dazu müssen wir erst einmal herausfinden, für welches der Elemente wir die Eigenschaften **Horizontaler Anker** und **Vertikaler Anker** auf die Eigenschaft **Beide** einstellen müssen. In der Tat weisen alle drei Teile dieses Steuerelements die Eigenschaften **Vertikaler Anker** und **Horizontaler Anker** auf. Es macht allerdings nur Sinn, diese für das Navigationsunterformular auf **Beide** einzustellen. Die Breite des Bereichs **NavigationControl** ändert sich ohnehin analog zu der des Navigationsunterformulars.

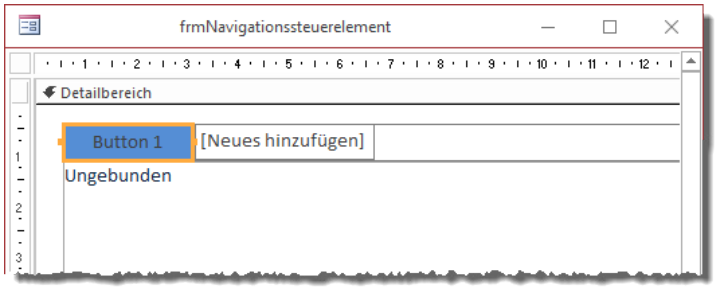


Bild 7: Ein neuer Navigationsbutton

Navigationsschaltfläche anlegen

Eine neue Navigationsschaltfläche können Sie zumindest auf zwei Arten anlegen:

- Die erste ist das Eintragen der gewünschten Bezeichnung für die Platzhalter-Schaltfläche mit der Beschriftung **[Neues hinzufügen]**. Nach dem Abschließen der Eingabe erscheint die neue Schaltfläche mit blauer Hintergrundfarbe und rechts daneben wird ein neuer Platzhalter angezeigt (siehe Bild 7).

- Die zweite Methode ist die über das Kontextmenü. Hier finden Sie ganz oben den Eintrag **Navigationsschaltfläche einfügen**

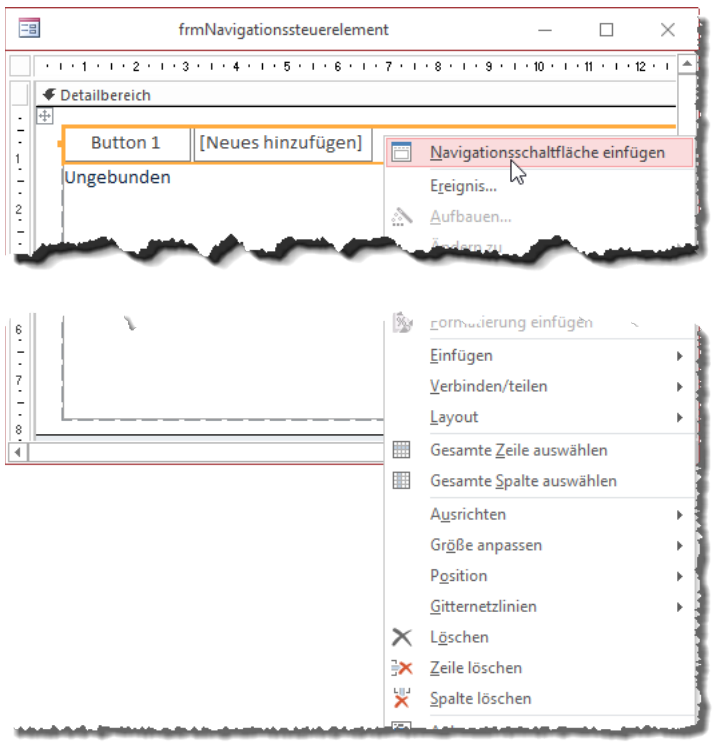


Bild 8: Kontextmenü zum Bearbeiten der Navigationsschaltflächen

(siehe Bild 8). Dieser Befehl fügt direkt einen neuen Navigationsbutton zum Navigationssteuerelement hinzu. Wenn Sie dabei das Kontextmenü für einen der vorhandenen Navigationsbuttons aufrufen und diesen Befehl auswählen, wird die neue Navigationsschaltfläche vor dem jeweiligen Element angelegt. Anderenfalls landet es links vom Element **[Neues hinzufügen]**.

Das Kontextmenü deutet direkt eine Reihe weiterer Möglichkeiten für den Umgang mit diesem Steuerelement an. Letztlich sind diese aber nicht wichtig für die Ausstattung des Navigationssteuerelements.

Anordnen der Navigationsschaltflächen

Sie brauchen sich beim Anlegen keine Sorgen um die Reihenfolge zu machen – Sie können die einzelnen Navigationsschaltflächen nach dem Einfügen immer noch per Drag and Drop auf der Navigationsleiste verschieben. Auch das Löschen versehentlich eingefügter Navigationsschaltflächen ist leicht möglich.

Eigenschaften der Navigationsschaltflächen

Nun möchten wir aber doch allmählich wissen, wie wir ein Unterformular für eine der Navigationsschaltflächen anlegen können. Dafür sehen beispielsweise die beiden Eigenschaften **Name des Navigationsziels** und **WHERE-Klausel für die Navigation** interessant aus (siehe Bild 9). Weitere Eigenschaften, die von denen für ähnliche Steuerelemente wie etwa Schaltflächen abweichen, gibt es nicht.

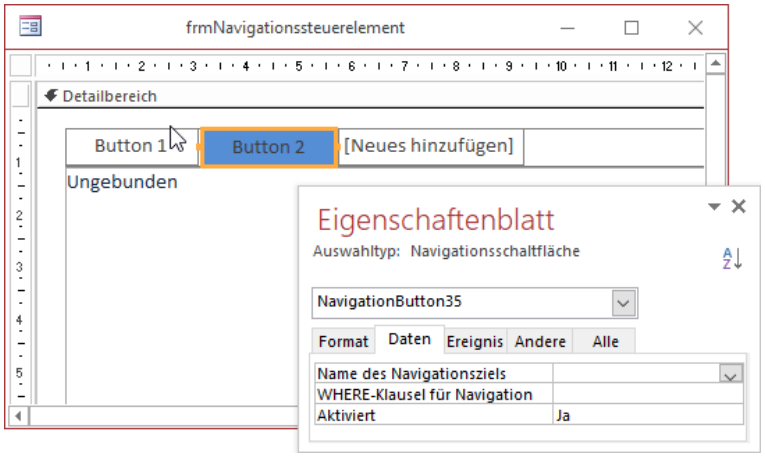


Bild 9: Eigenschaften des Navigationssteuerelements

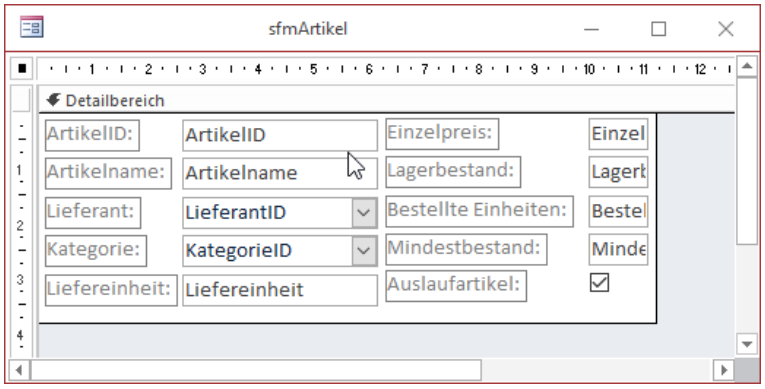


Bild 10: Formular, dass als Unterformular verwendet werden soll

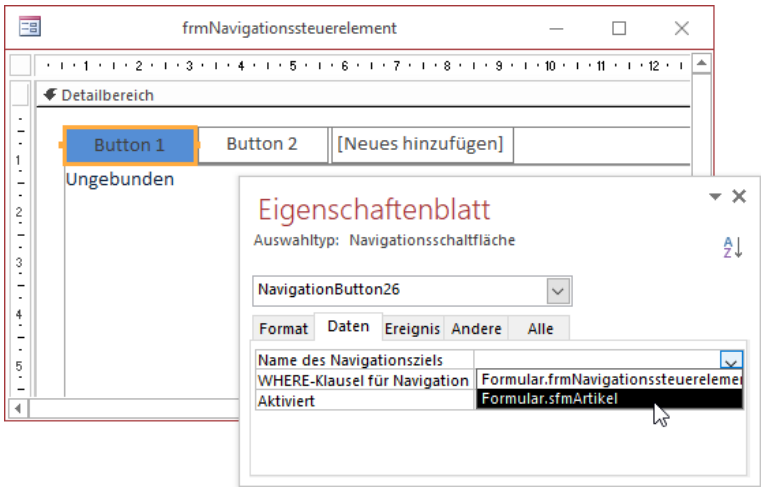


Bild 11: Auswahl des Unterformulars für eine Navigationsschaltfläche

Unterformular anlegen

Also legen wir ein neues Formular an, das wir gleich über eine der Navigationsschaltflächen als Unterformular einblenden können. Dazu verwenden wir ein Formular, dass die Tabelle **tblArtikel** als Datensatzquelle verwendet und alle Felder dieser Tabelle im Detailbereich des Entwurfs enthält (siehe Bild 10).

Unterformular auswählen

Kehren wir zurück zum Entwurf unseres Beispielformulars, können wir dort nun aus dem Kombinationsfeld für die Eigenschaft Name des Navigationsziels das Formular **Formular.sfmArtikel** auswählen (siehe Bild 11).

Anzeige des Unterformulars

Wechseln wir nun zur Formularansicht, erscheint dort unser Unterformular für die Navigationsschaltfläche **Button 1** (siehe Bild 12). Wechseln wir zu **Button 2**, für den wir ja noch kein Formular festgelegt haben, wird der Bereich zur Anzeige der Unterformulare tatsächlich geleert. Unten werden nun noch die Navigationsschaltflächen des Unterformulars angezeigt – ebenso wie der Datensatzmarkierer. Benötigen wir diese oder können wir sie ausblenden? Das hängt vom Anwendungszweck ab. Weiter unten schauen wir uns verschiedene Konfigurationen an. Zunächst kümmern wir uns jedoch noch um die Darstellung der Navigationsschaltflächen.

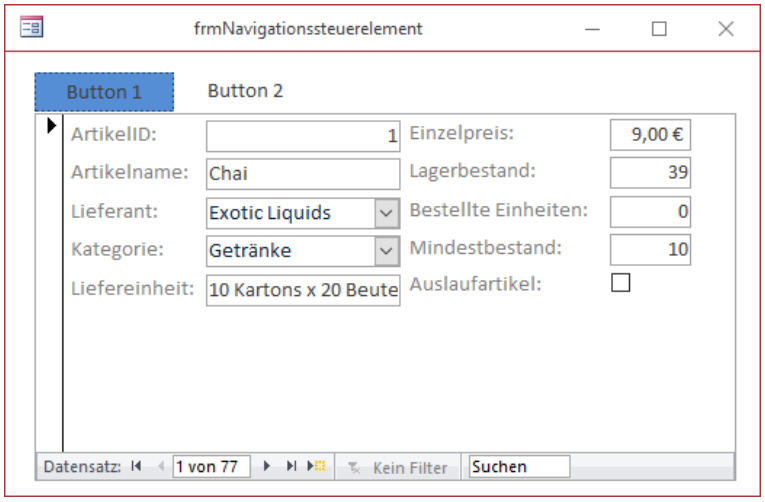


Bild 12: Anzeige eines Unterformulars für eine Navigationsschaltfläche

Darstellung der Navigationsschaltflächen

Die Navigationsschaltflächen können Sie auf verschiedene Arten anordnen.

Eine der dazu notwendigen Eigenschaften übersieht man fast – es ist die erste Eigenschaft auf der ersten Seite des Eigenschaftenblatts. Sie heißt **Bereich** und gibt an, ob die Navigationssteuerelemente horizontal oder vertikal angeordnet werden. Wenn wir diese Eigenschaft von **Horizontal** auf **Vertikal** ändern, erscheinen die bisher angelegten Navigationsschaltflächen untereinander – aber der Bereich, in dem sich diese befinden, liegt immer noch oberhalb vom Unterformular-Steuerelement (siehe Bild 13).

Es wäre nun schön, wenn sich diese vertikal angeordneten Schaltflächen auch am linken Rand neben dem Unterformular positionieren lassen würden. Auch dies lässt sich einstellen.

Navigationsbereich links

Dazu sind die folgenden Schritte nötig:

- Aktivieren Sie während der Entwurfsansicht des Formulars das Ribbon-Tab **Anordnen**.

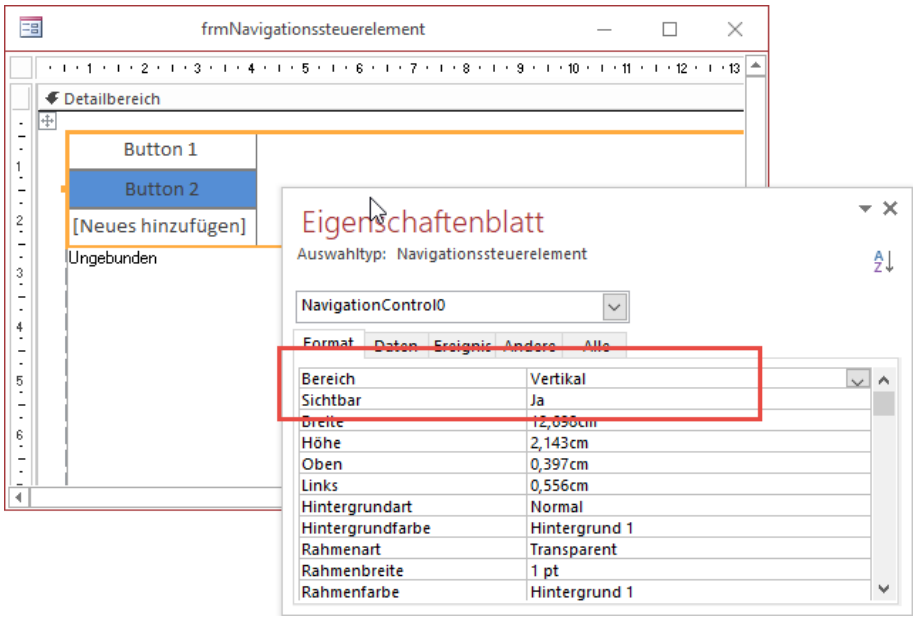


Bild 13: Anzeige eines Unterformulars für eine Navigationsschaltfläche

- Markieren Sie den unteren Bereich, also das Unterformular des Navigationssteuerelements.

- Betätigen Sie den Befehl **Anordnen|Zusammenfügen/Teilen|Horizontal teilen** des Ribbons.

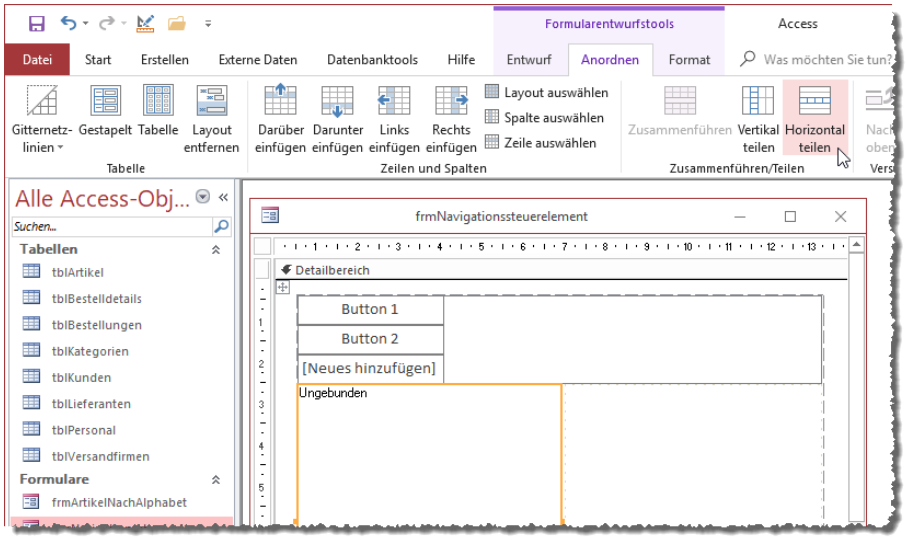


Bild 14: Aufteilen des unteren Bereichs

Damit erhalten Sie die Ansicht aus Bild 14 – wir haben unten nun zwei Bereiche, von denen der linke das Unterformular aufnimmt. Dieses wollen wir zunächst auf die rechte Seite verschieben. Dazu markieren Sie es und fahren mit der Maus auf den farbigen Rand, bis das Mauszeiger-Symbol zum Verschieben erscheint. Ziehen Sie das Unterformular dann auf den rechten Bereich.

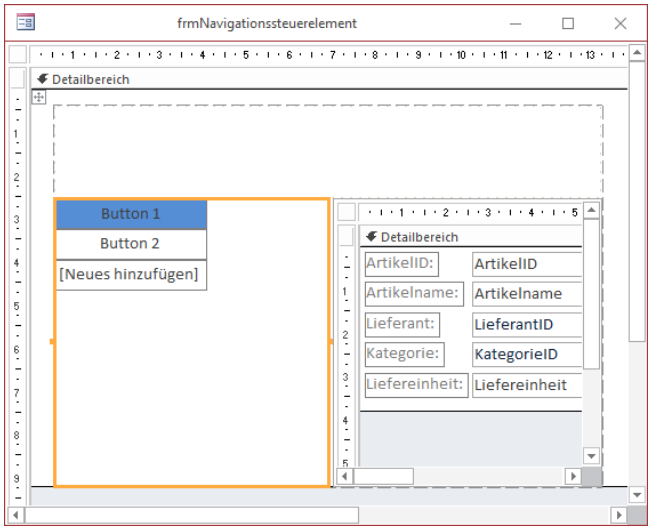


Bild 15: Verschieben des vertikalen Navigationsbereichs

Danach ziehen Sie den oberen Bereich, also den Navigationsbereich, auf die gleiche Weise vom oben nach links unten. Das Zwischenergebnis sieht wie in Bild 15 aus. Nun brauchen Sie nur noch den oberen Bereich zu löschen, am einfachsten mit dem Kontextmenü-Befehl **Zeile löschen**. Und wir passen natürlich noch die Breite des linken Bereichs an die Breite der enthaltenen Schaltflächen an. Schließlich sieht das Formular mit dem Navigationssteuerelement wie in Bild 16 aus.

Verschiedene Konfigurationen

Zwei denkbare Konfigurationen für den Einsatz des Navigationssteuerelements sind die folgenden:

- Sie zeigen für die verschiedenen Navigationsschaltflächen jeweils Unterformulare für verschiedene Elemente an – also beispielsweise Artikel, Kunden, Bestellungen, Lieferanten et cetera. Dann würden Sie jeweils einen Navigationsbutton für jedes Unterformular

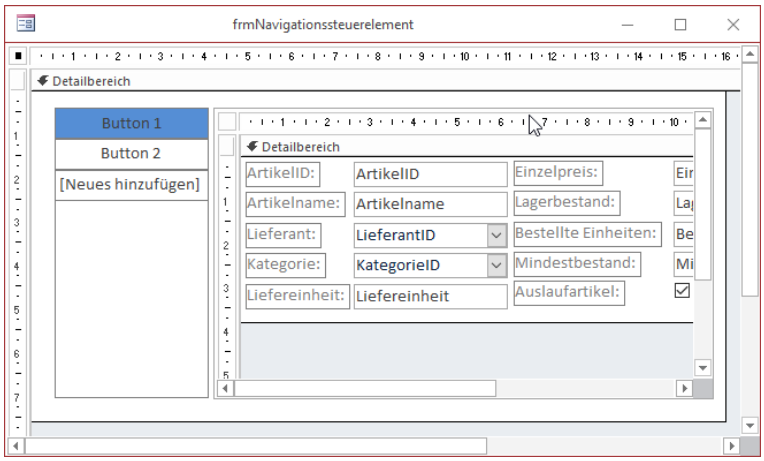


Bild 16: Navigationsbereich als Seitenleiste

Der neue Tabellenverknüpfungs-Manager

Still und heimlich wurde der alte Tabellen-Verknüpfungsmanager von Microsoft ausgetauscht. Der Tabellenverknüpfungs-Manager dient dazu, bestehende Verknüpfungen mit Tabellen aus anderen Datenbanken oder Dateien zu verwalten und auf dem aktuellen Stand zu halten. Die neue Version bietet eine Menge neuer Features, die wir in diesem Artikel vorstellen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1805_Tabellenverknuepfungsmanager.accdb** und im dazugehörigen Backend **BE.accdb**.

Tabellen verknüpfen

Der herkömmliche Weg, um eine Frontend-Anwendung mit den Tabellen eines Access-Backend zu verknüpfen, geht über die Assistenten zum Verknüpfen der Anwendung mit externen Datenquellen. Wenn Sie die aktuelle Anwendung etwa mit den Tabellen einer anderen Access-Datenbank verknüpfen wollen, führen Sie den Ribbon-Befehl **Externe Daten|Importieren und Verknüpfen|Neue Datenquelle|Aus Datenbank|Access** aus.

Der folgende Assistent führt Sie dann durch die notwendigen Schritte.

Sind die Tabellen erst einmal verknüpft, finden Sie diese wie in Bild 1 im Navigationsbereich von Access vor.

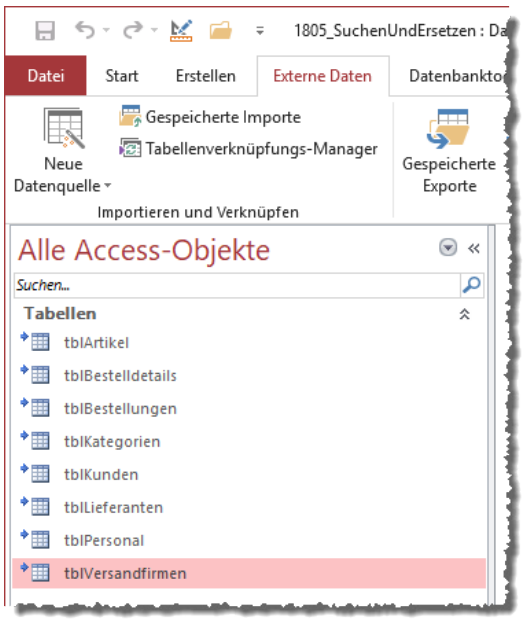


Bild 1: Verknüpfte Tabellen in einer Datenbank

Verknüpfung nicht verfügbar

Wir dann beispielsweise das Backend verschoben oder der Ordner, in dem es sich befindet, umbenannt, erscheint beim Versuch, auf diese Verknüpfung zuzugreifen, die Meldung aus Bild 2.

Was nun? Ganz einfach: Sie öffnen den Tabellenverknüpfungs-Manager. Diesen finden Sie über den Ribbon-Befehl **Externe Daten|Importieren und Verknüpfen|Tabellenverknüpfungs-Manager**. Bis

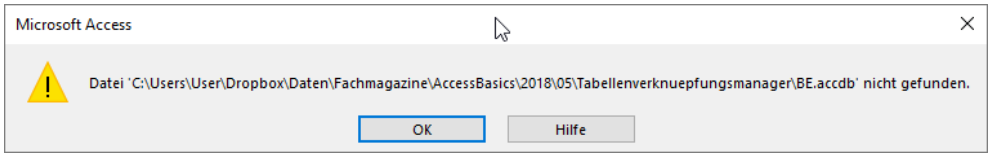


Bild 2: Fehlgeschlagener Zugriff auf eine verknüpfte Tabelle

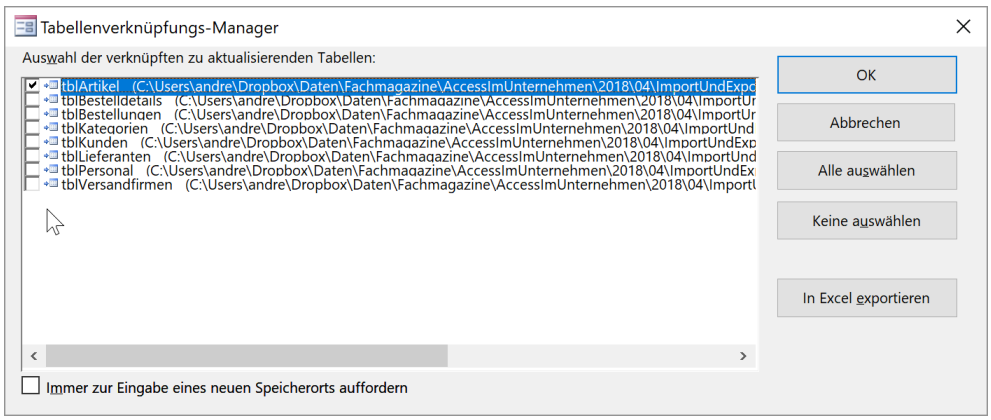


Bild 3: Der alte Tabellenverknüpfungs-Manager

vor kurzem sah der Tabellenverknüpfungs-Manager noch wie in Bild 3 aus.

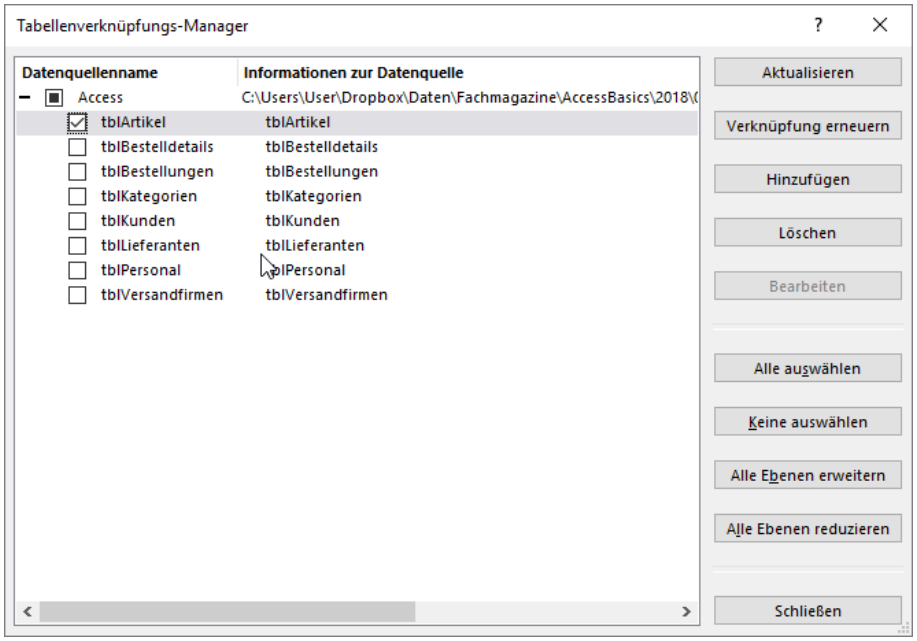


Bild 4: Der neue Tabellenverknüpfungs-Manager

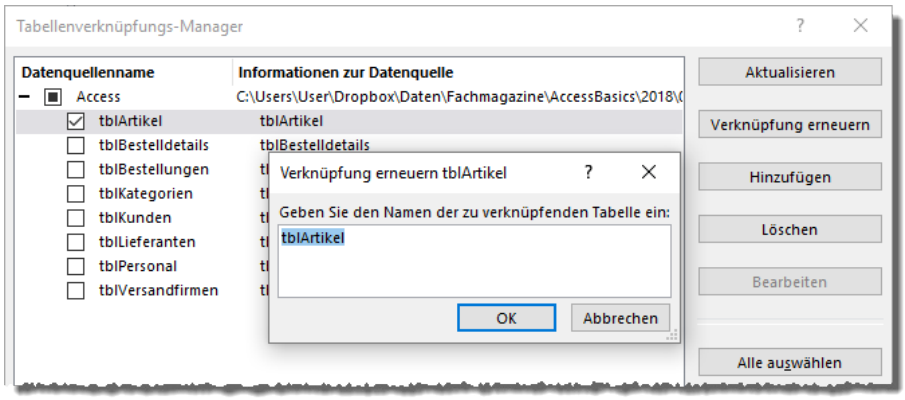


Bild 5: Versuch, eine einzige Verknüpfung zu aktualisieren

Der neue Tabellenverknüpfungs-Manager

Nun bietet sich allerdings ein völlig anderes Bild - siehe Bild 4. Die erste gravierende Änderung: Die Verknüpfungen werden nach der Quelle gruppiert und die Einträge einer jeden Quelle können ein- oder ausgeklappt werden. Das ist besonders hilfreich, wenn Sie Verknüpfungen aus mehreren Datenbanken haben – beispielsweise aus einem Access-Backend und einer SQL Server-Datenbank. Wenn Sie in diesem Fall früher etwa nur die Tabellen der SQL Server-Datenbank aktualisieren wollten, mussten Sie diese mühselig heraussuchen, was bei vielen Tabellen, die auch noch alphabetisch angeordnet waren, nicht gerade wenig fehleranfällig war. Nun finden Sie die Einträge zu einer Datenquelle nicht nur als Gruppe vor, sondern Sie können mit einem Klick auf die Datenquellendatei sogar noch alle enthaltenen Tabellen selektieren.

Nun finden Sie also als erste Änderung die Gruppierung nach der Quelldatenbank. Bekannt sind die Schaltflächen **Alle auswählen** und **Keine auswählen**. Davon ab gab es zuvor nur die Schaltfläche **OK**, mit der man die Aktualisierung der ausgewählten Tabellen starten konnte. In der neuen Version ist **Aktualisieren** eine eigene Schaltfläche, welche die gewählten Einträge aktualisiert. Im unteren Bereich können Sie mit den beiden Schaltflächen **Alle Ebenen erweitern** und **Alle Ebenen reduzieren** die in den jeweiligen Verknüpfungsquellen enthaltenen Tabellen ein- und ausblenden.

Neu sind außerdem die Schaltfläche **Verknüpfung erneuern**, **Hinzufügen**, **Löschen** und **Bearbeiten**. Diese schauen wir uns in den nächsten Abschnitten an.

Verknüpfung erneuern

Nehmen wir an, wir hätten den oben beschriebenen Fall, dass die Backend-Datei mit den verknüpften Tabellen den Speicherort gewechselt hat. Wir wollen die Verknüpfung nun aktualisieren. Wenn wir dies nur an einer einzigen Tabelle vollziehen wollen, gelingt das

Eingabeformate

Wenn Felder immer gleich aufgebaute Daten enthalten, können Sie dem Benutzer die Eingabe der Daten mit Eingabeformaten erleichtern. Dabei kann es sich um Telefonnummern, Postleitzahlen, Datumsangaben oder auch eigene, benutzerdefinierte Angaben handeln. Wenn Sie beispielsweise das Verfallsdatum von Lebensmitteln im Format mm/yyyy angeben wollen, also etwa 12/2018, dann kann ein Eingabeformat die Benutzer der Datenbank unterstützen, die Daten korrekt einzugeben. Dieser Artikel zeigt, wie Sie Eingabeformate definieren und wie Sie diese einsetzen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1805_Eingabeformate.accdb**.

Eingabeformate definieren

Eingabeformate definieren Sie an verschiedenen Stellen. Wenn Sie möchten, dass sich das Eingabeformat an allen Stellen wiederspiegelt, an denen das damit versehene Tabellenfeld auftaucht (also etwa in Abfragen oder Formularen), dann definieren Sie das Eingabeformat direkt in der Tabelle. Sie können das Eingabeformat auch erst in einer Abfrage definieren, die auf der Tabelle mit dem betroffenen Feld aufbaut.

Oder Sie gehen noch einen Schritt weiter und statten erst das Steuerelement eines Formulars, dass an das betroffene Feld gebunden ist, mit einem Wert für die Eigenschaft **Eingabeformat** aus. Wenn Sie das Eingabeformat also bereits im Tabellenentwurf definieren, wird das Eingabeformat auch in Abfragen und Formularen, die diese Tabelle als Datenherkunft verwenden, angewendet.

Gültige Zeichen in Eingabeformaten

Bevor wir uns ansehen, wie Sie ein Eingabeformat zuweisen, schauen wir und die verfügbaren Optionen für die Gestaltung eines Platzhalters an.

Es gibt die folgenden möglichen Zeichen:

- **0**: Pflichteingabe für eine Ziffer von 0 bis 9.
- **9**: Freiwillige Eingabe einer Ziffer von 0 bis 9.
- **#**: Freiwillige Eingabe einer Ziffer von 0 bis 9, eines Leerzeichens, eines Plus- oder Minuszeichens
- **L**: Pflichteingabe für einen Buchstaben.
- **?**: Freiwillige Eingabe eines Buchstaben.
- **A**: Verpflichtender Platzhalter für einen Buchstaben oder eine Ziffer.
- **a**: Freiwillige Eingabe eines Buchstabens oder einer Ziffer.
- **&**: Verpflichtende Eingabe eines Zeichens oder eines Leerzeichens.
- **C**: Freiwillige Eingabe eines Zeichens oder Leerzeichens.
- Punkt (**.**): Platzhalter für Tausender-Trennzeichen
- Komma (**,**): Platzhalter für Dezimalstellen
- Punkt (**.**), Doppelpunkt (**:**), Bindestrich (**-**) und Schrägstrich (**/**): Trennzeichen für Uhrzeit- und Datumsangaben
- **>**: Wandelt alle folgenden Zeichen in Großbuchstaben um.
- **<**: Wandelt alle folgenden Zeichen in Kleinbuchstaben um.

- !: Das Eingabeformat wird von links nach rechts aufgefüllt statt von rechts nach links.
- \: Das folgende Zeichen wird als Literal angezeigt.
- "<Zeichenkette>": <Zeichenkette> wird als Literal angezeigt.

Eingabeformat anlegen

Damit können wir nun unser erstes Eingabeformat anlegen, in diesem Fall für ein Feld namens **HaltbarBis**. Das Feld soll zwei Ziffern für den Monat, einen Schrägstrich und vier Ziffern für das Jahr aufnehmen. Dazu verwenden wir den folgenden Ausdruck:

```
00\/0000
```

Diesen tragen wir im Tabellenentwurf wie in Bild 1 für die Eigenschaft **Eingabeformat** ein. Die **0** sorgt dafür, dass der Benutzer nur verpflichtende Zahlenwerte eintragen kann, die Kombination aus Backslash und Schrägstrich (V) sorgt dafür, dass der Schrägstrich als Literal ausgegeben wird.

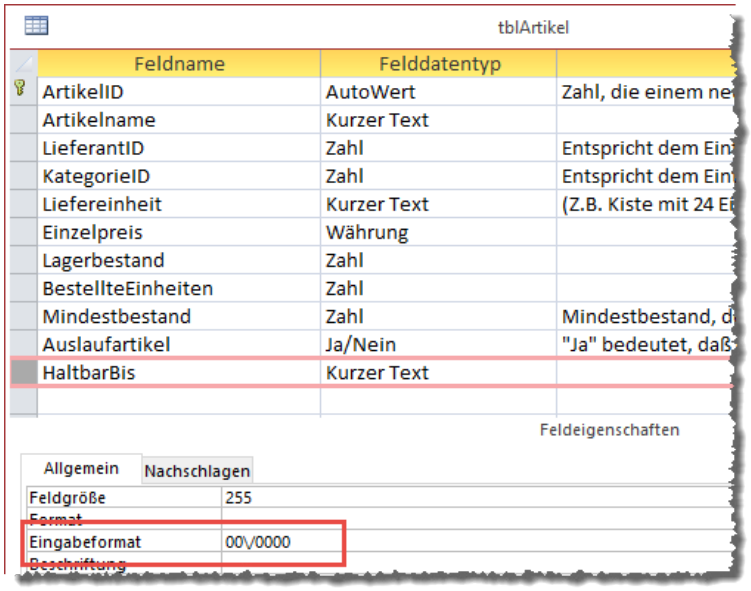


Bild 1: Formular-Entwurf unseres Beispielformulars

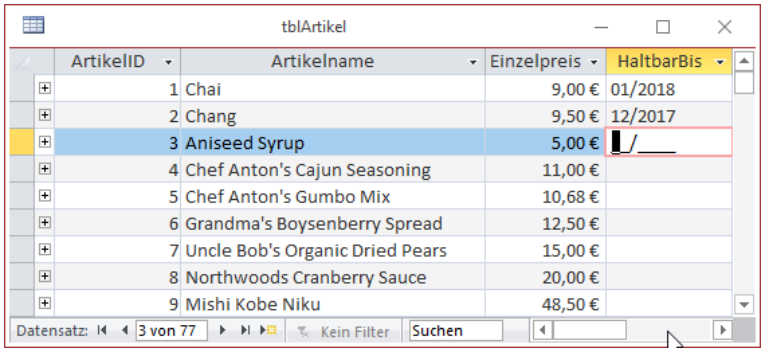


Bild 2: Eingabe mit Eingabeformat

Bei der Eingabe sieht das Eingabeformat dann wie in Bild 2 aus. Versucht der Benutzer, hier ein anderes Zeichen als eine Zahl einzugeben, gelingt dies nicht und es ertönt ein aktustisches Signal.

Wenn der Benutzer hingegen versucht, eine Abkürzung zu wählen und beispielsweise **1/18** eingeben will, erscheint eine entsprechende Meldung (siehe Bild 3). Diese Meldung könnte man gegebenenfalls noch optimieren, denn der Benutzer weiß vermutlich nichts mit dem Ausdruck **00V0000** anzufangen.

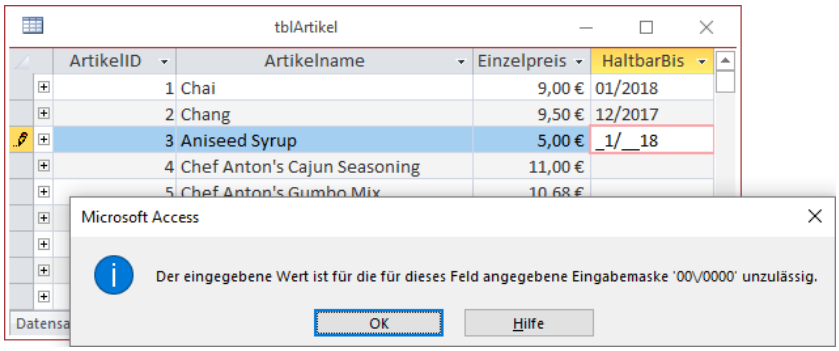


Bild 3: Meldung bei Verletzung des Eingabeformats

Platzhalterzeichen anpassen

Im Beispiel haben Sie nun gesehen, dass Access den Unterstrich als Platzhalterzeichen verwendet. Dies können Sie auch anpassen und etwa das Leerzeichen, einen Bindestrich

Kombinationsfeld um Suche erweitern

Kombinationsfelder sind schon eine praktische Einrichtung: Sie erlauben nicht nur die Auswahl von Einträgen, die schon nach dem Alphabet voreingestellt sind, sondern auch noch die Eingabe der Anfangsbuchstaben der gesuchten Inhalte. Was aber, wenn Sie doch gezielter nach einem Eintrag im Kombinationsfeld suchen wollen – beispielsweise nach Einträgen, die bestimmte Vergleichstexte an beliebiger Stelle enthalten? Dieser Artikel zeigt, wie Sie für ein Kombinationsfeld nach Wunsch per Doppelklick oder über eine zusätzliche Schaltfläche einen Dialog öffnen, mit dem Sie schnell den gesuchten Eintrag auswählen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1805_Kombisuche.accdb**.

Vorbereitung

Wir wollen uns die eingebauten Suchfeatures und usnere neuen Funktionen an einem ungebundenen Kombinationsfeld ansehen, das wir in einem neuen Formular namens **frmKombisuche** anlegen.

Das Kombinationsfeld soll **cboArtikel** heißen und als Datensatzherkunft die folgende SQL-Abfrage verwenden:

```
SELECT ArtikelID, Artikelname
FROM tblArtikel
ORDER BY Artikelname;
```

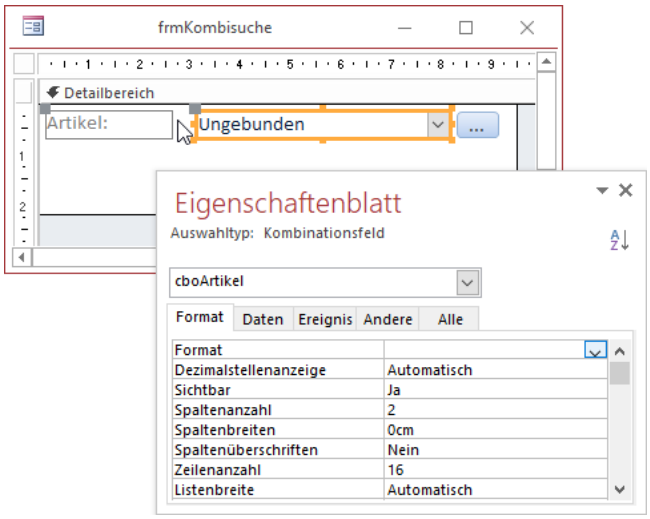


Bild 1: Entwurf des Formulars mit dem Kombinationsfeld

Es soll also die Einträge der Tabelle **tblArtikel** liefern, wobei wir nur die beiden Felder **ArtikelID** und **Artikelname** berücksichtigen. Diese sollen alphabetisch nach den Werten des Feldes **Artikelname** sortiert werden. Rechts neben dem Kombinationsfeld fügen wir noch eine Schaltfläche namens **cmdSuche** ein, der wir als Beschriftung schlicht drei Punkte hinzufügen.

Damit das Kombinationsfeld nur die Werte des Feldes **Artikelname** anzeigt, nicht jedoch die der gebundenen ersten Spalte **ArtikelID**, stellen wir die Eigenschaft **Spaltenanzahl** auf **2** und die Eigenschaft **Spaltenbreiten** auf **0cm** ein. Auf diese Weise wird die erste Spalte ausgeblendet (siehe Bild 1).

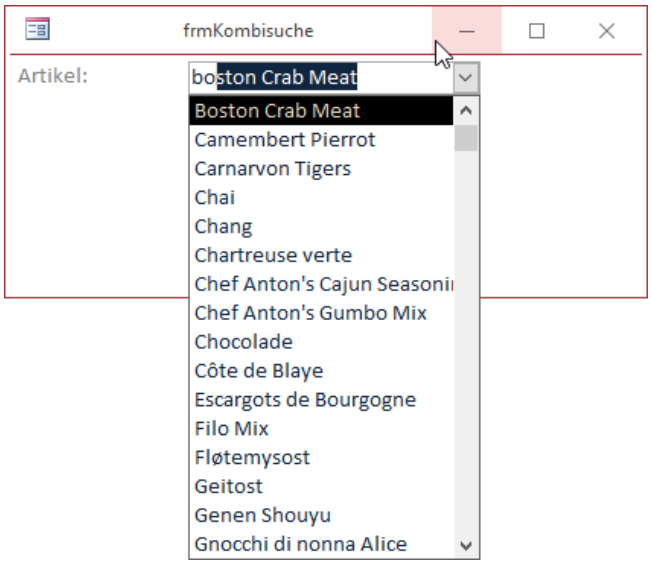


Bild 2: Markieren des Eintrags mit den eingegebenen Anfangsbuchstaben

Eingebaute Suchfeatures

Wenn wir nun in die Formularansicht des Formulars wechseln und das Kombinationsfeld aufklappen, zeigt dieses zunächst alle Einträge der Datensatzherkunft an. Geben Sie nun einen oder mehrere Buchstaben ein, springt das Kombinationsfeld automatisch zum ersten Eintrag, der mit den gleichen Buchstaben beginnt, und markiert diesen wie in Bild 2.

Das geschieht solange, bis sie einen Text eingegeben haben, der in den Werten des angezeigten Feldes nicht mehr gefunden wird – dann finden Sie nur noch den eingegebenen Text im Kombinationsfeld vor und die aufgeklappte Liste zeigt wieder alle Einträge der Datensatzherkunft an.

Diese Funktion ist also im Prinzip kein Filter, sondern nur ein Anspringen des ersten Eintrags, der den eingegebenen Zeichen entspricht. Damit wir noch gezielter suchen können, wollen wir nun noch ein zusätzliches Formular hinzufügen, dass per Doppelklick oder per Mausklick auf eine zusätzliche Schaltfläche neben dem Kombinationsfeld aufgerufen werden kann.

Suchformular anlegen

Das Suchformular wollen wir **frmArtikelsucheKombi** nennen. Es soll ein Textfeld enthalten, mir dem der Benutzer den Suchbegriff eingibt sowie ein Listenfeld, das die gefundenen Einträge der Datenherkunft liefert.

Da das Formular selbst nicht an eine Datenherkunft gebunden sein soll, können wir die Eigenschaften **Datensatzmarkierer**, **Navigationsschaltflächen**, **Trennlinien** und **Bildlaufleisten** auf den Wert **Nein** einstellen. Außerdem soll die Eigenschaft **Automatisch zentrieren** den Wert **Ja** erhalten.

Oben im Formular fügen wir das Textfeld namens **txtSuche** ein. Darunter legen wir ein Listenfeld namens **lstSuchergebnis** an. Dieses binden wir über die Eigenschaft **Datensatzherkunft** zunächst an die gleiche Abfrage, die wir bereits weiter oben für das Kombinationsfeld selbst genutzt haben:

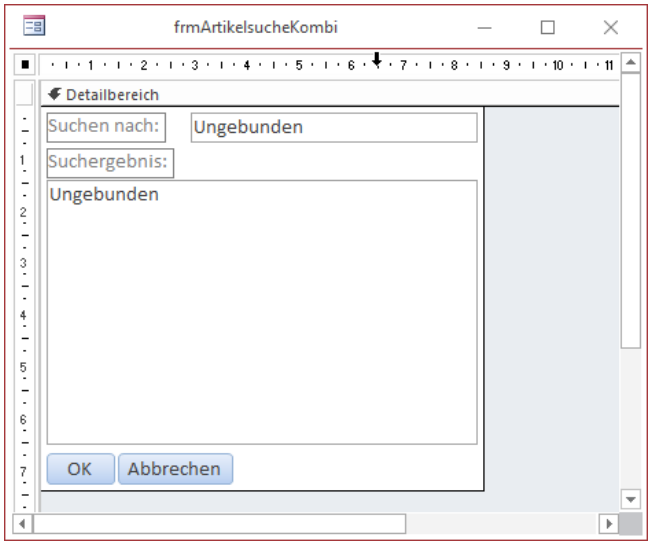


Bild 3: Entwurf des Suchformulars

```
SELECT ArtikelID, Artikelname
FROM tblArtikel
ORDER BY Artikelname;
```

Damit das Listenfeld, genau wie das Kombinationsfeld, nur die Artikelnamen anzeigt, aber nicht die Werte des Feldes **ArtikelID**, stellen wir auch hier die Eigenschaften **Spaltenanzahl** und **Spaltenbreiten** auf die Werte **2** und **0cm** ein.

Außerdem fügen wir noch eine Schaltfläche namens **cmdOK** und eine weitere Schaltfläche namens **cmdAbbrechen** zum Formularentwurf hinzu, der dann wie in Bild 3 aussieht.

Öffnen des Suchformulars

Bevor wir uns nun um die Funktionalität des Suchformulars kümmern, wollen wir erst einmal den Aufruf dieses Formulars realisieren. Dieser soll über zwei Möglichkeiten realisiert werden:

- einen Doppelklick auf das Kombinationsfeld und
- einen Klick auf die Schaltfläche **cmdSuche** rechts neben dem Kombinationsfeld.

Also hinterlegen wir eine Ereignisprozedur für die Ereigniseigenschaft **Beim Doppelklicken** des Kombinationsfeldes und eine für die Eigenschaft **Beim Klicken**

der Schaltfläche **cmdSuche**. Für beide tragen wir den Aufruf einer weiteren Prozedur namens Kombisuche ein:

```
Private Sub cboArtikel_DblClick(Cancel As Integer)
    Call Kombisuche
End Sub

Private Sub cmdSuche_Click()
    Call Kombisuche
End Sub
```

Der Hintergrund ist, dass beide die gleiche Aktion auslösen sollen – nämlich die Anzeige des Suchformulars und die Verarbeitung des dort gewählten Suchergebnisses. Also legen wir die dazu notwendigen Anweisung nur in einer einzigen Prozedur an, die dann von beiden Ereignisprozeduren aufgerufen wird. Auf diese Weise sparen wir uns redundanten Code.

Die durch diese beiden Ereignisprozeduren ausgelöste Prozedur sieht zunächst wie folgt aus:

```
Private Sub Kombisuche()

End Sub
```

Diese werden wir aber schon bald mit den notwendigen Anweisungen füllen. Vorher überlegen wir uns, was genau geschehen soll, wenn wir entweder einen Doppelklick auf dem Kombinationsfeld ausführen oder die Schaltfläche **cmdKombisuche** anklicken. Fest steht: Wir wollen das Formular **frmArtikelsucheKombi** öffnen, damit es uns die gezielte Suche nach einem der Einträge des Kombinationsfeldes erlaubt.

Was soll dann geschehen? Der Benutzer sucht nach einem Eintrag, der dann im Listenfeld markiert wird. Dann klickt er entweder auf die Schaltfläche **cmdOK** oder **cmdAbbrechen**. Klickt er auf **cmdOK** und es ist ein Eintrag markiert, soll dieser für das Kombinationsfeld ausgewählt werden. Das Formular muss dann noch geöffnet bleiben, damit wir den Wert auslesen können.

Klickt er auf **cmdOK** und es ist kein Eintrag markiert oder klickt er auf **cmdAbbrechen**, wird das Formular einfach geschlossen, denn dann brauchen wir ja auch keinen Wert auszulesen.

Wir nutzen also die bekannte Vorgehensweise: Wir öffnen das Formular als modalen Dialog, also in der Art, dass der Benutzer erst zu den übrigen Elementen der Benutzeroberfläche zurückkehren kann, wenn er das aufgerufene Formular entweder geschlossen oder es unsichtbar gemacht hat. In der aufrufenden Prozedur, die dann den Fokus zurückerhält und weiter ausgeführt wird, brauchen wir dann nur zu prüfen, ob das Formular noch geöffnet ist (allerdings unsichtbar) oder ob es nicht mehr geöffnet ist.

Dazu nutzen wir die folgende einfache Funktion, die wir in einem neuen Standardmodul namens **mdlTools** speichern:

```
Public Function IstFormularGeoeffnet(_
                    strFormular As String) As Boolean
    IstFormularGeoeffnet = SysCmd(_
       acSysCmdGetObjectState, acForm, strFormular) > 0
End Function
```

Die Prozedur, die durch unsere Ereignisprozeduren ausgelöst wird, gestalten wir dann wie in Listing 1.

Nach dem Öffnen des Formulars **frmArtikelsucheKombi** als modalen Dialog wird die Prozedur unterbrochen, bis das Formular **frmArtikelsucheKombi** entweder geschlossen oder ausgeblendet wurde. Im letzteren Fall ermitteln wir aus dem Listenfeld den Primärschlüsselwert des gewählten Eintrags und tra-

```
Private Sub Kombisuche()
    DoCmd.OpenForm "frmArtikelsucheKombi", WindowMode:=acDialog
    If IstFormularGeoeffnet("frmArtikelsucheKombi") Then
        Me!cboArtikel = Forms!frmArtikelsucheKombi!lstSuchergebnis
        DoCmd.Close acForm, "frmArtikelsucheKombi"
    End If
End Sub
```

Listing 1: Öffnen des Formulars für die Suche und Einstellen des gefundenen Wertes, wenn das Formular unsichtbar gemacht wurde

Formulare per VBA referenzieren

Wenn Sie Formulare und Steuerelemente programmieren wollen, müssen Sie wissen, wie Sie diese referenzieren. Nicht immer geschieht dies vom Klassenmodul des Formulars selbst aus – manchmal wollen Sie vielleicht auch von anderen Formularen auf die Steuerelemente eines anderen Formulars zugreifen oder auch von einer VBA-Prozedur in eine Standardmodul. Dieser Artikel zeigt, wie Sie von verschiedenen Orten aus auf die Formulare zugreifen können. In weiteren Artikeln betrachten wir dann etwa den Zugriff auf die darin enthaltenen Steuerelemente.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1803_FormsControlsVBA-Read.accdb**.

Innerhalb des Formulars

Wenn Sie ein Formular erstellen und diesem ein Klassenmodul zuweisen, das sogenannte Code behind-Modul, dann können Sie von diesem aus auf andere Weise auf die Elemente des Formulars zugreifen als dies etwa von einem anderen Formular oder auch von einem Standardmodul aus möglich ist.

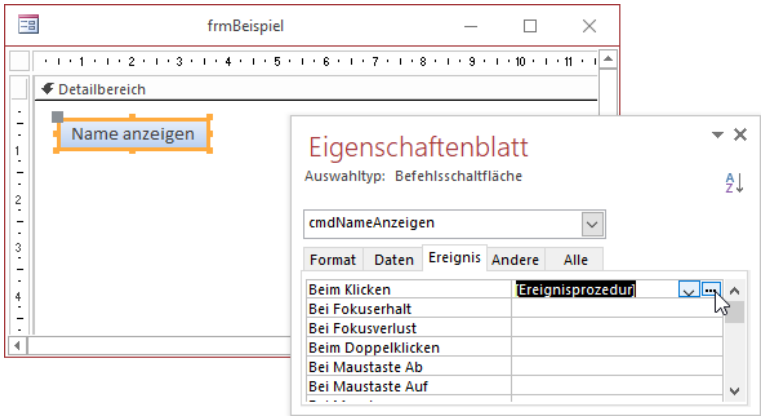


Bild 1: Hinzufügen einer Ereignisprozedur für eine Schaltfläche

Hier steht nämlich das **Me**-Schlüsselwort für den Zugriff auf die Objektvariable des aktuellen Formulars zur Verfügung. Zur Verdeutlichung legen Sie einfach einmal ein neues Formular in einer leeren Datenbank an und speichern dieses unter dem Namen **frmBeispiel**.

Diesem fügen wir nun eine Schaltfläche hinzu, die wir mit der Beschriftung **Name anzeigen** und dem Namen **cmdNameAnzeigen** versehen und für die wir eine neue Ereignisprozedur für das Ereignis **Beim Klicken** anlegen (siehe Bild 1).

Der Ereignisprozedur fügen wir nun eine Anweisung hinzu, die den Namen des Formulars in einem Meldungsfenster ausgeben soll:

```
Private Sub cmdNameAnzeigen_Click()
    MsgBox Me.Name
End Sub
```

Das hier wie selbstverständlich verwendete Schlüsselwort **Me** ist eine sehr praktische Vereinfachung für den Zugriff auf die Elemente des Formulars, auf das sich das Klassenmodul bezieht.

Mit diesem Schlüsselwort können Sie auf alle Eigenschaften und Methoden des Formulars zugreifen und auch auf die darin angelegten Steuerelemente – darauf gehen wir später ein.

Von außen

Wie aber greifen wir denn eigentlich von außen auf dieses Formular zu? Dazu müssen wir erst einmal wissen, ob das Formular geöffnet ist. Greifen wir auf ein Formular zu, das nicht geöffnet ist, erhalten wir einen Laufzeitfehler.

Um dies auszuprobieren, fügen Sie der Beispieldatenbank ein Standardmodul namens **mdlFormularzugriff** hinzu. Hier legen Sie die folgende Prozedur an:

```
Public Sub NameAnzeigen()
    MsgBox Forms!frmBeispiel.Name
End Sub
```

Die Prozedur erledigt nichts anders als die im Formular für die Schaltfläche **cmdName-Anzeigen** – Sie gibt ein Meldungsfenster mit dem Namen des Formulars aus.

Allerdings wissen wir, wenn wir auf die Schaltfläche des Formulars klicken, dass dieses logischerweise gerade geöffnet ist und brauchen dies nicht zu prüfen. Beim Zugriff von außen müssen wir dies jedoch vorher absichern. Rufen Sie die obige Prozedur auf, nachdem Sie das Formular geschlossen haben, löst dies den Fehler aus Bild 2 aus.

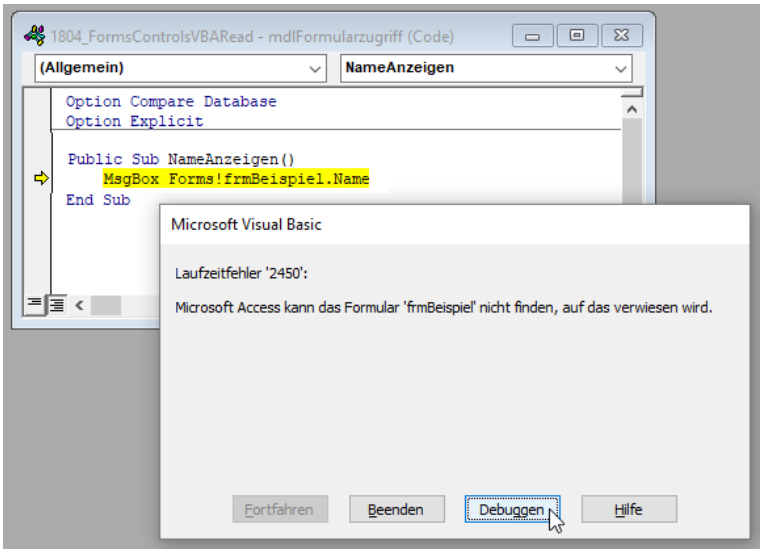


Bild 2: Fehler beim Zugriff auf ein nicht geöffnetes Formular

Prüfen, ob ein Formular geöffnet ist

Für die Prüfung, ob ein Formular geöffnet ist, hat sich die folgende Funktion etabliert. Sie erwartet den Namen des zu untersuchenden Formulars als Parameter. Sie nutzt die **SysCmd**-Funktion, der sie die Konstante **acSysCmdGetObjectState**, den Objekttyp undd den Namen des Formulars übergibt. Wenn das Ergebnis größer als **0** ist, ist das Formular geöffnet:

```
Public Function IstFormularGeoeffnet(strFormular As _
                                String) As Boolean
    IstFormularGeoeffnet = SysCmd( _
       acSysCmdGetObjectState, acForm, strFormular) > 0
End Function
```

Mit dieser Funktion können wir bei Zugriff von außen leicht prüfen, ob das Formular geöffnet ist. Die obige Prozedur erweitern wir dabei wie folgt:

```
Public Sub NameAnzeigen()
    If IstFormularGeoeffnet("frmBeispiel") Then
        MsgBox Forms!frmBeispiel.Name
    Else
        MsgBox "Das Formular ist nicht geöffnet."
    End If
End Sub
```

Wenn das Formular nicht geöffnet ist, weist die Prozedur per Meldungsfenster auf diesen Zustand hin.

Auf das aktuelle Formular zugreifen

In manchen Fällen möchten Sie auch einfach auf das aktuell geöffnete Formular zugreifen, um beispielsweise dessen Namen zu ermitteln. Hierfür stellt Microsoft eine Klasse mit verschiedenen Eigenschaften bereit. Diese heißt **Screen** und liefert beispielsweise über die Eigenschaft **ActiveForm** Zugriff auf das aktuell aktive Formular. Die folgende Prozedur gibt beispielsweise den Namen des aktuellen Formulars aus:

```
Public Sub AktuellesFormular()
    MsgBox Screen.ActiveForm.Name
End Sub
```

Wenn Sie diese Prozedur ausführen und es ist kein Formular geöffnet, erhalten Sie wiederum eine Fehlermeldung – diesmal die aus Bild 3. Dieser Fehler kann übrigens auch dann auftreten, wenn ein Formular geöffnet ist! Die **ActiveForm**-Eigenschaft ermittelt nämlich tatsächlich nur das aktive Formular, wenn es sich bei dem aktuell im Access-Fenster aktiven Element um ein Formular handelt. Wenn gerade eine Tabelle und ein Formular angezeigt werden, aber die Tabelle den Fokus hat, erhalten Sie den gleichen Fehler, als wenn gar kein Formular geöffnet ist.

Steuerelemente per VBA referenzieren

Wenn Sie Formulare und Steuerelemente programmieren wollen, müssen Sie wissen, wie Sie diese referenzieren. Nicht immer geschieht dies vom Klassenmodul des Formulars selbst aus – manchmal wollen Sie vielleicht auch von anderen Formularen auf die Steuerelemente eines anderen Formulars zugreifen oder auch von einer VBA-Prozedur in eine Standardmodul. Dieser Artikel zeigt, wie Sie die Unterformulare und Steuerelemente eines Formulars referenzieren und auf diese zugreifen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1805_FormsControlsVBA-Read.accdb**.

Innerhalb des Formulars

Im Artikel **Formulare per VBA referenzieren** haben Sie bereits erfahren, dass ein Formular ein Klassenmodul erhält, sobald Sie Ereignisprozeduren für das Formular oder die enthaltenen Steuerelemente anlegen. In diesem Klassenmodul können Sie mit dem Schlüsselwort **Me** auf das Formular und seine Eigenschaften zugreifen, während Sie von außen auf andere Art auf das Formular zugreifen müssen. Für den Zugriff auf die in einem Formular enthaltenen Elemente wie Unterformulare oder andere Steuerelemente gelten daher auch verschiedene Regeln, auch wenn Sie letztendlich auf das gleiche Element zugreifen wollen. Allerdings können wir das entschärfen, indem wir die Regeln dafür nochmal zusammenfassen. Wenn Sie also auf Steuerelemente in einem Formular zugreifen wollen, verwenden Sie abhängig vom Ausgangspunkt die folgenden Referenzen für das Formular:

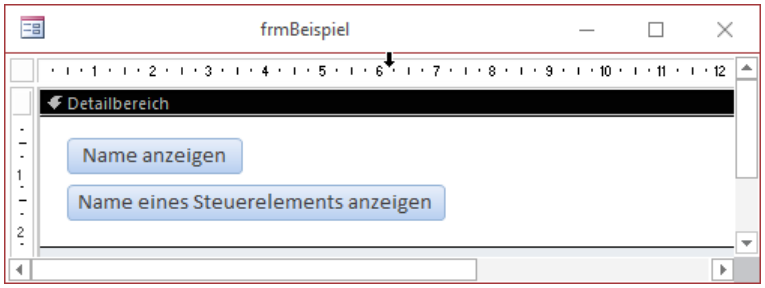


Bild 1: Schaltfläche zum Ermitteln des Namens einer anderen Schaltfläche

Im eigenen Klassenformular:

```
Me...
```

Von außen:

```
Forms!<Formularname>...
Forms("<Formularname>")...
```

Etwas komplizierter wird es noch, wenn sich die zu referenzierenden Steuerelemente in einem Unterformular befinden – dazu jedoch später mehr.

Über den Namen auf ein Steuerelement zugreifen

Die einfachste Methode, auf ein Steuerelement zuzugreifen, ist die über den Namen des Steuerelements. Dazu fügen wir dem Beispielformular **frmBeispiel** zwei Schaltflächen hinzu, von denen die zweite den Namen der ersten Schaltfläche ausgeben soll (siehe Bild 1). In der Ereignisprozedur, die durch die zweite Schaltfläche ausgelöst wird, referenzieren wir die erste Schaltfläche dann über die Syntax mit dem Ausrufezeichen und geben den Namen per **MsgBox** aus:

```
Private Sub cmdNameEinesSteuerelements_Click()
    MsgBox Me!cmdNameAnzeigen.Name
End Sub
```

Die zweite Möglichkeit ist die Angabe des Steuerelementnamens als Parameter des **Me**-Schlüsselworts:

```
MsgBox Me("cmdNameAnzeigen").Name
```

Dies ist allerdings auch nur möglich, weil die Auflistung **Controls** die Standardeigenschaft des mit dem Schlüsselwort **Me** referenzierten Objekts, in diesem Fall des Formulars, ist. Die ausgeschriebene Form dieser Schreibweise bezieht die **Controls**-Auflistung dann explizit mit ein:

```
MsgBox Me.Controls("cmdNameAnzeigen").Name
```

Dabei fällt mir ein, dass man mal prüfen könnte, welchen Typ das Schlüsselwort **Me** eigentlich hat. Ist es **Form**? Nein: Es handelt sich um die Code behind-Klasse des Formulars selbst:

```
Debug.Print TypeName(Me)
  Form_frmBeispiel
```

Für weitere Experimente fügen wir dem Formular noch ein Textfeld namens **txtBeispiel** hinzu sowie eine weitere Schaltfläche namens **cmdTextfeldAusgeben** (siehe Bild 2). Den Inhalt des Textfeldes legen wir in einer Ereignisprozedur fest, die beim Laden des Formulars ausgelöst wird:

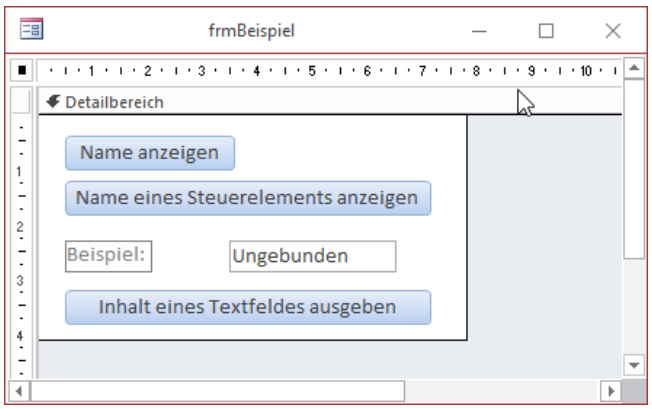


Bild 2: Neues Textfeld zum Experimentieren

```
Private Sub Form_Load()
    Me!txtBeispiel = "Beispieltext"
End Sub
```

Warum Ausrufezeichen und kein Punkt?

Theoretisch kommen Sie in den meisten Fällen auch ohne Fehler aus, wenn Sie zwischen Formularbezeichner (also etwa **Me** oder **Forms!<Formularname>**) und Steuerelementname den Punkt als Trennzeichen angeben. Warum aber verwenden wir hier so beharrlich das Ausrufezeichen? Dieses sollten Sie immer dann nutzen, wenn Sie Elemente von Auflistungen referenzieren, die Sie selbst angelegt haben beziehungsweise anlegen können – also etwa Formulare (**Forms!<Formularname>**) oder Steuerelemente in Formularen (**Forms!<Formularname>!<Steuerelementname>**).

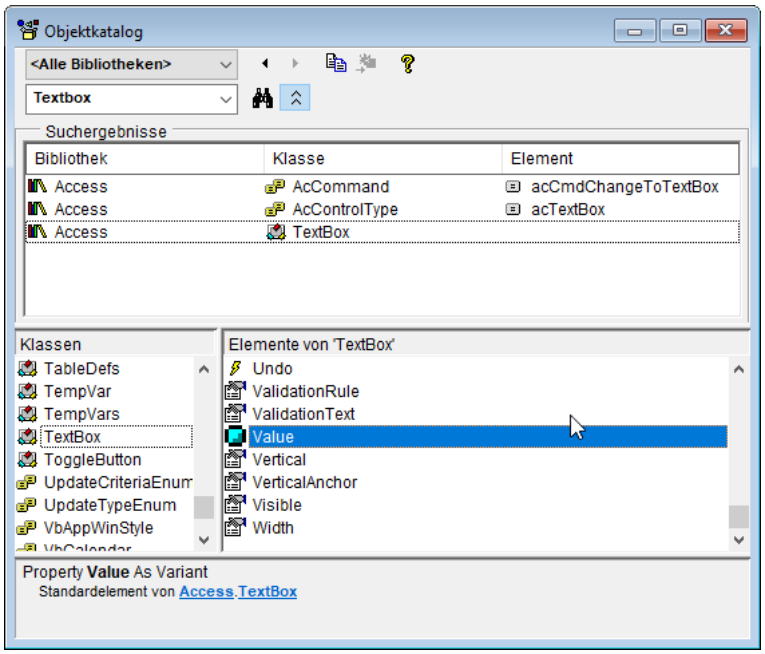


Bild 3: Standardeigenschaft einer Klasse im Objektkatalog

Zugriff auf Textfelder von verschiedenen Stellen

Wenn wir vom Formular selbst – in diesem Fall über die Schaltfläche **cmdTextfeldAusgeben** – auf den Inhalt dieses Textfeldes zugreifen wollen, verwenden wir, wie oben gelernt, das Schlüsselwort **Me** sowie das Ausrufezeichen und den Namen des **Textfeldes**:

```
Private Sub cmdTextfeldAusgeben_Click()
    MsgBox Me!txtBeispiel
End Sub
```

Tatsächlich ist dies wiederum eine Kurzform. Wir geben ja nicht das Textfeld **txtBeispiel** aus, sondern den Inhalt dieses Textfeldes. Die Kurzform funktioniert deshalb, weil die Eigenschaft **Value** des Textfeldes die Standardeigenschaft der Textfeld-Klasse ist. Ohne diese Kurzform müssten Sie also mit dem folgenden Asudruck arbeiten:

```
MsgBox Me!txtBeispiel.Value
```

Woher wissen wir so etwas? Dies erfahren wir beispielsweise im Objektkatalog, den Sie über den Menüeintrag **Ansicht|Objektkatalog** öffnen (siehe Bild 3). In der Liste der Eigenschaften finden Sie zum Beispiel für das Textfeld (**TextBox**) ein spezielles Icon für die Eigenschaft **Value**. Unten finden Sie dann den Text **Standardelement von Access.TextBox** – das ist eindeutig.

Das waren nun zwei Kurzform. Genau wie beim Referenzieren von Formularen können Sie aber auch noch über die Schreibweise mit Klammern auf die Steuerelemente eines Formulars zugreifen. Die erste Variante nimmt den Namen des Formulars entgegen:

```
Debug.Print Me("txtBeispiel").Value
```

Das war wiederum die Kurzform für die Variante mit der **Controls**-Auflistung:

```
Debug.Print Me.Controls("txtBeispiel").Value
```

Und wenn Sie den Index-Wert des Steuerelements kennen, können Sie auch noch einen numerischen Wert angeben:

```
Debug.Print Me.Controls(2).Value
```

Zugriff auf das aktive Steuerelement

Im Artikel zum Thema Formulare haben Sie die Eigenschaft **ActiveForm** des **Screen**-Objekts kennengelernt, mit der Sie das aktive Formular referenzieren konnten. Ein Pendand zu dieser Eigenschaft gibt es auch für Steuerelemente. Wenn Sie also das aktuell aktive Steuerelement ermitteln wollen, nutzen Sie die Eigenschaft **ActiveControl** des **Screen**-Objekts:

```
Debug.Print Screen.ActiveControl.Name
```

Dies liefert beispielsweise den Namen des aktiven Steuerelements. Was geschieht hier beispielsweise, wenn derzeit kein Formular, sondern etwa eine Tabelle den Fokus hat? Dann liefert **Screen.ActiveControl** den Namen des Feldes der Tabelle, das gerade markiert ist! Im Falle der Tabelle **tblBeispiel** unserer Beispieldatenbank sieht der Aufruf im Direktbereich etwa wie folgt aus:

```
Debug.Print Screen.ActiveControl.Name
ID
```

Können wir darüber auch auf den Namen der Tabelle zugreifen? Ja, auch das gelingt:

```
Debug.Print Screen.ActiveControl.Parent.Name
tblBeispiel
```

Dadurch angespornt lassen wir auch gleich noch den mit **TypeName** ermittelten Typ dieses Tabellenobjekts ausgeben:

```
Debug.Print TypeName(Screen.ActiveControl.Parent)
T_tblBeispiel
```

Dies ist dann also das Pendant zu dem Typ **Form_...**, der für ein Formular zurückgeliefert wird.

Wenn Sie also etwa per Code über **Screen.ActiveControl** auf ein Steuerelement eines Formulars zugreifen wollen, sollten Sie zuvor mit **Screen.ActiveForm** sicherstellen, dass auch gerade ein Formular den Fokus hat.

Steuerelemente durchlaufen

Sie haben nun schon gesehen, dass es eine Auflistung namens **Controls** gibt, mit der Sie Steuerelemente referenzieren können. Für Auflistungen können Sie auch die Anzahl der enthaltenen Elemente

bestimmen und Sie können die Elemente in verschiedenen Schleifentypen durchlaufen.

Wir fügen dem Formular zwei weitere Schaltflächen hinzu. Die erste heißt **cmdSteuerelementeDurchlaufenForNext** und löst die folgende Ereignisprozedur aus:

```
Private Sub cmdSteuerelementeDurchlaufenForNext_Click()
    Dim intCount As Integer
    Dim i As Integer
    intCount = Me.Controls.Count
    For i = 0 To intCount - 1
        Debug.Print Me.Controls(i).Name
    Next i
End Sub
```

Dies ist die Variante zum Durchlaufen der enthaltenen Steuerelemente, bei der wir zuerst die Anzahl der Steuerelemente ermitteln und dann eine **For...Next**-Schleife entsprechend der Anzahl der Steuerelemente durchlaufen. Innerhalb der Schleife referenzieren wir dabei das jeweilige Steuerelement über den Index.

Da dieser 0-basiert ist, beginnt die Schleife bei **0** und wir legen den Endpunkt der Schleife auf **intCount - 1** fest. **intCount** ermitteln wir zuvor über die Eigenschaft **Count** der **Controls**-Auflistung.

Die zweite Variante lösen wir mit der Schaltfläche **cmSteuerelementeDurchlaufenForEach** aus. Diese hat einen etwas eleganteren Ansatz: Sie durchläuft die **Controls**-Auflistung mit einer **For Each**-Schleife. Diese füllt die Elemente der Auflistung dann automatisch in die hinter **For Each** angegebene Variable, hier die Variable **ctl** vom Datentyp **Control**.

Control ist ein allgemeiner Datentyp, den Sie mit allen möglichen Steuerelementen füllen können. Innerhalb der Schleife gibt die Prozedur dann den Namen des mit **ctl** referenzierten Steuerelements aus:

```
Private Sub cmdSteuerelementeDurchlaufenForEach_Click()
    Dim ctl As Control
    For Each ctl In Me.Controls
```

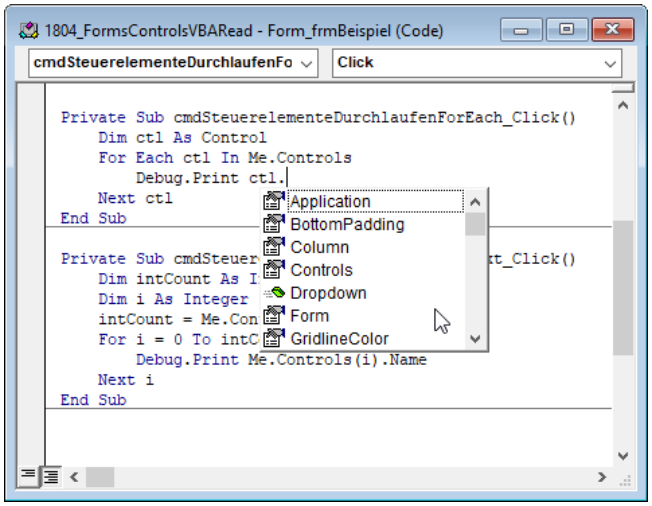


Bild 4: Eigenschaften der **Control**-Klasse

```
        Debug.Print ctl.Name
    Next ctl
End Sub
```

Steuerelementtypen

Wenn Sie sich die Auswahl der für die Klasse **Control** verfügbaren Eigenschaften, Methoden und Ereignisse ansehen (etwas per IntelliSense wie in Bild 4), scheint dies nur eine Teilmenge der für die verschiedenen Steuerelementtypen verfügbaren Member zu sein – in der Tat sind es nur rund 40. Schauen Sie sich dann etwas im Objektkatalog die Eigenschaften des Steuerelements **TextBox** an, finden Sie schon circa 160 Eigenschaften. Da scheint es logisch, dass Sie für volle Flexibilität auf das Steuerelement mit dem spezifischen Datentyp zugreifen wollen. Wenn Sie gezielt auf einzelne Steuerelemente zugreifen, wissen Sie natürlich, welchen Typ das jeweilige Steuerelement hat. Sie können es dann direkt mit dem passenden Datentyp deklarieren – beispielsweise wie folgt:

```
Dim txtBeispiel As TextBox
```

Was aber, wenn Sie etwa alle Steuerelemente durchlaufen wollen, aber dennoch die spezifischen Eigenschaften der jeweiligen Steuerelemente nutzen wollen? In diesem Fall müssen wir zuerst herausfinden, um welchen Steuerelementtyp es sich bei dem jeweiligen Objekt handelt.

Bedingte Formatierung per VBA

Im Artikel **Bedingte Formatierung** haben Sie in Ausgabe 11/2011 erfahren, wie Sie die bedingte Formatierung einsetzen, um die Steuerelemente in Formularen und Datenblättern in Abhängigkeit ihrer Werte mit bestimmten Formatierungen zu versehen – beispielsweise, um kritische Zahlenwerte mit einem roten Hintergrund zu versehen. Zwar können Sie seit Access 2010 bis zu 50 bedingte Formatierungen je Feld definieren, aber wer möchte das schon von Hand erledigen? Zu Glück können Sie auf diese Technik auch per VBA zugreifen und so die Formatierungen mit etwas mehr Hirnschmalz, aber weniger Handarbeit definieren.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1804_BedingteFormatierungPerVBA.accdb**.

Wunderwaffe VBA

Fast alles, was Sie unter Access über die Benutzeroberfläche erledigen können, lässt sich unter VBA regeln. In vielen Fällen ist das sehr hilfreich, vor allem dann, wenn Sie sich damit oft wiederholender oder umfangreicher Aufgaben entledigen können. Und das Setzen vieler bedingter Formatierungen für die Steuerelemente etwa eines Formulars oder Berichts ist eine der Aufgaben, die man lieber nicht von Hand erledigt.

Stattdessen könnte man die Zeit viel besser in die Programmierung stecken – zum Beispiel, um das Festlegen von bedingten Formatierungen zu automatisieren.

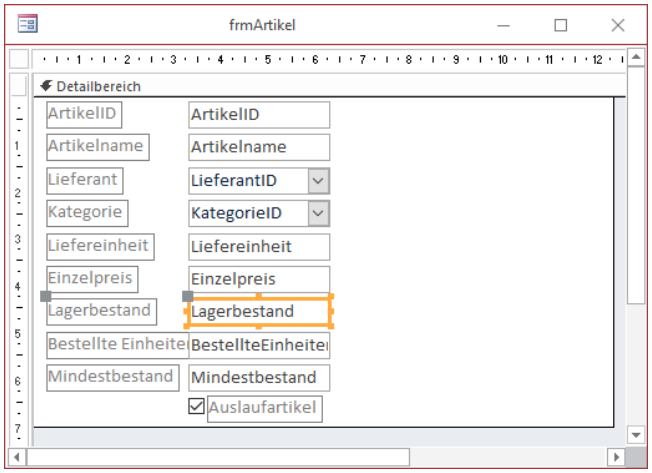


Bild 1: Formular-Entwurf unseres Beispielformulars

Erst lesen, dann schreiben

Um ins Thema einzusteigen, legen wir einmal zwei verschiedene Formatierungen für ein Feld eines Formulars in der Datenblattansicht an. Das Formular heißt **frmArtikel** und soll alle Datensätze der Tabelle **tblArtikel** anzeigen (siehe Bild 1). Wir wollen zunächst das Feld **Lagerbestand** mit einer bedingten Formatierung ausstatten, die dafür sorgt, dass das Feld rot angezeigt wird, wenn der Lagerbestand den Wert **0** aufweist. Dazu wählen wir aus dem Kontextmenü dieses Textfeldes den Eintrag **Bedingte Formatierung** aus und klicken im nun erscheinenden Dialog **Manager für Regeln zur bedingten Formatierung** auf die Schaltfläche **Neue Regel**.

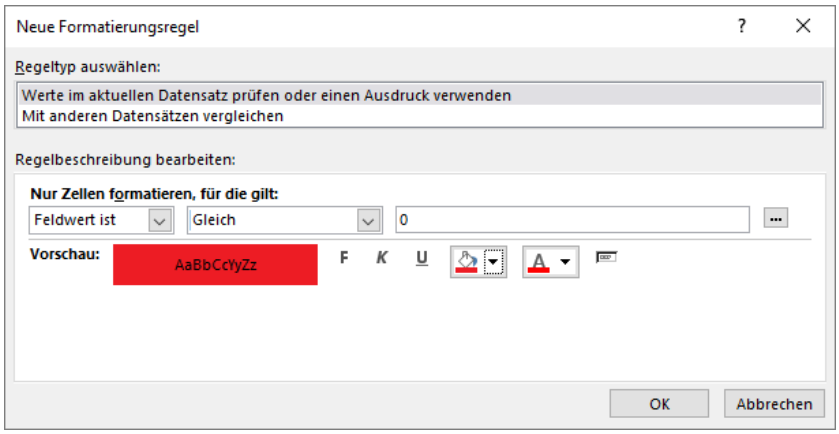


Bild 2: Anlegen einer neuen bedingten Formatierung

Dies öffnet den Dialog **Neue Formatierungsregel**, dem wir den Operator **Gleich** und den Vergleichswert **0** zuweisen. Außerdem stellen wir als Hintergrundfarbe die rote Farbe ein (siehe Bild 2).

Nach dem Mausklick auf die Schaltfläche **OK** schließt sich der Dialog und die neue Formatierungsregel wird in der Liste der Formatierungsregeln um aufrufenden Dialog angezeigt (siehe Bild 3).

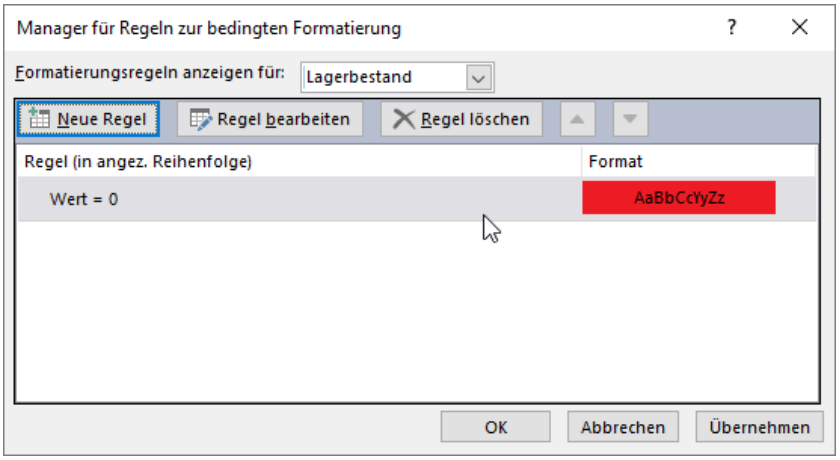


Bild 3: Liste der angelegten Formatierungsregeln

Für die übrigen Steuerelemente legen wir noch eine weitere Formatierungsregel an, welche die übrigen Steuerelemente mit Ausnahme des Kontrollkästchens für das Feld **Auslaufartikel** ebenfalls rot färbt, wenn das Feld **Lagerbestand** den Wert **0** annimmt.

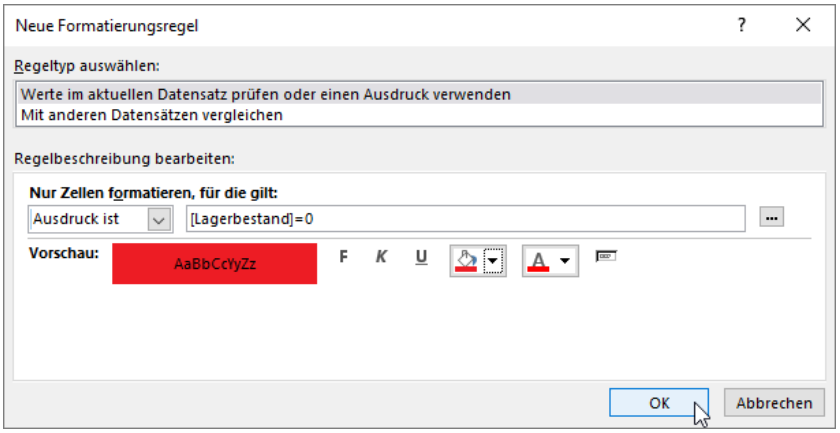


Bild 4: Anlegen einer weiteren neuen bedingten Formatierung

Hierbei ist interessant, dass beim Markieren mehrerer Steuerelemente gleichzeitig der Befehl **Bedingte Formatierung** nicht mehr im Kontextmenü vorhanden ist, aber immer noch im Ribbon unter **Format|Steuerelementformatierung|Bedingte Formatierung** erscheint. Für diese Felder legen wir eine bedingte Formatierung an, welche den Ausdruck **[Lagerbestand]=0** enthält und dabei den Hintergrund rot färbt (siehe Bild 4). Damit haben wir nun Spielmaterial, das wir per VBA erforschen können.

Bedingte Formatierung per VBA

Wir gehen davon aus, dass es unter VBA der Benutzeroberfläche zum Anlegen bedingter Formatierungen entsprechende Elemente gibt – also beispielsweise eine Auflistung aller bedingten Formatierungen und auch ein Objekt, das eine einzelne bedingte Formatierung bereitstellt und ihre Bearbeitung ermöglicht. Die Auflistung sollte ihrerseits das Hinzufügen, Referenzieren oder Entfernen von bedingten Formatierungen erlauben. Also werfen wir einen Blick in den Objektkatalog des VBA-Editors und suchen nach einem Schlüsselwort wie etwa **Format**.

Im ersten Schritt finden wir so schon einmal die drei Eigenschaften namens **ConditionalFormats** für die Steuerelemente **TextBox**, **ComboBox** und **NavigationControl** (siehe Bild 5).

Klicken wir hier auf den Link mit der Beschriftung **FormatConditions** im unteren Bereich, landen wir schnel bei der Auflistung **FormatConditions** mit den Methoden **Add** und **Delete** und den Eigenschaften **Count**, **Item** und **Parent** und direkt darüber entdecken wir die Klasse **FormatCondition** (siehe Bild 6). Hier finden wir alles, was wir brauchen, um die Schritte zum Hinzufügen und Auslesen von bedingten Formatierungen nachzubilden.

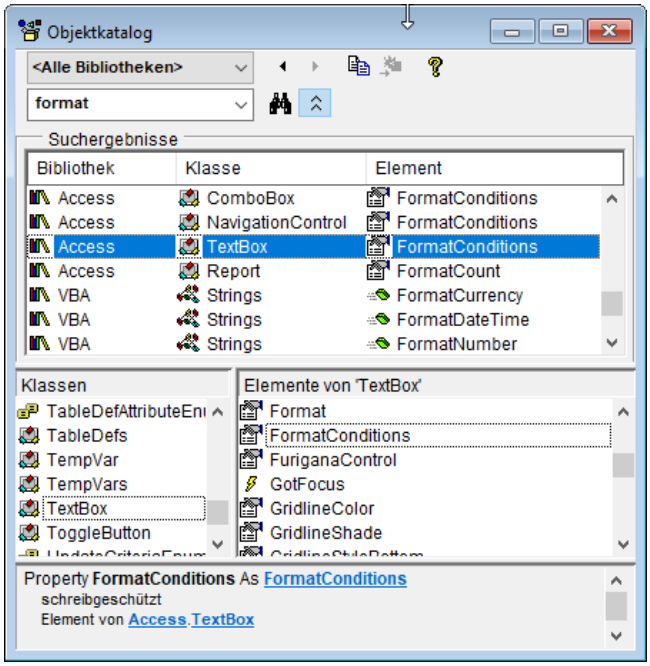


Bild 5: Suche nach der Auflistung mit dem bedingten Formatierungen

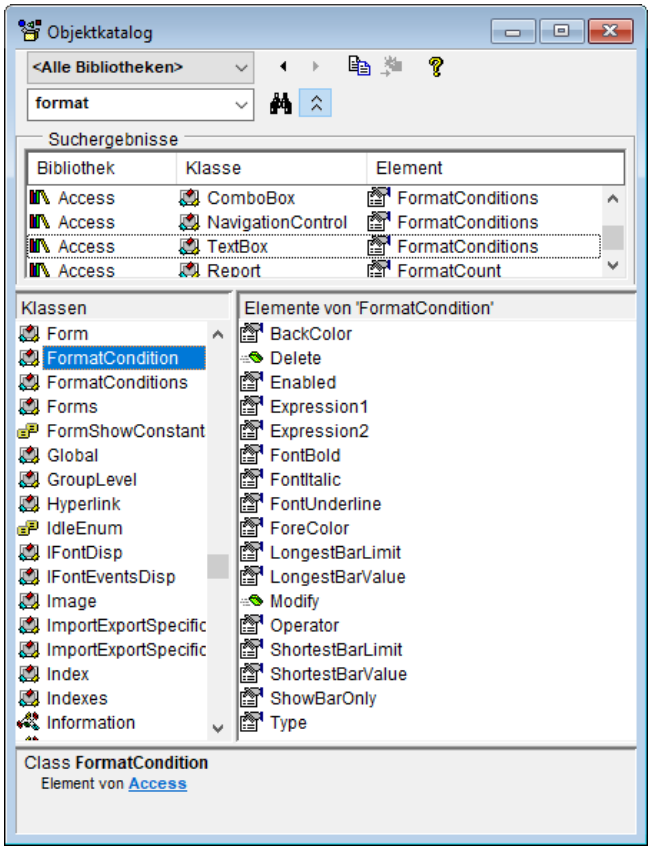


Bild 6: Die Klasse **FormatCondition** mit ihre Eigenschaften

Bedingte Formatierungen auslesen

Als Erstes lesen wir einmal die bereits angelegten bedingten Formatierungen aus. Dazu fügen wir dem VBA-Projekt der Beispieldatenbank ein neues Standardmodul namens **mdlBedingteFormatierung** hinzu.

Hier legen wir dann eine Prozedur namens **BedingteFormatierungenLesen** an. Diese soll unser Beispielformular öffnen, die Steuerelemente durchlaufen und die Eigenschaften der für die Steuerelemente definierten bedingten Formatierungen ausgeben.

Hinweis: Die Grundlagen zur folgenden Prozedur, mit der wir die Steuerelemente eines Formulars durchlaufen, finden Sie im Artikel **Steuerelemente per VBA referenzieren**.

Die Prozedur finden Sie in Listing 1. Sie deklariert eine Variable namens **frm** für das zu untersuchende Formular und eine namens **ctl** für ein **Control**-Element, also ein allgemeines Steuerelement. Außerdem benötigen wir noch eine Variable, mit der wir beim Durchlaufen der Auflistung der bedingten Formatierungen den jeweils aktuellen Eintrag referenzieren können – diese heißt **objFormatCondition** und hat den Datentyp **FormatCondition**. Anschließend öffnen wir mit der **OpenForm**-Methode des **DoCmd**-Objekts das Formular **frmArtikel** und weisen dieses der Variablen **frm** zu. Auf diese Weise ist es später leichter zu handhaben. In einer ersten **For Each**-Schleife durchlaufen wir nun die **Controls**-Auflistung des mit **frm** referenzierten Formulars und weisen das jeweils aktuelle **Control**-Element der Variablen **ctl** zu.

Hier prüfen wir dann, ob **ctl** den Typ **acTextBox** oder **acComboBox** aufweist, also ob es sich um eine Textbox oder um ein Kombintaionsfeld handelt. Für Kontrollkästchen etwa können Sie nämlich keine bedingten Formatierungen festlegen. Damit wir wissen, zu welchem Steuerelement die nachfolgend ausgegebenen Eigenschaften der bedingten Formatierungen gehören, geben wir hier schon den Namen des Steuerelements im Direktbereich aus.

```
Public Sub BedingteFormatierungenLesen()
    Dim frm As Form
    Dim ctl As Control
    Dim objFormatCondition As FormatCondition
    DoCmd.OpenForm "frmArtikel"
    Set frm = Forms!frmArtikel
    For Each ctl In frm.Controls
        Select Case ctl.ControlType
            Case acTextBox, acComboBox
                Debug.Print ctl.Name
                For Each objFormatCondition In ctl.FormatConditions
                    Debug.Print "  BackColor:        " & objFormatCondition.BackColor
                    Debug.Print "  Enabled:          " & objFormatCondition.Enabled
                    Debug.Print "  Expression1:      " & objFormatCondition.Expression1
                    Debug.Print "  Expression2:      " & objFormatCondition.Expression2
                    Debug.Print "  FontBold:         " & objFormatCondition.FontBold
                    Debug.Print "  FontItalic:       " & objFormatCondition.FontItalic
                    Debug.Print "  FontUnderline:    " & objFormatCondition.FontUnderline
                    Debug.Print "  ForeColor:        " & objFormatCondition.ForeColor
                    Debug.Print "  LongestBarLimit:  " & objFormatCondition.LongestBarLimit
                    Debug.Print "  LongestBarValue:  " & objFormatCondition.LongestBarValue
                    Debug.Print "  Operator:         " & objFormatCondition.Operator
                    Debug.Print "  ShortestBarLimit: " & objFormatCondition.ShortestBarLimit
                    Debug.Print "  ShortestBarValue: " & objFormatCondition.ShortestBarValue
                    Debug.Print "  ShowBarOnly:      " & objFormatCondition.ShowBarOnly
                    Debug.Print "  Type:             " & objFormatCondition.Type
                Next objFormatCondition
        End Select
    Next ctl
    DoCmd.Close acForm, "frmArtikel"
End Sub
```

Listing 1: Prozedur zum Ausgeben der Eigenschaften von bedingten Formatierungen

Damit können wir in die zweite **For Each**-Schleife über alle bedingten Formatierungen aus der Auflistung **FormatConditions** einsteigen, die jeweils in der Variablen **objFormatCondition** landen.

Von dieser geben wir die Inhalte aller Eigenschaften im Direktbereich des VBA-Editors aus. Die Programmierung der Ausgabe wird uns durch die Verwendung von IntelliSense erleichtert, was uns die verschiedenen Eigenschaften zur Auswahl anbietet (siehe Bild 7). Nach dem Durchlaufen aller bedingten Formatierungen aller Steuerelemente wird das Formular

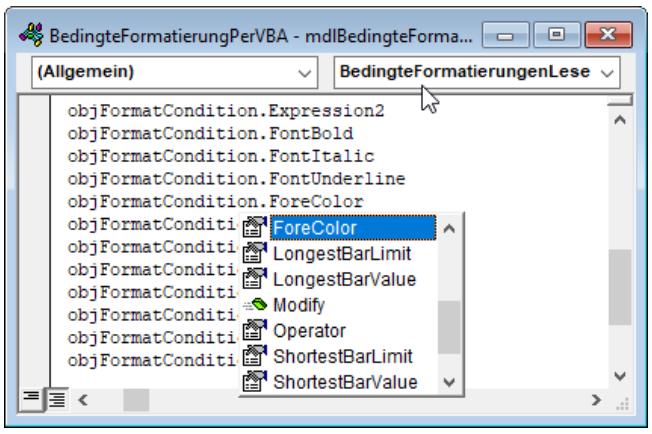


Bild 7: Auswahl per IntelliSense

schließlich wieder geschlossen. Das Ergebnis für zwei der Steuerelemente finden Sie in Bild 8 vor.

Die Eigenschaften haben die folgenden Bedeutungen, wobei wir zunächst die möglichen Änderungen bei erfüllter Bedingung für Hintergrund und Schrift auflisten:

- **BackColor**: Hintergrundfarbe, die beim Eintreten der Bedingung angezeigt werden soll.
- **Enabled**: Gibt an, ob das Steuerelement bei Eintreten der Bedingung deaktiviert dargestellt werden soll.
- **FontBold**: Gibt an, ob die Schrift fett dargestellt werden soll.
- **FontItalic**: Gibt an, ob die Schrift kursiv dargestellt werden soll.
- **FontUnderline**: Gibt an, ob die Schrift unterstrichen dargestellt werden soll.
- **ForeColor**: Gibt die Schriftfarbe an.

Diese möglichen Formatierungen finden Sie im Dialog zum Anlegen und Bearbeiten von Formatierungsregeln, wenn oben der erste Regeltyp ausgewählt wurde (siehe Bild 9). In diesem Fall fragt der Dialog auch die Eigenschaften für die Bedingung ab, welche durch die folgenden VBA-Eigenschaften angegeben werden. Die Eigenschaft **Type** nimmt einen der Werte **acDataBar** (3), **acExpression** (1), **acFieldHasFocus** (2) oder **acFieldValue** (0) an.

Der Wert **acFieldValue** wird gesetzt, wenn für die Bedingung Werte im aktuellen Datensatz geprüft werden und im ersten Kombinationsfeld der Wert **Ausdruck ist** ausgewählt ist. Der Wert **acExpression** wird gesetzt, wenn für die Bedingung Werte im aktuellen Datensatz geprüft werden und im ersten Kombinationsfeld der Wert **Ausdruck ist** ausgewählt ist. Der Wert **acFieldHasFocus** kommt zum Einsatz, wenn für die Bedingung Werte im aktuellen Datensatz geprüft werden und im ersten Kombinationsfeld der Wert **Feld hat Fokus** ausgewählt wurde.

Der Wert **acDataBar** schließlich wird für die Eigenschaft **Type** festgelegt, wenn oben die Option **Mit anderen Datensätzen vergleichen** festgelegt ist.

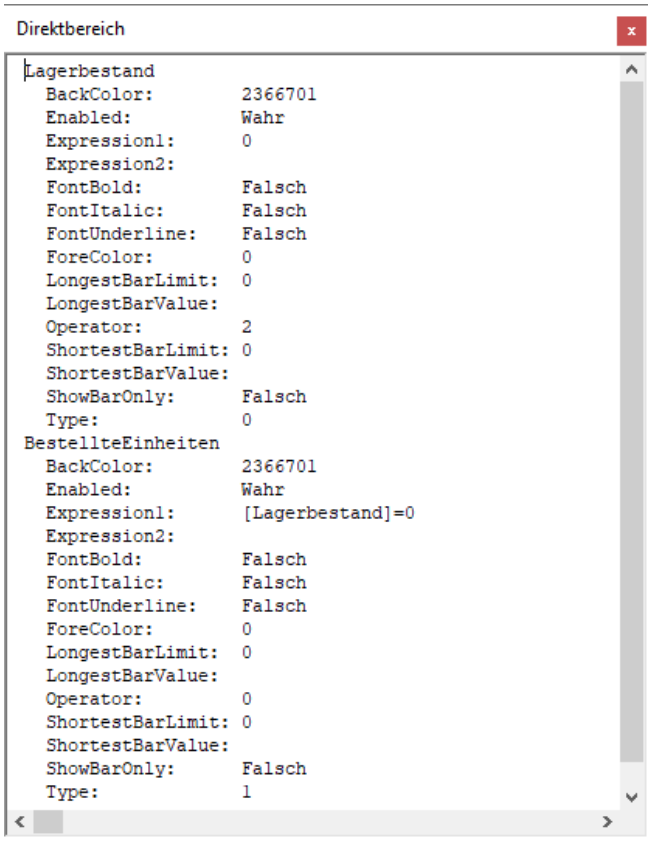


Bild 8: Ausgabe der Eigenschaften für bedingte Formatierungen

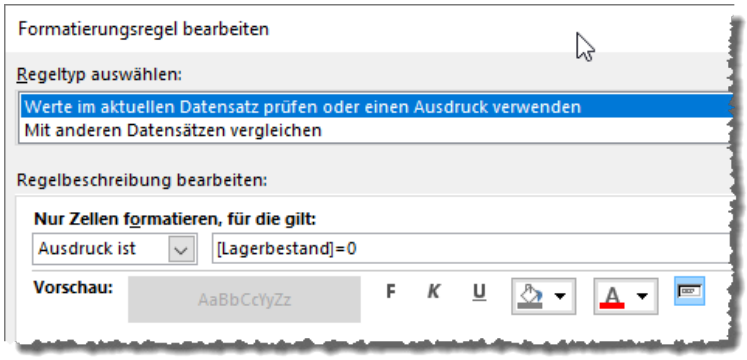


Bild 9: Format-Eigenschaften

Suchen und ersetzen per Access-Dialog

Der Access-Dialog zum Suchen und Ersetzen in Tabellen, Abfragen und Formularen ist eine praktische Sache. Aber man nutzt ihn viel zu selten! Das wollen wir ändern, indem wir in diesem Artikel einmal ausführlich auf diesen praktischen Dialog eingehen. Dabei zeigen wir nicht nur, wie Sie diesen bedienen, sondern auch, wie Sie ihn per VBA öffnen und einige der Voreinstellungen bereits vorab anpassen können.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1806_SuchenUndErsetzen.accdb**.

Suchen und Ersetzen in Datenblättern
Zum Suchen und Ersetzen in Datenblättern benötigen Sie im Prinzip nur eine einfache Tastenkombination, nämlich **Strg + F**. Damit öffnen Sie den **Suchen und Ersetzen**-Dialog.

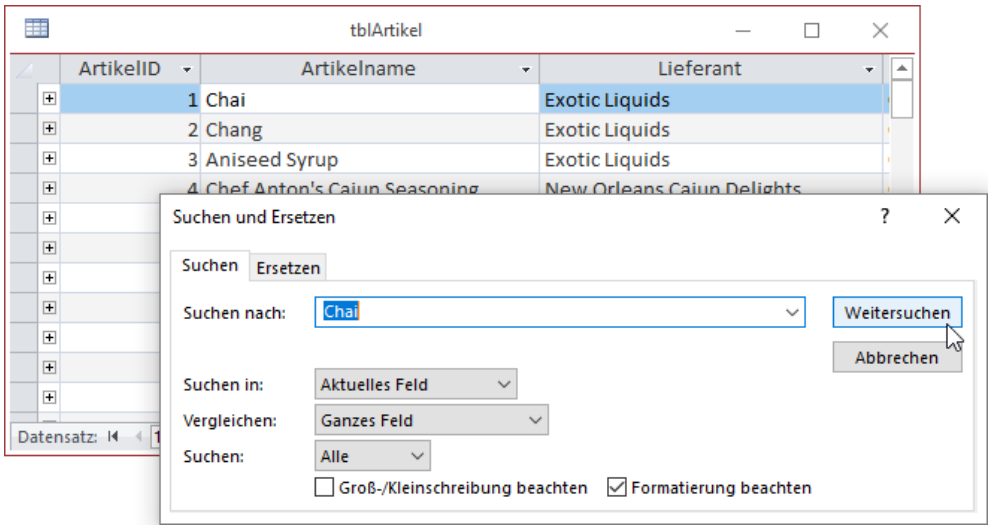


Bild 1: Aufruf des **Suchen und Ersetzen**-Dialogs

Dieser zeigt, wenn Sie zuvor den Inhalt oder einen Teilinhalt im markierten Feld markiert hatten, auch gleich den markierten Text im Feld **Suchen nach** an (siehe Bild 1). Unter dem Feld zur Eingabe des Suchbegriffs finden Sie einige weitere Optionen. Die erste namens **Suchen in** legt fest, wo genau gesucht werden soll.

Die Optionen lauten:

- **Aktuelles Feld**: Durchsucht nur im aktuell markierten Feld nach dem Suchbegriff.
- **Aktuelles Dokument**: Durchsucht alle Felder des aktuellen Dokuments, wobei Dokument eine Tabelle oder Abfrage in der Datenblattansicht oder ein Formular in beliebiger Ansicht bedeutet.

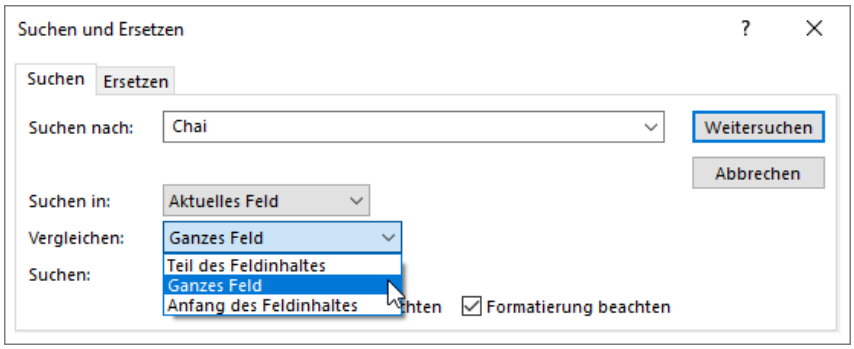


Bild 2: Spezifizieren des Vergleichs

Die Option **Vergleichen** lässt gleich drei verschiedene Werte zu (siehe Bild 2):

- **Teil des Feldinhaltes**: Sucht an beliebiger Stelle innerhalb der Inhalte des Feldes nach dem Suchbegriff. Der Suchbegriff **ha** würde also etwa **Chai** oder **Chang** als Ergebnis liefern.

- **Ganzes Feld**: Vergleicht den Suchbegriff mit dem kompletten Feldinhalt, der Suchbegriff **Chai** trifft also nur, wenn das Feld exakt den Text **Chai** enthält.

- **Anfang des Feldinhaltes**: Liefert nur Ergebnisse, bei denen der Feldinhalt mit dem Suchbegriff beginnt. Bei **Ch** also auch wieder **Chai** oder **Chang**.

Mit dem Feld **Suchen** legen Sie schließlich noch fest, ob aufwärts, abwärts oder überall gesucht werden soll.

Die Option **Groß-/Kleinschreibung beachten** gibt an, ob bei der Suche auf Groß-/Kleinschreibung geachtet werden soll. Ist diese Option aktiviert, findet der Suchbegriff **chai** zum Beispiel nicht **Chai**.

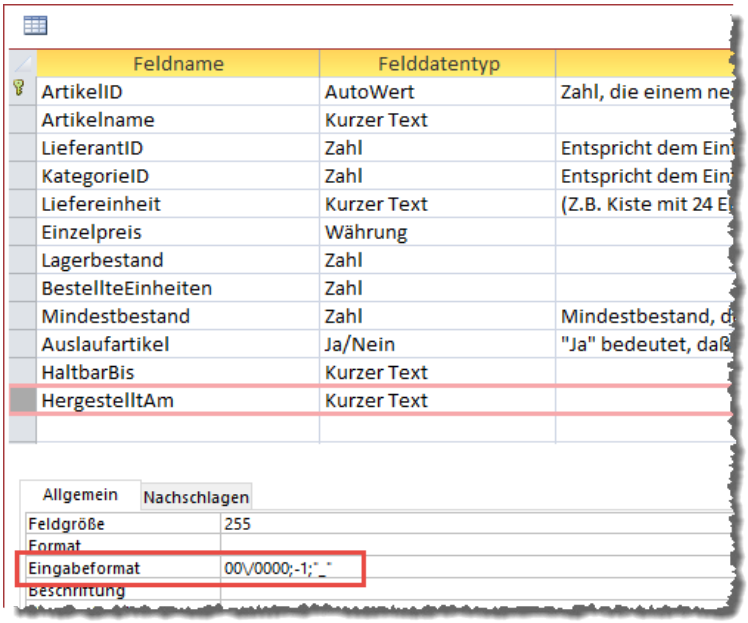


Bild 3: Spezifizieren des Eingabeformats

Formatierung beachten schließlich ist eine recht komplizierte Option. Das liegt auch daran, dass diese scheinbar willkürlich aktiviert oder deaktiviert wird. Schauen wir uns das an Beispielen an.

Wenn Sie in der Tabelle **tblArtikel** das Feld **ArtikelID** markieren und dann den **Suchen und Ersetzen**-Dialog öffnen, ist die Option selektiert, aber aktiviert, sodass Sie diese Einstellung ändern können.

Schließen Sie den **Suchen und Ersetzen**-Dialog und markieren Sie ein Feld in der Spalte **Artikelname**. Öffnen Sie den **Suchen und Ersetzen**-Dialog erneut. Die Option **Formatierung beachten** ist wiederum aktiviert und selektiert. Aktivieren Sie nun die Option **Groß-/Kleinschreibung beachten**, wird die Option **Formatierung beachten** jedoch deaktiviert, bleibt aber selektiert.

Die Eigenschaft »Formatierung beachten«

Schließlich schauen wir uns an, was geschieht, wenn wir ein Feld der Spalte **Lieferant** markieren und den **Suchen und Ersetzen**-Dialog öffnen. Nun ist die Option **Formatierung beachten** selektiert, aber deaktiviert! Was bedeutet dies nun?

Eigentlich ist diese Option für das Händeln von Eingabeformaten bei der Suche vorgesehen. Sie können ja für ein Feld verschiedene Eingabeformate festlegen (wie das geht, habe ich im Artikel **Eingabeformate** beschrieben). Ich nutze dieses Feature nie, aber zu Beispielzwecken fügen wir einfach ein Feld namens **HaltbarBis** zur Tabelle **tblArtikel** hinzu. Diesem weisen wir den Datentyp **Kurzer Text** und als Eingabeformat den Ausdruck **00\/0000;0;"_"** zu. Für ein weiteres Feld namens **HergestelltAm** legen wir dieses Eingabeformat fest: **00\/0000;-1;"_"** (siehe Bild 3). Der Unterschied ist, dass Ersteres die eingegebenen Zeichen mit den im Eingabeformat definierten Literalen speichert, Letzteres speichert nur die eingegebenen Zeichen ohne die Literale des Eingabeformats.

Nun schauen wir uns an, wie sich unterschiedliche Einstellungen der Eigenschaft **Formatierung beachten** auswirken. Als Erstes durchsuchen wir das Feld **HaltbarBis**, in dem wir die Literale des Eingabeformats mit im Feld speichern, also etwa **01/2018**. Wir aktivieren **Formatierung beachten** und suchen nach **01/2018**. Diesen Eintrag finden wir problemlos. Nun entfernen wir den Haken für die Eigenschaft **Formatierung beachten** und suchen erneut. Auch diesmal findet Access den gewünschten Eintrag. Wenn wir in

1:1-Beziehungen

1:1-Beziehungen können für eine ganze Reihe von Anwendungzwecken sinnvoll sein. Sie können damit beispielsweise die Liefer- und/oder die Rechnungsanschrift für einen Kundendatensatz in eigenen Tabellen speichern, um so verschiedene Ziele zu erreichen: beispielsweise die Anzahl der Felder je Tabelle zu verringern, verschiedene Teile der Daten mit unterschiedlichen Berechtigungen versehen und so weiter. In diesem Artikel schauen wir uns an, wie Sie eine 1:1-Beziehung herstellen und welche Stolpersteine es zu beachten gibt.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1806_11Beziehung.accdb**.

Aufbau von 1:1-Beziehungen
Eine 1:1-Beziehung besteht aus zwei Tabellen und ist im Prinzip eine eingeschränkte 1:n-Beziehung. Bei der 1:n-Beziehung können Sie dem Datensatz der Tabelle mit dem Primärschlüsselfeld der Beziehung, also etwa **tblKunden**, keinen, einen oder mehrere Datensätze der Tabelle mit dem Fremdschlüsselfeld der Beziehung zuweisen, also etwa **tblBestellungen**. In diesem Fall realisieren wir das über ein Fremdschlüsselfeld, das durch die Definition der Beziehung so angelegt wird, dass dieses Feld nur die Werte des Primärschlüsselfeldes der verknüpften Tabelle annehmen kann.

Bei der 1:1-Beziehung ist das prinzipiell genau das Gleiche, mit einem entscheidenden Unterschied: Das Fremdschlüsselfeld der Beziehung wird so ausgelegt, dass es jeden Wert nur einmal annehmen kann, es erhält also einen eindeutigen Index. Das würde im obigen Beispiel der Kunden und Bestellungen bedeuten, dass wir das Fremdschlüsselfeld **KundeID** der Tabelle **tblBestellungen** mit einem eindeutigen Index versehen.

Dann könnten Sie jedem Kunden allerdings nur eine Bestellung zuweisen, was definitiv keinen Sinn ergibt. In anderen Fällen jedoch schon: Wenn Sie etwa die Adresse des Kunden in einer eigenen Tabelle verwalten wollen (zum Beispiel namens **tblKundenLieferadresse**), würde ein mit einem eindeutigen Index versehenes Feld etwa namens **KundeID** als Fremdschlüsselfeld durchaus sinnvoll sein. Auf diese Weise ist sichergestellt, dass es für jeden Kunden nur eine Lieferadresse gibt. Was übrigens in den meisten Anwendungsfällen ausreicht – außer, Sie programmieren eine Datenbank für ein Shopsystem wie Amazon, wo ein Kunde auch Bestellungen an andere Empfänger schicken lassen kann.

Auf die gleiche Weise könnte man dann auch noch eine zweite Tabelle per 1:1-Beziehung mit der Tabelle **tblKunden** verknüpfen, zum Beispiel namens **tblKundenRechnungsadressen**, um zu jedem Kunden auch noch eine Rechnungsadresse speichern zu können. Auch hier würden wir das Fremdschlüsselfeld mit einem eindeutigen Index versehen.

Sinn von 1:1-Beziehungen
Welchen Sinn und Zweck haben solche 1:1-Beziehungen nun? Der erste ist, dass man eine Tabelle auf diese Weise ein wenig entschlacken kann. Das ist insbesondere wichtig, wenn es der Anwendungsfall tatsächlich erfordert, dass die Tabelle mehr als 255 Felder enthält (was in der Realität nicht oft vorkommen sollte – meist lassen sich dann ähnliche Felder in eine m:n-Tabelle auslagern). Sollte also eine Tabelle tatsächlich mehr als 255 Felder benötigen, legen Sie eine zweite Tabelle an, die per Fremdschlüsselfeld mit eindeutigem Index mit der ersten Tabelle verknüpft wird.

Ein weiterer Anwendungszweck sind Berechtigungen. Wenn Sie zum Beispiel Mitarbeiter in einer Datenbank verwalten, könnte die Tabelle mit den Mitarbeitern

auch sensible Daten wie etwa das Gehalt enthalten. Die Mitarbeiterdaten benötigen Sie aber auch für andere Zwecke wir etwa das Zuweisen von Aufgaben. Damit beispielsweise Projektleiter dies tun können, ohne Einblick in die Gehälter der Mitarbeiter nehmen zu können, müssten Sie die Tabellen mit entsprechenden Berechtigungen versehen. Wenn Sie das nicht auf Feldebene tun wollen, können Sie die Tabelle aufteilen und per 1:1-Beziehung verknüpfen und den Teil der Tabelle mit den sensiblen Daten entsprechend schützen.

Zugegeben: Das Thema ist nicht relevant, wenn Sie mit einer reinen Access-Datenbank arbeiten, da diese keine Möglichkeiten zur Vergabe von Berechtigungen bietet. Aber wenn Sie etwa mit dem SQL Server arbeiten, ist dies durchaus einen Alternative. Wenngleich Sie dort natürlich auch auf anderer Ebene sicherstellen können, dass Benutzer nur die für sie relevanten Daten sehen – etwa indem Sie per gespeicherter Prozedur nur solche Daten liefern, die der Benutzer benötigt.

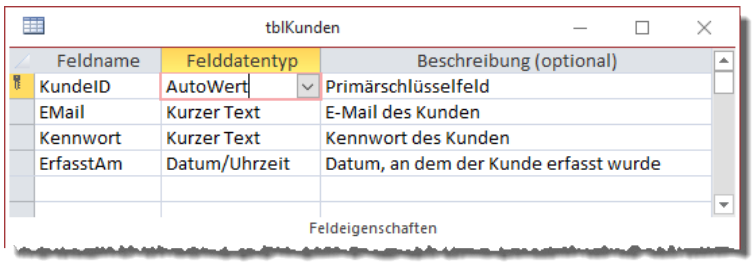


Bild 1: Entwurf der Tabelle **tblKunden**

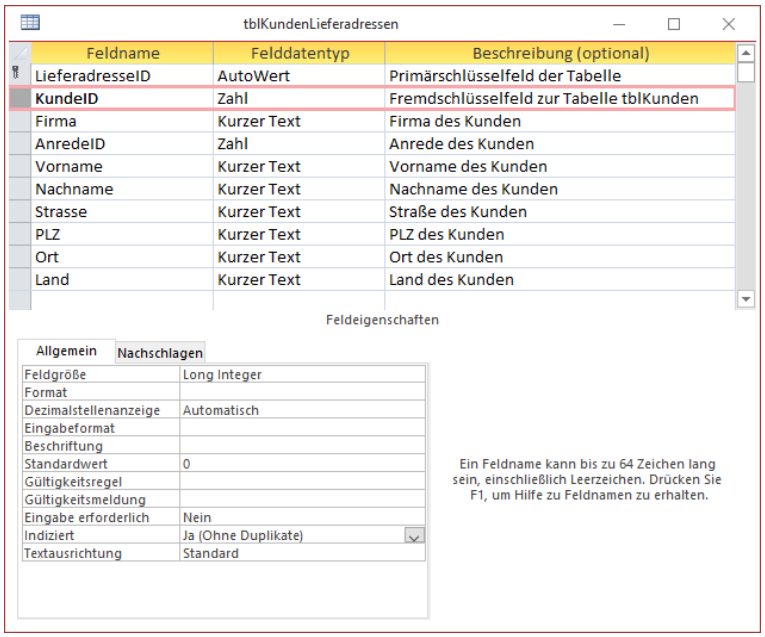


Bild 2: Entwurf der Tabelle **tblKundenLieferadressen**

Eine andere Variante, die sich ebenfalls die Verwaltung von Kunden anschneidet, ist die, bei der Personen in eine allgemeine Tabelle geschrieben werden und deren Informationen bezüglich ihrer Rolle, also etwa als Kunde, Mitarbeiter oder Lieferant in separaten Tabellen gespeichert werden. Die Tabelle **tblPersonen** würde dann nur Informationen wie die Anrede, Vorname, Nachname, Geburtsdatum und andere allgemeine Daten aufnehmen. In weiteren Tabellen wie **tblKunden**, **tblMitarbeiter** oder **tblLieferanten** fügen Sie dann die rollenspezifischen Daten hinzu. Auf diese Weise ist es dann auch einfach, einer Person mehrere Rollen zuzuweisen. Warum sollte ein Mitarbeiter nicht gleichzeitig auch Kunde sein?

1:1-Beziehung erstellen

Wir wir eine 1:1-Beziehung erstellen, schauen wir uns am Beispiel von Kunden und Lieferadressen an.

Die Tabelle **tblKunden** soll allgemeine Daten enthalten, wie sie etwa in einem Shopsystem Verwendung finden wie E-Mail-Adresse, das Kennwort oder das Datum der Aufnahme dieses Kunden in die Datenbank (siehe Bild 1).

Die Tabelle zum Speichern der Lieferadressen enthält die typischen Felder wie **Firma**, **Vorname**, **Nachname** und die Adressdaten. Zusätzlich enthält sie aber auch noch ein Fremdschlüsselfeld namens **KundeID**, das wir mit dem Feld **KundeID** der Tabelle **tblKunden** verknüpfen wollen. Der Clou hierbei ist, dass wir die Eigenschaft **Indiziert** auf den Wert **Ja (Ohne Duplikate)** festlegen (siehe Bild 2).

Die Beziehung könnten wir nun über den Nachschlagefeld-Assistenten für das zukünftige Fremdschlüs-

selfeld **KundeID** erstellen, aber in diesem Fall macht das keinen Sinn. Wir werden später in der Benutzeroberfläche, sprich in den Formularen, keinen Anwendungsfall finden, wo wir ein Nachschlagefeld beziehungsweise Kombinationsfeld benötigen, um den Kunden zu einer Lieferadresse auszuwählen.

In diesem seltenen Fall würde ich also die Beziehung direkt über das Beziehungen-Fenster erstellen, das wir über den Ribbon-Eintrag **Datenbanktools|Beziehungen|Beziehungen** öffnen.

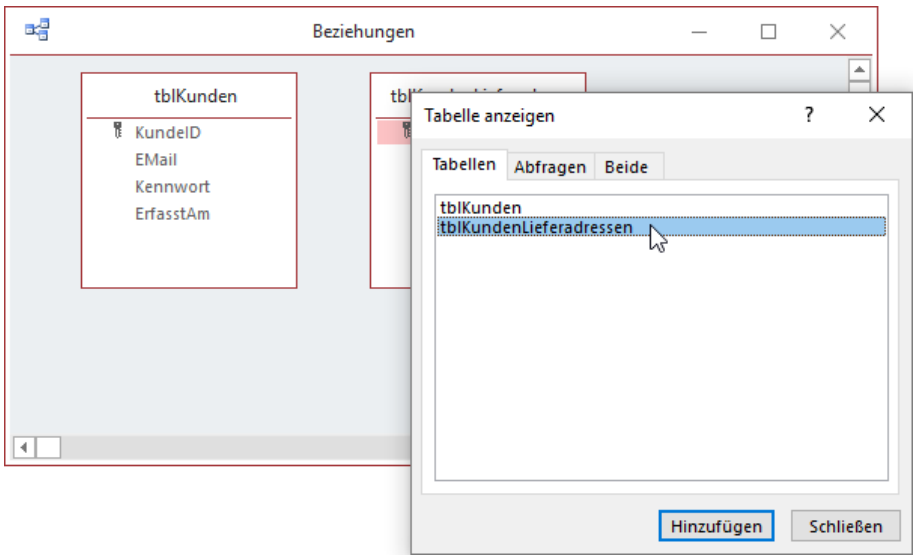


Bild 3: Einfügen der Tabellen in das Beziehungen-Fenster

Hier fügen wir die beiden Tabellen **tblKunden** und **tblKundenLieferadressen** per Doppelklick auf die entsprechenden Einträge im Dialog **Tabelle anzeigen** zum **Beziehungen**-Fenster hinzu (siehe Bild 3).

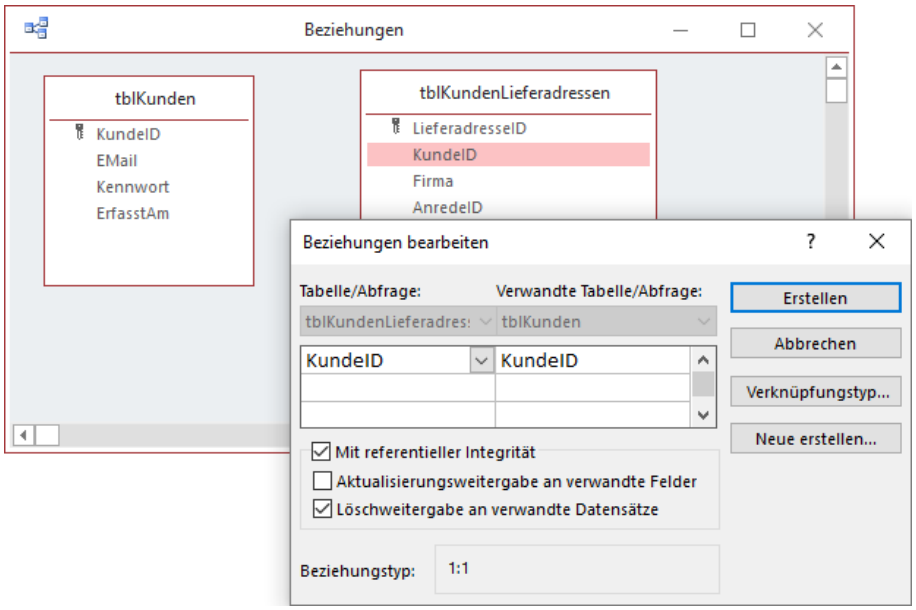


Bild 4: Definieren der Beziehung

Beziehung anlegen

Danach legen wir die Beziehung an, indem wir das Feld **KundeID** von der Tabelle **tblKundenLieferadressen** auf das Feld **KundeID** der Tabelle **tblKunden** ziehen. Es erscheint nun der Dialog **Beziehungen bearbeiten**, wo Sie weitere Einstellungen vornehmen können (siehe Bild 4).

Hier aktivieren wir die Option **Mit referenzieller Integrität** und auch die Option **Löschweitergabe an verwandte Datensätze**. Die erste sorgt dafür, dass das Fremdschlüsselfeld nur solche Werte aufnehmen kann, die auch im verknüpften Feld **KundeID** der Tabelle **tblKunden** enthalten sind. Die zweite sorgt dafür, dass beim Löschen eines Datensatzes der Tabelle **tblKunden** auch der damit vernküpften Datensatz der Tabelle **tblKundenLieferadressen** gelöscht wird. Auf diese bleiben keine verwaisten Datensätze in der Tabelle **tblKundenLieferadressen** übrig.

Sie sehen nun auch direkt, dass Access eine 1:1-Beziehung statt der sonst üblichen 1:n-Beziehung als Beziehungstyp in den Dialog eingetragen hat. Es hat also aufgrund des eindeutigen Indexes für das

Fremdschlüsselfeld automatisch erkannt, dass eine 1:1-Beziehung angelegt werden soll.

Damit können wir den Dialog **Beziehungen bearbeiten** nun schließen und der Beziehungspfeil für diese Beziehung erscheint im Beziehungen-Fenster (siehe Bild 5).

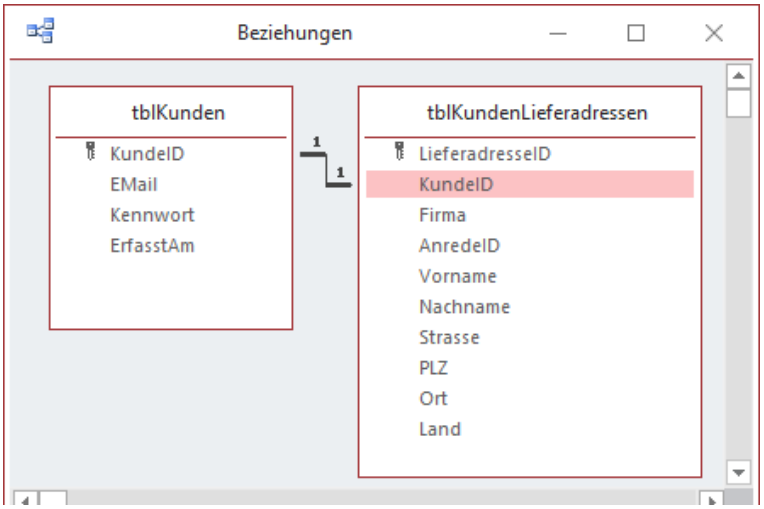


Bild 5: Beziehungspfeil der 1:1-Beziehung

1:1-Beziehung ausprobieren

Nun wollen wir die 1:1-Beziehung ausprobieren, indem wir zunächst einen Datensatz in der Tabelle **tblKunden** anlegen. Nachdem wir dies getan haben und den Datensatz speichern wollen, erscheint die Fehlermeldung aus Bild 6. Diese besagt, dass wir erst einen Datensatz in der Tabelle **tblKundenLieferadressen** anlegen müssen, der mit dem neuen Datensatz in Beziehung steht.

Das irritiert zunächst, aber dann wird klar, dass etwas an der Definition der Beziehung nicht stimmen kann. Dadurch, dass wir weiter oben das Feld **KundeID** aus der Tabelle **tblKundenLieferadressen** auf das Feld **KundeID** der Tabelle **tblKunden** gezogen haben, haben wir die Beziehung falsch aufgebaut. Korrekt wäre gewesen, den Beziehungspfeil vom Feld der Tabelle mit dem Primärschlüsselfeld auf die Tabelle mit dem Fremdschlüsselfeld zu ziehen. Das holen wir nun nach, indem wir die Beziehung im **Beziehungen**-Dialog markieren und löschen. Dann legen wir diese wie oben beschrieben neu an. Das Ergebnis ist die umgekehrte Angabe der beiden Tabellen im Dialog **Beziehung bearbeiten**, wo nun die Tabelle **tblKunden** unter **Tabelle/Abfrage** und die Tabelle **tblKundenLieferadressen** unter **Verwandte Tabelle/Abfrage** aufgeführt wird (siehe Bild 7).

Wenn wir nun erneut in die Datenblattansicht der Tabelle **tblKunden** wechseln, können wir auch den ersten Kundendatensatz anlegen. Und danach müssen wir zwar in der Tabelle **tblKundenLieferadressen** manuell den Primärschlüsselwert des Kundendatensatzes eintragen, aber zu Testzwecken ist das okay. Gleich schauen wir uns noch an, wie wir das per Abfrage vereinfachen (siehe Bild 8).

Test der Löschweitergabe

Da wir nun einen Datensatz eingegeben haben, wollen wir auch gleich noch die Löschweitergabe

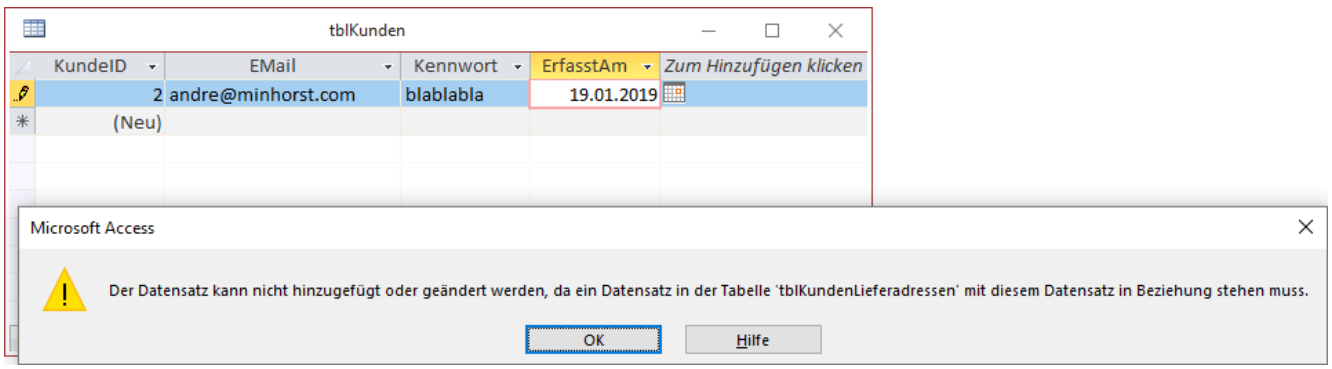


Bild 6: Fehlermeldung beim Anlegen des ersten Datensatzes

Einfacher Listenfeldfilter

Manchmal möchte Sie einen Filter auf die einfache Art setzen – ohne VBA und den Einsatz von Ereignisprozeduren. Das gelingt unter Access sehr leicht, aber nicht ohne Tücken – und auch nicht komplett ohne den Einsatz von VBA: So kann es zum Beispiel sein, dass das Listenfeld keine Daten anzeigt, wenn das für die Eingabe des Vergleichswertes verwendete Textfeld leer ist. Wir schauen uns einen einfachen Listenfeldfilter anhand eines Textfeldes an und zeigen, wie Sie die kleinen Hindernisse umschiffen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1806_EinfacherListenfeldfilter.accdb**.

Vorbereitung

Das Formular soll ein Textfeld namens **txtSucheNachFirma** enthalten sowie ein Listenfeld namens **lstKunden**, das zu Beginn alle Datensätze der Tabelle **tblKunden** anzeigt (siehe Bild 1). Dem Listenfeld weisen wir als Datensatzherkunft die folgende Abfrage zu, um die Spalten korrekt auszurichten:

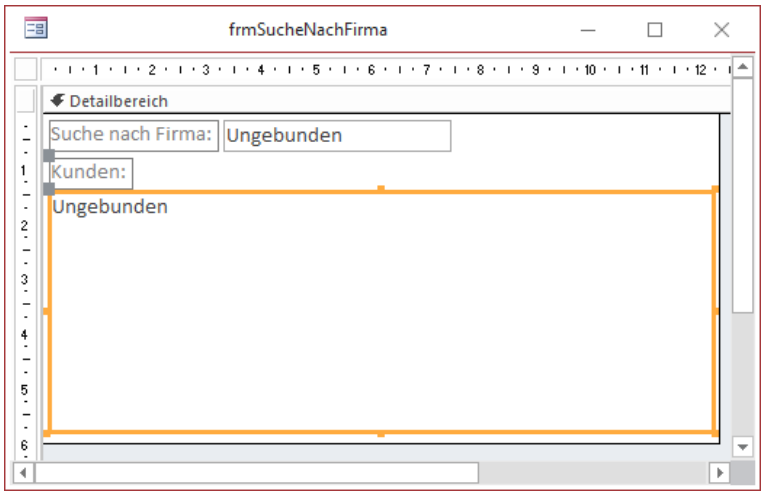


Bild 1: Entwurf des Beispielformulars

```
SELECT tblKunden.KundeID, tblKunden.KundenCode, tblKunden.Firma, tblKunden.Kontaktperson FROM tblKunden;
```

Dabei wollen hier nur die Spalten **KundeID**, **Kundencode**, **Firma** und **Kontaktperson** anzeigen. Damit die Spalte **KundeID** nur als gebundene Spalte verwendet, aber nicht angezeigt wird, stellen wir die Eigenschaft **Spaltenanzahl** auf **4** und **Spaltenbreiten** auf **0cm** ein. Dadurch wird die erste Spalte ausgeblendet und die übrigen Spalten teilen sich den verbleibenden Platz.

Das Ergebnis sieht in der Formularansicht nun wie in Bild 2 aus. Hier haben wir auch noch die Eigenschaften **Navigationsschaltflächen**, **Datensatzmarkierer**, **Bildlaufleisten** und **Trennlinien** auf **Nein** sowie **Automatisch zentrieren** auf **Ja** eingestellt. Hier zeigt sich, dass wir die Breiten der Spalten doch manuell einstellen müssen, um ein brauchbares Ergebnis zu erhalten. Wir stellen die Eigenschaft **Spaltenbreiten**

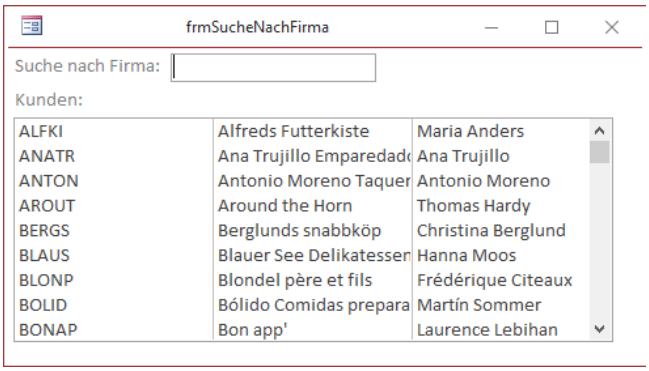


Bild 2: Formularansicht des Beispielformulars

auf **0cm;2cm** ein. Damit wird die Spalte mit dem Kundencode schmaler und die beiden übrigen Spalten teilen sich den verbleibenden Platz.

Filter einbauen

Den Filter bauen wir nun direkt in die Abfrage ein, die wir als Datensatzquelle des Listenfeldes angegeben

1:1-Beziehungen in Formularen

1:1-Beziehungen können für eine ganze Reihe von Anwendungzwecken sinnvoll sein. Sie können damit beispielsweise die Liefer- und/oder die Rechnungsanschrift für einen Kundendatensatz in eigenen Tabellen speichern, um so verschiedene Ziele zu erreichen: beispielsweise die Anzahl der Felder je Tabelle zu verringern, verschiedene Teile der Daten mit unterschiedlichen Berechtigungen versehen und so weiter. In diesem Artikel schauen wir uns an, wie Sie die Formular zur Darstellung einer 1:1-Beziehung gestalten können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1806_11BeziehungInFormularen.accdb**.

Voraussetzungen

Grundlage für diesen Artikel ist ein weiterer Artikel namens 1:1-Beziehungen. Hier erfahren Sie, wie Sie eine 1:1-Beziehungen erstellen. Kurz zusammengefasst: Sie legen eine Tabelle etwa namens **tblKunden** an, welche die Basisdaten des Kunden speichert.

Weitere Tabellen wie **tblKundenLieferadresse** und **tblKundenRechnungsadresse** enthalten jeweils ein mit einem eindeutigen Index versehenes Fremdschlüsselfeld, über das eine Beziehung zur Tabelle **tblKunden** hergestellt wird. Wichtig dabei ist, dass Sie die Beziehung im Beziehungen-Fenster anlegen, indem Sie das Primärschlüsselfeld der Tabelle **tblKunden** auf das Fremdschlüsselfeld der Tabellen **tblKundenLieferadressen** und **tblKundenRechnungsadressen** ziehen.

Vorbereitung

Als Vorbereitung legen wir eine Abfrage an, welche die drei Tabellen **tblKunden**, **tblKundenLieferadressen** und **tblKundenRechnungsadressen** enthält. Sie weist im Entwurfsraster alle Felder der Tabelle **tblKunden** sowie alle Felder der beiden übrigen Tabellen mit Ausnahme des jeweiligen Primärschlüssel- und Fremdschlüsselfeldes auf (siehe Bild 1). Die Abfrage speichern wir unter dem Namen **qryKundenLieferadressenRechnungsadressen**.

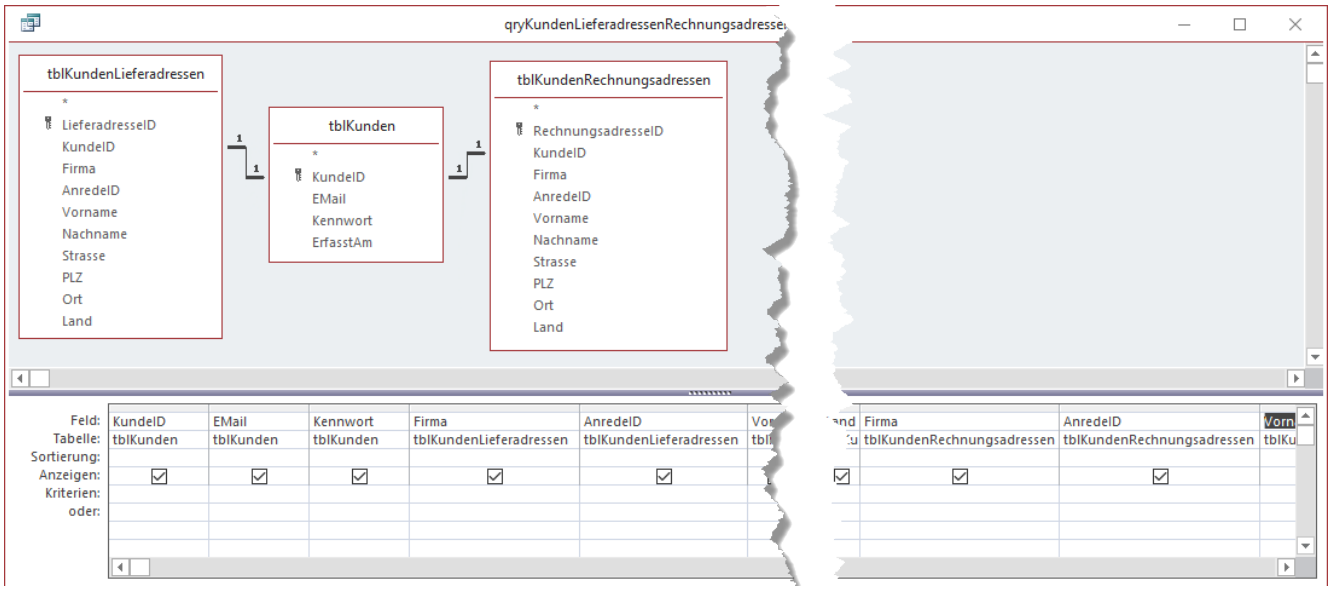


Bild 1: Entwurf der Abfrage **qryKundenLieferadressenRechnungsadressen**

Anschließend nehmen wir noch eine wichtige Änderung vor. Wenn wir die Abfrage so nutzen, wie Sie aktuell konfiguriert ist, liefert diese nur solche Datensätze der Tabelle **tblKunden**, für die es sowohl in der Tabelle **tblKundenLieferadressen** als auch in der Tabelle **tblKundenRechnungsadressen** einen verknüpften Datensatz gibt.

Daher müssen wir noch die Verknüpfungseigenschaften für die beiden Verknüpfungen einstellen. Das erledigen wir, indem wir doppelt auf einen der Verknüpfungspfeile klicken. Dies öffnet den Dialog aus Bild 2. Hier wählen wir die folgende Option aus:

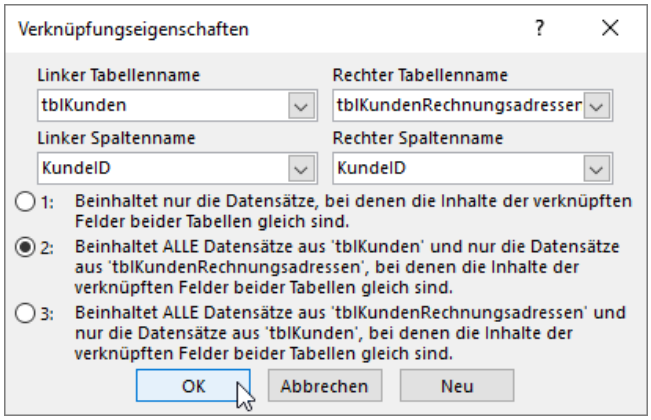


Bild 2: Verknüpfungseigenschaften der 1:1-Beziehung zwischen den Tabellen **tblKunden** und **tblKundenRechnungsadressen**

```
Beinhaltet ALLE Datensätze aus 'tblKunden' und nur die Datensätze aus 'tblKundenRechnungsadressen', bei denen die Inhalte der verknüpften Felder beider Tabellen gleich sind.
```

Die gleiche Einstellung nehmen wir auch für die Beziehung zwischen den Tabellen **tblKunden** und **tblKundenLieferadressen** vor. Danach sieht der obere Teil des Abfrageentwurfs wie in Bild 3 aus. Wenn jetzt nur ein Eintrag in der Tabelle **tblKunden** vorliegt, aber keiner oder nur einer in den beiden übrigen Tabellen, werden die Felder der entsprechenden Tabelle im Abfrageergebnis leer angezeigt.

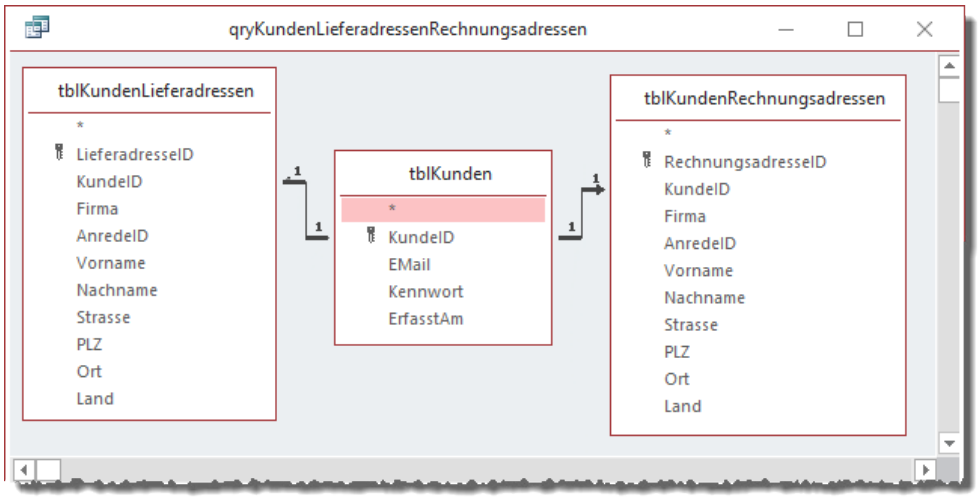


Bild 3: Abfrage mit angepassten Verknüpfungseigenschaften

Fehler beim Ausführen der Abfrage

Das war zumindest die Theorie, denn in der Tat erscheint beim Wechsel in die Datenblattansicht der Abfrage die folgende Fehlermeldung:

```
Die SQL-Anweisung konnte nicht ausgeführt werden, da sie mehrdeutige äußere Verknüpfungen enthält. Damit eine der VErknüpüfungen zuerst ausgeführt wird, müssen Sie eine separate Abfrage erstellen, die die erste Verknüpfung ausführt, und dann diese Abfrage in die SQL-Anweisung einschließen.
```

Das heißt also: Wir müssen erst eine Abfrage erstellen, welche die Daten der Tabelle **tblKunden** mit einer der beiden per 1:1-Beziehung verknüpften Tabellen liefert und diese dann in einer zweiten Abfrage mit der anderen verknüpften Tabelle zusammenführen.

Die erste Abfrage ist schnell erstellt. Dazu löschen Sie einfach die Tabelle **tblKundenRechnungsadressen** aus der Abfrage **qryKundenLieferadressenRechnungsadressen** und ändern den Namen auf **qryKundenLieferadressen**. Der Entwurf sieht dann wie in

Bild 4 aus. Der Wechsel in die Datenblattansicht birgt die nächste Überraschung: Auch hier taucht diese Fehlermeldung auf.

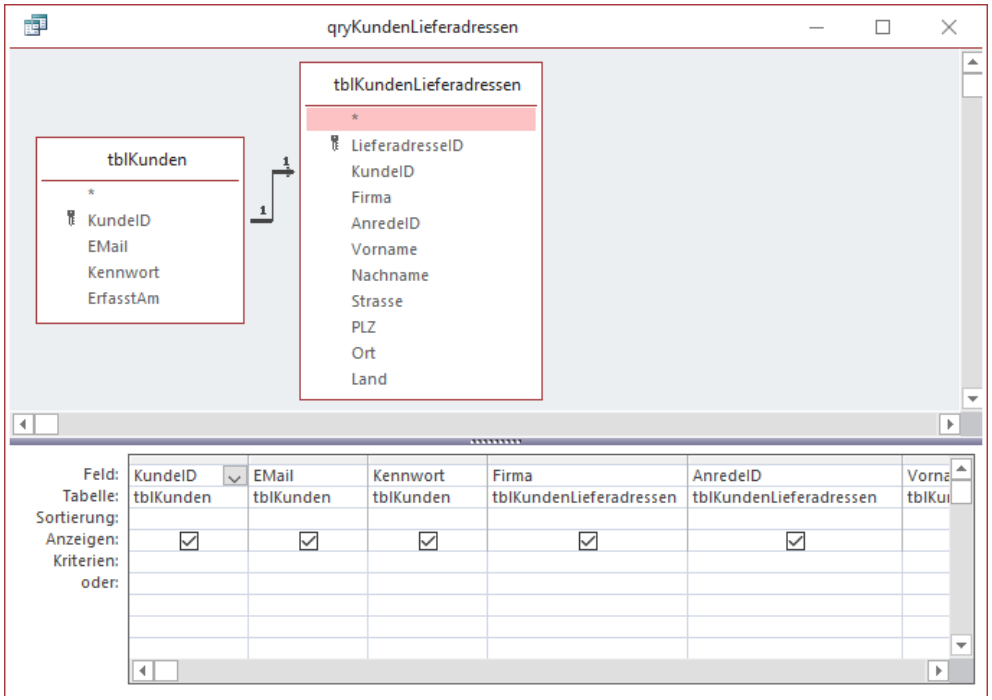


Bild 4: Erster Teil der Abfrage mit zwei 1:1-Beziehungen

Können wir die Daten der 1:1-Beziehung also gar nicht so darstellen, dass nur die Daten einer Tabelle ausgegeben werden, wenn es in der zweiten Tabelle keine verknüpften Daten gibt? Das wäre ungünstig, weil wir ja dann nur neue Datensätze in einem Formular auf Basis dieser Abfrage anlegen könnten. Wenn wir hier nur die Daten zur Tabelle **tblKunden** hinzufügen und kein Feld der Tabelle **tblKundenLieferadressen** füllen, können wir dies auch nachträglich nicht mehr erledigen, weil dieser Datensatz der Tabelle **tblKunden** schlicht nicht mehr angezeigt werden kann – zumindest nicht in einer Abfrage zusammen mit der Tabelle **tblKundenLieferadressen**.

Formular zur Anzeige der Daten

Nun haben wir durch das obige Problem zwei Möglichkeiten: Wir verwenden eine Version der Abfrage, welche die Daten der drei Tabellen zusammenführt, als Datensatzquelle des Formulars und sorgen gleich beim Anlegen der ersten Werte für die Felder der Tabelle **tblKunden** dafür, dass jeweils ein Datensatz in den Tabellen **tblKundenLieferadressen** und **tblKundenRechnungsadressen** erstellt wird.

Dann hätten wir allerdings nicht den Vorteil, dass nur für solche Datensätze, die tatsächlich eine Lieferadresse und/oder Rechnungsadresse erhalten, auch ein Datensatz in einer der Tabellen **tblKundenLieferadressen** und/oder **tblKundenRechnungsadressen** angelegt wird. Sprich: Wir könnten auch gleich eine einzige Tabelle anlegen, die wieder alle Felder des Kunden inklusive Rechnungsadresse und Lieferadresse enthält.

Die Alternative ist, mit zwei Unterformularen zu arbeiten, welche die Daten der Tabellen **tblKundenRechnungsadressen** und **tblKundenLieferandressen** enthalten. Wir schauen uns einfach beide Lösungen an.

Formular mit automatischen Datensätzen

Als Datensatzquelle des ersten Formulars verwenden wir also eine Abfrage, die alle drei Tabellen, also **tblKunden**, **tblKundenRechnungen** und **tblKundenLieferadressen** enthält (siehe **qryKundenRechnungsadressenLieferadressen**).

Dann fügen wir die Felder etwa wie in Bild 5 zum Formularentwurf des neuen Formulars namens **frmKunden1zu1EinFormular** hinzu. Dadurch, dass die doppelt vorkommenden Felder in der Abfrage alle noch mit dem Tabellennamen versehen werden, haben wir allerdings noch eine kleine Fleißaufgabe für die Bezeichnungsfelder vor uns.

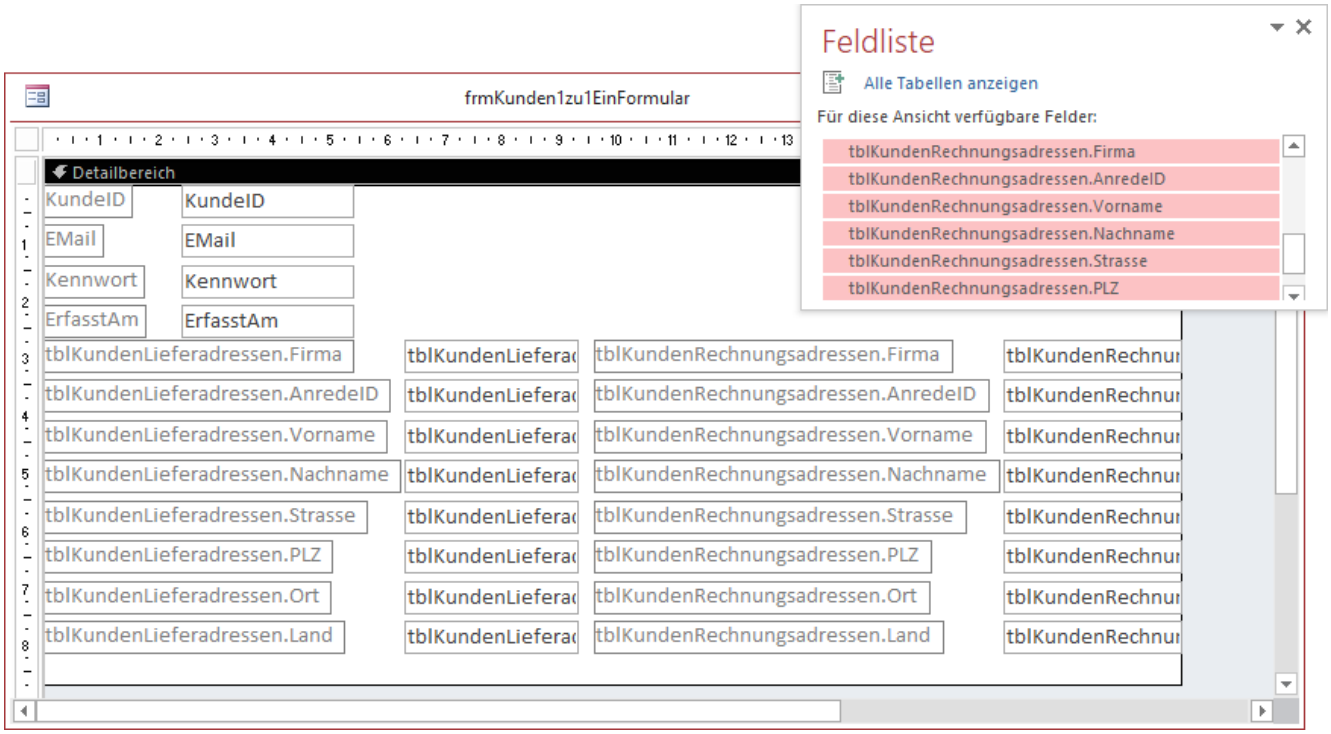


Bild 5: Erster Entwurf des Formulars

Nachdem diese erledigt ist und wir noch je einen Rahmen für die Felder der Lieferadresse und der Rechnungsadresse hinzugefügt haben, sieht das Formular in der Formularansicht wie in Bild 6 aus.

Hier haben wir außerdem noch die Beschriftungsfelder angepasst sowie die Aktivierreihenfolge richtig eingestellt. Nun experimentieren wir ein wenig, indem wir einen neuen Datensatz anlegen.

Experimente im Formular mit einer Abfrage über die beiden 1:1-Beziehungen

Wir legen einen neuen Datensatz an, wobei wir mit den oberen Feldern beginnen und dann die Felder mit der Lieferadresse und der Rechnungsadresse füllen. In dieser Reihenfolge gelingt das ohne Probleme – der Datensatz wird korrekt mit den verknüpften Datensätzen in den drei Tabellen gespeichert und der Datensatz erscheint

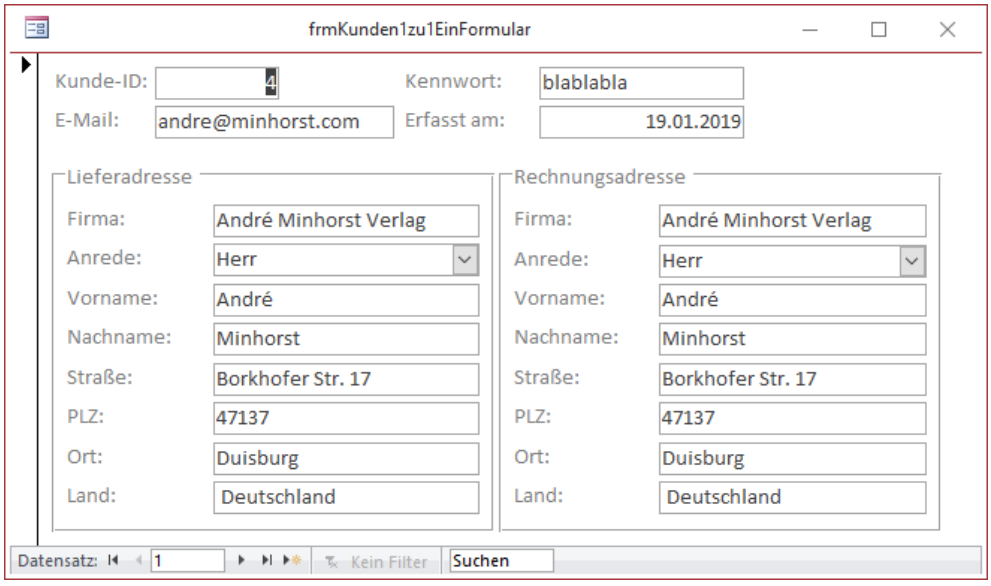


Bild 6: Formular optisch aufbereitet

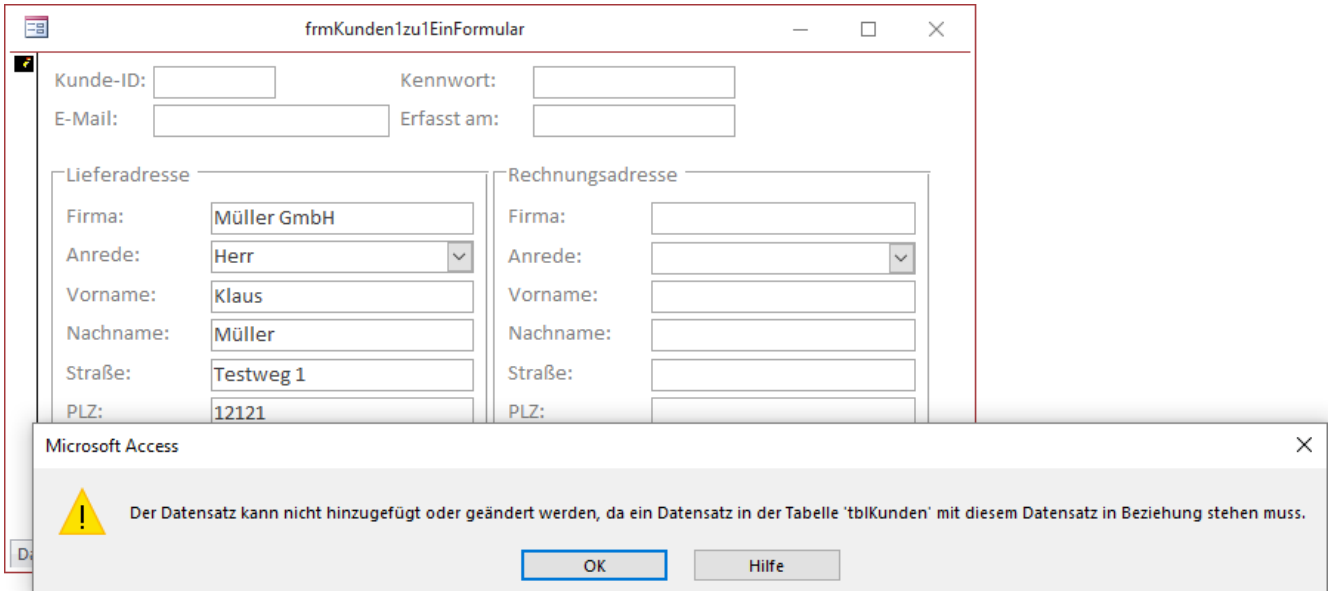


Bild 7: Fehler beim Versuch, nur Daten in die Tabelle **tblKundeLieferadresse** zu schreiben

nach dem Verlassen und dem erneuten Anzeigen wieder im Formular **frmKunden1zu1EinFormular**.

Nun beginnen wir mit dem Anlegen bei den Feldern der Tabelle **tblLieferadresse**, also im Rahmen mit der Überschrift **Lieferadresse**, tragen dann die Rechnungsadresse ein und erst dann die Daten der Tabelle **tblKunden**. Auch das gelingt – der Datensatz wird gespeichert und im Formular angezeigt.

Wenn wir nur die Daten der Lieferadresse und/oder der Rechnungsadresse eintragen, ohne auch nur ein Feld der Tabelle **tblKunden** auszufüllen und dann versuchen, den Datensatz zu speichern, erhalten wir die Fehlermeldung aus Bild 7.

Dieses Problem beheben wir gleich mit ein paar Zeilen VBA-Code.

Eine weitere Variante ist, Daten in die Felder der Tabelle **tblKunden** einzugeben, aber dann nur die Felder keiner oder nur einer der beiden Tabellen **tblKundenLieferadressen** oder **tblKundenRechnungsadressen** zu füllen.

Das funktioniert zwar ohne Fehler und die Daten werden auch in den betroffenen Tabellen gespeichert.

Wenn allerdings nicht in jeder der Tabellen **tblKundenLieferadressen** und **tblKundenRechnungsadressen** ein Datensatz zu dem Datensatz aus **tblKunden** enthalten ist, werden die Daten nicht von der Abfrage **qryKundenRechnungsadressenLieferadressen** zurückgeliefert. Diese Abfrage liefert nur solche Daten, für die es in allen drei Tabellen entsprechend verknüpfte Datensätze gibt.

Auch hier können wir per VBA dafür sorgen, dass zumindest ein ansonsten leerer Datensatz in der nicht vom Benutzer ausgefüllten Tabelle **tblKundenLieferadressen** oder **tblKundenRechnungsadressen** angelegt wird. Wie das geht, lesen Sie weiter unten.

Speichern ohne Kundendaten verhindern

Um den Fehler zu verhindern, der beim Versuch auftritt, die Lieferadresse oder Rechnungsadresse ohne Angabe der Basisdaten zum Kunden zu speichern, gibt es zwei Möglichkeiten: Entweder wir fangen den Fehler ab und ersetzen diesen durch eine benutzerdefinierte, gut verständliche Fehlermeldung, nach dessen Anzeige der Fokus in eines der noch auszufüllenden Felder verschoben wird.

Oder wir legen automatisch einen Datensatz in der Tabelle **tblKunden** an, sobald der Benutzer mit der

Bearbeitung eines der Felder der Tabellen **tblKundenLieferadressen** oder **tblKundenRechnungsadressen** begonnen hat.

Die erste Variante zeigen wir im Formular **frmKunden1zu1EinFormular_FehlerAbfangen**. Diesem Formular fügen wir für die Ereigniseigenschaft **Bei Fehler** eine Ereigniseigenschaft hinzu, die wie folgt aussieht:

```
Private Sub Form_Error(DataErr As Integer, _
        Response As Integer)
    Debug.Print DataErr
End Sub
```

Die dort enthaltene Anweisung nutzen wir zunächst, um herauszufinden, wie die Fehlernummer des ausgelösten Fehlers lautet. Wenn wir das Formular nun in der Formularansicht öffnen und in einem Feld einer der beiden Tabellen **tblKundenLieferadressen** oder **tblKundenRechnungsadressen** einen Wert eintragen, bevor wir den neu angelegten Datensatz speichern, löst dies den oben genannten Fehler aus.

Im Direktbereich können wir dann die Nummer des Fehlers auslesen, der in diesem Fall **3201** lautet.

Diesen Fehler behandeln wir dann in der folgenden Version der Ereignisprozedur **Form_Error** gezielter, indem dafür einen eigenen Zweig in einer neu hinzugefügten **Select Case**-Bedingung anlegen:

```
Private Sub Form_Error(DataErr As Integer, _
                                Response As Integer)
    Select Case DataErr
        Case 3201
            MsgBox "Bitte geben Sie einen Wert in _
               eines der Felder im oberen Bereich ein, _
                    bevor Sie den Datensatz speichern."
            Me!EMail.SetFocus
            Response = acDataErrContinue
    End Select
End Sub
```

Damit erhalten wir die Meldung mit dem folgenden Text, wenn der Benutzer versucht, den Datensatz ohne Eingabe von Werten in die Felder der Tabelle **tblKunden** zu speichern:

```
Bitte geben Sie einen Wert in eines der Felder im oberen Bereich ein, bevor Sie den Datensatz speichern.
```

Außerdem verschiebt die Prozedur den Fokus auf das Textfeld, das an das Feld EMail der Tabelle **tblKunden** gebunden ist.

Datensatz in tblKunden automatisch anlegen

Die zweite Variante finden Sie im Formular **frmKunden1zu1EinFormular_DSAnlegen**. Wir wollen hier dafür sorgen, dass bei der Eingabe eines Wertes in eines der Felder der Tabellen **tblKundenLieferadressen** oder **tblKundenRechnungsadressen** auch ein Datensatz in der Tabelle **tblKunden** angelegt wird. Das ist allerdings weniger einfach als gedacht.

Wenn wir in einer einfachen Tabelle etwa in der Datenblattansicht das erste Feld bearbeiten, wird durch die Autowert-Funktion direkt der Primärschlüsselwert in die Tabelle eingetragen. Das ist einer Abfrage mit den wie in unserem Beispiel verknüpften Daten nicht der Fall. Hier können wir Werte für die Felder einer beliebigen der drei beteiligten Tabellen eingeben, die Autowert-Funktionen für die Tabellen werden erst später aufgerufen.

In der Abfrage **qryKundenRechnungsadressenLieferadressen** ist das bei den beiden Tabellen **tblKundenLieferadressen** und **tblKundenRechnungsadressen** beispielsweise erste der Fall, wenn wir den jeweiligen Datensatz der verknüpften Tabelle verlassen.

Wenn wir also etwa das Feld **Firma** der Tabelle **tblKundenLieferadressen** füllen und dann den Fokus auf das Feld **Firma** der Tabelle **tblKundenRechnungsadressen** verschieben, wird das Feld **LieferadresseID** mit einem Autowert gefüllt. Das Fremdschlüsselfeld **KundeID** der Tabelle **tblKundenLieferadressen** erhält dann zunächst den Wert **0**, da ja auch das Primärschlüsselfeld **KundeID** der Tabelle **tblKunden** noch nicht gefüllt wurde.

Formularansicht filtern

Access bietet einige Möglichkeiten, um die Daten in Formularen zu filtern – zum Beispiel die Möglichkeiten, die auftauchen, wenn Sie das Kontextmenü eines Feldes in der Formularansicht anzeigen oder die Funktionen, die über das Ribbon zur Verfügung stehen. Wir schauen uns diese Möglichkeiten in diesem Artikel einmal im Detail an und schauen, wie wir diese Funktionen per VBA aufrufen können.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1806_FormulareFiltern.accdb**.

Beispielformular
Für die Beispiele verwenden wir ein Formular namens **frmArtikelFiltern**, das wir mit der Tabelle **tblArtikel** als Datensatzquelle versehen und mit dem wir alle Felder dieser Tabelle anzeigen. Wenn wir das Formular in der Formularansicht anzeigen und dann mit der rechten Maustaste etwa auf das Textfeld **Artikelname** klicken, finden wir bereits verschiedene Filterfunktionen vor, die wir direkt anwählen können – zum Beispiel **Ist gleich "Chai"** oder **Ist ungleich "Chai"**. Es gibt aber auch noch das Menü **Textfilter**, das weitere Befehle wie **Gleich...**, **Nicht gleich...**, **Beginnt mit...** und so weiter anbietet (siehe Bild 1).

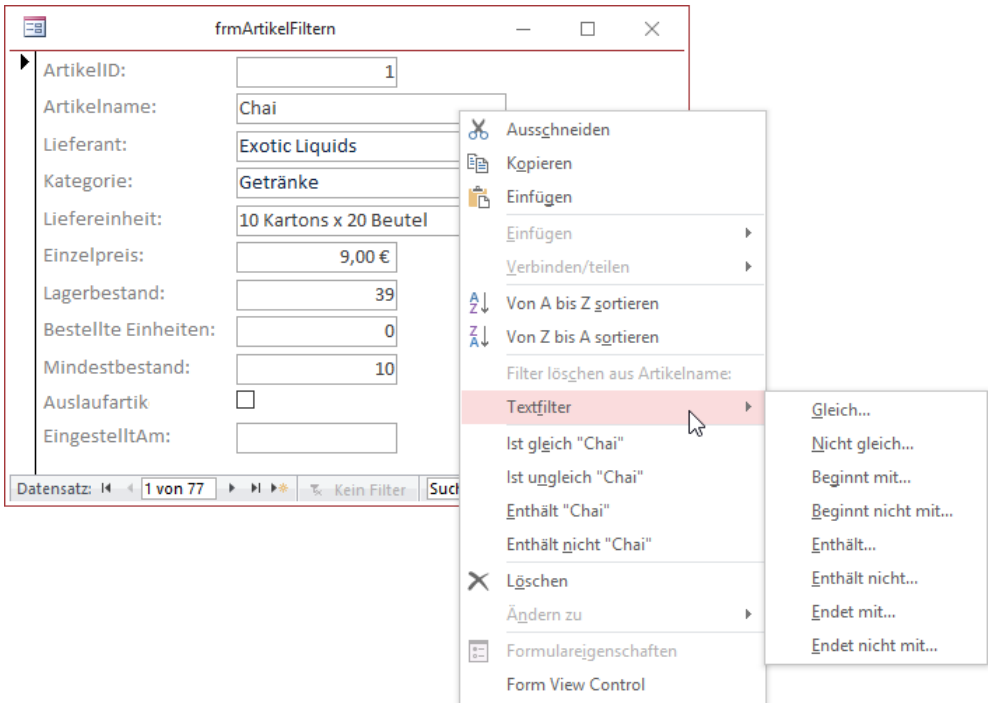


Bild 1: Formular-Entwurf unseres Beispielformulars

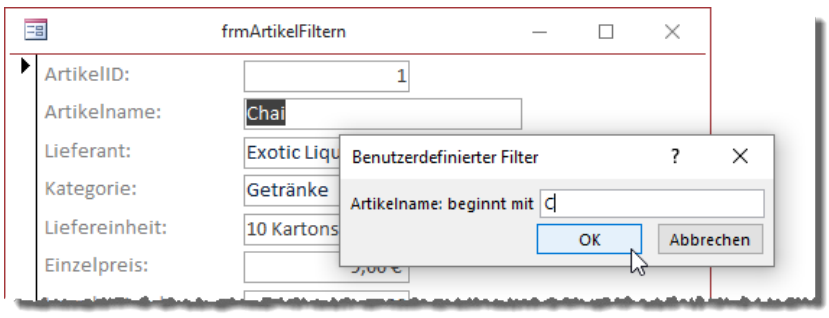


Bild 2: Eingabe eines Filtervergleichswertes

Die Einträge im Hauptmenü filtern direkt mit Ausdrücken, die den aktuellen Wert als Vergleichswert enthalten, die im Untermenü ermöglichen die Anzeige eines kleinen Eingabefeldes, mit dem der Vergleichswert für die gewählte Option ausgewählt werden können. Hier können Sie dann beispielsweise den Wert eingeben, mit dem die zu suchenden Einträge für dieses Feld beginnen (siehe Bild 2). Nach der Eingabe des Vergleichswertes und dem Betätigen der **OK**-Schaltfläche wird der

Filter genau wie bei den Schnellfiltern aus dem ersten Menü auf alle Datensätze des Formulars angewendet.

Was ist passiert?

Was genau haben wir durch diese Aktion erledigt? Wir haben einen Filter definiert und diesen auf die Daten des Formulars angewendet. Können wir irgendwie herausfinden, welcher Filterausdruck nun genau wirkt? Ja, das geht – und zwar per VBA. Wir brauchen einfach nur den Wert der Eigenschaft **Filter** des Formulars auszugeben. Das gelingt etwa mit dem folgenden Befehl im Direktbereich des VBA-Editors (anzuzeigen über die Tastenkombination **Strg + G**):

```
? Forms!frmArtikelFiltern.Filter
([tblArtikel].[Artikelname] Like "C*")
```

Der Filterausdruck vergleicht also den Inhalt des Feldes **Artikelname** mit der Zeichenkette **C***.

Was geschieht, wenn wir einen weiteren Filter für genau das gleiche Feld definieren – beispielsweise einen, der alle Artikel liefert, die den Buchstaben **A** enthalten? Dann wählen wir als Erstes den Kontextmenü-Eintrag **Textfilter|Enthält...** für das Feld **Artikelname**. Den resultierenden Filterausdruck ermitteln wir dann mit dem gleichen Befehl wie oben:

```
? Forms!frmArtikelFiltern.Filter
(([tblArtikel].[Artikelname] Like "C*")) AND
                ([tblArtikel].[Artikelname] Like "*A*")
```

Wenn wir einen zweiten Filter festlegen, fügt Access diesen also zum bestehenden ersten Filter hinzu – und zwar mit einer **AND**-Verknüpfung.

Filter entfernen

Wie können wir den Filter wieder aus dem Formular entfernen – beispielsweise, um danach einen komplett neuen Filterausdruck zu definieren? Wenn es nur um die Filter geht, die Sie für ein bestimmtes Steuerelement angelegt haben, betätigen Sie den Kontextmenü-Eintrag **Filter löschen aus Artikelname**. Das Formular entfernt dann den Filter für das Feld **Artikelname**.

Und wenn wir nicht nur Filterausdrücke für das Feld **Artikelname** definiert haben, sondern auch noch für andere Felder? Probieren wir es aus, indem wir einen Filter für das Feld **Artikelname** und einen für **Lieferant** einstellen. Dann geben wir den Filterausdruck wieder im Direktbereich des VBA-Editors aus:

```
? Forms!frmArtikelFiltern.Filter
(([tblArtikel].[Artikelname] Like "C*")) AND
          ([Lookup_LieferantID].[Firma] Like "*A*")
```

Hier ist schon der zweite Teil interessant – aber den schauen wir uns gleich an. Erst einmal entfernen wir nun den Filter für das Feld **Artikelname**. Danach sieht der Filterausdruck wie folgt aus:

```
? Forms!frmArtikelFiltern.Filter
(([Lookup_LieferantID].[Firma] Like "*A*"))
```

Der Filter für ein Feld wird also sorgfältig aus dem gesamten Filterausdruck entfernt.

Lookup-Filter

Damit kommen wir zu dem interessanten Ausdruck, der als Filter für das Feld **Lieferant** eingestellt wurde – und der ohne Klammern wie folgt aussieht:

```
[Lookup_LieferantID].[Firma] Like "*A*"
```

Dieser Ausdruck nimmt also auf ein Feld namens **Lookup_LieferantID** Bezug und ermittelt dann dessen Werte für das Feld **Firma**.

Wenn wir etwa in der Ereignisprozedur, die durch das Ereignis **Beim Laden** des Formulars ausgelöst wird, den Filter so setzen, funktioniert das ebenfalls:

```
Private Sub Form_Load()
    Me.Filter = "Lookup_LieferantID.Firma LIKE '*A*'"
    Me.FilterOn = True
End Sub
```

Filter für Zahlenfelder

Bei Zahlenfeldern sehen die Filtermöglichkeiten im Kontextmenü etwas anders aus. Hier finden Sie vier

Datenblatt per VBA: Spalten ein- und ausblenden

Für die Datenblattansicht unter Access gibt es eine Menge Einstellungsmöglichkeiten. Diese haben wir ausführlich im Artikel »Datenblattansicht im Griff« in Ausgabe 10/2011 beschrieben. Nun werfen wir einen Blick darauf, wie Sie diese Einstellungen per VBA einlesen und einstellen und welche Nutzungsmöglichkeiten sich daraus ergeben – zum Beispiel das einfache Ein- und Ausblenden der Spalten des Datenblatts.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1806_DatenblattPerVBA.accdb**.

Vorbereitung

Wir verwenden eine Beispieldatenbank mit den Tabellen der Suedsturm-Datenbank und dort speziell mit den Tabellen **tblArtikel**, **tblKategorien** und **tblLieferanten**.

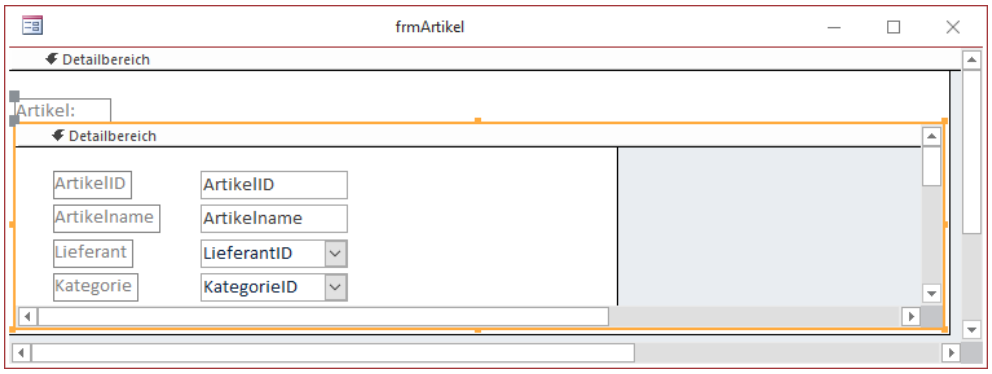


Bild 1: Entwurf der Datenblattansicht

Wir stellen ein Unterformular namens sfmArtikel, das wir über die Eigenschaft Datensatzquelle an die Tabelle **tblArtikel** binden. Danach ziehen wir alle Felder dieser Tabelle in den Detailbereich des Entwurfs des neuen Formulars. Nach dem Einstellen der Eigenschaft **Standardansicht** auf **Datenblatt** speichern und schließen wir das Formular.

Nun erstellen wir das Hauptformular für dieses Formular und speichern es unter dem Namen **frmArtikel**. Das Formular **sfmArtikel** ziehen wir aus dem Navigationsbereich in den Entwurf des Hauptformulars, sodass dieses im Entwurf nun wie in Bild 1 aussieht.

Außerdem stellen wir für das Hauptformular die Eigenschaften **Datensatzmarkierer**, **Navigationsschaltflächen** und **Bildlaufleisten** jeweils auf **Nein** ein und **Automatisch zentrieren** auf **Ja**. Schließlich soll das Unterformular sich mit dem Hauptformular vergrößern lassen, daher legen wir für die Eigenschaften **Horizontalter Anker** und **Vertikaler Anker** jeweils den Wert **Beide** fest.

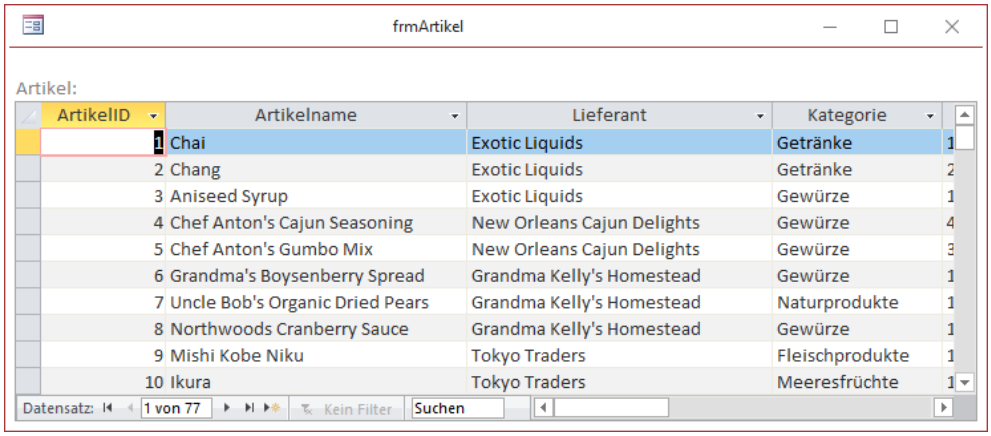


Bild 2: Datenblatt in der Formularansicht

Nach dem Wechsel in die Formularansicht des Hauptformulars erwartet uns dann die Ansicht aus Bild 2.

Spalten ein- und ausblenden per Benutzeroberfläche

Wie können wir nun per VBA eine Spalte ein- oder ausblenden? Über die Benutzeroberfläche geht das ganz einfach: Dort klicken wir mit der rechten Maustaste auf die entsprechende Spaltenüberschrift und wählen den Eintrag **Felder ausblenden** aus dem Kontextmenü aus (siehe Bild 3).

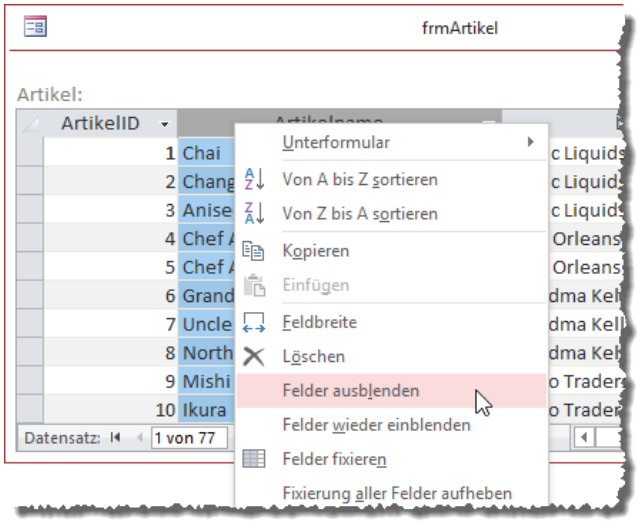


Bild 3: Feld ausblenden per Kontextmenü

Spalten ein- und ausblenden per VBA

Und wie bekommen wir das per VBA hin? Wir können es ja einfach mal über das Direktfenster probieren, indem wir dort entsprechende Eigenschaften ändern. Um zu prüfen, ob wir etwa die Spalte **Artikelname** korrekt referenzieren, wollen wir einmal den Inhalt des aktuellen Feldes ausgeben, was auch gelingt:

```
Debug.Print Forms!frmArtikel!sfmArtikel.Form!Artikelname
 Chai
```

Können wir so auch etwa die Breite des Feldes einstellen? Die Breite können wir zumindest ausgeben:

```
Debug.Print Forms!frmArtikel!sfmArtikel.Form!Artikelname.width
 1701
```

Lässt sie sich so auch ändern? Der folgende Befehl, der die Spaltenbreite verdoppel sollte, zeigt jedenfalls keine Wirkung:

```
Forms!frmArtikel!sfmArtikel.Form!Artikelname.Width = 3400
```

Damit gehen wir dann zum Ein- und Ausblenden über – funktioniert das wenigstens? Dazu nutzen wir die **Visible**-Eigenschaft, erst einmal zum Auslesen des aktuellen Wertes:

```
Debug.Print Forms!frmArtikel!sfmArtikel.Form!Artikelname.Visible
Wahr
```

Dann versuchen wir, die Spalte mit der folgenden Anweisung auszublenden:

```
Forms!frmArtikel!sfmArtikel.Form!Artikelname.Visible = False
```

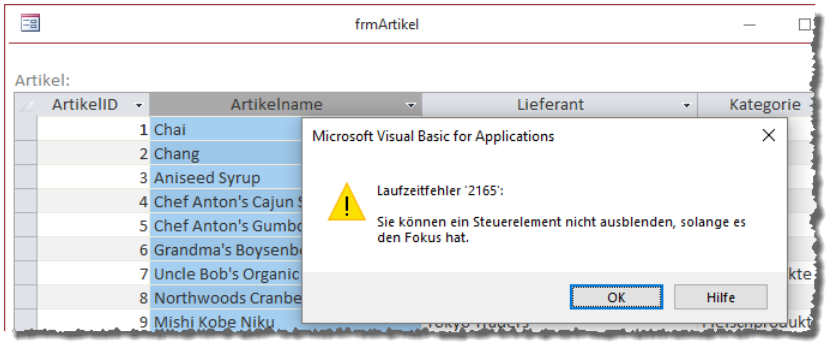


Bild 4: Fehler beim Zugriff auf die **Visible**-Eigenschaft eines aktiven Steuerelements

Das liefert immerhin einmal eine Reaktion – wenn auch nur in Form einer Fehlermeldung (siehe Bild 4). Wir können das Feld natürlich nicht ausblenden, wenn dieses den Fokus hat. Also verschieben wir diesen auf ein anderes Feld, zum Beispiel **ArtikelID**, und führen den Befehl erneut aus. Und es geschieht wieder nichts! Offensichtlich kommen wir an die Spalten nicht über den Verweis auf die dahinterliegenden Steuerelemente heran.

Eigenschaften für Datenblattspalten

Da stellt sich doch die Frage, ob die für Steuerelemente in der Formularansicht gültigen Eigenschaften überhaupt in der Datenblattansicht wirken. Also durchsuchen wir einmal den Objektkatalog etwa nach den Eigenschaften für das Textfeld-Steuerelement. Und hier werden wir dann auch schnell fündig: Mit **ColumnHidden**, **ColumnOrder** und **ColumnWidth** finden wir Eigenschaften, die sich direkt auf Spalten beziehen (siehe Bild 5).

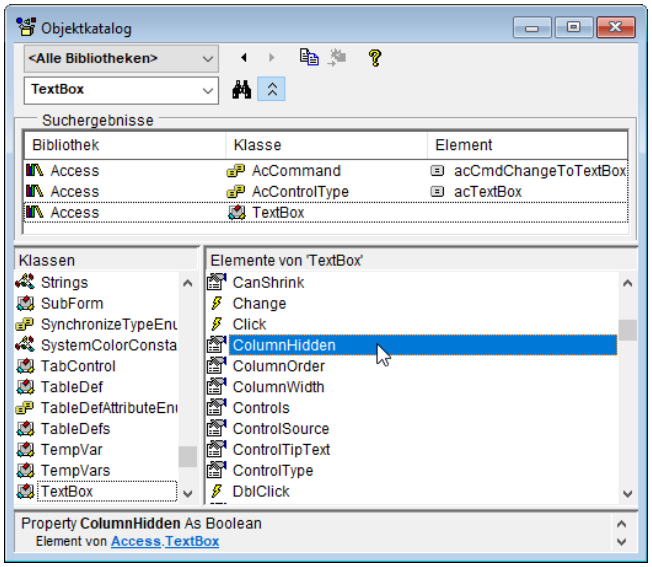


Bild 5: Einstellungen für Datenblattspalten

Die Eigenschaft **ColumnHidden** für die Spalte **Artikelname** sollte also zum Beispiel den Wert **Falsch** ausgeben, was auch geschieht:

```
? Forms!frmArtikel!sfmArtikel.Form!Artikelname.
                                        ColumnHidden
Falsch
```

Nun wollen wir die Spalte einmal ausblenden. Dazu verwenden wir diesen Befehl:

```
Forms!frmArtikel!sfmArtikel.Form!Artikelname.
                                    ColumnHidden = True
```

Und es gelingt – die Spalte ist nicht mehr zu sehen (siehe Bild 6). Auf die gleiche Weise können wir die Spalte auch wieder einblenden, dazu brauchen wir die Eigenschaft **ColumnHidden** nur wieder auf **False** einzustellen.

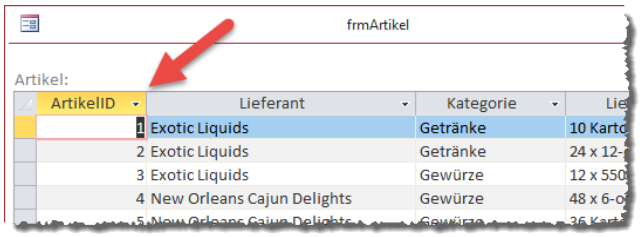


Bild 6: Ausblendete Spalte

Komfortables Ein- und Ausblenden

Wie können wir dem Benutzer das Ein- und Ausblenden der Spalten noch komfortabler machen? Dazu wollen wir diesem ein Listenfeld mit einer Liste der Spalten bereitstellen, in dem die aktuell sichtbaren Spalten markiert sind und die nicht sichtbaren nicht. Genau genommen ist das auch eine schöne Fingerübung, um das Listenfeld einmal genauer kennenzulernen.

Listenfeld mit ein- und ausgeblendeten Spalten

Also fügen wir dem Hauptformular im oberen Teil ein Listenfeld hinzu, welches die Steuerelemente im Unterformular auflistet. Das Listenfeld soll **lstSpalten** heißen und für die Eigenschaft **Herkunftstyp** den Wert **Wertliste** aufweisen.

Im Entwurf sieht das Formular nun wie in Bild 7 aus. Nun wollen wir das Listenfeld noch füllen. Das erledigen wir, wenn das Formular geöffnet wird, also in der Ereignisprozedur, die durch das Ereignis **Beim Laden** des Formulars ausgelöst wird. Diese erhält den folgenden Code:

```
Private Sub Form_Load()
    Dim sfm As Form
    Dim ctl As Control
    Dim strControlSource As String
    Dim strSpalten As String
    Set sfm = Me!sfmArtikel.Form
    For Each ctl In sfm.Controls
        strControlSource = ""
```

```
        On Error Resume Next
        strControlSource = ctl.ControlSource
        On Error GoTo 0
        If Len(strControlSource) > 0 Then
            strSpalten = strSpalten & ";" _
                & ctl.Name
        End If
    Next ctl
    If Len(strSpalten) > 0 Then
        strSpalten = Mid(strSpalten, 2)
    End If
    Me!lstSpalten.RowSource = strSpalten
End Sub
```

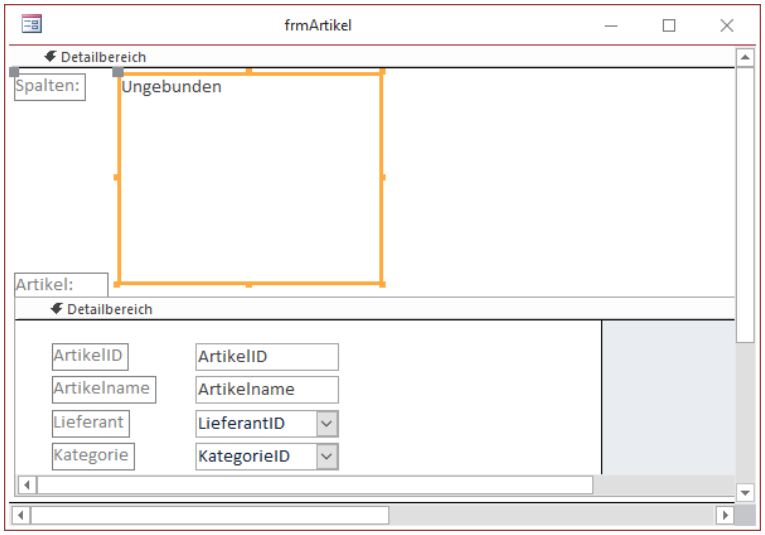


Bild 7: Listenfeld zum Ein- und Ausblenden der Spalten

In dieser Prozedur referenzieren wir zunächst das Unterformular mit der Variablen **sfm** vom Datentyp **Form**, damit wir leichter darauf zugreifen können.

Das Datenblatt enthält nicht nur Textfelder, Kombinationsfelder und Kontrollkästchen als Steuerelemente, sondern auch Bezeichnungsfelder. Wir wollen aber nur diejenigen Steuerelemente im Listenfeld auflisten, die auch in Form einer Spalte im Datenblatt erscheinen und Daten anzeigen. Diese ermitteln wir, indem wir prüfen, ob ihre Eigenschaft **Steuerelementinhalt**, unter VBA **ControlSource**, einen Wert enthält. Nur in diesem Fall ist ein Steuerelement auch ein gebundenes Steuerelement.

Leider ist es nicht damit getan, einfach zu prüfen, ob ein Steuerelement einen Wert in der Eigenschaft **ControlSource** besitzt. Einige Steuerelemente, darunter auch das Bezeichnungsfeld, weisen die Eigenschaft **ControlSource** gar nicht auf. Kein Wunder – man kann sie ja auch nicht an ein Feld binden, sie sollen nur Texte anzeigen. Wenn wir dennoch für ein solches Steuerelement auf die Eigenschaft **ControlSource** zugreifen, lösen wir damit einen Fehler aus. Um dies zu umgehen, greifen wir zu einem Trick: Wir deaktivieren mit **On Error Resume Next** die eingebaute Fehlerbehandlung, lesen den Inhalt von **ControlSource** in die Variable **strControlSource** ein und aktivieren die Fehlerbehandlung dann mit **On Error Goto 0** wieder. Danach brauchen wir nur noch zu prüfen, ob **strControlSource** einen Wert enthält – wenn das Steuerelement die Eigenschaft **ControlSource** nicht aufweist, wird die Variable auch nicht gefüllt. Vorher müssen wir allerdings sicherstellen, dass die Variable auch leer ist und nicht noch den Wert der Eigenschaft **ControlSource** vom zuvor untersuchten Steuerelement enthält. Dafür leeren wir **strControlSource** vorher durch das Einstellen auf eine leere Zeichenkette.

Das alle geschieht innerhalb einer **For Each**-Schleife über alle Element der **Controls**-Auflistung des mit **sfm** referenzierten Unterformulars. Darin referenzieren wir jeweils das aktuelle Element mit der Variablen **ctl**. Für jedes Element, dass eine gefüllte **ControlSource**-Eigenschaft enthält, fügen wir in dieser Schleife ein Semikolon und den Namen des Steuerelements an die Variable **strSpalten** an. Nach ein paar Durchläufen enthält diese dann etwa den folgenden Wert:

```
;ArtikelID;Artikelname;LieferantID;KategorieID;Lieferei
nheit;Einzelpreis;Lagerbestand;BestellteEinheiten;Minde
stbestand;Auslaufartikel
```

Hier wollen wir nun noch das führende Semikolon entfernen, was wir mithilfe der **Mid**-Funktion erledigen. Diese erwartet die zu bearbeitende Zeichenkette sowie als zweiten Parameter die Position, ab welcher

Berechnete Felder in Tabellen

Berechnete Felder in Tabellen sind ein Feature, das mit der Version 2010 von Access hinzugekommen ist. Dabei handelt es sich um die Möglichkeit, die Inhalte der übrigen Felder in einem Feld wie in einem berechneten Feld in einer Abfrage zusammenzuführen. So können Sie etwa den Nettopreis und den Mehrwertsteuersatz eines Artikels in zwei Feldern angeben und in einem dritten Feld, das als berechnetes Feld ausgelegt wird, den Bruttopreis speichern. Dieser Artikel zeigt, wie berechnete Felder funktionieren und welche Vor- und Nachteile diese haben.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1901_BerechneteFelder.accdb**.

Berechnete Felder

Wenn Sie ohne den Einsatz berechneter Felder Informationen aus mehreren Feldern zusammenführen wollen, verwenden Sie eine Abfrage. Wenn wir etwa eine Tabelle namens **tblKunden** mit den Feldern **AnredeID**, **Vorname** und **Nachname** verwenden und diese über das Feld **AnredeID** mit der Tabelle **tblAnreden** verknüpft ist, die wiederum das Feld **Anrede** besteuert, dann können wir mit einer Abfrage leicht berechnete Felder mit Inhalten wie **<Nachname>, <Vorname>** oder **<Anrede> <Vorname> <Nachname>** hinzufügen. Dies sieht dann in der Entwurfsansicht etwa wie in Bild 1 aus.

Die passende Datenblattansicht finden Sie in Bild 2.

Kundennamen in Tabelle zusammenstellen

Können wir das auch mit berechneten Feldern in einer Tabelle erreichen? Teilweise auf direktem Wege, teilweise sind Einschränkungen vorhanden.

qryKundenMitBerechnetenFeldern

KundeID	AnredeID	Vorname	Nachname	Kunde	KundeMitAnrede
1	Herr	Andre	Minhorst	Minhorst, Andre	Herr Andre Minhorst
2	Frau	Claudia	Müller	Müller, Claudia	Frau Claudia Müller
(Neu)					

Datensatz: 1 von 2 | Kein Filter | Suchen

Bild 2: Berechnete Felder in einer Abfrage

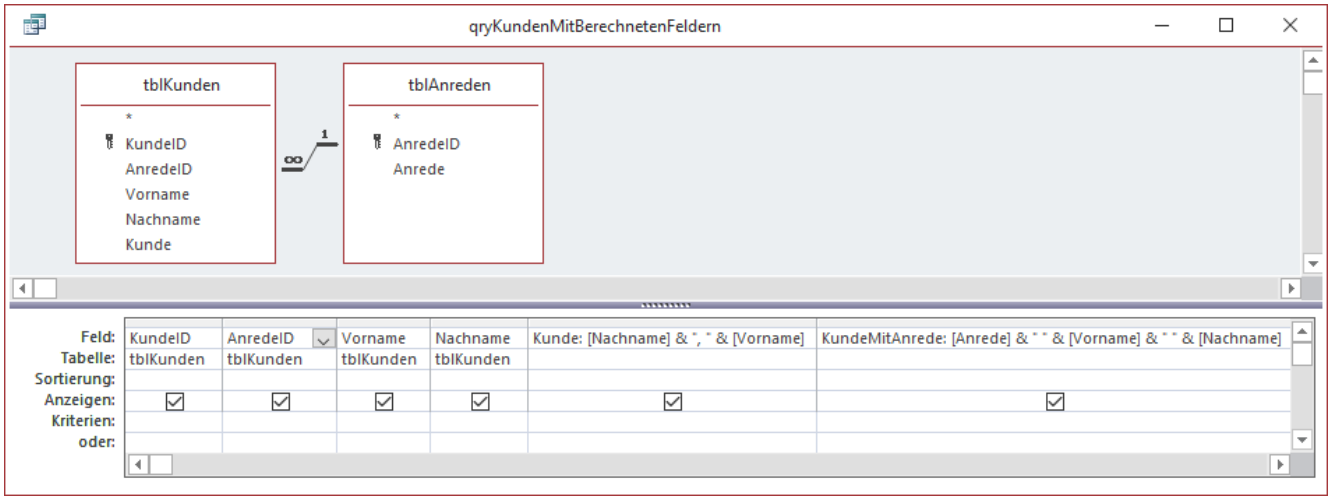


Bild 1: Abfrage mit berechneten Feldern

Wir schauen uns das an genau den beiden berechneten Feldern der Abfrage aus dem vorherigen Beispiel an, in dem wir zunächst zwei Felder der aktuellen Tabelle zusammengeführt und um ein zusätzliches Literal ergänzt haben (ein Komma und ein Leerzeichen zwischen **Nachname** und **Vorname**) und dann in einem weiteren berechneten Feld noch den Wert des Feldes **Anrede** aus einer verknüpften Tabelle ermittelt haben – auch hier mit zwei zusätzlichen Literalen, in diesem Fall mit zwei Leerzeichen.

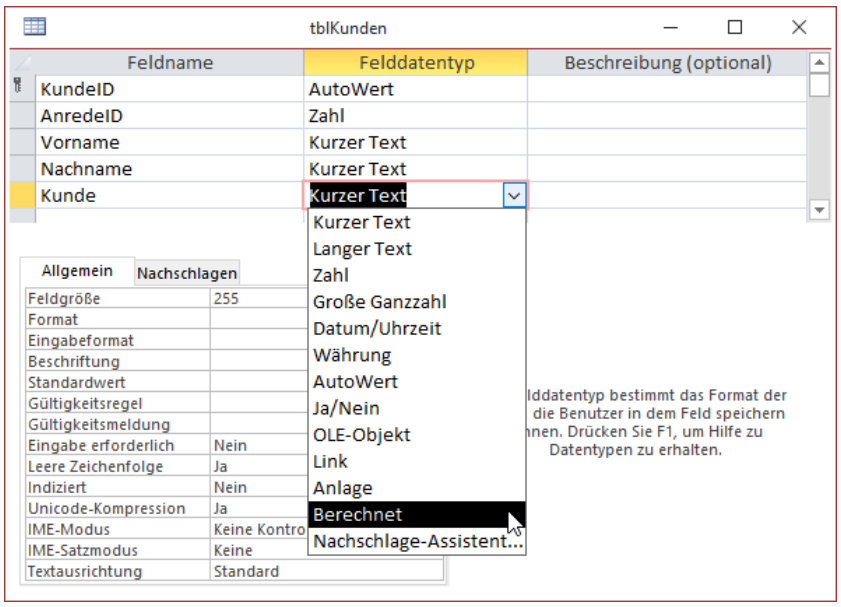


Bild 3: Auswählen des Datentyps **Berechnet**

Berechnetes Feld anlegen

Ein neues berechnetes Feld legen Sie zunächst wie jedes andere Feld an, indem Sie den Feldnamen in einer neuen, leeren Zeile der Entwurfsansicht der betroffenen Tabelle hinzufügen.

In diesem Fall geben wir den Namen **Kunde** in die Tabelle **tblKunden** ein und wählen dann als Datentyp den Wert **Berechnet** aus (siehe Bild 3).

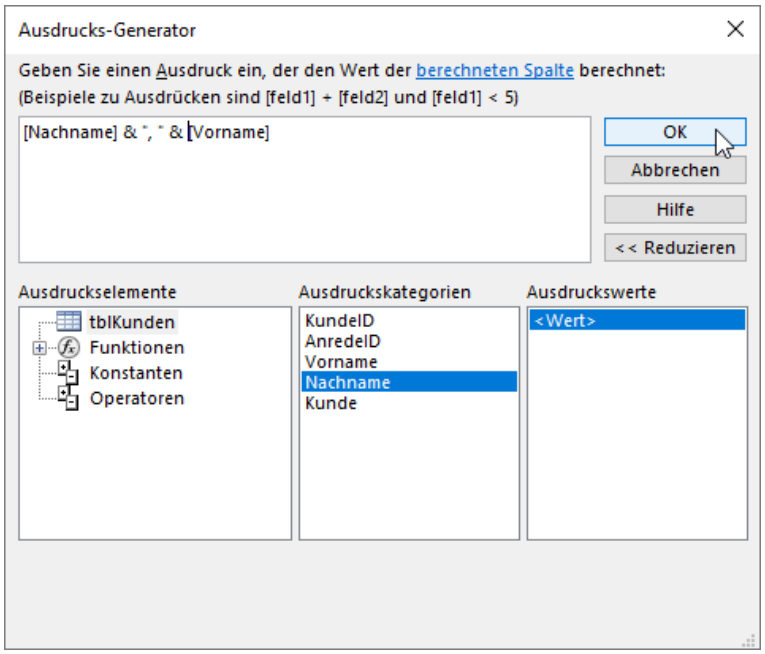


Bild 4: Eingabe des zu berechnenden Ausdrucks mit dem Ausdrucks-Generator

Danach erscheint der Ausdrucks-Generator, der im Gegensatz zum Ausdrucks-Generator etwa für Kriterien in Abfragen einige Einträge weniger aufweist (siehe Bild 4).

Wir benötigen in diesem ersten Beispiel die beiden Felder **Nachname** und **Vorname**, die wir jeweils per Doppelklick hinzufügen. Der Inhalt des oberen Fensters sieht dann so aus:

```
[Nachname] «Ausdr» [Vorname]
```

Den mittleren Teil ersetzen Sie dann wie im Bild, sodass das Ergebnis wie folgt aussieht:

```
[Nachname] & ", " & [Vorname]
```

Schließen wir den Ausdrucks-Generator mit einem Klick auf die **OK**-Schaltfläche, landet der Ausdruck in der Eigenschaft **Ausdruck** des berechneten Feldes im Tabellenentwurf. Auch wenn die Tabelle bereits zuvor einige Datensätze enthielt, wird die Berechnung der Inhalt für das neue Feld ausgeführt, sodass die Datenblattansicht nun wie in Bild 5 aussieht.

Laufende Summen in Tabellen

Immer wieder taucht die Frage auf, wie man in Access die Inhalte von Zahlenfelder mehrerer Datensätze aufsummiert. Unter Excel ist das einfach – dort trägt man einfach die Summe aus der Summe bis zur vorherigen Zeile plus dem Wert der aktuellen Zeile ein. Ändert man Werte, ändern sich auch die Summen. In diesem Artikel schauen wir uns an, wie dies unter Access mit Abfragen gelingt.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1901_Kumulieren.accdb**.

Aufsummieren an verschiedenen Stellen

Die erste Unterscheidung, die wir treffen müssen, ist die bezüglich des Ortes, an dem der kumulierte Wert erscheint. Sollen die kumulierten Werte in der Tabelle zum jeweiligen Datensatz gespeichert werden? Oder soll eine Abfrage jeweils das aktuelle Ergebnis in einer eigenen Spalte liefern? Vielleicht sollen die kumulierten Zahlen auch nur in einem Formular oder Bericht erscheinen?

Wichtig ist hier, die Bedeutung des Speicherns kumulierter Werte zu erkennen. Wenn Sie das tun, stellen Sie quasi eine Abhängigkeit verschiedener Datensätze voneinander her, wenn Sie beispielsweise die Werte von drei Datensätze wie folgt kumulieren:

Wert	Summe
12	12
5	17
23	40

Die Daten müssen nun immer in dieser Reihenfolge angezeigt werden, außerdem dürfen keine Datensätze gelöscht werden oder die Werte in der zu kumulierenden Spalte geändert werden, da sonst das Ergebnis nicht mehr stimmt. Wenn Sie etwa den Wert der zweiten Zeile von **5** auf **4** ändern, stimmen die kumulierten Werte der zweiten und dritten Zeile nicht mehr. Gleiches gilt, wenn Sie einen der Datensätze löschen. Nur beim letzten Datensatz wirkt sich dies nicht fehlerhaft aus.

In einer Abfrage kann man durch geschickten Einsatz von Berechnungsformeln dafür sorgen, dass die kumulierten Werte dynamisch erzeugt werden und dass sich Änderungen an den Daten oder auch das Hinzufügen oder Löschen von Datensätzen nicht in Form von Fehlern auswirken, sondern sich allenfalls die Spalte mit den kumulierten Werten ändert.

Beispieltabellen

Wir verwenden die Tabelle **tblAusgaben** als Beispiel, die neben dem Primärschlüsselfeld noch ein Feld mit dem Datum, den Zweck und den Betrag der Ausgabe enthält sowie ein Feld mit einer Kategorie, das auf eine weitere Tabelle verweist (siehe Bild 1).

Kumulieren in Tabellen

Wir schauen uns als Erstes an, wie wir die Zahlen in Tabellen aufsummieren können. Dazu haben wir der Tabelle **tblAusgaben** ein paar Datensätze hinzuge-

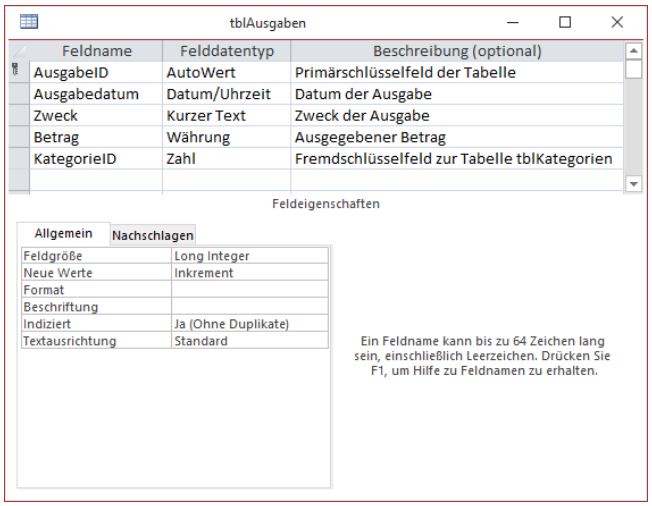
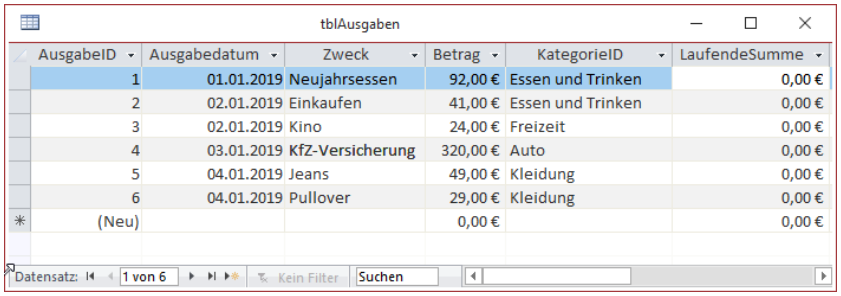


Bild 1: Tabelle für Beispielzwecke

fügt. Außerdem legen wir dort noch ein Feld namens **LaufendeSumme** an (siehe Bild 2).



tblAusgaben

AusgabeID	Ausgabedatum	Zweck	Betrag	KategorieID	LaufendeSumme
1	01.01.2019	Neujahrsessen	92,00 €	Essen und Trinken	0,00 €
2	02.01.2019	Einkaufen	41,00 €	Essen und Trinken	0,00 €
3	02.01.2019	Kino	24,00 €	Freizeit	0,00 €
4	03.01.2019	KfZ-Versicherung	320,00 €	Auto	0,00 €
5	04.01.2019	Jeans	49,00 €	Kleidung	0,00 €
6	04.01.2019	Pullover	29,00 €	Kleidung	0,00 €
(Neu)			0,00 €		0,00 €

Datensatz: 1 von 6 | Kein Filter | Suchen

Bild 2: Tabelle noch ohne laufende Summe

Laufende Summe per VBA

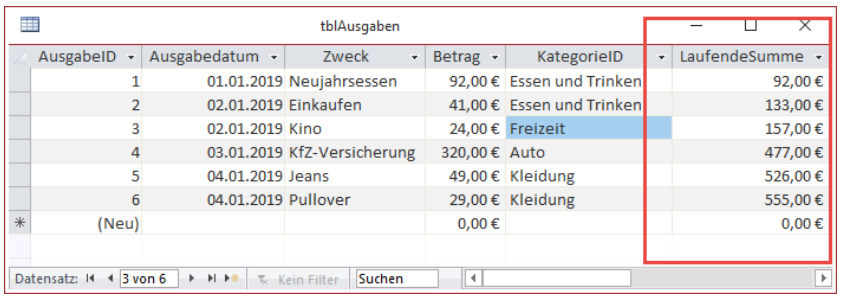
Wer sich mit VBA und dem DAO-Objektmodell auskennt, kann die Spalte LaufendeSumme nun recht schnell füllen. Dazu verwenden wir beispielsweise die Prozedur aus Listing 1.

```
Public Sub FeldAufsummieren()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim curLaufendeSumme As Currency
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM tblAusgaben", dbOpenDynaset)
    Do While Not rst.EOF
        rst.Edit
        curLaufendeSumme = curLaufendeSumme + rst!Betrag
        rst!LaufendeSumme = curLaufendeSumme
        rst.Update
        rst.MoveNext
    Loop
End Sub
```

Listing 1: Aufsummieren der Werte des Feldes einer Tabelle

Hier deklarieren und füllen wir ein Recordset mit den Daten der Tabelle **tblAusgaben**. Dadurch, dass wir keine Sortierung angeben, werden die Datensätze nach dem Primärschlüsselwert sortiert. Danach durchlaufen wir die Datensätze in einer **Do While**-Schleife und versetzen dabei zunächst den aktuellen Datensatz mit der **Edit**-Methode in den Bearbeitungsmodus. Dann addieren wir den Wert des Feldes **rst!Beitrag** zu dem in der Variablen **curLaufendeSumme** gespeicherten Wert.

Im ersten Durchlauf ist dieser noch **0**, sodass der Wert nach dem Ausführen der Anweisung den Wert des Feldes **Betrag** enthält. Bei den folgenden durchlaufen wird hier jeweils der Wert des Feldes **Betrag** hinzuaddiert. Außerdem landet dieser Wert im Feld **LaufendeSumme** des jeweiligen Datensatzes. Die **Update**-Methode speichert die Änderungen und die **MoveNext**-Methode verschiebt den Datensatzzeiger auf den nächsten Datensatz, damit dieser im folgenden Durchlauf der **Do While**-Schleife verarbeitet werden kann.



tblAusgaben

AusgabeID	Ausgabedatum	Zweck	Betrag	KategorieID	LaufendeSumme
1	01.01.2019	Neujahrsessen	92,00 €	Essen und Trinken	92,00 €
2	02.01.2019	Einkaufen	41,00 €	Essen und Trinken	133,00 €
3	02.01.2019	Kino	24,00 €	Freizeit	157,00 €
4	03.01.2019	KfZ-Versicherung	320,00 €	Auto	477,00 €
5	04.01.2019	Jeans	49,00 €	Kleidung	526,00 €
6	04.01.2019	Pullover	29,00 €	Kleidung	555,00 €
(Neu)			0,00 €		0,00 €

Datensatz: 3 von 6 | Kein Filter | Suchen

Bild 3: Tabelle mit Werten im Feld **LaufendeSumme**

Das Ergebnis nach dem Ausführen der Prozedur (etwa durch Betätigen der Taste **F5**, während sich die Einfügemarke in der Prozedur befindet) finden Sie in Bild 3.

Problem: Neue Datensätze

Wenn Sie nun feststellen, dass Sie vergessen haben, eine Ausgabe für ein zurückliegendes Datum, etwa am 2.1.2019, einzutragen, landet der neue Datensatz zwar ganz unten.

Das ist kein Problem, aber Sie wollen die Ausgaben nach dem Datum sortiert sehen und nicht nach der Reihenfolge des Anlegens. Sortieren Sie die Daten

also nach dem Anlegedatum, erscheint der neue Datensatz zwischen den anderen Datensätzen (siehe Bild 4).

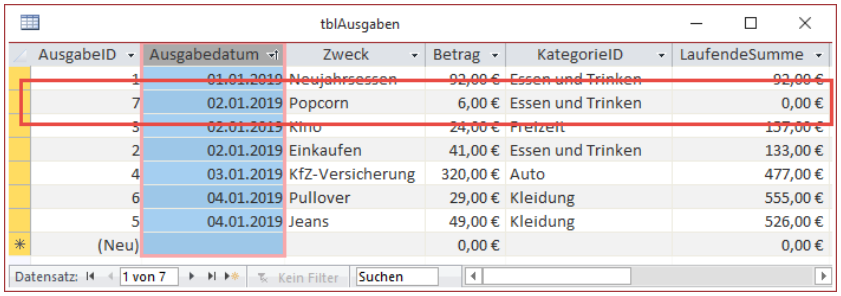


Bild 4: Neuer Datensatz zwischen den anderen Datensätzen

Wir müssen die Prozedur **FeldAufsummieren** also nochmals aufrufen. Diese liefert dann allerdings ein in dieser Sortierung falsches Ergebnis, weil die Prozedur in der Reihenfolge der Primärschlüsselwerte aufsummiert, wir aber die Summe nach dem Feld **Ausgabedatum** sortiert benötigen (siehe Bild 5).

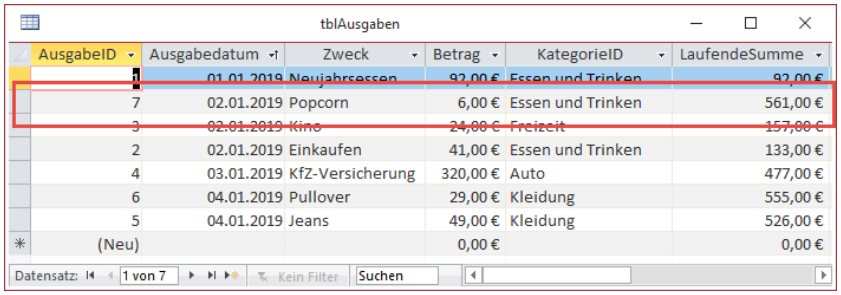


Bild 5: Falsche Reihenfolge beim Aufsummieren

Also ändern wir die Prozedur, indem wir die Zeile zum Erstellen des Recordset-Objekts wie folgt um eine **ORDER BY**-Klausel erweitern:

```
Set rst = db.OpenRecordset("SELECT * FROM tblAusgaben
ORDER BY Ausgabedatum", dbOpenDynaset)
```

Damit erhalten wir dann nach erneutem Aufruf der Prozedur **FeldAufsummieren** die korrekten Summen im Feld **LaufendeSumme**.

LaufendeSumme per Datenmakro

Ab Access 2010 gibt es die sogenannten Datenmakros. Dabei handelt es sich um einen speziellen Typ von Makros, der durch verschiedene Aktionen wie das Anlegen, Löschen oder Ändern eines Datensatzes ausgelöst werden kann. Das ist in unserem Fall interessant, weil wir ja sicherstellen sollen, dass die Daten im Feld **LaufendeSumme** aktuell sind – auch wenn der Benutzer zwischendurch die Daten geändert hat.

Das Problem der Sortierung lösen wir damit allerdings nicht, denn es gibt kein Ereignis für Datenmakros, das nach dem Ändern der Sortierung von Datensätzen einer Tabelle ausgelöst wird.

Wir wollen zunächst ein Makro entwickeln, dass beim Ändern des Wertes im Feld **Betrag** eines der Datensätze ausgelöst wird und dann dafür sorgt, dass die Werte im Feld **LaufendeSumme** neu berechnet werden.

Dazu erstellen wir eine Kopie der Tabelle **tblAusgaben** unter dem Namen **tblAusgabenDatenmakro**.

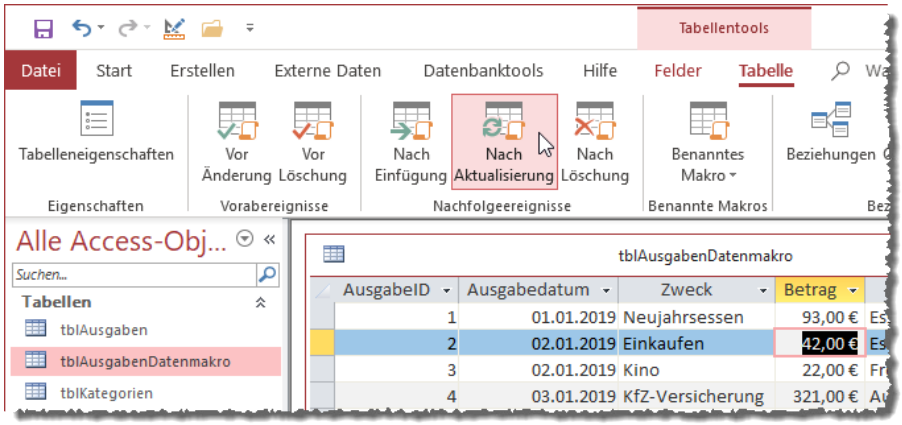


Bild 6: Anlegen eines Datenmakros

Danach wechseln wir bei geöffnete Datenblattansicht der so kopierten Tabelle zum Ribbon-Element **Tabelle|Nachfolgeereignisse|Nach Aktualisierung** und klicken dieses an (siehe Bild 6).

Daraufhin erscheint die Entwurfsansicht für Makros. Hier geben Sie zunächst den Befehl **FestlegenLokaleVar** ein. Mit diesem Stellen Sie den Wert einer lokalen Variablen namens **varLaufendeSumme** auf **0** ein. Darunter fügen Sie den Befehl **Für jeden Datensatz in** hinzu. Wählen Sie dafür die Tabelle **tblAusgabenDatenmakro** aus. Als ersten Befehl innerhalb dieser Schleife erhöhen wir den Wert der Variablen **varLaufendeSumme** um den Wert des Feldes **Betrag**. Dann bearbeiten wir mit dem Befehl **FestlegenFeld** den aktuellen Datensatz mit dem Makrobefehl **DatensatzBearbeiten** und stellen den Wert des Feldes **LaufendeSumme** auf den aktuellen Wert der Variablen **varLaufendeSumme** ein (siehe Bild 7).

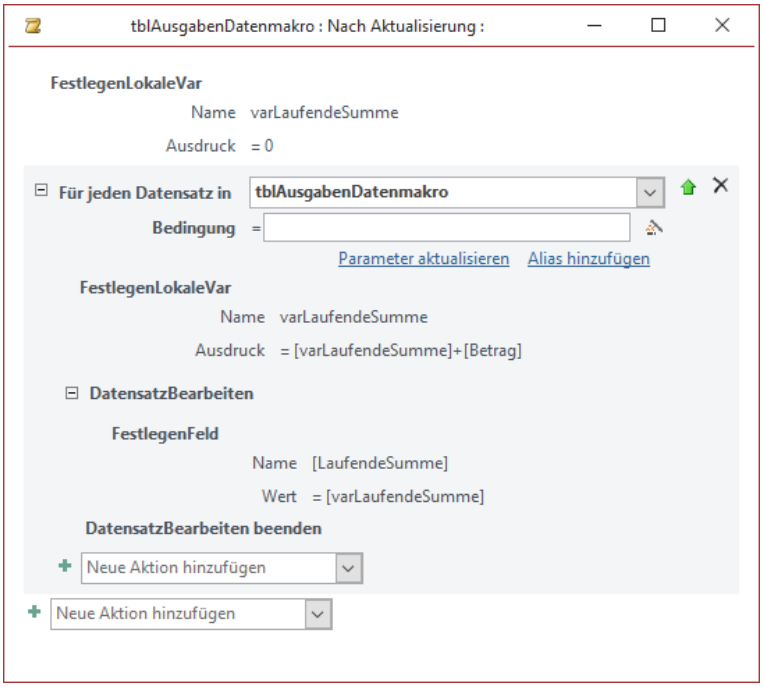


Bild 7: Erstellen des Datenmakros

Schließen Sie nun den Makro-Editor und bestätigen Sie gegebenenfalls die Meldung zum Speichern der Änderungen. Ändern Sie nun den Wert des Feldes **Betrag** für einen der Datensätze, sollte das Datenmakro eigentlich die laufende Summe in das Feld **LaufendeSumme** für die Datensätze der Tabelle eintragen. Aber es geschieht nichts. In einem solchen Fall schauen wir immer in den Backstage-Bereich der Anwendung nach, um zu sehen, ob es Einträge im Anwendungsprotokoll gibt. Die Schaltfläche aus Bild 8 wird dann auch nur in diesem Fall angezeigt.

Bild 8: Es gibt Fehler im Anwendungsprotokoll.

Klicken wir die Schaltfläche an, erscheint die Tabelle aus Bild 9. Hier können wir der Meldung entnehmen, dass sich das Makro offensichtlich immer wieder selbst aufgerufen hat. Das ist auch logisch, denn wenn wir ein Makro anlegen, dass durch das Ereignis **Nach Aktualisierung** eines Datensatzes ausgelöst wird und welches dadurch eine weitere Aktualisierung anstößt, triggert dies wiederum das Ereignis **Nach Aktualisierung**. In der Folge werden alle Änderung, die durch das Makro durchgeführt wurden, verworfen.

Wir müssen also noch eine Sicherung einbauen, die prüft, welches Feld geändert wurde beziehungsweise die sicherstellt, dass die Änderung nicht durch das Feld **LaufendeSumme** ausgelöst wurde. Dazu

Laufende Summen in Abfragen

Immer wieder taucht die Frage auf, wie man in Access die Inhalte von Zahlenfelder mehrerer Datensätze aufsummiert. Unter Excel ist das einfach – dort trägt man einfach die Summe aus der Summe bis zur vorherigen Zeile plus dem Wert der aktuellen Zeile ein. Ändert man Werte, ändern sich auch die Summen. Im Artikel »Laufende Summen in Tabellen« haben wir uns angesehen, wie dies mit einem zusätzlichen Feld in Tabelle gelingt. In diesem Artikel schauen wir uns an, wie dies unter Access unter Verwendung von Abfragen gelingt.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1901_KumulierenInAbfragen.accdb**.

Beispieltabelle

Wie im oben bereits erwähnten Artikel **Laufende Summen in Tabellen** verwenden wir auch in diesem Artikel die Tabelle **tblAusgaben** als Beispiel (siehe Bild 1).

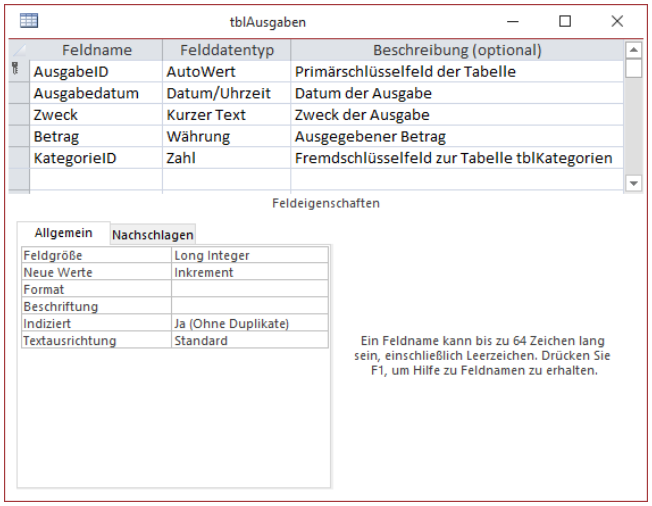


Bild 1: Tabelle für Beispielzwecke

Laufende Summe per Abfrage

Dort finden Sie allerdings kein Feld namens **LaufendeSumme**, mit dem wir im Artikel **Laufende Summen in Tabellen** gezeigt haben, wie Sie die Werte eines Feldes in einem anderen Feld aufsummieren. Stattdessen wollen wir ein solches Feld mithilfe einer Abfrage hinzufügen.

Während wir bei der Methode mit dem Aufsummieren der Werte direkt in der Tabelle die Datensätze der Tabelle durchlaufen und diese dabei Datensatz für Datensatz aufsummiert haben, sieht die Technik bei Abfragen ein wenig anders aus.

Hier können wir zwar durch eine geschickt formulierte Abfrage mit einem berechneten Feld auf die Werte des vorherigen Datensatzes zugreifen, aber nicht auf den Wert, den das berechnete Feld für den vorherigen Datensatz geliefert hat.

Laufende Summe nach Primärschlüsselwert

Stattdessen müssen wir, wenn wir etwa den Wert eines Feldes über alle Datensätze aufsummieren wollen, immer die Summe über das Feld für alle Datensätze ermitteln, die sich in der Reihenfolge vor dem aktuellen Datensatz befinden. Dazu nutzen wir eine Abfrage, die alle Felder der Tabelle **tblAusgaben** enthält. Außerdem fügen wir dieser Abfrage ein weiteres Feld hinzu, das wir als berechnetes Feld auslegen.

Also geben wir den Namen für die anzuzeigende Spalte vorn an, gefolgt von einem Doppelpunkt und dem Ausdruck, der die anzuzeigenden Werte festlegt.

Für den ersten Datensatz soll dieser Ausdruck den Wert des Feldes **Betrag** des aktuellen Datensatzes ausgeben, für die übrigen Datensätze jeweils die Summe aller Werte des Feldes **Betrag** aller Datensätze, die in der Reihenfolge über dem aktuellen Datensatz liegen und dazu noch den Wert des Feldes **Betrag** des aktuellen Datensatzes addieren. Dieser Ausdruck sieht wie folgt aus:

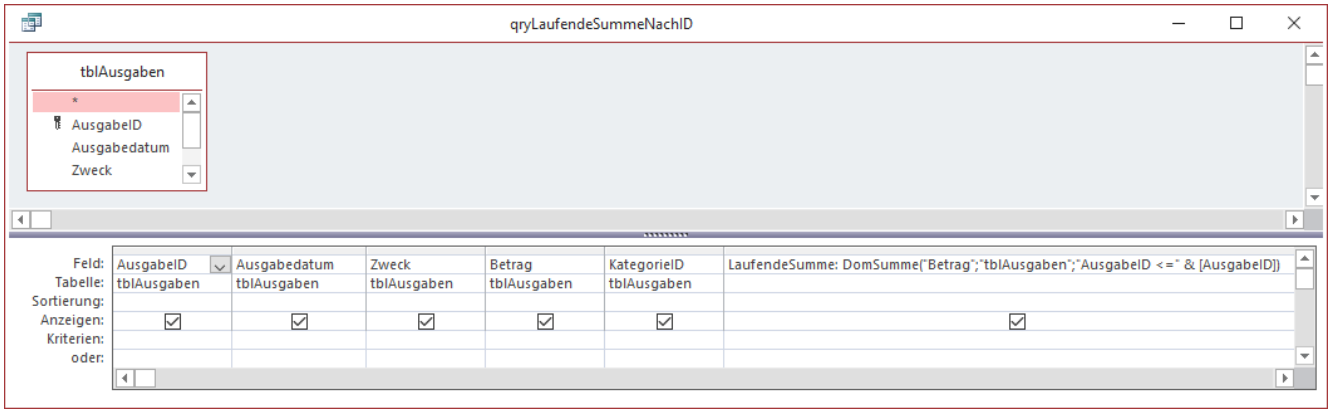


Bild 2: Abfrage mit laufender Summe, sortiert nach dem Primärschlüsselwert

```
LaufendeSumme: DomSumme("Betrag";"tblAusgaben";"Ausgabe
ID <=" & [AusgabeID])
```

Die Funktion **DomSumme** entspricht der **DSum**-Funktion unter VBA. Sie erwartet als ersten Parameter den Namen des Feldes, dessen Werte aufsummiert werden sollen. Der zweite Parameter gibt den Namen der Tabelle oder Abfrage an, welche die Werte liefert.

Und der dritte Parameter liefert das Kriterium, das festlegt, welche Datensätze von der Summenbildung berücksichtigt werden sollen. In diesem Fall lautet dieser **"AusgabeID <=" & AusgabeID**.

Das bedeutet, dass der Wert des Feldes **AusgabeID** der Datensätze der Datenquelle aus dem zweiten Parameter mit dem Wert des Feldes **AusgabeID** des aktuellen Datensatzes verglichen wird.

Der Entwurf dieser Abfrage sieht schließlich wie in Bild 2 aus. Für den Datensatz mit dem Wert 4 im Feld **AusgabeID** liefert diese Funktion also die Summe der Werte des Feldes **Betrag** für die Datensätze, die im Feld **AusgabeID** einen Wert kleiner oder gleich 4 aufweisen.

Das Ergebnis dieser Abfrage sieht für unsere Beispieldaten wie in Bild 3 aus.

Laufende Summe nach Datum

Um die laufende Summe nach dem Datum zu ermitteln, benötigen wir erst einmal ein entsprechendes Kriterium für die Sortierreihenfolge der Datensätze. In diesem Fall wollen wir die Daten zuerst nach dem Datum sortieren, das im Feld **Ausgabedatum** gespeichert ist, und bei gleichen Datumsangaben nach dem Primärschlüsselwert. Der Entwurf der dazu notwendigen Abfrage sieht wie in Bild 4 aus.

Wichtig ist hier, dass wir nicht einfach die beiden Felder **AusgabeID** und **Ausgabedatum** mit dem Wert **Aufsteigend** in der Zeile **Sortierung** versehen können. Dann würde die Sortierung nämlich erst nach dem Primärindex erfolgen und dann nach dem Ausgabedatum, was praktisch bedeuten würde, dass die Datensätze nur nach dem Primärschlüsselfeld sortiert

qryLaufendeSummeNachID

AusgabeID	Ausgabedatum	Zweck	Betrag	KategorieID	LaufendeSu
1	01.01.2019	Neujahrsessen	92,00 €	Essen und Trinken	92
2	02.01.2019	Einkaufen	41,00 €	Essen und Trinken	133
3	02.01.2019	Kino	24,00 €	Freizeit	157
4	03.01.2019	KfZ-Versicherung	320,00 €	Auto	477
5	04.01.2019	Jeans	49,00 €	Kleidung	526
6	04.01.2019	Pullover	29,00 €	Kleidung	555
7	02.01.2019	Popcorn	6,00 €	Essen und Trinken	561
(Neu)			0,00 €		

Datensatz: 1 von 7 Kein Filter Suchen

Bild 3: Ergebnis der Summenbildung, sortiert nach dem Primärschlüsselwert

werden würden – die Werte dieses Feldes sind ja eindeutig und somit ist kein zweites Sortierkriterium erforderlich.

Da wir aber dennoch wollen, dass das Feld **AusgabeID** ganz links als erstes Feld angezeigt wird, müssen wir es ein zweites Mal zur Abfrage hinzufügen. Dieses zweite Feld können wir dann mit der aufsteigenden Sortierung versehen. Außerdem sollten wir seine Anzeige deaktivieren, indem wir das Häkchen aus dem Kontrollkästchen in der Zeile **Anzeigen** entfernen.

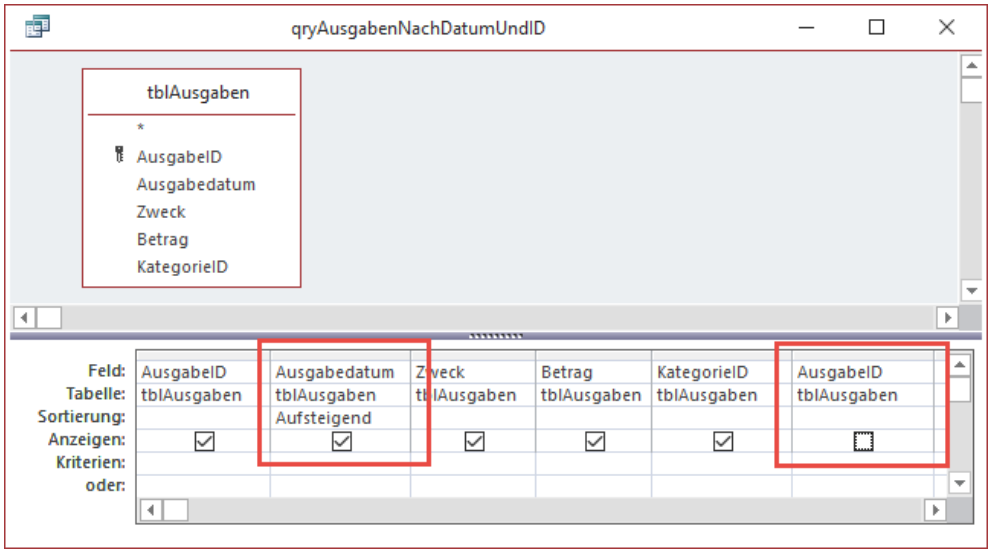


Bild 4: Sortierung nach **Ausgabedatum** und **AusgabeID**

Summenfeld hinzufügen

Nun fügen wir der Abfrage noch ein weiteres Feld hinzu, das als berechnetes Feld ausgelegt wird. Das heißt, dass wir das Feld nicht aus der Feldliste in das Entwurfsraster ziehen, sondern den Ausdruck selbst formulieren. Wo ist der Unterschied dieses Summenfelds im Vergleich zur ersten Abfrage, wo wir nach dem Primärschlüsselfeld sortiert haben? Wir müssen das Kriterium ändern, da wir ja nun nach dem Feld Ausgabedatum sortieren.

Normalerweise sollte dies also funktionieren, wenn wir das Feld zur Ermittlung der laufenden Summe etwa wie folgt hinzufügen:

```
LaufendeSumme: DomSumme("Betrag";"tblAusgaben";"Ausgabedatum <=" & [Ausgabedatum])
```

Das Ergebnis sieht allerdings wie in Bild 5 aus – wir erhalten **#Fehler** als Ergebnis. Zum Glück erhalten

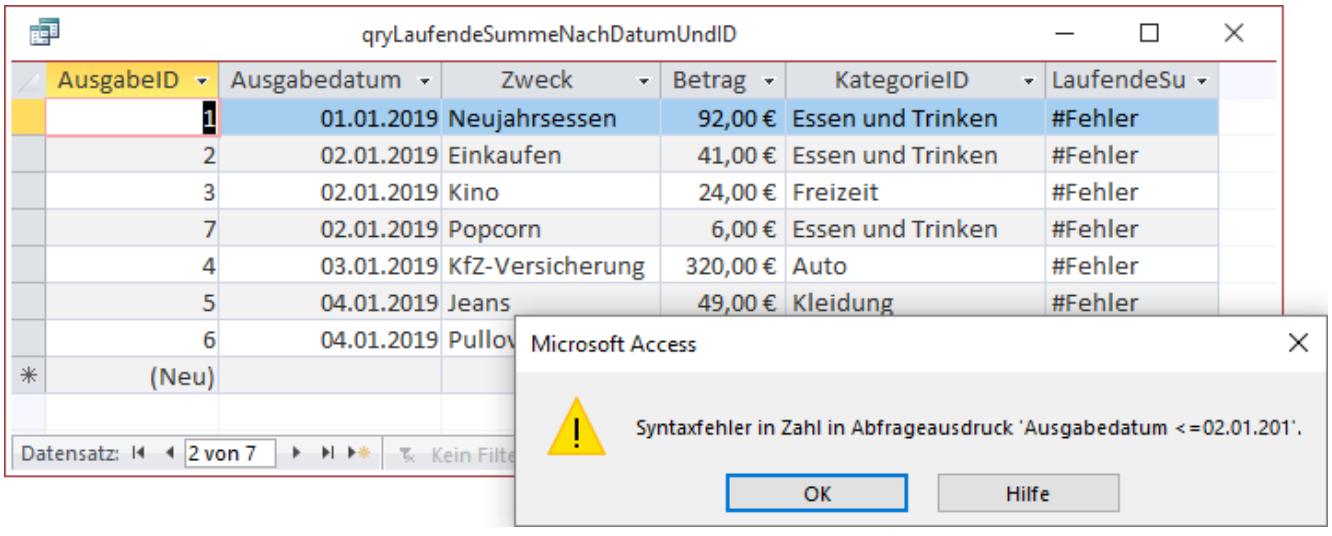
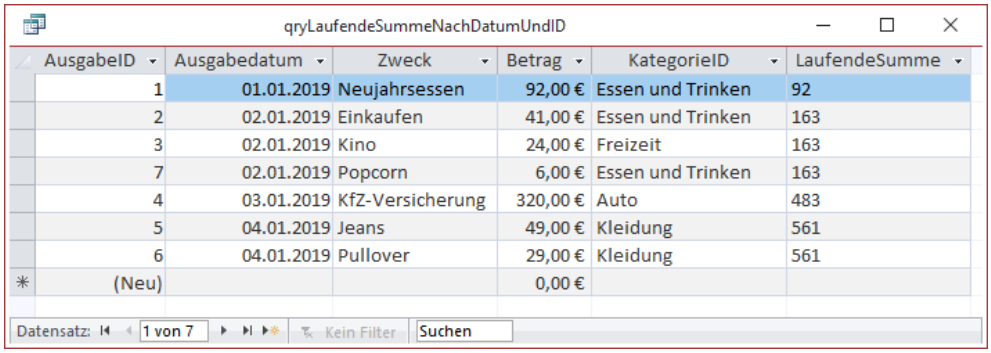


Bild 5: Sortierung nach **Ausgabedatum** und **AusgabeID** mit Fehler

wir, nachdem wir auf den Wert **#Fehler** klicken, noch eine aufklärende Fehlermeldung. Diese gibt den Abfrageausdruck aus, der im Falle des ersten Datensatzes so lautet (die letzte Stelle wurde aus offensichtlich aus Platzgründen abgeschnitten):


qryLaufendeSummeNachDatumUndID

AusgabeID	Ausgabedatum	Zweck	Betrag	KategorieID	LaufendeSumme
1	01.01.2019	Neujahrsessen	92,00 €	Essen und Trinken	92
2	02.01.2019	Einkaufen	41,00 €	Essen und Trinken	163
3	02.01.2019	Kino	24,00 €	Freizeit	163
7	02.01.2019	Popcorn	6,00 €	Essen und Trinken	163
4	03.01.2019	KfZ-Versicherung	320,00 €	Auto	483
5	04.01.2019	Jeans	49,00 €	Kleidung	561
6	04.01.2019	Pullover	29,00 €	Kleidung	561
(Neu)			0,00 €		

Datensatz: 1 von 7 Kein Filter Suchen

Bild 6: Laufende Summe nach dem Datum – noch nicht perfekt.

```
Ausgabedatum <= 02.01.201
```

Das sieht logisch aus – mit dem Ausdruck **<= 02.01.2019** hätten wir auch in der Zeile **Kriterien** des Entwurfs einer Abfrage Probleme, weil **02.01.2019** kein gültiges Datumsformat ist.

Also wandeln wir den Ausdruck noch in eine gültige Form um. Wir könnten das Datum in Raute-Zeichen einfassen, aber das ist nicht zuverlässig genug. Also wandeln wir das Datum einfach in das Format um, in dem Datumswerte auch in den Tabellen gespeichert werden, nämlich in **Long**.

Wenn Sie Datum und Zeit verwenden wollen, benötigen Sie den Datentyp **Double** wegen der Nachkommastellen, welche die Uhrzeit repräsentieren.

Die Spalte sieht nun also wie folgt aus:

```
LaufendeSumme: DomSumme("Betrag";"tblAusgaben";"Ausgabedatum <=" & ZLong([Ausgabedatum]))
```

Damit erhalten wir nun das Ergebnis aus Bild 6. Wir erhalten zwar nicht #Fehler als Ergebnis, aber die laufende Summe ist auch noch nicht korrekt. Sie ermittelt die laufende Summe nämlich immer für alle Tage eines Datums plus der vorhergehenden Tage. Auch das ist logisch, denn wir haben ja das Kriterium so formuliert.

Also ist noch eine Anpassung notwendig, nämlich die Ergänzung um das Primärschlüsselfeld als weiteres Kriterium. Das Feld **LaufendeSumme** sieht dann wie folgt aus:

```
LaufendeSumme: DomSumme("Betrag";"tblAusgaben";"Ausgabedatum <=" & ZLong([Ausgabedatum] & " AND AusgabeID <= " & [AusgabeID]))
```

Wir fügen also mit dem **AND**-Schlüsselwort ein weiteres Kriterium hinzu, mit dem wir festlegen, dass die **AusgabeID** der aufsummierten Einträge auch noch kleiner sein soll als die **AusgabeID** des aktuellen Datensatzes. Das schlägt allerdings fehl (wieder mit **#Fehler** als Ergebnis). Warum das? Ein Anklicken des Feldes mit dem Inhalt **#Fehler** liefert leider keine Fehlermeldung wie beim vorherigen Versuch.

Eine kurze Analyse unseres Ausdrucks ergibt dann auch einen Denkfehler: Wir wollen nicht die laufende Summe der Beträge für alle Einträge, deren **AusgabeID** und **Ausgabedatum** kleiner sind als die Werte im aktuellen Datensatz plus dem Betrag des aktuellen Datensatzes ermitteln, sondern zunächst die Werte für alle Datensätze, deren Datum kleiner oder gleich als das Datum des aktuellen Datensatzes ist (plus dem Betrag des aktuellen Datensatzes) und dann noch die Reihenfolge der Primärschlüsselwerte berücksichtigen. Das lässt sich in Form zweier einzelner Kriterien in der **DomSumme**-Funktion nicht realisieren.

Hier benötigen wir einen kleinen Trick. Wie können wir das Primärschlüsselfeld **AusgabeID** berücksichtigen, wenn es in der Priorität der Sortierung dem Feld **Ausgabedatum** untergeordnet ist?

Linien und Rechtecke

Linien und Rechtecke sind unscheinbare Elemente, die man vermeintlich nie oder nur selten benutzt. Das gilt vor allem für Formulare, in Berichten kommen diese Steuerelemente schon eher zum Einsatz, um beispielsweise Trennlinien zwischen zwei Datensätzen zu ziehen oder zusammenhängende Elemente einzurahmen. Dieser Artikel zeigt, wie Sie Rechtecke und Linien anlegen und welche Eigenschaften diese Steuerelemente aufweisen und wie Sie diese einstellen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1901_LinienUndRechtecke.accdb**.

Linien

Um einem Formular oder einem Bericht eine Linie hinzuzufügen, klicken Sie wie üblich auf das gewünschte Steuerelement der Steuerelemente-Liste und ziehen dann einen Rahmen im Zielobjekt auf, um die Position und die Größe festzulegen (siehe Bild 1).

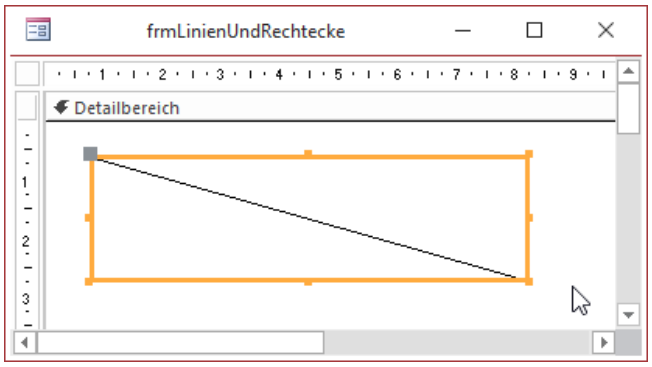


Bild 1: Eine neue Linie in einem Formular

Für das Finetuning der Position, Länge und vor allem der Ausrichtung der Linie verwenden Sie entweder die Maus und ziehen den Rahmen der Markierung an den acht ausgefüllten Rechtecken, bis die Linie die gewünschte Position, Ausrichtung und Länge hat.

Das gleiche können Sie auch erreichen, indem Sie die Eigenschaften **Oben**, **Links**, **Breite** und **Höhe** im Eigenschaftenblatt einstellen. Wenn Sie die Eigenschft **Höhe** auf **0** einstellen, erhalten Sie eine horizontal ausgerichtete Linie, wenn Sie die Eigenschaft **Breite** auf **0** einstellen, ist die Linie vertikal ausgerichtet.

Bild 2: Die Eigenschaft **Neigung**

Wenn Sie eine Linie erstellt haben, die von links oben nach rechts unten verläuft und Sie möchten dies so ändern, dass diese von links unten nach rechts oben verläuft, brauchen Sie nur die Eigenschaft **Neigung** von **/** auf den Wert **** zu verändern (siehe Bild 2).

Formular durchstreichen

Wenn Sie das komplette Formular aus irgendwelchen Gründen als durchgestrichen kennzeichnen wollen, können Sie etwa eine Linie von links unten nach rechts oben und eine von links oben nach rechts

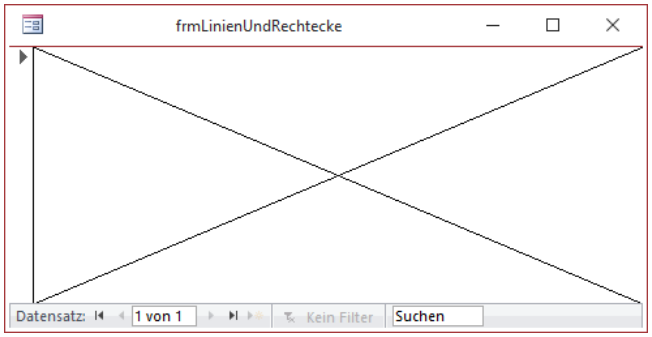


Bild 3: Durchgestrichenes Formular

Formularansicht filtern, Teil II

Access bietet einige Möglichkeiten, um die Daten in Formularen zu filtern – zum Beispiel die Möglichkeiten, die auftauchen, wenn Sie das Kontextmenü eines Feldes in der Formularansicht anzeigen oder die Funktionen, die über das Ribbon zur Verfügung stehen. Wir schauen uns diese Möglichkeiten in diesem Artikel einmal im Detail an und schauen, wie wir diese Funktionen per VBA aufrufen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1806_FormulareFiltern.accdb**.

Beispielformular

Für die Beispiele verwenden wir ein Formular namens **frmArtikelFiltern**, das wir mit der Tabelle **tblArtikel** als Datensatzquelle versehen und mit dem wir alle Felder dieser Tabelle anzeigen.

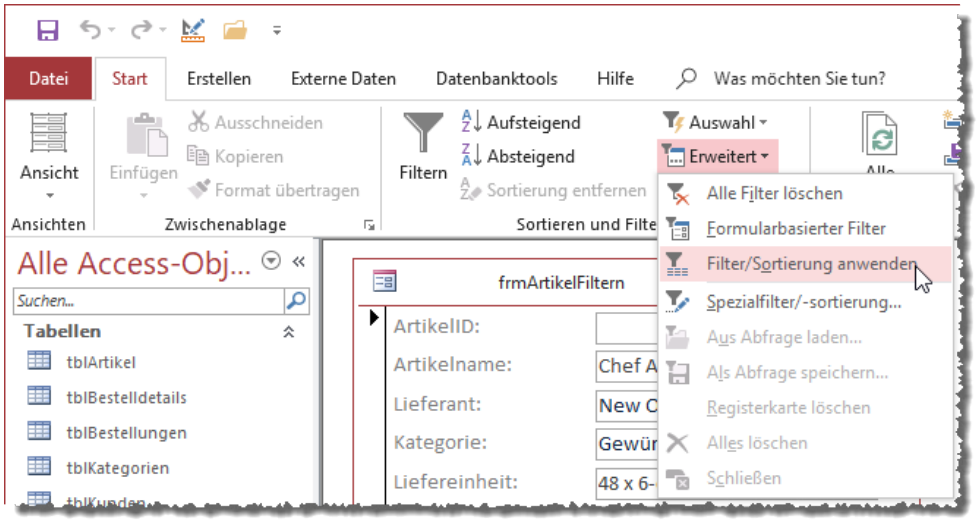


Bild 1: Ribbon-Befehle zum erweiterten Filtern

Formularbasierte Filter

Im ersten Teil dieser Artikelreihe haben wir bereits angedeutet, dass wir uns noch die weiteren Befehle im Ribbon ansehen wollen. Wenn Sie ein Formular geöffnet und eines der gebundenen Steuerelemente aktiviert haben, sind im Ribbon-Menü unter **Start|Sortieren und Filtern|Erweitert** zunächst die Befehle **Alle Filter löschen**, **Formularbasierter Filter**, **Filter/Sortierung anwenden** und **Spezialfilter/-sortierung...** aktiviert (siehe Bild 1). Noch nicht aktiviert sind die Befehle **Aus Abfrage laden...** und **Als Abfrage speichern...**. Wann diese aktiviert werden, schauen wir uns später an.

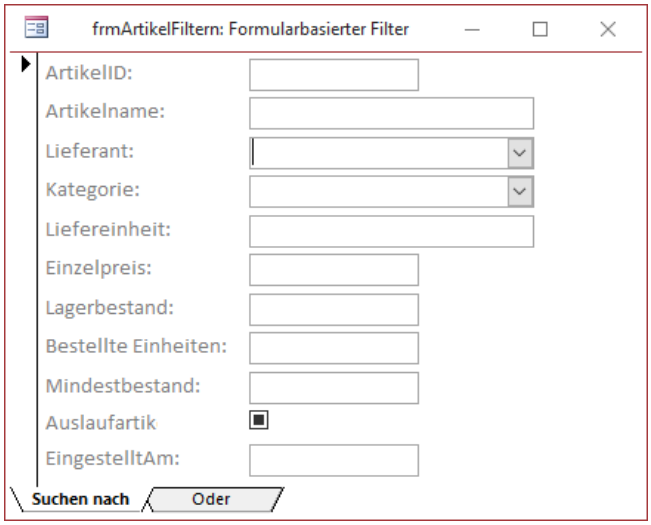


Bild 2: Formularbasierter Filter

Klicken Sie auf den Eintrag **Formularbasierter Filter**, erscheint eine neue Ansicht des Formulars. Diese enthält die gleichen Steuerelemente, aber keine Daten mehr. Die Überschrift des Formulars wird mit dem Text **Formularbasierter Filter** erweitert (siehe Bild 2). Außerdem finden wir unten zwei Registerreiter vor, und zwar mit den Beschriftungen **Suchen nach** und **Oder**. Wir können hier nun zunächst einmal einen Filterausdruck für den Artikelnamen eingeben, zum Bespiel **C*** für alle Artikel, deren Name mit dem

Buchstaben **C** beginnt, und dann den Filter aktivieren. Nach der Eingabe fragen wir uns allerdings, wie genau wir den Filter aktivieren können.

Interessant ist jedenfalls erstmal, dass der Vergleichswert nach dem Wechsel in ein anderes Steuerelement gleich in einen Vergleichsausdruck umgewandelt wird, in diesem Fall in den Ausdruck **Wie "C*"** (siehe Bild 3).

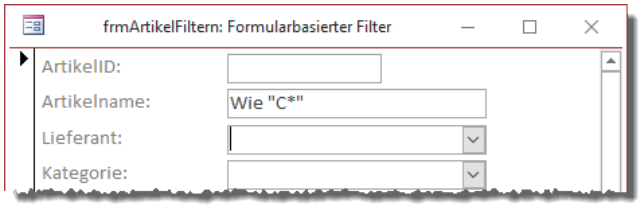


Bild 3: Umwandlung des Kriteriums in einen Vergleichsausdruck

Um den Filter nun auszuführen, betätigen Sie den Ribbon-Befehl **Start|Sortieren und Filtern|Filter ein/aus**. Danach verschwindet die Ansicht zum Eingeben der Filterkriterien wieder und das Formular zeigt nur noch die den Filterkriterien entsprechenden Datensätze an (siehe Bild 4).

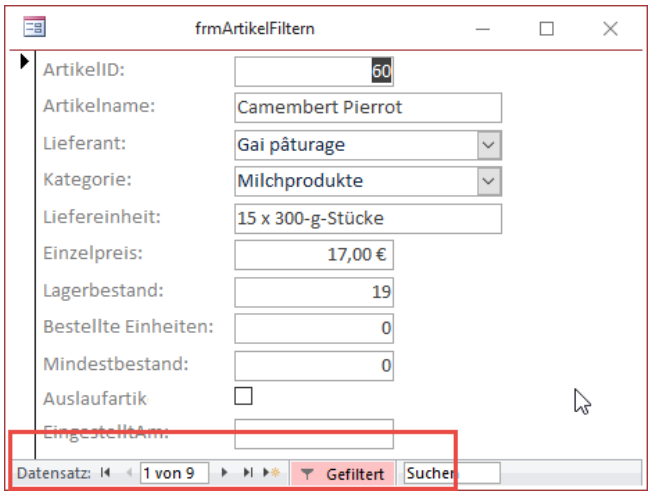


Bild 4: Nach dem Anfangsbuchstaben **C** gefilterte Datensätze

Weitere Filterkriterien eingeben

Um weitere Filterkriterien einzugeben, wechseln Sie nun einfach wieder mit dem Ribbonbefehl **Start|Sortieren und Filtern|Erweitert|Formularbasierter Filter** zur Eingabe der formularbasierten Filter. Hier können wir nun noch weitere Kriterien für die anderen Felder angeben, zum Beispiel indem wir über das Kombinationsfeld für das Feld **Kategorie** noch den Eintrag **Getränke** auswählen. Wenn wir diese Eingabe abschließen, erzeugt Access übrigens keinen Ausdruck wie **Wie "Getränke"**, sondern belässt nur den Vergleichsausdruck im Steuerelement (siehe Bild 5). Der Unterschied resultiert daraus, dass wir hier keinen Platzhalter angegeben haben. Der Vergleichsoperator **Wie** wird also nur ausgegeben, wenn ein Platzhalter vorhanden ist. Der Vergleichsoperator **=** ist also quasi der Standardoperator, der nicht explizit angegeben wird.

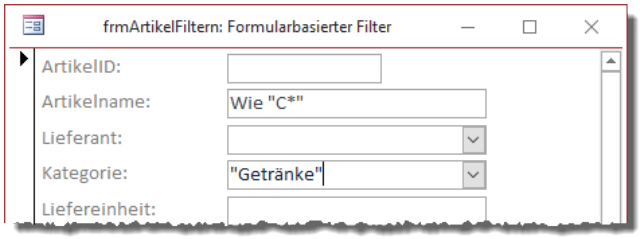


Bild 5: Zusätzlich nach der Kategorie filtern

Grundsätzlich gilt: Alle Filterkriterien, die auf einer Registerseite angegeben werden, verknüpft Access mit dem Und-Operator. Das können Sie auch für den aktuellen Filter leicht herausbekommen, wenn Sie den Filter aktivieren und dann im VBA-Editor den folgenden Befehl eingeben:

```
? Forms!frmArtikelFiltern.Filter
```

```
((tblArtikel.Artikelname Like "C*") AND (Lookup_KategorieID.Kategoriename="Getränke"))
```

Filterkriterien mit Oder verknüpfen

Wenn Sie einen Satz von Filterkriterien auf der ersten Registerseite mit **Und** verknüpft haben und dann weitere Kriterien hinzufügen wollen, die dann mit **Oder** verknüpft werden, klicken Sie einfach unten auf die Registerlasche **Oder**. Zu Beispielzwecken wollen wir alle Artikel liefern, deren Artikelname mit den Buchstaben **C** oder **A** beginnt. Auf der ersten Registerseite

VBA-Funktionen der Information-Klasse

In dieser neuen Reihe schauen wir uns verschiedene VBA-Funktionen an und zeigen, wie Sie diese sinnvoll einsetzen können. Im ersten Teil widmen wir uns der Klasse Information, die einige wichtige Funktionen liefert. Das bekannteste Element dürfte die IsNull-Funktion sein. Aber auch die anderen Funktionen sollten Sie kennenlernen, damit Sie im Fall der Fälle wissen, wo Sie diese finden und nicht etwa die Funktionen selbst programmieren.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1901_VBAInformationKlasse.accdb**.

Funktionen im Überblick

Wenn Sie den VBA-Editor einer Access-Anwendung mit der Tastenkombination **Alt + F11** öffnen und dann die Taste **F2** betätigen, erscheint der Objektkatalog von Access. Hier wählen Sie im Kombinationsfeld oben links den Eintrag **VBA** aus und dann unter Klassen den Eintrag **Information**. Danach erscheinen im rechten Fenster alle Funktionen dieser Klasse (siehe Bild 1). Hier sind die Funktionen im Überblick:

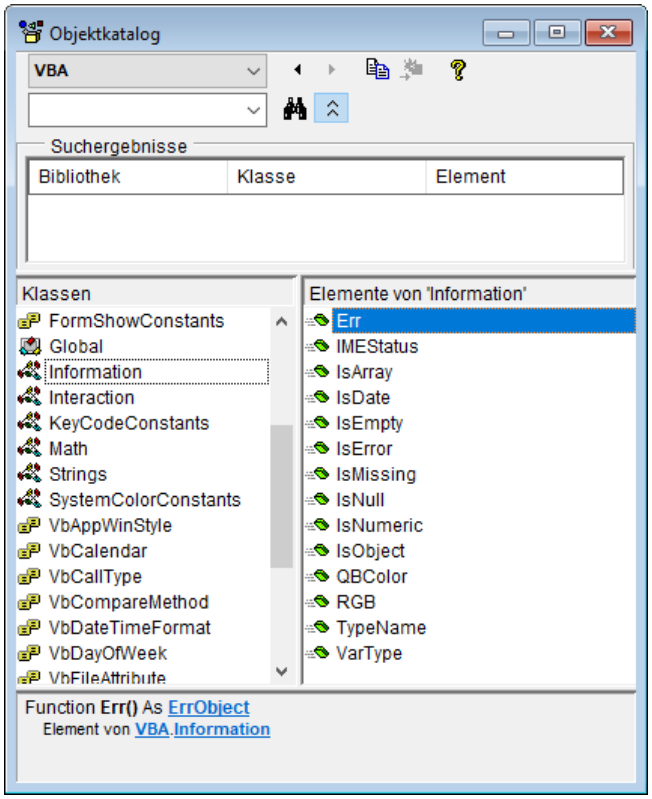


Bild 1: Die Funktion der **Information**-Klasse im Objekt-Katalog

- **Err**: Liefert ein Objekt des Typs **ErrObject**, über dessen Eigenschaften Sie Informationen über den zuletzt aufgetretenen Fehler auslesen können.
- **IMEStatus**: Wird nur in fernöstlichen Versionen von Access genutzt.
- **IsArray**: Gibt mit einem **Boolean**-Wert an, ob es sich bei der als Parameter übergebenen Variablen um ein Array handelt.
- **IsDate**: Die Funktion prüft, ob es sich bei dem mit dem Parameter übergebenen Wert um ein gültiges Datum handelt.
- **IsEmpty**: Diese Funktion prüft, ob eine Variant-Variable bereits durch Zuweisen eines Wertes initialisiert wurde.
- **IsError**: Die Funktion prüft, ob das Ergebnis einer Funktion ein mit der Funktion **CVErr** erzeugtes Objekt des Typs **Error** ist.
- **IsMissing**: Gibt an, ob ein optionaler Parameter einer Routine mit dem Datentyp **Variant** beim Aufruf angegeben wurde.
- **IsNull**: Prüft, ob ein Ausdruck den Wert **Null** hat.
- **IsNumeric**: Prüft, ob ein Ausdruck als Zahl interpretiert werden kann.
- **IsObject**: Gibt an, ob der Parameter ein **Variant** mit dem Untertyp **Objekt** ist.

- **QBColor**: Liefert Farbcodes für die Zahlen von 1 bis 15, die den wichtigsten Farben entsprechen.
- **RGB**: Ermittelt aus den Rot-, Grün- und Blau-Anteilen den VBA-Farbcode.
- **TypeName**:
- **VarType**:

Die Funktion Err

Die Funktion **Err** liefert ein Objekt des Typs **ErrObject** zurück. Sie können damit Informationen über den Fehlerstatus unter VBA abfragen. **ErrObject** enthält immer die Informationen über den zuletzt aufgetretenen Fehler, zum Beispiel die Fehlernummer und die Fehlerbeschreibung.

ErrObject hat als Standardeigenschaft die Eigenschaft **Number**. Wenn Sie also einfach die **Err**-Funktion aufrufen, liefert dies die Nummer des zuletzt aufgetretenen Fehlers. Im Direktbereich sieht das so aus, wenn seit Beginn der Session kein Fehler aufgetreten ist:

```
? Err
 0
```

Das **ErrObject**-Objekt speichert nur die Informationen über Fehler, die nicht über die eingebaute Fehlerbehandlung von Access behandelt werden. Wenn Sie also im Direktbereich etwa mit der folgenden Anweisung einen Fehler auslösen, erscheint die eingebaute Fehlermeldung mit dem Text **Laufzeitfehler 11: Division durch Null**:

```
? 1/0
```

Die Funktion **Err** liefert danach aber immer noch den Wert **0**. Err nimmt nur Fehlerinformationen auf, wenn sie vorher die eingebaute Fehlerbehandlung deaktivieren, beispielsweise mit der folgenden Anweisung:

```
Public Sub BeispielError()
    On Error Resume Next
    Debug.Print 1 / 0
    Debug.Print Err.Number, Err.Description
End Sub
```

Im Direktfenster erscheint dann:

```
11          Division durch Null
```

Das **ErrObject**-Objekt hat die folgenden Methoden und Eigenschaften:

- **Clear**: Leert das **ErrObject**-Objekt.
- **Description**: Liefert die Beschreibung des Fehlers.
- **Number**: Liefert die Nummer des Fehlers.
- **Source**: Gibt die Quelle des Fehlers an. In der Regel ist diese Eigenschaft leer, aber wenn Sie etwa mit der **Raise**-Methode einen Fehler auslösen, können Sie dort einen Wert für **Source** angeben.
- **Raise**: Löst den Fehler mit der angegebenen Nummer aus. Sie können dieser Methode unter anderem die Fehlernummer, eine Angabe der Quelle und die Beschreibung übergeben.

IsArray

Wenn Sie prüfen möchten, ob eine Variable ein Array enthält, können Sie die **IsArray**-Funktion verwenden. Die Funktion erwartet als Parameter den Namen der zu prüfenden Variablen.

Warum aber sollte man diese Funktion benötigen? Steht denn nicht immer fest, ob eine Variable ein Array ist oder nicht? Nein, denn es kann sein, dass etwa beim Aufruf einer Funktion einem **Variant**-Parameter entweder ein Array oder eine einfache Variable übergeben wird.

In diesem Fall können Sie mit der Funktion **IsArray** sicherstellen, ob es sich um ein Array handelt, bevor Sie etwa mit Funktionen wie **UBound** oder **LBound** auf die Variable zugreifen.

Die folgende Funktion erwartet eine **Variant**-Variable, der man sowohl einfache Variablen als auch Array-Variablen übergeben kann. Die Funktion prüft mit der **IsArray**-Funktion, ob **var** ein Array ist oder nicht. Im Falle eines Arrays durchläuft die Funktion alle Einträge des Arrays und gibt dessen Elemente im Direktbereich aus. Falls es sich um eine einfache Variable handelt, gibt sie einfach ihren Inhalt aus:

```
Public Function ArrayOderNicht(var As Variant)
    Dim i As Integer
    If IsArray(var) Then
        For i = LBound(var) To UBound(var)
            Debug.Print var(i)
        Next i
    Else
        Debug.Print var
    End If
End Function
```

Wir testen die Funktion zuerst mit der folgenden Prozedur, die eine einfache Variable übergibt:

```
Public Sub ArrayOderNichtMitVariable()
    Dim strTest As String
    strTest = "String-Variable"
    ArrayOderNicht strTest
End Sub
```

Dies gibt den Text **String-Variable** im Direktbereich aus. Danach rufen wir die Funktion mit der folgenden Prozedur auf. Diese stellt ein Array mit drei Elementen zusammen und übergibt es an die Funktion **ArrayOderNicht**:

```
Public Sub ArrayOderNichtMitArray()
    Dim strArray(2) As String
    strArray(0) = "Wert 1"
    strArray(1) = "Wert 2"
    strArray(2) = "Wert 3"
    ArrayOderNicht strArray
End Sub
```

In diesem Fall erscheinen im Direktbereich die drei Einträge des Arrays:

```
Wert 1
Wert 2
Wert 3
```

Die Funktion IsDate

Mit dieser Funktion prüfen Sie, ob es sich bei dem mit dem Parameter übergebenen Wert um einen gültigen Datumswert handelt. Dabei ist die Funktion sehr großzügig. Die folgenden Aufrufe zeigen, wie wir das meinen, denn sie liefern alle den Wert **True** zurück:

```
Debug.Print IsDate("3/20/2019")
Debug.Print IsDate("März 20")
Debug.Print IsDate("20-3")
Debug.Print IsDate("20/3/19")
Debug.Print IsDate("3/20")
Debug.Print IsDate("20/3")
```

Und nun raten Sie mal, welches Datum all diese Datumsangaben liefern, wenn Sie diese mit der Funktion **CDate** in das Standarddatumsformat konvertieren:

```
Debug.Print CDate("3/20/2019")
Debug.Print CDate("März 20")
Debug.Print CDate("20-3")
Debug.Print CDate("20/3/19")
Debug.Print CDate("3/20")
Debug.Print CDate("20/3")
```

Genau: Es kommt immer das Datum **20.3.2019** heraus. Die **IsDate**-Funktion ist sehr tolerant und frisst alles, was irgendwie als Datum interpretierbar ist. Das führt manchmal auch zu unerwarteten Ergebnissen.

Wenn Sie etwa folgendes Datum prüfen, würden Sie wohl kaum mit einem positiven Ergebnis rechnen, weil es 2019 keinen 29. Februar gibt:

```
? IsDate("29.02.19")
Wahr
```

Und auch VBA erfindet keinen neuen Tag zum Kalender hinzu, den es noch nicht gibt, sondern legt die Regeln einfach großzügig aus und tauscht Tag und Jahr aus:

```
? CDate("29.02.19")
19.02.2029
```

IsDate können Sie etwa nutzen, wenn Sie Datumseingaben des Benutzers in ein Formular validieren wollen.

Die Funktion IsEmpty

Mit der **IsEmpty**-Funktion können Sie prüfen, ob eine **Variant**-Variable initialisiert wurde, also ob ihr nach der Deklaration bereits ein Wert zugewiesen wurde.

Die folgende Prozedur prüft zunächst den Wert für eine **Variant**-Variable, die noch nicht initialisiert wurde und dann für die gleiche Variable, der wir einen Wert zugewiesen haben. Im letzten Schritt initialisieren wir die Variable wieder, indem wir ihr den Wert **Empty** zuweisen:

```
Public Sub Test_IsEmpty()
    Dim varTest As Variant
    Debug.Print IsEmpty(varTest), varTest
    varTest = "Test"
    Debug.Print IsEmpty(varTest), varTest
    varTest = Empty
    Debug.Print IsEmpty(varTest), varTest
End Sub
```

Die Funktion IsError

Diese Funktion ist uns im Rahmen der Recherchen zu diesem Artikel zum ersten Mal aufgefallen. Sie arbeitet zusammen mit einer weiteren Funktion namens **CVErr**. Die Zusammenarbeit sieht etwa so aus: In einer Funktion können Sie, wenn diese einen Fehler auslösen würde, einen benutzerdefinierten Fehlercode erstellen, den Sie als Ergebnis zurückliefern.

Dieser benutzerdefinierte Fehlercode ist eine Zahl, die Sie mit der Funktion **CVErr** kodieren. Wenn das geschehen ist, können Sie diesen mit **CVErr** kodierten Zahlenwert oder das korrekte Ergebnis in der aufrufenden Prozedur mit der **IsError**-Methode prüfen. Liefert **IsError** den Wert **True**, handelt es sich um einen mit **CVErr** kodierten Fehler, sonst um das gewünschte Ergebnis.

Wie das gelingt, sehen Sie im folgenden Beispiel. Die Funktion **Division** soll die beiden per Parameter übergebenen Zahlen dividieren. Wenn der zweite Parameter den Wert **0** enthält, würde dies einen Fehler wegen der Division durch 0 auslösen, aber wir prüfen vorher, ob der Wert **0** lautet. In diesem Fall geben wir von der Funktion **Division** den Wert **CVErr(123)** als Ergebnis zurück:

```
Public Function Division(dblZahl1 As Double, _
        dblZahl2 As Double)
    If dblZahl2 = 0 Then
        Division = CVErr(123)
    Else
        Division = dblZahl1 / dblZahl2
    End If
End Function
```

Dann rufen wir die Funktion **Division** mit den beiden Werten **1** und **0** auf, was den Fehler bewirken soll, und speichern das Ergebnis in der Variablen **varErgebnis**. Diese prüfen wir dann mit der **IsError**-Funktion. Liefert diese den Wert **True**, geben wir eine Fehlermeldung aus, sonst den berechneten Wert:

```
Public Sub DivisionTesten()
    Dim varErgebnis As Variant
    varErgebnis = Division(1, 0)
    If IsError(varErgebnis) Then
        MsgBox ("Es ist ein Fehler beim Dividieren 
                                        aufgetreten.")
    Else
        MsgBox "Das Ergebnis lautet " & varErgebnis
    End If
End Sub
```

Die Funktion **CVErr** stellt der als Parameter übergebenen Zahl übrigens scheinbar lediglich die Zeichenkette **Fehler**, gefolgt von einem Leerzeichen, voran:

```
? CVErr(123)
Fehler 123
```

Die Funktion **IsError** prüft dann offensichtlich, ob es sich bei dem Parameter um die Zeichenkette **Fehler**

und eine Zahl handelt. »Offensichtlich« deshalb, weil wohl doch mehr dazu gehört. Der folgende Aufruf von **IsError** bestätigt die Annahme nämlich nicht, sonst würden wir hier den Wert **True** erhalten:

```
? IsError("Fehler 1")
Falsch
```

CVErr(1) liefert aber den erwarteten Wert:

```
? CVErr(1)
Fehler 1
```

Und darauf die **IsError**-Funktion angewendet funktioniert auch wie erwartet:

```
? IsError(CVErr(1))
Wahr
```

Wo liegt also der Denkfehler? Ganz einfach: **CVErr** liefert keinen String etwa mit dem Inhalt **Fehler 1**, sondern ein Objekt des Type **Error**, der wiederum ein Untertyp des Datentyps **Variant** ist. Tatsache ist: Sie können nicht einfach eine Zeichenkette wie **Fehler 1** zurückgeben statt einer **Variant**-Variable, die mit dem Ergebnis der Funktion **CVErr** gefüllt ist – sonst erhalten Sie eine falsche Interpretation durch die **IsError**-Funktion.

Die Funktion IsMissing

Diese Funktion können Sie in **Sub**- oder **Function**-Routinen verwenden, um zu prüfen, ob der Benutzer für optionale **Variant**-Parameter einen Wert angegeben hat. Die folgende Funktion hat zwei optionale Parameter, einen mit dem Datentyp **Variant** und einen mit dem Datentyp **String**. Die **If...Then**-Bedingungen prüfen jeweils die Variable mit der **IsMissing**-Funktion und geben einen entsprechenden Wert im Direktfenster aus:

```
Public Function ParameterPruefen(Optional var As Variant, _
                                 Optional strTest As String)
    If IsMissing(var) Then
        Debug.Print "var ist leer."
    Else
        Debug.Print "var ist gefüllt."
    End If
    If IsMissing(strTest) Then
        Debug.Print "strTest ist leer."
    Else
        Debug.Print "strTest ist gefüllt."
    End If
End Function
```

Nun rufen wir diese Funktion mit den Anweisungen der folgenden Prozedur auf:

```
Public Sub Test_IsMissing()
    ParameterPruefen
    ParameterPruefen "test", "test"
End Sub
```

Das Ergebnis sieht so aus:

```
var ist leer.
strTest ist gefüllt.
var ist gefüllt.
strTest ist gefüllt.
```

Wenn der **Variant**-Parameter gefüllt wird, liefert **IsMissing** also **False**, wenn er leer ist, den Wert **True**. Bei dem **String**-Parameter liefert die Funktion **IsMissing** immer den Wert **True**. Der Grund ist, dass die **String**-Variable immer bereits mit einer leeren Zeichenkette initialisiert ist. Das können Sie auch mit der weiter oben vorgestellten Funktion **IsEmpty** ausprobieren. Der Einsatz der **IsMissing**-Funktion ist also nur sinnvoll, wenn Sie Parameter mit dem Datentyp **Variant** testen.

Die Funktion IsNull

IsNull ist wohl die bekannteste Funktion der **Information**-Klasse. Damit prüfen Sie unter anderem, ob ein Feld der Tabelle einer Datenbank leer ist – was ein großer Unterschied ist zu einem Feld, dass eine leere Zeichenkette enthält. Angenommen, in einer Tabelle namens **tblBeispiel** hätten wir ein Textfeld namens **Textfeld**. Hat dieses Textfeld einen Wert, liefert die **IsNull**-Funktion für dieses Textfeld, dass wir mit dem folgenden Konstrukt referenzieren, den Wert **Falsch**:

```
? IsNull(CurrentDb.OpenRecordset("tblBeispiel").
Fields("Textfeld"))
Falsch
```

Leeren Sie das Textfeld, erhalten wir **Wahr** für die **IsNull**-Funktion:

```
? IsNull(CurrentDb.OpenRecordset("tblBeispiel").
Fields("Textfeld"))
Wahr
```

Nun testen wir mit einem Formular, das die Felder dieser Tabelle enthält. Der Fokus liegt auf dem Steuerelement **Textfeld**, damit wir mit **Screen.ActiveControl** darauf zugreifen können. Wir geben zunächst den Inhalt im Direktbereich des VBA-Editors aus und untersuchen das Textfeld dann mit der **IsNull**-Funktion (siehe Bild 2):

```
? Screen.ActiveControl
Null

? IsNull(Screen.ActiveControl)
Wahr
```

Mit der **IsNull**-Funktion können Sie auch den Inhalt von Variablen des Datentyps **Variant** untersuchen. Das erledigen wir wie folgt, wobei wir erst die nicht initialisierte **Variant**-Variable **var** testen, dann die mit einer Zeichenkette und die mit **Null** gefüllte Variable:

```
Public Sub Test_IsNull()
    Dim var As Variant
    Debug.Print IsNull(var)
    var = "123"
    Debug.Print IsNull(var)
    var = Null
    Debug.Print IsNull(var)
End Sub
```

Das Ergebnis ist interessant:

```
Falsch
Falsch
Wahr
```

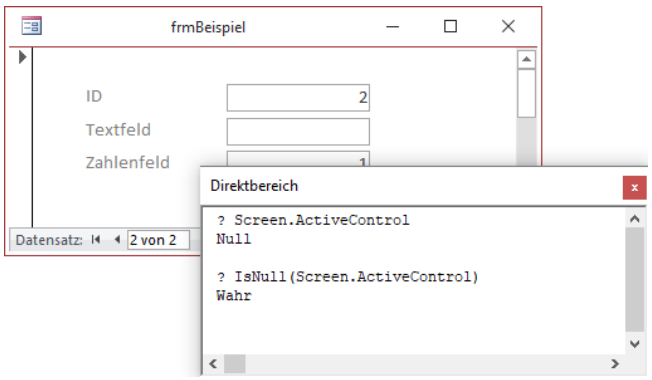


Bild 2: Testen der **IsNull**-Funktion

Bei der nicht initialisierten Variablen hätte wir auch erwartet, dass der Wert **Null** entspricht und **IsNull** das Ergebnis **Wahr** liefert. Allerdings ist die Variable schlicht nicht initialisiert und kann deshalb noch keinen Wert enthalten! Dementsprechend liefert dafür auch, wie weiter oben erläutert, **IsEmpty** das Ergebnis **True**. Gefüllt mit einem Text liefert **IsNull** dann den Wert **Falsch**, und wenn wir der Variablen **Null** zuweisen, liefert **IsNull** natürlich **Wahr**.

Oben haben Sie gesehen, dass wir die Variable **var** auf **Null** eingestellt haben, indem wir dieser einfach **Null** zugewiesen haben. Könnten wir dann nicht durch einen Vergleich mit **Null** prüfen, ob die Variable den Wert **Null** enthält – etwa wie folgt?

```
Public Sub Test_Null()
    Dim var As Variant
    Debug.Print var = Null
    var = Null
    Debug.Print var = Null
End Sub
```

Hier wird es interessant. Das Ergebnis beider Vergleiche ist nämlich nicht **Wahr** oder **Falsch**, sondern **Null**! Wie kann das sein? Das ist per Definition so: Ausdrücke, die den Wert **Null** enthalten, sind ebenfalls gleich Null.

Eine Ausnahme erhalten Sie etwa, wenn dieser Ausdruck mit dem &-Operator verknüpft ist – dann wird nur der Teil links oder rechts vom &-Operator gleich **Null** gesetzt, der Rest nicht:

Das FileSystemObject: Verzeichnisse

Die VBA-eigenen Befehle wie **Dir**, **MkDir**, **Kill**, **Copy** und so weiter erlauben grundlegende Dateioperationen. Wenn es etwas komfortabler sein darf, gibt es jedoch noch andere Möglichkeiten. Eine davon ist die Klasse **FileSystemObject**, die Sie in der Bibliothek **Microsoft Scripting Runtime** finden. Dieser Artikel zeigt, welche Möglichkeiten diese Bibliothek bietet.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1902_FileSystemObject.accdb**.

Late Binding und Early Binding

Wie bei vielen anderen Bibliotheken können Sie auch die Elemente der Bibliothek **Microsoft Scripting Runtime** per Late Binding oder per Early Binding nutzen. Die Version mit Late Binding verwendet keinen Verweis und bindet die Bibliothek erst beim Aufruf ein. Das könnte etwa wie folgt aussehen:

```
Public Sub FSOLateBinding()
    Dim objFSO As Object
    Set objFSO = _
        CreateObject("Scripting.FileSystemObject")
    With objFSO
        Debug.Print .FolderExists("c:")
    End With
End Sub
```

Hier deklarieren wir die Objektvariable mit dem Datentyp **Object**. Das hat den Nachteil, dass wir nicht per IntelliSense auf die Elemente dieser Klasse zugreifen können. In diesem Beispiel nutzen wir die **FolderExists**-Funktion, um zu prüfen, ob das Verzeichnis **c:** vorhanden ist.

Wenn Sie lieber mit IntelliSense arbeiten, was beim Programmieren wesentlich komfortabler ist, fügen Sie einen Verweis auf die Bibliothek **Microsoft Scripting Runtime** hinzu. Dazu öffnen Sie den VBA-Editor, etwa mit der Tastenkombination **Alt + F11**, und aktivieren dann mit dem Menübefehl **Extras|Verweise** die Anzeige des **Verweise**-Dialogs für das aktuelle VBA-Projekt. Hier wählen Sie den Eintrag **Microsoft Scripting Runtime** aus und aktivieren diesen (siehe Bild 1).

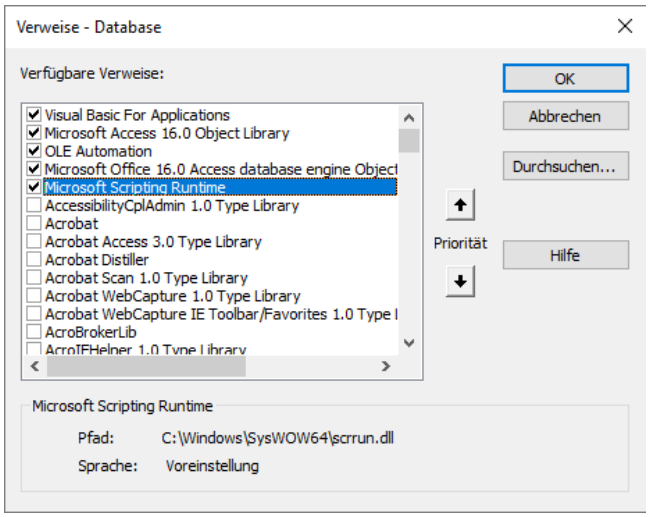


Bild 1: Verweis auf die Bibliothek **Microsoft Scripting Runtime**

Danach können Sie die Objektvariable für die Klasse **FileSystemObject** auch mit diesem Datentyp deklarieren:

```
Public Sub FSOEarlyBinding()
    Dim objFSO As FileSystemObject
    Set objFSO = New FileSystemObject
    With objFSO
        Debug.Print .FolderExists("c:")
    End With
End Sub
```

Ein weiterer Unterschied ist, dass Sie nun die Methoden der Klasse per IntelliSense anzeigen können (siehe Bild 2). Alternativ können Sie sich natürlich auch über den Objektkatalog einen Überblick über die Methoden und Eigenschaften des Objekts einer Bibliothek verschaffen. Den Objektkatalog blenden Sie mit der Tastenkombination **F2** ein. Um die Elemente der gewünschten Bibliothek zu betrachten,

wählen Sie diese oben im Kombinationsfeld aus. Dann können Sie unten die entsprechende Klasse anklicken und finden alle Element im rechten Listenfeld vor (siehe Bild 3).

Elemente der FileSystemObject-Klasse

Die Klasse, die wir uns in diesem Artikel ansehen, hat die folgenden Elemente:

- **BuildPath**: Stellt Verzeichnisnamen zusammen auf Basis eines Verzeichnisnamens und des Namens für das Unterverzeichnis. Das Backslash-Zeichen (\) wird dabei automatisch eingefügt.
- **CopyFile**: siehe **Das FileSystemObject: Dateien**
- **CopyFolder**: Kopiert das im ersten Parameter per Pfad angegebene Verzeichnis in das im zweiten Parameter angegebene Verzeichnis.
- **CreateFolder**: Erstellt das als Parameter angegebene Verzeichnis.
- **CreateTextFile**: siehe **Das FileSystemObject: TextStream**
- **DeleteFile**: siehe **Das FileSystemObject: Dateien**
- **DeleteFolder**: Löscht den Ordner mit dem angegebenen Pfad.
- **DriveExists**: Prüft, ob ein Laufwerk vorhanden ist.
- **Drives**: Liefert eine Auflistung aller Laufwerke.
- **FileExists**: siehe **Das FileSystemObject: Dateien**
- **FolderExists**: Prüft, ob das Verzeichnis mit dem als Parameter angegebenen Namen existiert.
- **GetAbsolutePathName**: Ermittelt den Pfad ohne abschließenden Backslash.
- **GetBasePathName**: siehe **Das FileSystemObject: Dateien**

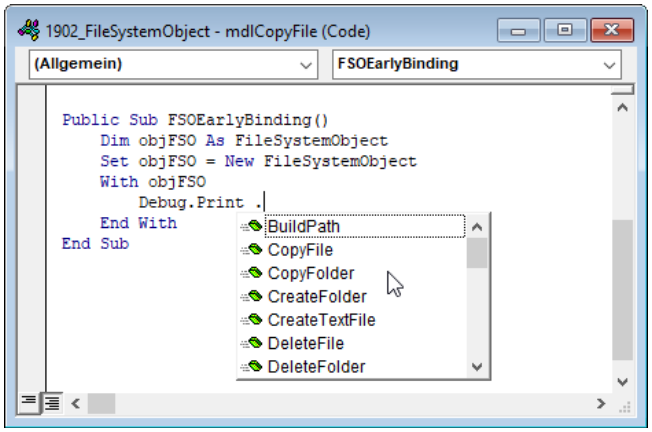


Bild 2: **FileSystemObject** mit IntelliSense nutzen

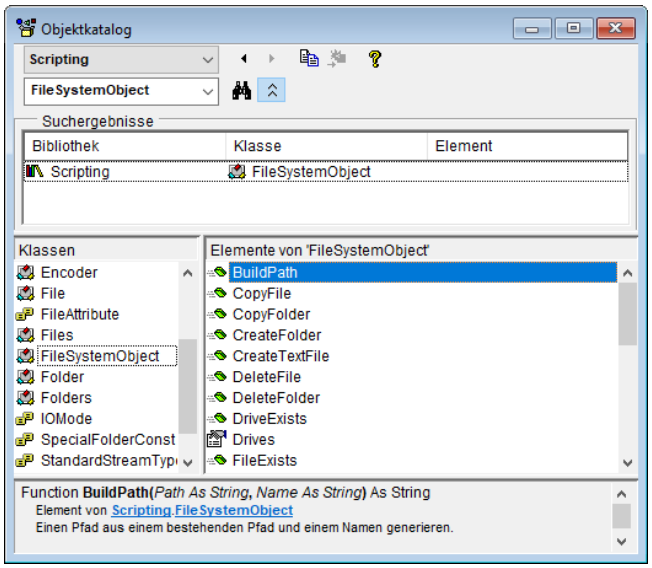


Bild 3: Per Objektkatalog in den Methoden einer Klasse stöbern

- **GetDrive**: Ermittelt das Drive-Objekt zu einem Laufwerksbuchstaben.
- **GetDriveName**: Ermittelt den Laufwerksbuchstaben eines Pfades.
- **GetExtensionName**: siehe **Das FileSystemObject: Dateien**
- **GetFile**: siehe **Das FileSystemObject: Dateien**
- **GetFileName**: siehe **Das FileSystemObject: Dateien**

- **GetFileVersion**: siehe **Das FileSystemObject: Dateien**
- **GetFolder**: Referenziert das als Parameter angegebene Verzeichnis.
- **GetParentFolderName**:
- **GetSpecialFolder**: Ermittelt spezielle Verzeichnisse wie etwa das **Windows**-Verzeichnis.
- **GetStandardStream**: siehe **Das FileSystemObject: TextStream**
- **GetTempName**: siehe **Das FileSystemObject: Dateien**
- **MoveFile**: siehe **Das FileSystemObject: Dateien**
- **MoveFolder**: Verschiebt das als ersten Parameter angegebene Verzeichnis und fügt es als das mit dem zweiten Parameter angegebene Verzeichnis wieder ein.
- **OpenTextFile**: siehe **Das FileSystemObject: TextStream**

Mit Laufwerken arbeiten

Wir beginnen mit der obersten Ebene im Dateisystem – den Laufwerken. Dazu nutzen wir gleich das einzige Element der Klasse **FileSystemObject**, das keine Funktion oder Methode ist, nämlich die Auflistung **Drives**. Die folgende Prozedur ermittelt die Anzahl der Laufwerke und gibt diese im Direktbereich aus. Danach durchlaufen wir die Laufwerke in einer **For Each**-Schleife, wobei wir das aktuelle Laufwerk jeweils der Variablen **objDrive** mit dem Datentyp **Drive** zuweisen. Dieses liefert einige Eigenschaften zum jeweiligen Laufwerk, von denen wir hier beispielhaft den Laufwerkbuchstaben, den Typ, die Bezeichnung des Filesystems und den verfügbaren Speicherplatz im Direktbereich ausgeben:

```
Public Sub Laufwerke()
    Dim objFSO As FileSystemObject
    Dim objDrive As Scripting.Drive
    Set objFSO = New FileSystemObject
    Debug.Print objFSO.Drives.Count
    For Each objDrive In objFSO.Drives
        With objDrive
            On Error Resume Next
            Debug.Print .DriveLetter,
            Debug.Print .DriveType,
            Debug.Print .FileSystem,
            If Err.Number = 71 Then
                Debug.Print "Nicht bereit.",
            End If
            Err.Clear
            Debug.Print .FreeSpace
            If Err.Number = 71 Then
                Debug.Print "Nicht bereit."
            End If
        End With
    Next objDrive
End Sub
```

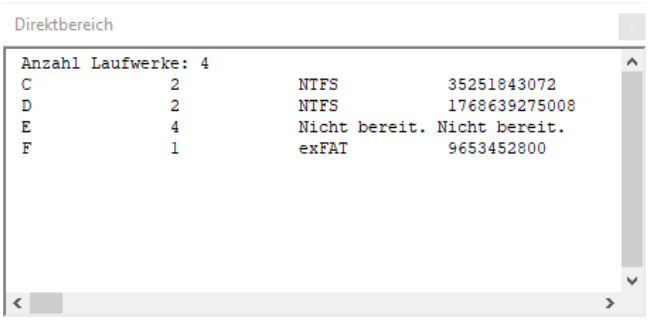


Bild 4: Ausgabe einiger Laufwerk-Eigenschaften

Im Falle der beiden Eigenschaften **FileSystem** und **FreeSpace** tritt ein Fehler auf, wenn das jeweilige Laufwerk nicht bereit ist. Das ist etwa der Fall, wenn wir ein DVD-Laufwerk untersuchen, das aktuell keine DVD enthält. Diesen Fehler fangen wir ab und geben stattdessen den Text **Nicht bereit.** aus. Wir könnten auch in einer **If...Then**-Bedingung prüfen, ob die Eigenschaft **IsReady** den Wert **True** zurückgibt. Dann würden wir uns die Fehlerbehandlung sparen:

```
With objDrive
    Debug.Print .DriveLetter,
    Debug.Print .DriveType,
```

```
    If .IsReady Then
        Debug.Print .FileSystem,
        Debug.Print .FreeSpace,
        Debug.Print .VolumeName,
    Else
        Debug.Print "Nicht bereit.",
        Debug.Print "Nicht bereit.",
        Debug.Print "Nicht bereit.",
    End If
    Debug.Print .ShareName
End With
```

Das Ergebnis sieht etwa wie in Bild 4 aus. Die ersten beiden Laufwerke sind Festplatten mit dem Format **NTSF**, das dritte ein DVD-Laufwerk und das vierte ein USB-Stick, der im Format **exFAT** formatiert ist. Die Zahlenwerte für die Eigenschaft **DriveType** bedeuten **CDRom** (**4**), **Fixed** (**2**), RamDisk (**5**), **Remove** (**3**), **Removable** (**1**) und **UnknownType** (**0**). Die **Drive**-Klasse bietet die folgenden Eigenschaften:

- **AvailableSpace**: Verfügbarer Speicherplatz
- **DriveLetter**: Großbuchstabe für das Laufwerk (zum Beispiel **C** oder **D**)
- **DriveType**: Typ des Laufwerks
- **FileSystem**: Bezeichnung des Dateisystems
- **FreeSpace**: Verfügbarer Speicherplatz
- **IsReady**: Gibt an, ob das Laufwerk bereit ist
- **Path**: Großbuchstabe für das Laufwerk plus Doppelpunkt, also etwa **C:** oder **D:**
- **RootFolder**: Liefert ein **Folder**-Objekt, welches das Basisverzeichnis repräsentiert, also etwa **C:**. Damit können Sie weitere Optionen ausführen wie etwa die Unterverzeichnisse ermitteln.
- **SerialNumber**: Serielle Nummer des Laufwerks.
- **ShareName**: Netzwerkfreigabename des Laufwerks
- **TotalSize**: Gesamter Speicherplatz
- **VolumeName**: Benutzerdefinierter Name des Laufwerks

Zur Vereinfachung: FileSystemObject als Property

Damit wir nicht in jeder Beispielprozedur erneut das **FileSystemObject** deklarieren und erzeugen müssen, legen wir in einem Standardmodul die folgende private Variable an:

```
Private m_FSO As FileSystemObject
```

Zusätzlich fügen wir dem Modul die folgende **Property Get**-Prozedur hinzu, die prüft, ob **m_FSO** bereits mit dem **FileSystemObject**-Objekt gefüllt ist. Falls nicht, wird es nun erstellt und in **m_FSO** geschrieben. Danach gibt die **Property**-Prozedur den Verweis zurück:

```
Public Property Get FSO() As FileSystemObject
    If m_FSO Is Nothing Then
        Set m_FSO = New FileSystemObject
    End If
    Set FSO = m_FSO
End Property
```

Sie können nun in den folgenden Prozeduren einfach mit **FSO** auf das **FileSystemObject** zugreifen.

Prüfen, ob ein Laufwerk vorhanden ist

Wenn Sie prüfen möchten, ob ein Laufwerk mit einem bestimmten Laufwerksbuchstaben vorhanden ist, können Sie dazu die **DriveExists**-Funktion verwenden und entweder nur den Laufwerksbuchstaben übergeben oder zusätzlich weitere Zeichen:

```
Debug.Print FSO.DriveExists("C")
Debug.Print FSO.DriveExists("C:")
Debug.Print FSO.DriveExists("C:\")
```

Laufwerk zu bestimmtem Verzeichnis

Wenn Sie etwa das Laufwerk ermitteln wollen, auf dem sich der aktuelle Datenbankordner befindet,

Das FileSystemObject: Dateien

Die VBA-eigenen Befehle wie **Dir**, **MkDir**, **Kill**, **Copy** und so weiter erlauben grundlegende Dateioperationen. Wenn es etwas komfortabler sein darf, gibt es jedoch noch andere Möglichkeiten. Eine davon ist die Klasse **FileSystemObject**, die Sie in der Bibliothek **Microsoft Scripting Runtime** finden. Im Artikel **Das FileSystemObject: Verzeichnisse** haben wir die Grundlagen zu dieser Bibliothek und zum Umgang mit Laufwerken und Verzeichnissen erläutert. Dieser Artikel zeigt, welche Möglichkeiten diese Bibliothek im Hinblick auf den Umgang mit Dateien bietet.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1902_FileSystemObject_Files.accdb**.

Der oben genannte Artikel hat die Grundlagen zum **FileSystemObject** und zu den Auflistungen **Drives** und **Folders** sowie den Klassen **Drive** und **Folder** erläutert. Nun gehen wir einen Schritt weiter und schauen uns an, wie wir mit den in den Verzeichnissen enthaltenen Dateien arbeiten können. Diese referenzieren wir über die **Files**-Auflistung beziehungsweise im Falle einer einzelnen Datei über die **File**-Klasse.

Schneller Zugriff auf ein FileSystemObject-Objekt

Wie schon im Artikel **Das FileSystemObject: Verzeichnisse** beschrieben, verwenden wir die folgende private Variable und die öffentliche **Property Get**-Prozedur, um schnell Zugriff auf eine Instanz der **FileSystemObject**-Klasse zu erhalten:

```
Private m_FSO As FileSystemObject
Public Property Get FSO() As FileSystemObject
    If m_FSO Is Nothing Then
        Set m_FSO = New FileSystemObject
    End If
    Set FSO = m_FSO
End Property
```

Sie können dann einfach über den Ausdruck **FSO** auf das **FileSystemObject** zugreifen – also etwa so:

```
Debug.Print FSO.Drives.Count
```

Dateien auflisten

Wenn wir Dateien auflisten wollen, benötigen wir zunächst ein **Folder**-Objekte, also ein Verzeichnis, in dem die zu untersuchenden Dateien enthalten sind. Wir beschäftigen uns mit dem Verzeichnis der aktuellen Datenbank, das wir mit dem VBA-Befehl **CurrentProject.Path** auslesen. Das dazugehörige **Folder**-Objekte erhalten wir mit der **GetFolder**-Methode:

```
Public Sub DateienAuflisten()
    Dim objFile As File
    Dim objFolder As Folder
    Set objFolder = FSO.GetFolder(CurrentProject.Path)
    For Each objFile In objFolder.Files
        Debug.Print objFile.Name
    Next objFile
End Sub
```

In der Prozedur ermitteln wir das **Folder**-Objekt und durchlaufen alle in der **Files**-Auflistung dieses Verzeichnisses enthaltenen **File**-Objekte. Wir geben zu jedem **File**-Objekt den Namen aus der Eigenschaft **Name** im Direktbereich aus. Auf die gleiche Weise können wir die folgenden Informationen des **File**-Objekts ausgeben lassen:

- **Attributes**: Gibt einen Zahlenwert für verschiedene Attributwerte zurück.
- **DateCreated**: Gibt das Erstellungsdatum zurück.
- **DateLastAccessed**: Gibt das Datum des letzten Zugriffs zurück.

- **DateLastModified**: Gibt das Datum der letzten Änderung zurück.
- **Name**: Gibt den Namen der Datei aus.
- **Path**: Gibt den Pfad der Datei zurück.
- **ShortName**: Gibt den Dateinamen im 8.3-Format zurück.
- **ShortPath**: Gibt den Pfad zur Datei im 8.3-Format zurück.
- **Size**: Gibt die Größe der Datei zurück.
- **Type**: Gibt den Dateityp zurück.

Wenn Sie all diese Informationen für die Dateien aus der obigen Prozedur ausgeben wollen, erweitern Sie den Inhalt der **For Each**-Schleife wie folgt:

```
With objFile
    Debug.Print .Name, .Attributes, .DateCreated, _
        .DateLastAccessed, .DateLastModified, .Path, _
                        .ShortName, .ShortPath, .Type
End With
```

Für die Eigenschaft **Type** erhalten wir Texte wie etwa **Microsoft Access Database**, **LACCDB-Datei** oder **Snagit Editor Image**. Teilweise werden hier Typen ausgeben, die mit der standardmäßig zu öffnenden Anwendung zusammenhängen, teilweise nur die Dateiendung.

Eigenschaften mit Objekten

Es gibt zwei Eigenschaften, die Verweise auf Objekte zurückliefern:

- **Drive**: Gibt das **Drive**-Objekt zu der Datei zurück.
- **ParentFolder**: Gibt das **Folder**-Objekt zurück, in dem sich die Datei befindet.

Methoden und Funktionen des File-Objekts

Das **File**-Objekt enthält neben den bereits erwähnten Eigenschften noch die folgenden Methoden und Funktionen:

- **Copy**: Kopiert die Datei an den angegebenen Zielpfad.
- **Delete**: Löscht die referenzierte Datei.
- **Move**: Verschiebt die Datei an den angegebenen Zielpfad.
- **OpenAsTextStream**: Öffnet die Datei als Text-Stream.

Elemente der FileSystemObject-Klasse für den Umgang mit File-Objekten

Und auch die **FileSystemObject**-Klasse stellt noch Eigenschaften, Methoden und Funktionen bereit, die Sie in Zusammenhang mit **File**-Objekten verwenden können:

- **CopyFile**: Kopiert eine Datei. Erwartet Quell- und Zielpfad als Parameter.
- **CreateTextFile**: Erstellt ein **TextStream**-Objekt, das als Textdatei gespeichert wird.
- **DeleteFile**: Löscht die als Pfad angegebene Datei.
- **FileExists**: Untersucht, ob eine Datei existiert.
- **GetBaseName**: Liefert den Dateinamen ohne Dateiendung.
- **GetExtensionName**: Liefert die Dateiendung.
- **GetFile**: Liefert das **File**-Objekt zur Datei mit dem als Parameter angegebenen Pfad.
- **GetFileName**: Liefert den kompletten Dateinamen.

- **GetFileVersion**: Liefert die Dateiversion von **.exe**- und **.dll**-Dateien.
- **GetStandardStream**: siehe **Das FileSystemObject: Textdateien**
- **GetTempName**: Ermittelt einen temporären Dateinamen.
- **MoveFile**: Verschiebt die Datei an den als Parameter angegebenen Pfad.
- **OpenTextFile**: siehe **Das FileSystemObject: Textdateien**

Elemente der Folder-Klasse für den Umgang mit File-Objekten

Und schließlich finden wir auch in der **Folder**-Klasse noch Methoden und Auflistungen für den Umgang mit **File**-Objekten. Die **Files**-Auflistung haben wir ja bereits weiter oben beschrieben:

- **CreateTextFile**: Erstellt ein **TextStream**-Objekt, das als Textdatei gespeichert wird.
- **Files**: Enthält eine Auflistung aller **File**-Objekte des **Folder**-Objekts.

Dateien als File-Objekte referenzieren

Eine erste Möglichkeit, wie Sie Dateien als **File**-Objekte referenzieren können, haben Sie ja bereits weiter oben kennengelernt – nämlich über die Auflistung **Files** eines **Folder**-Objekts.

Sie können aber auch auf ein **File**-Objekt zugreifen, wenn Sie nur seinen Pfad kennen. Dazu verwenden Sie die **GetFile**-Methode der **FileSystemObjekt**-Klasse:

```
Public Sub DateiReferenzieren()
    Dim objFile As File
    Dim strPfad As String
    strPfad = FSO.BuildPath(CurrentProject.Path, _
                                        "pic001.png")
    Set objFile = FSO.GetFile(strPfad)
    Debug.Print objFile.Path
End Sub
```

Damit können Sie dann etwa alle Eigenschaften der mit **objFile** referenzieren Datei ermitteln oder auch die Methoden und Funktionen der **File**-Klasse nutzen. Diese lernen Sie in den folgenden Abschnitten kennen. Hier haben wir die **BuildPath**-Methode genutzt, um einen Pfad aus dem Verzeichnis der aktuellen Datenbank und der darin enthaltenen Datei **pic001.png** zu erstellen. Damit brauchen Sie sich zum Beispiel nicht mehr darum zu kümmern, ob **CurrentProject.Path** nun mit oder ohne abschließendes Backslash-Zeichen kommt. Danach ermitteln wir mit **GetFile** das **File**-Objekt zu der in **strPfad** angegebenen Datei und geben ihren Pfad im Direktbereich des VBA-Editors aus.

Prüfen, ob eine Datei existiert

Unter VBA verwenden Sie die **Dir**-Funktion, um zu prüfen, ob eine Datei oder ein Verzeichnis existiert. Die **FileSystemObject**-Klasse bietet dazu spezifischere Methoden wie die im oben genannten Artikel beschriebene Funktion **FolderExists** und die nun vorgestellte Funktion **FileExists**. Diese erwartet die Angabe des Pfades zu der zu untersuchenden Datei als Parameter:

```
Dim strPfad As String
strPfad = FSO.BuildPath(CurrentProject.Path, "pic001.png")
Debug.Print FSO.FileExists(strPfad)
```

Informationen zur Datei

Über das **FileSystemObject**-Objekt lassen sich noch weitere Informationen zu einer Datei ermitteln, wenn Sie den Pfad zu dieser Datei kennen. Die folgenden Anweisungen liefern einige Informationen:

```
Dim strPfad As String
strPfad = FSO.BuildPath(CurrentProject.Path, "pic001.png")
Debug.Print "BaseName: " & FSO.GetBaseName(strPfad)
Debug.Print "ExtensionName: " _
                    & FSO.GetExtensionName(strPfad)
Debug.Print "FileName: " & FSO.GetFileName(strPfad)
Debug.Print "FileVersion: " & FSO.GetFileVersion(strPfad)
```

Das FileSystemObject: Textstream

Die VBA-eigenen Befehle wie **Open**, **Write**, **Input**, **Close** und so weiter erlauben das Erstellen, Schreiben und Lesen von Textdateien. Wenn es etwas komfortabler sein darf, gibt es jedoch noch andere Möglichkeiten. Eine davon ist die Klasse **FileSystemObject**, die Sie in der Bibliothek **Microsoft Scripting Runtime** finden. In den Artikeln **Das FileSystemObject: Verzeichnisse** und **Das FileSystemObject: Dateien** haben wir die Grundlagen zu dieser Bibliothek und zum Umgang mit Laufwerken, Verzeichnissen und Dateien erläutert. Dieser Artikel zeigt, welche Möglichkeiten diese Bibliothek im Hinblick auf den Umgang mit Textdateien bietet.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1902_FileSystemObject_Textstream.accdb**.

Die TextStream-Klasse

Beim Aufruf der Methode **CreateTextFile** der **FileSystemObject**-Klasse und der **Folder**-Klasse werden Objekte des Typs **TextStream** zurückgeliefert.

Diese weisen die folgenden Eigenschaften auf:

- **AtEndOfLine**: Gibt an, ob sich der Zeiger am Ende der aktuellen Zeile befindet.
- **AtEndOfStream**: Gibt an, ob sich der Zeiger am Ende des Streams befindet.
- **Close**: Schließt einen offenen Textstream.
- **Column**: Gibt die aktuelle Spalte des Zeigers aus.
- **Line**: Gibt die Nummer der aktuellen Zeile des Zeigers aus.
- **Read**: Liest die mit dem Parameter angegebene Menge an Zeichen.
- **ReadAll**: Liest den kompletten Inhalt des **TextStream**-Objekts.
- **ReadLine**: Liest die jeweils folgende Zeile des **TextStream**-Objekts.
- **Skip**: Verschiebt den Zeiger um die angegebene Anzahl Zeichen in Richtung Ende des Streams.
- **SkipLine**: Verschiebt den Zeiger auf den Anfang der nächsten Zeile.
- **Write**: Fügt den Inhalt des Parameters hinter dem letzten vorhandenen Zeichen in einem **TextStream**-Objekt ein.
- **WriteBlankLines**: Fügt hinter dem letzten Zeichen des **TextStream**-Objekts leere Zeilen entsprechend der im Parameter angegebenen Anzahl ein.
- **WriteLine**: Fügt den als Parameter angegebenen Text hinter dem letzten Zeichen des **TextStream**-Objekts ein und hängt ein Zeilenumbruchszeichen hinten an.

Erstellen einer Textdatei und Füllen mit Text aus einer Variablen

Wenn Sie einfach nur den Inhalt einer **String**-Variablen in eine neue Textdatei schreiben wollen, gehen Sie wie folgt vor:

```
Dim strDatei As String
Dim strPfad As String
Dim strText As String
Dim objTextstream As TextStream
strDatei = Replace(FSO.GetTempName, ".tmp", ".txt")
strPfad = FSO.BuildPath(CurrentProject.Path, strDatei)
strText = "Dies ist ein Beispieltext." & vbCrLf
strText = strText & "Und noch eine Zeile." & vbCrLf
```

```
strText = strText & "Und die letzte Zeile."
Set objTextstream = FSO.
CreateTextFile(strPfad)
With objTextstream
    .Write strText
End With
objTextstream.Close
```

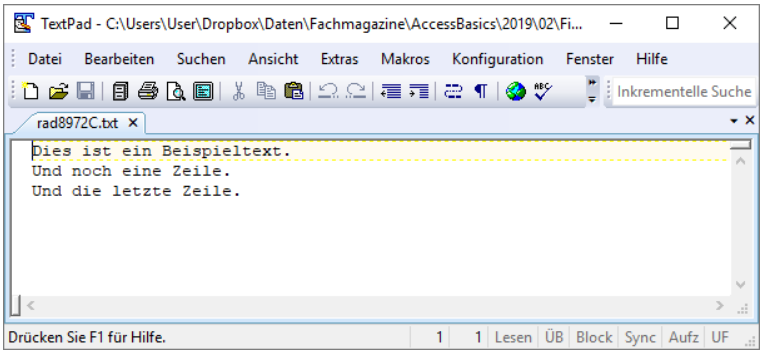


Bild 1: Eine erste Textdatei mit Inhalt

Hier ermitteln wir einen temporären Dateinamen, bei dem wir die Dateiendung **.tmp** durch **.txt** ersetzen. Dann fügen wir diese über die Funktion **BuildPath** mit dem Verzeichnis der aktuellen Datenbank zu einem Pfad zusammen. Außerdem fügen wir in der Variablen **strText** einen mehrzeiligen Text zusammen, den wir dann zur Datei hinzufügen wollen.

Die Textdatei erstellen wir mit der **CreateTextFile**-Methode des **FileSystemObject**-Objekts, der wir als Parameter den Pfad aus **strPfad** übergeben.

Dann nutzen wir die **Write**-Methode des mit der Variablen **objTextstream** erstellten **TextStream**-Objekts, um den Inhalt der Variablen **strText** einzufügen.

Schließlich schließen wir die Datei mit der **Close**-Methode. Das Ergebnis ist eine Textdatei mit dem Text aus **strText** (siehe Bild 1).

TextStream-Objekt für Folder erzeugen

Nicht nur das **FileSystemObject**-Objekt, sondern auch das **Folder**-Objekt bietet die Methode **CreateTextFile** an.

Dieser brauchen Sie dann nur noch den Dateinamen zu übergeben, da das Zielverzeichnis ja schon im **Folder**-Objekt steckt:

```
Dim strDatei As String
Dim objFolder As Folder
Dim objTextstream As TextStream
strDatei = Replace(FSO.GetTempName, ".tmp", ".txt")
Set objFolder = FSO.GetFolder(CurrentProject.Path)
Set objTextstream = objFolder.CreateTextFile(strDatei)
objTextstream.Close
```

Text in TextStream-Objekt schreiben

Was geschieht, wenn wir die drei Zeilen in einzelnen **Write**-Anweisungen zum **TextStream**-Objekt hinzufügen?

```
With objTextstream
    .Write "Dies ist ein Beispieltext."
    .Write "Und noch eine Zeile."
    .Write "Und die letzte Zeile."
End With
```

Dann wird der Text einfach ohne Zeilenumbruch aneinander gehängt:

```
Dies ist ein Beispieltext.Und noch eine Zeile.Und die
letzte Zeile.
```

Wenn der Text jeweils in einer neuen Zeile geschrieben werden soll, verwenden Sie die **WriteLine**-Methode:

```
With objTextstream
    .WriteLine "Dies ist ein Beispieltext."
    .WriteLine "Und noch eine Zeile."
    .WriteLine "Und die letzte Zeile."
End With
```

Dies hängt an den Text jeweils noch einen Zeilenumbruch an. Genau genommen könnten wir uns den bei der letzten Zeile sparen:

```
With objTextstream
    .WriteLine "Dies ist ein Beispieltext."
```

Auflistungen mit dem Dictionary-Objekt

Die Bibliothek **Microsoft Scripting Runtime** liefert neben den Objekten für den Umgang mit Laufwerken, Verzeichnissen, Dateien und Textstreams noch ein weiteres Objekt an, nämlich das **Dictionary**-Objekt. Dabei handelt es sich um eine Alternative zur **Collection**-Auflistung der VBA-Bibliothek. Das **Dictionary**-Objekt liefert allerdings ein paar zusätzliche Features. Dieser Artikel zeigt, wie Sie mit dem **Dictionary**-Objekt programmieren und wozu Sie es nutzen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1902_Dictionary.accdb**.

Andere Auflistungsarten

Es gibt neben dem **Dictionary**-Objekt andere Möglichkeiten, um Auflistungen von Elementen oder Informationen unterzubringen. Da wäre zum Beispiel das Array, über das wir beispielsweise im Artikel **Programmieren mit Arrays** berichtet haben. Und die ebenfalls in VBA eingebaute Collectioon haben wir im Artikel **Programmieren mit Collections** ausführlich beschrieben.

Verweis auf die Microsoft Scripting Runtime

Wenn Sie das **Dictionary**-Objekt der Bibliothek **Microsoft Scripting Runtime** nutzen wollen, können Sie dieses per Late Binding oder Early Binding tun. Wenn Sie die Early Binding-Variante verwenden wollen, gehen Sie wie folgt vor: Fügen wir dem VBA-Projekt der Datenbank einen Verweis auf die entsprechende Bibliothek hinzu, und zwar im **Verweise**-Dialog (VBA-Editor, Menüpunkt **Extras|Verweise**) – siehe Bild 1. Dann können Sie die Variable für das **Dictionary**-Objekt mit dem entsprechenden Typ deklarieren und auch mit dem Schlüsselwort New eine neue Instanz dieser Klasse erstellen:

```
Dim objDictionary As Dictionary
Set objDictionary = New Dictionary
```

Wenn Sie aus irgendwelchen Gründen keinen Verweis setzen wollen, können Sie auch mit Late Binding arbeiten. In diesem Fall verwenden Sie die folgenden Anweisungen statt der oben verwendeten:

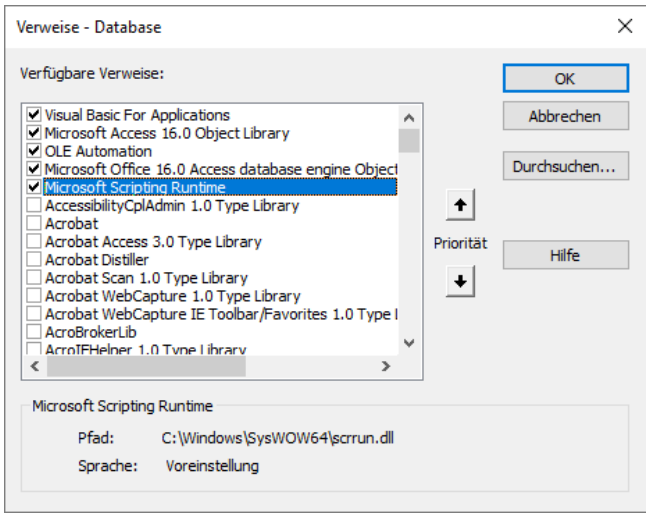


Bild 1: Verweis auf die Bibliothek **Microsoft Scripting Runtime**

```
Dim objDictionary As Object
Set objDictionary = CreateObject("Scripting.Dictionary")
```

Das Dictionary-Objekt

Das **Dictionary**-Objekt enthält die folgenden Elemente:

- **Add**: Fügt ein neues Element hinzu. Erwartet den Schlüssel und den Wert für das neue Element als Parameter.
- **CompareMode**: Modus für das Vergleichen der Schlüsselwerte. Es gibt die Werte **vbTextCompare** (unterscheidet nicht zwischen Groß- und Kleinschreibung) und **vbBinaryCompare** (unterscheidet zwischen Groß- und Kleinschreibung).
- **Count**: Liefert die Anzahl der Einträge im **Dictionary**-Objekt.

- **Exists**: Prüft, ob das Element mit dem angegebenen Schlüssel existiert.
- **Item**: Erlaubt den Zugriff über den Schlüssel des jeweiligen Elements.
- **Items**: Auflistung aller Werte, die über den Index angesteuert werden kann.
- **Key**: Dient zum Zuweisen eines neuen Schlüssels für ein vorhandenes Element.
- **Keys**: Auflistung aller Schlüssel, die über den Index angesteuert werden kann.
- **Remove**: Entfernt den Eintrag mit dem angegebenen Schlüssel aus dem **Dictionary**-Objekt.
- **RemoveAll**: Entfernt alle Einträge aus dem **Dictionary**-Objekt.

Erstellen und Füllen eines Dictionary-Objekts

Zum Hinzufügen eines Eintrags zu einem **Dictionary**-Objekts verwenden Sie standardmäßig die **Add**-Methode. Diese gibt über die zur Verfügung stehenden Parameter, die Sie nach Eingabe der **Add**-Methode etwa per IntelliSense einsehen können, die Struktur eines Eintrags vor (siehe Bild 2).

Dieser enthält also ein **Key**- und ein **Item**-Element. Für **Key** geben Sie einen Namen oder einen Wert an, über den Sie den Eintrag später referenzieren wollen.

Wenn Sie also etwa eine Reihe von Informationen über einen Benutzer in einem **Dictionary**-Objekt speichern wollen, beginnen Sie mit dem **Key** mit der Bezeichnung **Vorname** und weisen dem Element mit diesem **Key** den entsprechenden Wert zu, beispielsweise **André**:

```
Dim objDictionary As Dictionary
Set objDictionary = New Dictionary
With objDictionary
    .Add "Vorname", "André"
End With
```

Öffentliches Dictionary-Objekt deklarieren

Damit wir mit dem Dictionary, dessen Daten gegebenenfalls für die komplette Sitzung mit der Access-Anwendung zur Verfügung stehen sollen, experimentieren können, deklarieren wir es außerhalb einer Prozedur in einem Standardmodul als öffentliche Variable:

```
Public objDictionary As Dictionary
```

Dann erstellen wir eine Instanz der **Dictionary**-Klasse und weisen diese unserer Objektvariablen zu:

```
Set objDictionary = New Dictionary
```

Dies können Sie in einer Prozedur erledigen, aber auch etwa über den Direktbereich des VBA-Editors.

Eintrag hinzufügen mit der Add-Methode

Danach fügen wir dort den ersten Eintrag hinzu:

```
objDictionary.Add "Vorname", "André"
```

Damit können wir nun schon einmal die **Count**-Eigenschaft nutzen, um die Anzahl der Elemente zu ermitteln – ebenfalls im Direktbereich:

```
Debug.Print objDictionary.Count
 1
```

Wir fügen ein weiteres Element hinzu:

```
objDictionary.Add "Nachname", "Minhorst"
```

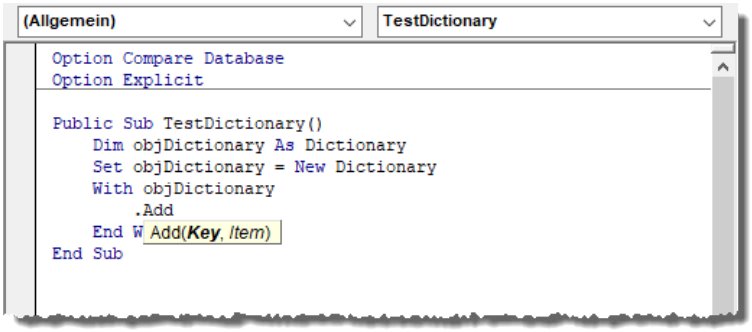


Bild 2: Nutzen von IntelliSense mit dem **Dictionary**-Objekt

Danach liefert **Count** logischerweise diesen Wert:

```
Debug.Print objDictionary.Count
 2
```

Wenn wir allerdings versuchen, ein Element zum **Dictionary**-Objekt hinzuzufügen, dessen Schlüssel bereits vorhanden ist, löst dies den Fehler mit der Nummer **457** aus (**Dieser Schlüssel ist bereits einem Wert dieser Auflistung zugeordnet.**). Auch dies können wir leicht umgehen, indem wir zuvor mit der **Exists**-Funktion prüfen, ob das entsprechende Element bereits vorhanden ist:

```
Public Sub ElementNurHinzufuegenWennNochNichtVorhanden()
    If Not objDictionary.Exists("Vorname") Then
        objDictionary.Add "Vorname", "Klaus"
    Else
        MsgBox "Element 'Vorname' bereits vorhanden."
    End If
End Sub
```

Auf Elemente per Key zugreifen

Wie greifen wir nun auf die einzelnen Elemente des **Dictionary**-Elements zu? Dazu verwenden wir beispielsweise die **Item**-Eigenschaft:

```
? objDictionary.Item("Vorname")
André
? objDictionary.Item("Nachname")
Minhorst
```

Anderen Wert zuweisen

Wenn Sie den Wert eines **Dictionary**-Eintrags mit einem bestimmten Schlüssel ändern wollen, referenzieren Sie das Element über den Schlüssel als Parameter der Eigenschaft **Item** und weisen diesem dann den neuen Wert zu:

```
objDictionary.Item("Vorname") = "Andreas"
```

Danach ist der Wert geändert:

```
? objDictionary.Item("Vorname")
Andreas
```

Was aber geschieht, wenn wir einen Wert zu einem Element zuweisen wollen, das noch gar nicht existiert – wie in folgendem Beispiel?

```
objDictionary.Item("Strasse") = "Borkhofer Str. 17"
```

Dann wird automatisch ein neuer Eintrag mit dem angegebenen Schlüssel angelegt und diesem der nachfolgende Wert zugewiesen. Das können Sie leicht an der Anzahl der Einträge prüfen:

```
Debug.Print objDictionary.Count
 3
```

Elemente des Dictionary-Objekts durchlaufen

Können wir die Elemente des **Dictionary**-Elements auch per Schleife durchlaufen? Wir probieren es zunächst mit einer einfachen **For Next**-Schleife über den Index-Wert, wobei wir **1** als untersten Wert annehmen und den obersten Wert mit der **Count**-Eigenschaft von **objDictionary** ermitteln:

```
Public Sub DictionaryDurchlaufen()
    Dim i As Integer
    For i = 1 To objDictionary.Count
        With objDictionary
            Debug.Print .Item(i)
        End With
    Next i
End Sub
```

Dies gibt allerdings nur leere Zeilen im Direktbereich aus. Wie ist das möglich? Das ist eine Eigenart der **Dictionary**-Klasse. Diese legt beim Versuch des Zugriffs auf ein Element, das nicht vorhanden ist, einfach ein neues Element mit dem angegebenen Schlüssel an und weist diesem keinen Wert zu.

Wenn wir vor dem Aufruf der obigen Prozedur also drei Elemente zum **Dictionary**-Objekt hinzugefügt hatten, befinden sich nun sechs Elemente darin. Das belegt auch der Test mit der **Count**-Eigenschaft:

```
? objDictionary.Count
 6
```

Formulare mehrfach öffnen mit Dictionary

In einem früheren Artikel namens **Mehrere Instanzen eines Formulars** haben wir gezeigt, wie Sie ein Formular einer Datenbank mehr als einmal öffnen können. Dabei haben wir ein **Collection**-Objekt verwendet, um die Objektvariablen mit den verschiedenen Instanzen des Formulars zu speichern. Im vorliegenden Artikel schauen wir uns an, wie wir diese Objektvariablen in einem **Dictionary**-Objekt statt in einem **Collection**-Objekt verwalten können und welche Vorteile dies bringt.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1902_DictionaryFormularMehrfach.accdb**.

Das Dictionary-Objekt

Das **Dictionary**-Objekt erläutern wir im Detail im Artikel **Auflistungen mit dem Dictionary-Objekt** – daher hier nur die notwendigsten Informationen für seine Nutzung. Um das **Dictionary**-Objekt zu verwenden, fügen Sie dem VBA-Projekt einen Verweis auf die Bibliothek **Microsoft Scripting Runtime** hinzu (siehe Bild 1).

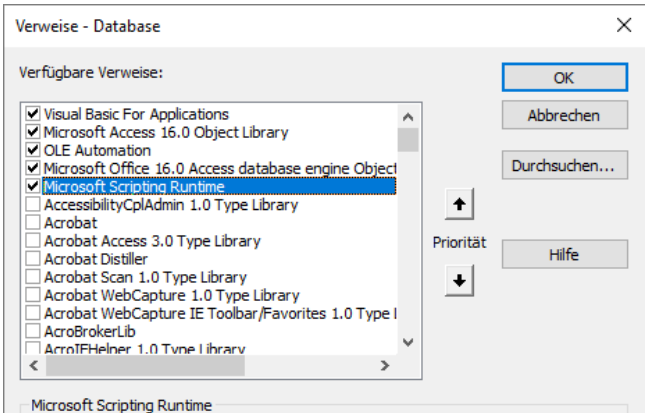


Bild 1: Verweis auf die Bibliothek **Microsoft Scripting Runtime**

Die Verweise auf die vom Hauptformular **frmKundenuebersicht** aus geöffneten Instanzen des Formulars **frmKunde** (siehe Bild 2) speichern wir in einem **Dictionary**-Objekt, dessen Deklaration wir im Klassenmodul des Formulars **frmKundenuebersicht** unterbringen. Dazu legen wir dort im Kopf des Moduls die folgende Anweisung an:

```
Dim dicForms As Dictionary
```

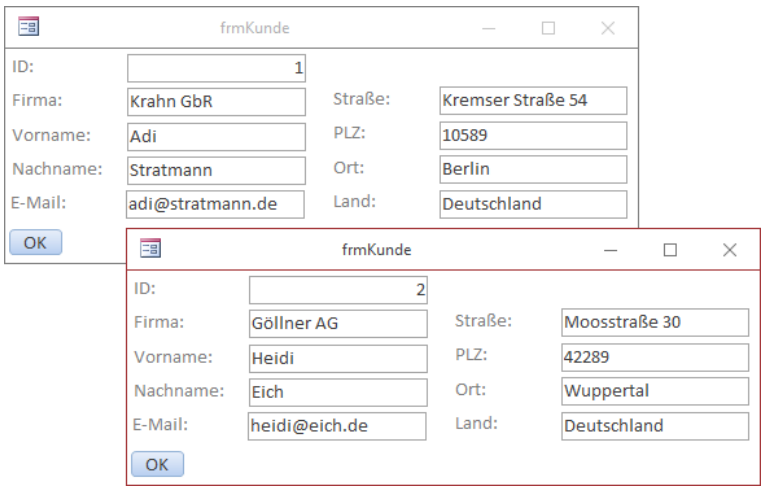


Bild 2: Mehrfaches Öffnen des gleichen Formulars mit verschiedenen Datensätzen

Die Position der neu zu erstellenden Formulare speichern wir in zwei Variablen namens **intPosX** und **intPosY**, die wir ebenfalls dort deklarieren:

```
Dim intPosX As Integer
Dim intPosY As Integer
```

Das Hauptformular **frmKundenuebersicht** zeigt die Einträge für die Kunden in einem Listenfeld an (siehe Bild 3).

Per Doppelklick auf einen der Listeneinträge kann der Benutzer das Formular zum jeweiligen Kunden öffnen. Dies löst die Prozedur

aus Listing 1 aus. Damit diese korrekt funktioniert, muss allerdings beim Laden des Formulars die Variable für das **Dictionary**-Objekt initialisiert worden sein:

```
Private Sub Form_Load()
    Set dicForms = New Dictionary
End Sub
```

Danach führt der Doppelklick auf einen der Einträge des Listenfeldes gleich zum ersten Zugriff auf das **Dictionary**-Objekt und hier

Bild 3: Formular zum Öffnen der Detailformulare

```
Private Sub lstKunden_DblClick(Cancel As Integer)
    Dim frm As Form_frmKunde
    Dim bolVorhanden As Boolean
    Dim i As Integer
    Static lngOffsetY As Long
    Dim lngXFaktor As Long
    Dim lngYFaktor As Long
    If Not dicForms.Exists(Me!lstKunden) Then
        Set frm = New Form_frmKunde
        With frm
            .Visible = True
            .Caption = Me!lstKunden & " - " & Me!lstKunden.Column(1)
            .Filter = "KundeID = " & Me!lstKunden
            .FilterOn = True
        End With
        dicForms.Add Me!lstKunden, frm
        GetScreenInfo lngXFaktor, lngYFaktor
        intPosX = intPosX + 1
        intPosY = intPosY + 1
        On Error Resume Next
        DoCmd.MoveSize intX * (intPosX - 1) * lngXFaktor, intY * (intPosY - 1) * lngYFaktor + lngOffsetY * lngYFaktor
        If Err.Number = 2498 Then
            intPosX = 1
            intPosY = 1
            lngOffsetY = intY + 30
            DoCmd.MoveSize intX * (intPosX - 1) * lngXFaktor, intY * (intPosY - 1) * lngYFaktor + lngOffsetY * lngYFaktor
        End If
    Else
        dicForms.Item(Me!lstKunden).SetFocus
    End If
End Sub
```

Listing 1: Öffnen eines neuen Formulars

Access-Daten nach Excel verknüpfen

Wir haben uns bereits in verschiedenen Artikeln angesehen, wie Sie von Access aus auf die Daten einer Excel-Datei zugreifen können – ob per VBA oder auch per Verknüpfung oder Import. In diesem Artikel drehen wir den Spieß nun einmal um und zeigen, wie Sie die Daten einer Access-Tabelle in einer Excel-Tabelle einbinden und diese auf dem aktuellen Stand halten. Dabei schauen wir uns sowohl an, wie Sie dies von Excel aus erledigen, als auch wie Sie das Herstellen und Aktualisieren der Verknüpfung von Access aus erledigen.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1902_Access-UndExcel.zip**.

Verknüpfung von Excel aus initiieren
Wenn Sie die Verknüpfung von Excel aus einrichten wollen, legen Sie zunächst eine neue Excel-Datei über die Benutzeroberfläche von Excel an.

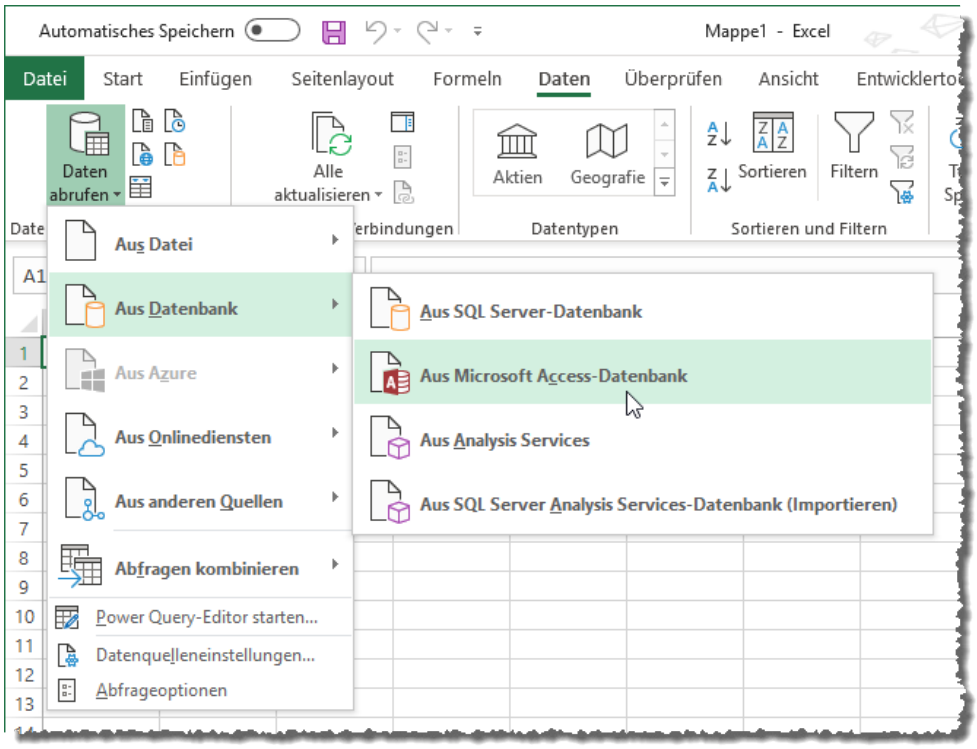


Bild 1: Abrufen von Daten aus einer Access-Datenbank

Dazu wählen Sie den Ribbon-Eintrag **Daten|Daten abrufen und transformieren|Daten abrufen|Aus Datenbank|Aus Microsoft Access-Datenbank** aus (siehe Bild 1).

Im nun erscheinenden **Daten importieren**-Dialog wählen Sie die Access-Datenbank aus, welche die Tabelle oder Abfrage enthält, deren Daten Sie in Excel anzeigen wollen.

Danach erscheint ein Dialog namens **Navigator**, der auf der linken Seite alle Tabellen und Abfragen der Quelldatenbank anzeigt und auf der rechten Seite gleich die Daten der aktuell ausgewählten Datenquelle liefert (siehe Bild 2).

Klicken Sie in diesem Dialog auf die Schaltfläche **Laden**, nimmt sich Excel ein paar Sekunden Zeit und präsentiert die Daten aus der ausgewählten Quelltabelle ordentlich formatiert in einer neuen Excel-Tabelle (siehe Bild 3).

Daten aktualisieren

Wenn Sie nun einen Datensatz in der Access-Tabelle anpassen oder einen Datensatz hinzufügen oder löschen, wollen Sie die Änderungen auch in der Excel-Instanz sehen. Dazu klicken Sie nach dem Durchführen der Änderungen in Access ein-

fach auf den Ribbon-Eintrag **Entwurf|Externe Tabellendaten|Aktualisieren**. Kurz darauf erscheinen die Änderungen dann auch in der Excel-Tabelle.

Die Daten können so zwar in Excel angezeigt und dort auch bearbeitet werden, allerdings wirken sich Änderungen auf der Excel-Seite nicht auf die in der

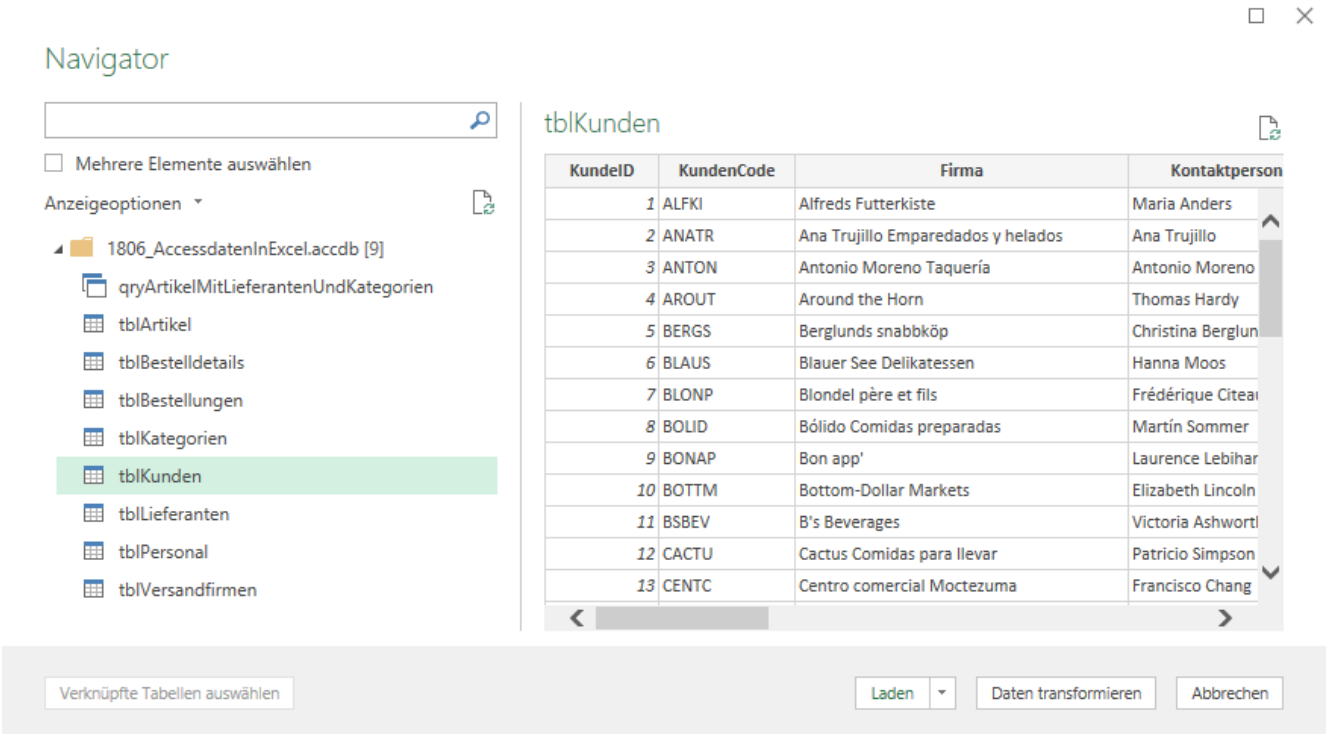


Bild 2: Auswahl der abzufragenden Daten

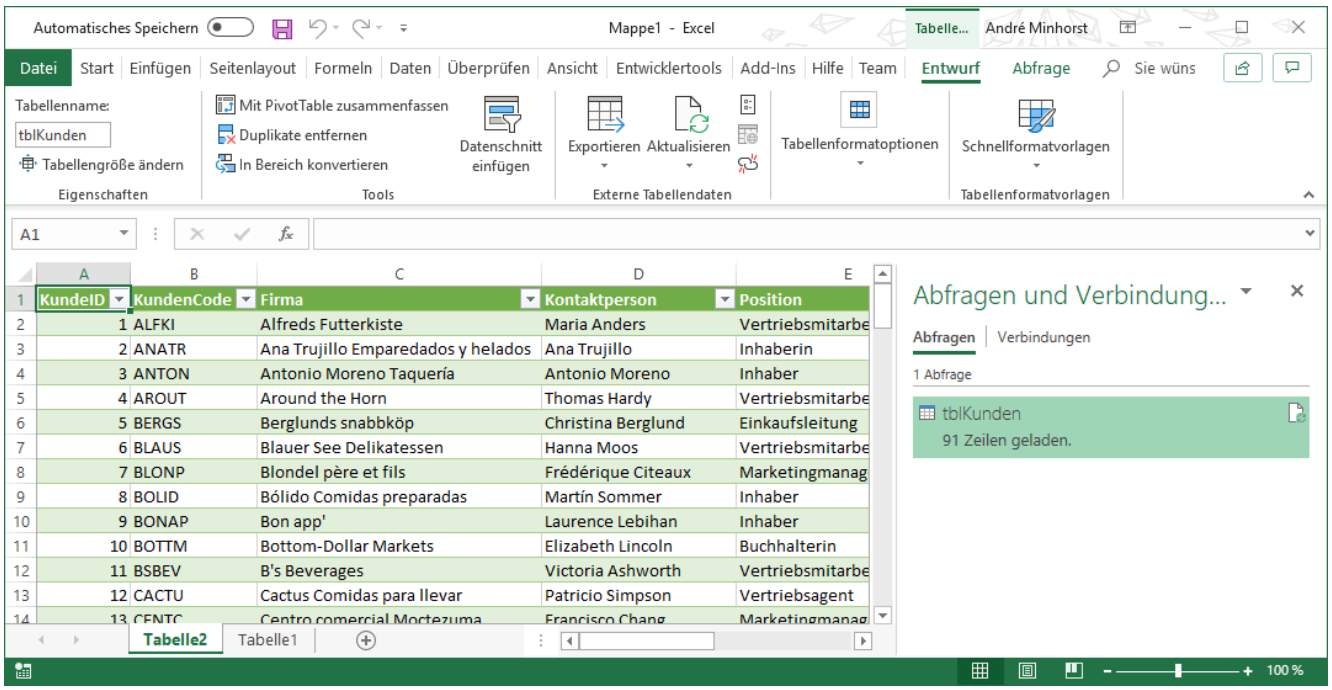


Bild 3: Die Daten der Access-Tabelle in Excel

Access-Datenbank gespeicherten Daten aus. Das ändert sich auch nicht, wenn Sie die Daten in Excel ändern und die Excel-Datei speichern – es bleibt eine Verknüpfung zu den Access-Daten, die von Excel aus nicht dauerhaft geändert werden kann.

Allerdings können Sie natürlich dennoch davon profitieren, dass die Daten nun in Excel vorliegen. Sie können damit beispielsweise Berechnungen durchführen oder Diagramme auf Basis dieser Daten erstellen.

Access-Verknüpfung in Excel von Access aus erstellen

Nun wollen Sie dem Benutzer vielleicht nicht die Arbeit aufbürden, Excel zu öffnen und von Hand die Verknüpfung zu den Daten der Access-Tabelle zu erstellen.

Deshalb zeigen wir Ihnen nun, wie Sie dies bequem von der Access-Anwendung aus erledigen können – per Mausklick auf eine Schaltfläche in einem Formular. Dieses soll direkt die in Excel anzuzeigenden Daten in einem Unterformular anzeigen.

Bevor wir das erledigen, fügen wir einen Verweis auf die Bibliothek **Microsoft Excel x.0 Object Library** hinzu.

Dazu öffnen Sie den VBA-Editor etwa mit der Tastenkombination **Alt + F11** und betätigen dann den Menübefehl **Extras|Verweise**. Hier suchen Sie nach dem gewünschten Eintrag und aktivieren diesen (siehe Bild 4).

Anschließend erstellen wir das Formular frmKundenVerknuepfung, dem wir das Unterformular **sfmKunden** hinzufügen. Dieses enthält als Datensatzquelle die Tabelle **tblKunden** und zeigt die darin enthaltenen Daten in der Datenblattansicht an. Dem Hauptformular fügen wir zunächst eine Schaltfläche namens **cmdDatenInExcelVerknuepfen** hinzu (siehe Bild 5).

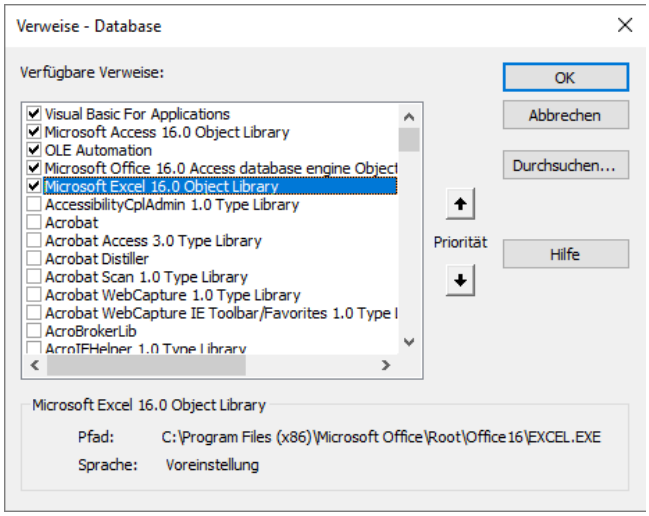


Bild 4: Verweis auf die Excel-Bibliothek

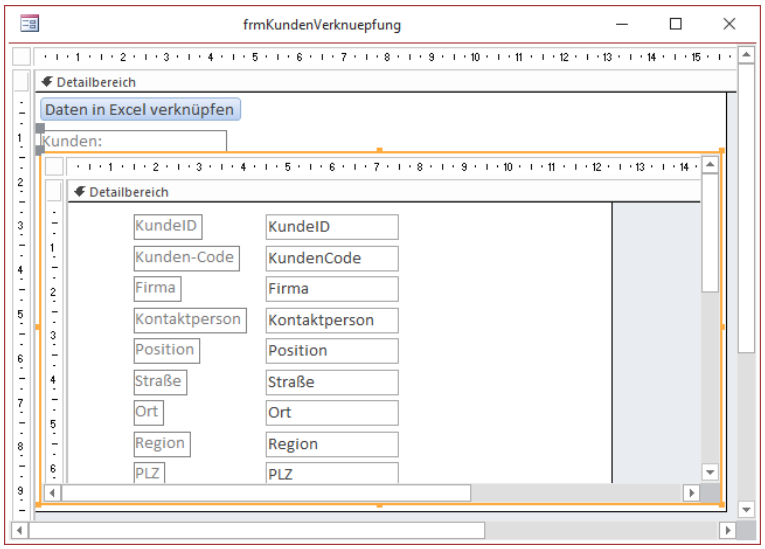


Bild 5: Entwurf des Formulars zum Öffnen der Verknüpfung in Excel

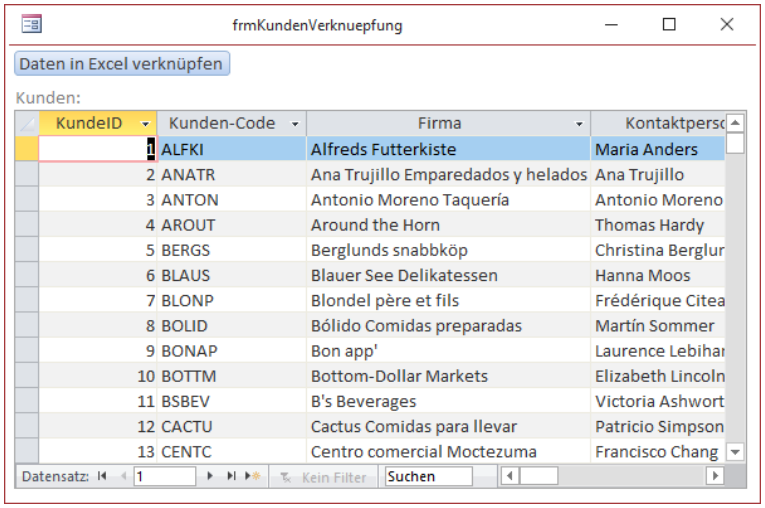


Bild 6: Formular mit den Daten, die in der Excel-Verknüpfung angezeigt werden sollen

Standards für Steuerelemente einfach festlegen

Sie können in Access abweichend von den voreingestellten Standards festlegen, mit welchen Standardwerten die Eigenschaften von Steuerelementen beim Einfügen ausgestattet werden sollen. Wenn Sie etwa wünschen, dass Textfelder eine Schriftgröße von 9 statt der üblichen 11 aufweisen, sind dazu nur wenige Schritte nötig. Noch schneller gelingt es mit einem kleinen Trick, den wir Ihnen in diesem Artikel vorstellen.

Wenn Sie die Standardwerte für Eigenschaften eines Steuerelements für ein Formular festlegen wollen, klicken Sie im Formularentwurf die Schaltfläche zum Hinzufügen des Steuerelements in der Toolbox im Ribbon an, fügen das Steuerelement aber nicht hinzu. Auf diese Weise erscheint im Eigenschaftenblatt nicht der Name des Steuerelements und unter **Auswahltyp** steht nun **Standard: Textfeld** statt einfach nur **Textfeld** (siehe Bild 1).

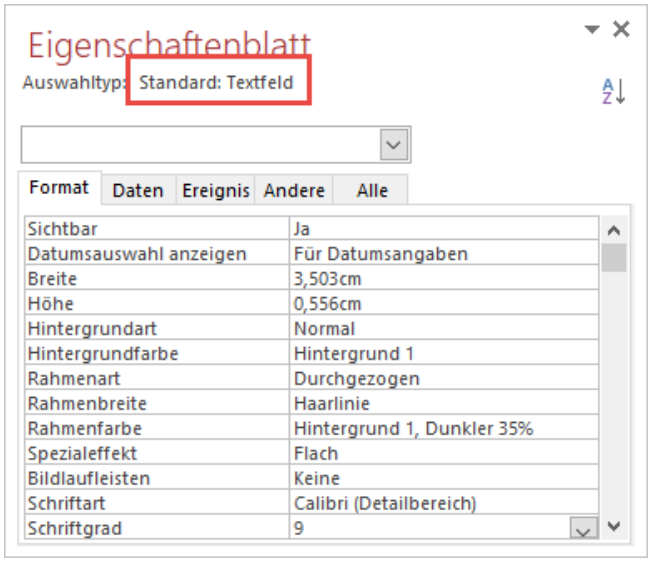


Bild 1: Standard-Eigenschaften eines Steuerelements

Wenn Sie hier nun als **Schriftgrad** den Wert **9** statt **11** einstellen und das Steuerelement dann mit einem Klick in das Entwurfsraster zum Formular hinzufügen, bleibt diese Eigenschaft erhalten. Aber es geht noch weiter: Jedes neu zu diesem Formular hinzugefügte Textfeld erhält ebenfalls den Wert **9** als Schriftgrad.

Wenn Sie dieses Formular nun unter dem Namen **Normal** speichern, gelten die für das Steuerelement festgelegten Standardeigenschaften auch für alle von nun an neu erstellten Formulare. Diese Vorgehensweise haben wir schon im Artikel **Formular- und Berichtsvorlagen** beschrieben.

Standardwerte vorab ansehen

Was aber, wenn Sie die Standardwerte für die Eigenschaften des Steuerelements nicht nur vorab einstellen wollen, also ohne die Einstellungen direkt zu sehen, sondern ein Steuerelement mit den von Ihnen gewünschten Einstellungen ausgestattet haben und diese dann auf das Standardsteuerelement dieses Typs übertragen wollen? Dann hätten Sie unter älteren Access-Versionen noch eine geeignete Schaltfläche vorgefunden, mit der Sie die für ein vorhandenes Steuerelement festgelegten Eigenschaften einfach auf das Standard-Steuerelement übertragen können.

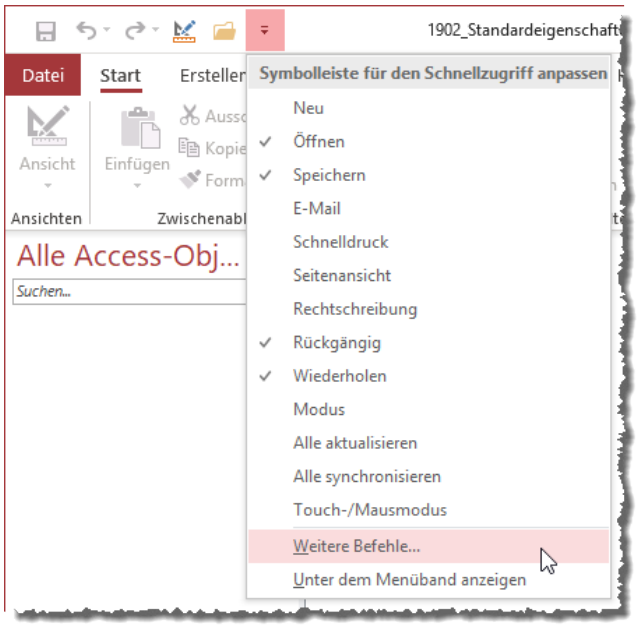


Bild 2: Einblenden weiterer Befehle

Im Ribbon von Access 2016 etwa finden Sie diesen Befehl, der sich Steuerelementvorgaben festlegen nennt, allerdings nicht mehr – zumindest nicht in den standardmäßig vorhandenen Ribbon-Befehlen.

Wenn wir allerdings wie in Bild 2 den Befehl **Weitere Befehle...** anklicken, erscheint der Dialog **Access-Optionen**, wo Sie links direkt den Bereich **Menüband anpassen** anklicken. Hier wählen Sie links unter **Befehle auswählen** den Eintrag **Alle Befehle** aus und navigieren in der Liste zum Befehl **Steuerelementvorgaben festlegen**.

Rechts wählen Sie Sie unter **Menüband anpassen:** den Eintrag **Registerkarten für Tools** aus und legen unter **Formularentwurfstools|Entwurf** mit der Schaltfläche **Neue Gruppe** eine neue Gruppe an.

Dieser benutzerdefinierten Gruppe fügen Sie dann den Eintrag **Steuerelementvorgaben festlegen** hinzu (siehe Bild 3).

Wenn Sie den Dialog nun schließen, ein neues, leeres Formular öffnen und diesem ein Textfeld hinzufügen, finden Sie rechts die aktiviere Schaltfläche **Steuerelementvorgaben festlegen** vor (siehe Bild 4).

Damit können Sie nun, wenn Sie einige Eigenschaften des hinzugefügten Steuerelements so angepasst haben, dass Sie die neuen Werte als Standardwerte für die Eigenschaften dieses Steuerelements übernehmen wollen, per Mausklick die gewünschten Eigenschaftswerte übertragen. Danach werden die neuen Werte auch bei neu hinzugefügten Steuerelementen übernommen. Wie Sie diese Vorgaben

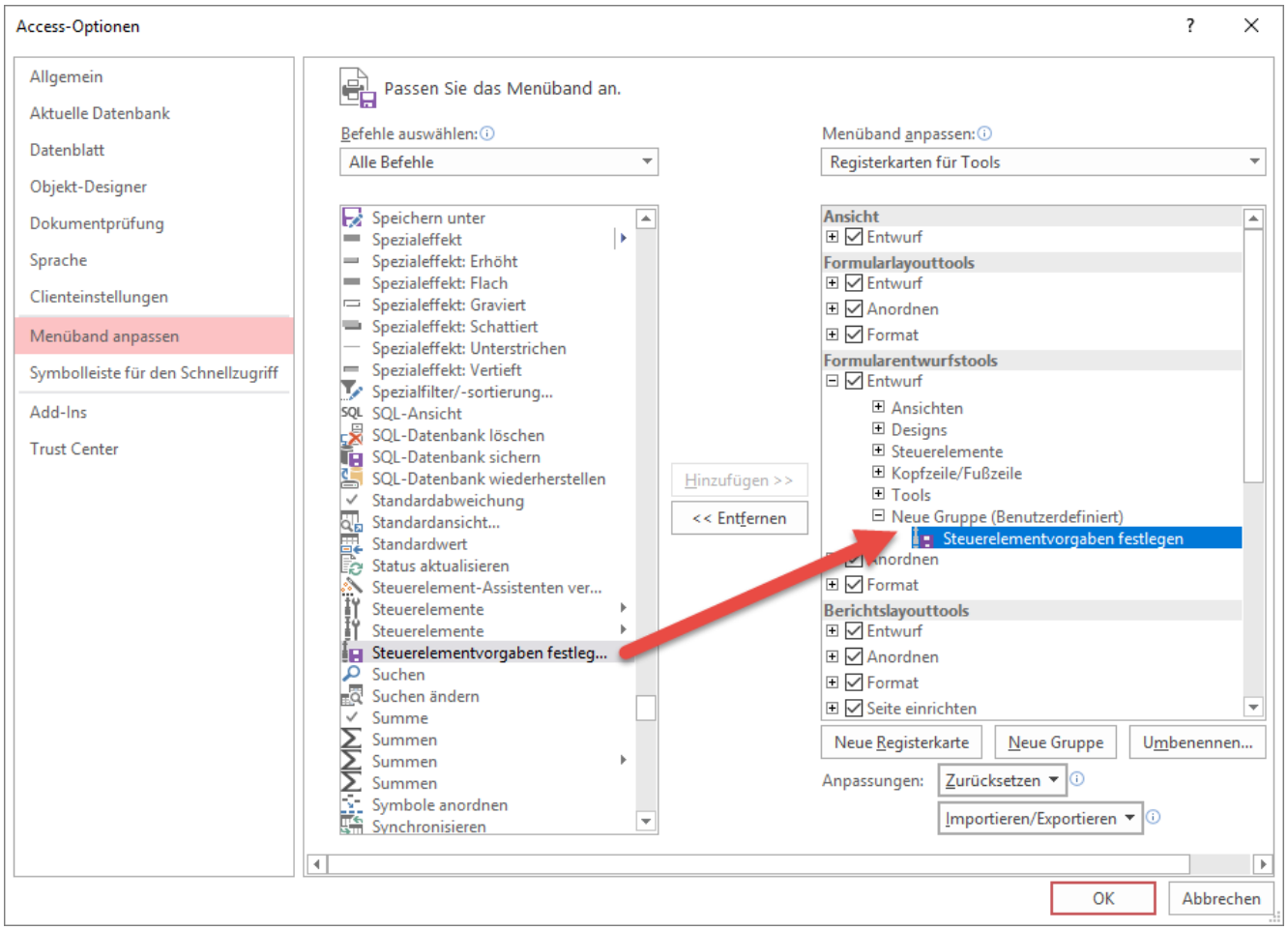


Bild 3: Hinzufügen eines Befehls zum Ribbon

XML: Eine Einführung

XML ist neben JSON eine der Sprachen, mit der Daten strukturiert in Textdateien gespeichert werden können. XML ähnelt HTML, der Auszeichnungssprache für Internetseiten. Die Inhalte werden in beiden Fällen dort durch bestimmte, in spitzen Klammern eingefasste Bezeichnungen gekennzeichnet und so zum Beispiel im Fall von HTML mit entsprechenden Formatierungen versehen. Unter Access und anderen Anwendungen dient XML der Vereinfachung beim Austausch von Daten zwischen verschiedenen Anwendungen. Wenn beide Anwendungen das gleiche XML-Format kennen, können die Daten etwa aus den Tabellen einer Datenbank in das XML-Format übertragen und dann von der empfangenden Anwendung in diesem Format eingelesen werden.

Beispieldatenbank
Die Beispiele dieses Artikels finden Sie in der Datenbank **1903_XML.accdb**.

XML: Extensible Markup Language
Die Abkürzung **XML** steht für **Extensible Markup Language**. Genauso wie bei **HTML** (**Hypertext Markup Language**) steht das **M** für **Markup**. **Markup** steht dabei für Markierung. Die Markup-Elemente in einem XML-Dokument sorgen dabei für die Strukturierung der darin enthaltenen Daten.

XML-Dokumente können sowohl von Maschinen als auch von Menschen recht gut gelesen werden, wobei es für den Menschen natürlich bessere Formate gibt – aber die lassen sich ja durchaus mit der Nutzung der Markup-Elemente aus einem XML-Dokument ableiten.

Das folgende ist ein Beispiel für ein XML-Dokument. Hier finden Sie ein übergeordnetes Element namens **Kunden**, das mehrere **Kunde**-Elemente enthält. Für das **Kunde**-Element ist ein Attribut namens **Kundennummer** definiert, welches die Kundennummer aufnimmt.

Außerdem gibt es innerhalb des **Kunde**-Elements weitere Elemente wie zum Beispiel eines für den Vornamen und eines für den Nachnamen. Das **Kunde**-Element enthält noch eine Element namens **Bestellungen**, das weitere Unterelemente namens Bestellung liefert. Diese enthalten wiederum ein Attribut namens **Bestellnummer** und ein Element namens Bestelldatum. Eine Bestellung enthält meist noch weitere Informationen etwa über die Bestellpositionen et cetera, aber für ein einfaches Beispiel wollen wir uns auf die bisher gezeigten Elemente beschränken:

```
<Kunden>
  <Kunde Kundennummer="123">
    <Nachname>Minhorst</Nachname>
    <Vorname>André</Vorname>
    <Bestellungen>
      <Bestellung Bestellnummer="234">
        <Bestelldatum>29.07.2019</Bestelldatum>
        ...
      </Bestellung>
    </Bestellungen>
  </Kunde>
  <Kunde Kundennummer="124">
    ...
  </Kunde>
</Kunden>
```

Sie sehen, dass XML nicht nur etwa Daten in Listenform aufnehmen kann, sondern auch Daten in hierarchischer Beziehung.

In einer Access-Datenbank würden Sie die Kundendaten in einer Kunden-Tabelle speichern und die Bestelldaten in einer Bestellungen- und einer Bestellpositionen-Tabelle – wobei diese in der Regel noch

eine Artikel-Tabelle referenziert. Unter XML können Sie durch geeignete Strukturierung die Daten mehrerer hierarchisch verknüpfter Tabellen in einer einzigen Datei unterbringen. Die Daten der Kunden-Tabelle landen dabei in den **Kunde**-Elementen unmittelbar unter dem **Kunden**-Element. Zum **Kunde**-Element gibt es ein **Bestellungen**-Element, das kein, ein oder mehrere Elemente des Typs **Bestellung** enthält – und so weiter.

Öffnende und schließende Elemente

Für XML-Dokumente gibt es einige grundlegende Regeln und außerdem können Sie noch benutzerdefinierte Regel festlegen. Zu den grundlegenden Regeln gehört, dass jedes **Tag**-Element, als eines der in spitzen Klammern eingefasstes Elemente, durch ein entsprechendes schließendes Element ergänzt werden muss.

Das schließende Element besteht dabei aus einer öffnenden spitzen Klammer, einem Schrägstrich, dem Elementnamen und einer schließenden spitzen Klammer. In unserem Beispiel ist das etwa beim **Kunden**-Element der Fall:

```
<Kunden>
  . . .
 </Kunden>
```

Dazwischen kann das Element beliebige weitere Elemente enthalten. Wichtig ist dabei, dass auch diese immer durch ein schließendes Element ergänzt werden, auch End-Tag genannt:

```
<Kunden>
  <Kunde  . . .>
    . . .
  </Kunde>
  <Kunde  . . .>
    . . .
  </Kunde>
</Kunden>
```

Bei Elementen, die keine weiteren Elemente einschließen, sondern beispielsweise komplett leer sind oder auch nur Attribute enthalten, können Sie das öffnende und das schließende Element zusammenfassen. Dazu fügen sie vor der schließenden spitzen Klammer einen Schrägstrich ein:

```
<Kunde/>
```

Ein solches Element kann auch Attribute enthalten:

```
<Kunde Kundennummer="123"/>
```

Interpretieren des XML-Dokuments

Ein Mensch könnte ein XML-Dokument, bei dem mal ein schließendes Element fehlt oder bei dem ein Tag falsch geschrieben ist, womöglich noch korrekt interpretieren. Wenn Sie das XML-Dokument hingegen maschinell einlesen wollen, dürfen Sie sich keine Fehler erlauben – sonst kann das Dokument nicht interpretiert werden.

Für XML gibt es für alle bekannten Programmiersprachen Bibliotheken, mit denen Sie den Inhalt eines XML-Dokuments lesen können. Dabei ist es nicht nötig, wie etwa bei HTML das ganze Dokument einzulesen und zu verarbeiten, um das gewünschte Ergebnis zu erhalten – nämlich eine optisch ansprechend formatierte Ausgabe im Internetbrowser.

Sie können etwa nur die **Kunde**-Elemente durchlaufen und die Kundendaten ausgeben oder verarbeiten, ohne die Bestelldaten zu berücksichtigen.

Sie können XML-Dokumente auch auf verschiedene Arten erstellen: Die auf den ersten Blick einfachste wäre, einfach einige Zeilen Text zusammenzustellen, welche die öffnenden und schließenden Elemente sowie die gewünschten Informationen enthalten.

Viel effizienter ist es jedoch, die Elemente über das sogenannte **Document Object Model** (**DOM**) zu erstellen und durch untergeordnete Elemente zu ersetzen.

Genau dieses **Document Object Model** können Sie auch nutzen, um auf die Elemente eines bestehenden XML-Dokuments zuzugreifen.

XML in Webservices

Einer der Einsatzzwecke von XML-Dokumenten ist der Austausch von Daten mittels Webservice. Ein Webservice ist ein Service, der Anfragen über das HTTP- oder HTTPS-Protokoll entgegennimmt und die Antwort über das gleiche Protokoll zurücksendet.

Wenn die zu sendenden Informationen strukturiert sein sollen, können Sie diese in ein nach den Anforderungen des Webservices gestaltetes XML-Dokument einbetten. Die Antwort kann dann auch in Form eines XML-Dokuments gesendet werden, welches Sie dann per Code interpretieren.

Wohlgeformte XML-Dokumente

Damit Programme XML-Dokumente parsen können, müssen diese wohlgeformt sein. Sie haben bereits erfahren, dass XML-Dokumente aus Elementen bestehen, die ein Starttag (**<Kunde>**) und ein Endtag enthalten (**</Kunde>**), die aber auch durch ein einziges Element ersetzt werden können, wenn dieses keine weiteren Elemente oder Inhalte aufweist. Das heißt dann auch Leer-Tag (**<Kunde/>**).

Damit ein XML-Dokument wohlgeformt ist, gelten einige weitere Regeln. So darf ein XML-Dokument nur ein Wurzelelement enthalten, also ein Element, das alle anderen Elemente einschließt.

Sie könnten in einem XML-Dokument also beispielsweise nicht einfach mehrere **Kunde**-Element anlegen:

```
<Kunde ...>...</Kunde>
...
<Kunde ...>...</Kunde>
```

Das wären dann zwei Wurzelelemente. Deshalb muss beispielsweise ein übergeordnetes Element namens **Kunden** vorhanden sein:

```
<Kunden>
  <Kunde ...>...</Kunde>
  ...
  <Kunde ...>...</Kunde>
</Kunden>
```

Eher würde man jedoch noch ein Element wie **<Kundenverwaltung>** nutzen:

```
<Kundenverwaltung>
  <Kunden>
    ...
  </Kunden>
</Kundenverwaltung>
```

Öffnende und schließende Elemente müssen sich genau wie etwa bei **If...Then**-Bedingungen oder Schleifen auf der gleichen Ebene befinden. Sie können also nicht ein **Kunde**-Element öffnen, darin ein **Bestellung**-Element öffnen und dann das **Kunde**-Element schließen, bevor nicht das **Bestellung**-Element geschlossen ist.

Es kann zwar mehrere Elemente gleichen Namens sowohl auf der gleichen Ebene geben als auch auf verschiedenen Ebenen, aber Attribute dürfen nur einmal je Element verwendet werden. Wenn Sie also Informationen unterbringen möchten, die mehrmals je Element vorkommen, müssen Sie dazu Unterelemente nutzen. Wenn Sie also etwa etwa einfach nur angeben wollen, welche Bestellnummern zu einem Kunden gehören, dann geht das nicht:

```
<Kunde Kundennummer="123" Bestellnummer="1" Bestellnummer="2" ...>
```

Hier benötigen Sie Unterelemente:

```
<Kunde Kundennummer="123">
  <Bestellung Bestellnummer="1"/>
  <Bestellung Bestellnummer="2"/>
  ...
</Kunde>
```

Werte von Attributen müssen immer entweder in Anführungszeichen oder Hochkommata eingefasst werden:

```
<Kunde Kundennummer="123">
```

oder

XML: Austausch mit Access-Bordmitteln

Im Artikel XML: Eine Einführung haben wir uns angesehen, wie XML-Dokumente aufgebaut sind und was bei ihrer Erstellung zu beachten ist. In diesem Teil der Artikelreihe wollen wir uns ansehen, welche Möglichkeiten die Bordmittel von Access für die Erstellung und das Einlesen von XML-Dokumenten bieten. Dabei verwenden wir die Tabellen der Südsturm-Datenbank und schauen uns an, welche Ergebnisse wir beim Exportieren erhalten.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1903_XMLMitBordmitteln.accdb**.

Tabelle exportieren

Wenn wir uns das Kontextmenü einer Tabelle im Navigationsbereich von Access ansehen, finden wir gleich die erste Möglichkeit, die Daten einer Tabelle im XML-Format zu exportieren.

Dazu wählen wir im Kontextmenü einfach den Eintrag **Exportieren|XML-Datei** aus (siehe Bild 1).

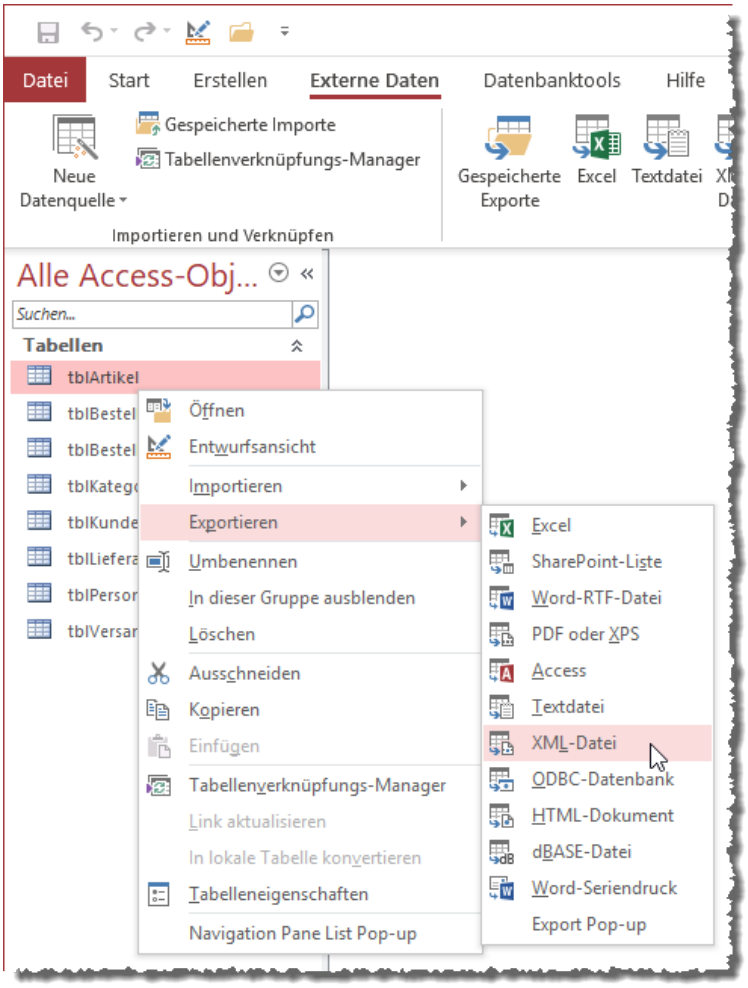


Bild 1: Export einer Tabelle in das XML-Format

Im nächsten Schritt fragt Access den Speicherort für das zu exportierende XML-Dokument ab (siehe Bild 2), wo Sie sich Unterstützung durch einen **Durchsuchen...**-Dialog holen können.

Im folgenden Dialog wählen Sie dann aus, welche Informationen in der zu erstellenden XML-Datei gespeichert werden sollen (siehe Bild 3).

Hier haben Sie folgende Optionen zur Auswahl:

- **Daten (XML)**: Gibt die Daten aus, also eine XML-Datei mit Elementen und den Inhalten der Tabellenfelder

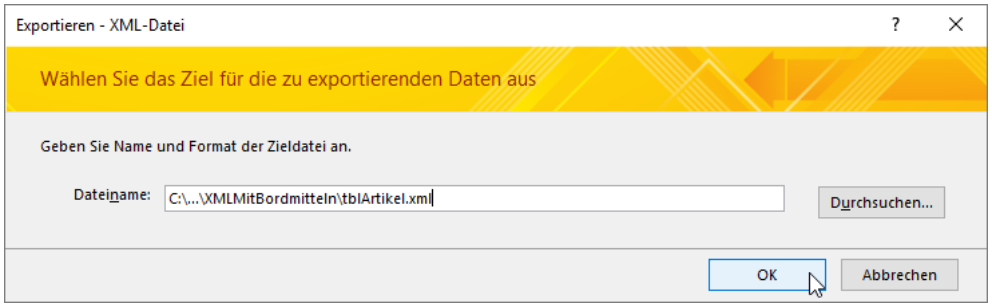


Bild 2: Angabe des Speicherortes für das zu exportierende XML-Dokument

- **Schema der Daten (XSD)**: Legt fest, dass das Schema der Daten ausgegeben werden soll.

- **Präsentation der Daten (XSL)**: Gibt eine XSL-Datei zur Definition der Präsentation der Daten aus.

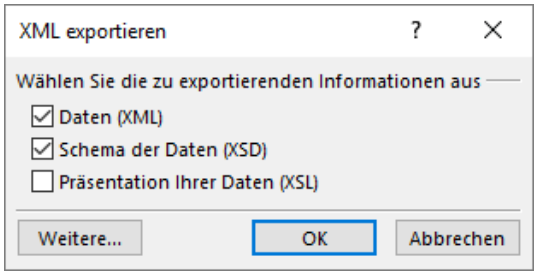


Bild 3: Festlegen der Optionen

Sie können auch noch auf die Schaltfläche **Weitere...** klicken. Dies öffnet einen weiteren Dialog, mit dem Sie für die drei Optionen des vorherigen Dialogs weitere Optionen einstellen können.

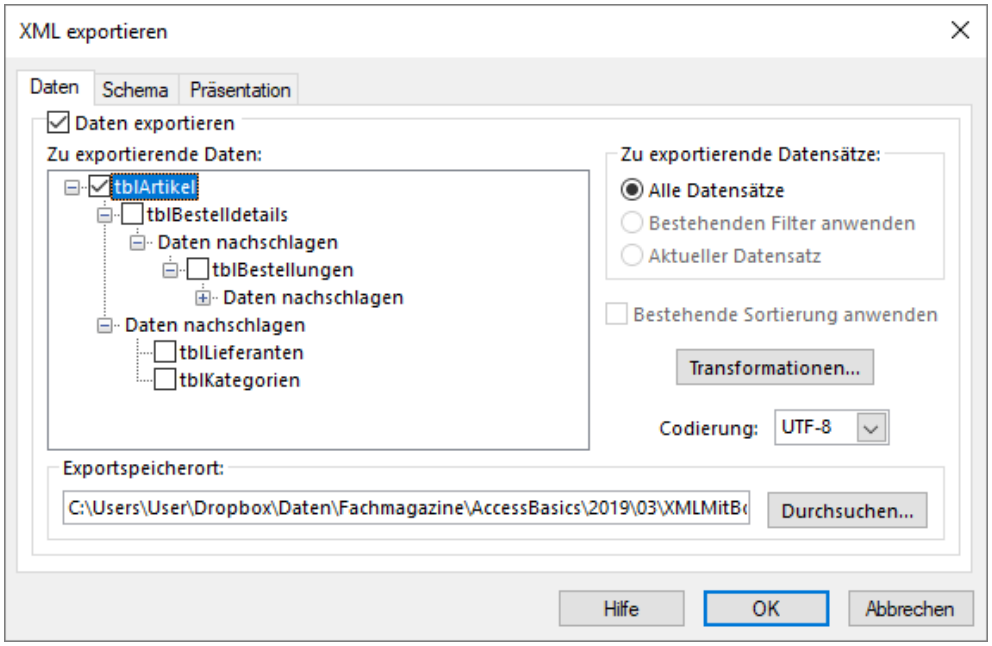


Bild 4: Weitere Optionen für den Export einer XML-Datei

Bild 4 zeigt etwa die Optionen für den Export der Daten in der XML-Datei. Hier können Sie auch nochmals explizit festlegen, ob Sie die Daten exportieren wollen. Außerdem zeigt der Bereich **Zu exportierende Daten** an, welche Tabellen noch mit der zu exportierenden Tabelle verknüpft sind und bietet die Möglichkeit, diese ebenfalls für den Export vorzusehen. Klicken Sie auf die Plus-Zeichen, um weitere aktuell nicht sichtbare Einträge einzublenden.

Codierung einstellen

Außerdem können Sie hier auch einstellen, unter welcher Codierung die exportierte XML-Datei gespeichert werden soll. Hier stehen die beiden Möglichkeiten **UTF-8** und **UTF-16** zur Auswahl.

Zu exportierende Datensätze einstellen

Der Bereich Zu exportierende Datensätze ist im Screenshot noch deaktiviert. Das liegt daran, dass die Tabelle mit den zu exportierenden Daten aktuell nicht geöffnet ist. Daher gibt es auch keinen bestehenden Filter und auch keinen aktuellen Datensatz. Zu Beispielzwecken wollen wir zunächst erst einmal einen Datensatz exportieren, deshalb schließen wir den Dialog wieder und öffnen erst einmal die Tabelle **tblArtikel** in der Datenblattansicht.

Wenn wir dann erneut den Kontextmenü-Eintrag **Exportieren|XML-Datei** auswählen, ist nun auch der Eintrag **Aktueller Datensatz** markiert. Die Option Bestehenden Filter anwenden ist jedoch immer noch deaktiviert. Damit wir diese nutzen können, müssen wir vor dem Öffnen des Dialogs einen Filter auf die Daten der Tabelle anwenden – zum Beispiel, indem wir nach den Datensätzen filtern, deren Artikelname mit dem Buchstaben **C** beginnt.

Sortierung einstellen

Wenn wir dann noch eine Sortierung für das Datenblatt einstellen, bevor wir den Kontextmenü-Befehl

Exportieren|XML-Datei betätigen, aktiviert der Dialog auch noch die Option **Bestehende Sortierung** anwenden.

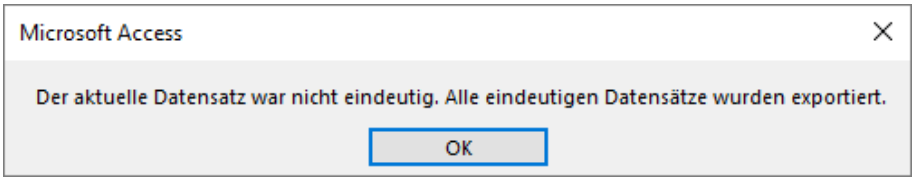


Bild 5: Irreführende Meldung

Unter diesen Einstellungen können wir den ersten Export testen – zunächst nur mit aktivierter Option **Daten (XML)** und nur für den aktuellen Datensatz (Option **Aktueller Datensatz**). Dabe erhalten wir die irreführende Meldung aus Bild 5. Diese erklärt, dass der aktuelle Datensatz nicht eindeutig war. Ich welcher Hinsicht, erläutert die Meldung nicht – aber sie ist irreführend, weil natürlich jeder Datensatz durch den eindeutigen Index im Primärschlüsselfeld eindeutig ist.

Erster Export

Das Ergebnis sieht jedenfalls wie folgt aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<dataroot xmlns:od="urn:schemas-microsoft-
      com:officedata" generated="2019-07-29T16:11:03">
  <tblArtikel>
    <ArtikelID>1</ArtikelID>
    <Artikelname>Chai</Artikelname>
    <LieferantID>1</LieferantID>
    <KategorieID>1</KategorieID>
    <Liefereinheit>10 Kartons x 20 Beutel</Liefereinheit>
    <Einzelpreis>9</Einzelpreis>
    <Lagerbestand>39</Lagerbestand>
    <BestellteEinheiten>0</BestellteEinheiten>
    <Mindestbestand>10</Mindestbestand>
    <Auslaufartikel>0</Auslaufartikel>
  </tblArtikel>
  <tblArtikel>
    ...
  </tblArtikel>
  ...
</dataroot>
```

Access erstellt also ein **dataroot**-Element, das als Wurzel-Element dient, und fügt darunter die einzelnen **tblArtikel**-Elemente ein, die jeweils einen Datensatz der Tabelle **tblArtikel** repräsentieren. Für jedes Feld der Tabelle wird dabei ein Unterelement im Element **tblArtikel** angelegt. Die Daten des jeweiligen Feldes landen dabei im jeweiligen Element.

Als Codierung wird **UTF-8** angegeben – genau wie im Dialog mit den Export-Optionen angegeben war.

Test auf Sonderzeichen

Bevor wir weitere Optionen testen, wollen wir prüfen, was geschieht, wenn die Tabellenfelder Werte mit unzulässigen Sonderzeichen wie dem Größer- oder Kleiner-Zeichen enthalten. Dazu ändern wir den Wert des Feldes **Artikelname** im ersten Datensatz wie folgt:

```
Chai<Test>
```

Wenn wir den Export nun wiederholen, erhalten wir folgendes Ergebnis:

```
...
<tblArtikel>
  <ArtikelID>1</ArtikelID>
  <Artikelname>Chai&lt;Test&gt;</Artikelname>
  ...
</tblArtikel>
...
```

Diese Sonderzeichen werden also schon einmal berücksichtigt.

Export mit verknüpften Tabellen

Da der Export-Dialog schon die Option anbietet, die Daten verknüpfter Tabellen gleich mit zu exportieren, wollen wir dies auch einmal ausprobieren.

Dazu markieren wir diesmal neben der Tabelle tblArtikel auch noch die verknüpften Tabellen tblBestellde-

tails und **tblBestellungen** sowie die Lookup-Tabellen **tblLieferanten** und **tblKategorien** (siehe Bild 6).

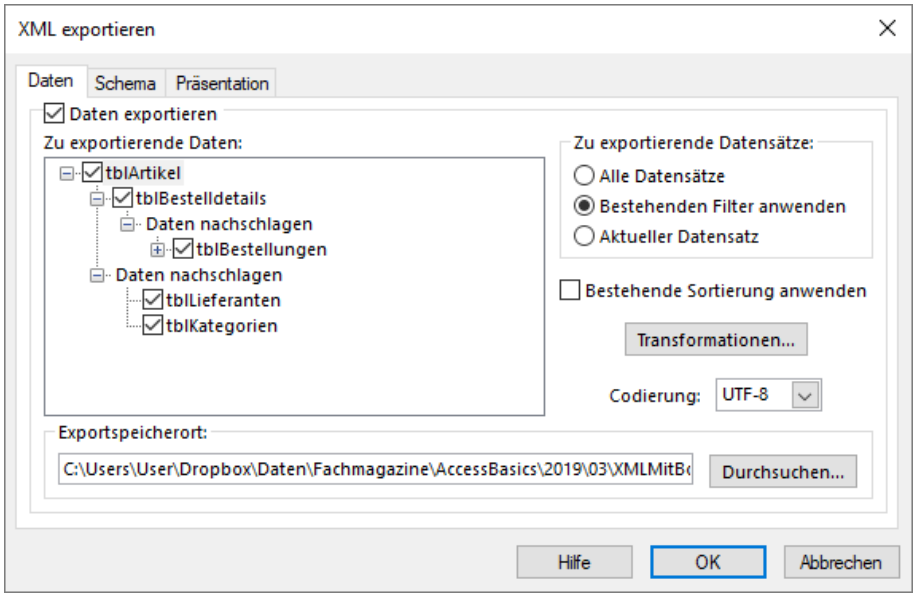


Bild 6: Einstellung für das nächste Export-Beispiel

Wir haben die Tabelle zuvor geöffnet und nur nach dem Artikel **Chai** gefiltert. Da wir die Option **Bestehenden Filter anwenden** aktiviert haben, sollen nur die Daten für diesen einen Artikel exportiert werden.

Die so erstellte Datei enthält zunächst die Artikeldaten, wobei diese in Unterelementen namens **tblBestelldetails** die Einträge der Tabelle **tblBestelldetails** enthalten, die dem jeweiligen Artikel zugeordnet sind:

```
<?xml version="1.0" encoding="UTF-8"?>
<dataroot xmlns:od="urn:schemas-microsoft-
com:officedata" generated="2019-07-29T20:08:52">
  <tblArtikel>
    <ArtikelID>1</ArtikelID>
    <Artikelname>Chai</Artikelname>
    <LieferantID>1</LieferantID>
    <KategorieID>1</KategorieID>
    <Liefereinheit>10 Kartons x 20 Beutel</Liefereinheit>
    <Einzelpreis>9</Einzelpreis>
    <Lagerbestand>39</Lagerbestand>
    <BestellteEinheiten>0</BestellteEinheiten>
    <Mindestbestand>10</Mindestbestand>
    <Auslaufartikel>0</Auslaufartikel>
    <tblBestelldetails>
      <BestellungID>10285</BestellungID>
      <ArtikelID>1</ArtikelID>
      <Einzelpreis>7.2</Einzelpreis>
      <Anzahl>45</Anzahl>
      <Rabatt>0.2</Rabatt>
    </tblBestelldetails>
    <tblBestelldetails>
      ...
    </tblBestelldetails>
    ...
  </tblArtikel>
```

Danach folgen alle Einträge der Tabelle **tblBestellungen**. Man sollte meinen, dass nur die Bestellungen ausgegeben werden, in denen auch der aktuelle Artikel vorkommt.

Offensichtlich werden aber alle Bestellungen ausgegeben:

```
  <tblBestellungen>
    <BestellungID>10248</BestellungID>
    <KundeID>90</KundeID>
    <PersonalID>5</PersonalID>
    <Bestelldatum>1996-07-04T00:00:00</Bestelldatum>
    <Lieferdatum>1996-08-01T00:00:00</Lieferdatum>
    <Versanddatum>1996-07-16T00:00:00</Versanddatum>
    <VersandUeber>3</VersandUeber>
    <Frachtkosten>16.19</Frachtkosten>
    <Empfaenger>Vins et alcools Chevalier</Empfaenger>
    <Strasse>59 rue de l&apos;Abbaye</Strasse>
    <Ort>Reims</Ort>
    <PLZ>51100</PLZ>
    <Bestimmungsland>Frankreich</Bestimmungsland>
  </tblBestellungen>
  <tblBestellungen>
```

XML exportieren per VBA

Im Artikel **XML: Austausch mit Access-Bordmitteln** haben wir gezeigt, wie Sie die Daten einer Access-Tabelle oder auch die Daten mehrerer verknüpfter Access-Tabellen in ein XML-Dokument exportieren können. Möglicherweise möchten Sie für einen solchen Export nicht immer den Export-Assistenten bemühen. Deshalb haben wir uns angesehen, wie Sie den Export per VBA durchführen können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1903_XMLExportieren-PerVBA.accdb**.

Exportieren per VBA-Methode

Für den Export der Daten aus Tabellen oder Abfragen stellt Access die Methode **ExportXML** des **Application**-Objekts bereit. Diese Methode hat die folgenden Pflichtparameter (siehe Bild 1):

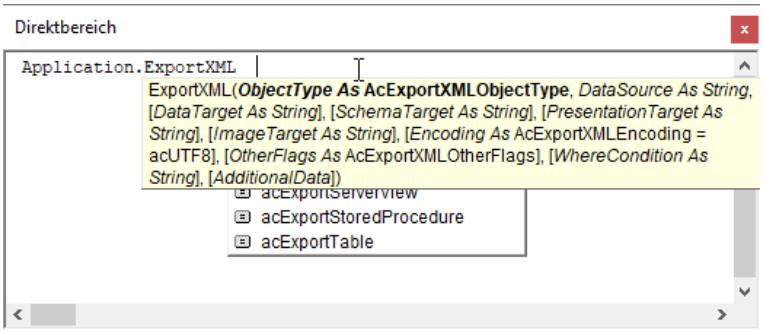


Bild 1: Aufruf der Methode **ExportXML** des **Application**-Objekts

- **ObjectType**: Einer der Werte **acExportForm**, **acExportFunction**, **acExportQuery**, **acExportReport**, **acExportServerView**, **acExportStoredProcedure** oder **acExportTable**. Wir interessieren uns hauptsächlich für den Export von Tabellen, also verwenden wir **acExportTable** als Wert des ersten Parameters. Wenn Sie beispielsweise das Ergebnis einer Abfrage exportieren wollen, verwenden Sie den Parameterwert **acExportQuery**.

- **DataSource**: Gibt den Namen des zu exportierenden Objekts an, in unserem Fall also etwa den Namen einer Tabelle oder Abfrage.

- **DataTarget**: Gibt den Namen der zu erstellenden Datei an. Bei diesem Parameter handelt es sich zwar um einen optionalen Parameter, aber wenn Sie ihn weglassen, liefert Access beim Aufruf von **ExportXML** eine Fehlermeldung (**Ungültige Parameter für ExportXML**).

Damit können wir bereits den ersten Test starten und die folgende Anweisung im Direktbereich des VBA-Editors absetzen:

```
Application.ExportXML acExportTable, "tblKategorien", CurrentProject.Path & "\tblKategorien.xml"
```

Wir wollen also ein Element des Typs **acExportTable** exportieren, das den Namen **tblKategorien** aufweist.

Für die Zieldatei haben wir mit **CurrentProject.Path** das Verzeichnis der aktuellen Datenbank ermittelt und den Dateinamen **tblKategorien.xml** angegeben.

Das Ergebnis des Exports finden Sie in Bild 2. Hier ist anzumerken, dass die **ExportXML**-Anweisung eventuell vorhandene Dateien gleichen Namens ohne Rückfrage überschreibt.

Die exportierte XML-Datei unterscheidet sich nicht von einer mit dem Export-Assistenten exportieren Tabelle.

Weitere Parameter von ExportXML

Die Methode stellt noch weitere Parameter bereit, die weitgehend die Optionen des Dialogs **XML exportieren** und des Dialogs zur Anzeige der erweiterten Einstellungen des XML-Exports wiedergeben:

- **SchemaTarget**: Wenn Sie eine Schema-Datei erzeugen lassen wollen, geben Sie hier den Namen der Schema-Datei an. Im Dialog **XML exportieren** erledigen Sie das einfach durch Anhaken der Option **Schema der Daten (XSD)** – siehe Bild 3. Den Namen geben Sie in den erweiterten Eigenschaften unter **Schema** ein.

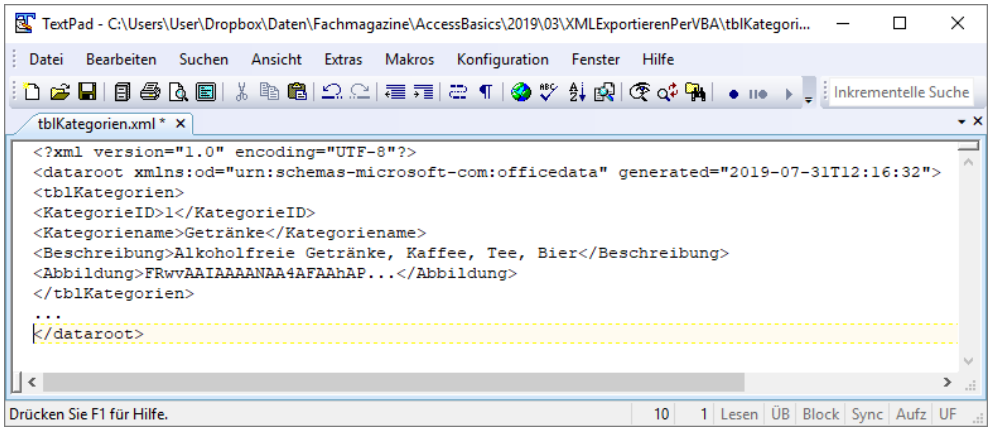


Bild 2: Export der Tabelle **tblKategorien** (gekürzte Fassung)

- **PresentationTarget**: Dafür gilt das gleiche wir für die Option **SchemaTarget**: Sie geben hier den Namen der zu erstellenden XSL-Datei an. Dies entspricht dem Aktivieren der Option **Schema der Daten (XSD)** aus dem Dialog **XML exportieren**. Dort geben Sie den Namen der Präsentationsdatei im erweiterten Dialog unter Präsentation ein.

- **ImageTarget**: Erlaubt laut Onlinehilfe die Angabe eines Ordners für Bilddateien. Wir konnten allerdings keine sinnvolle Anwendung für diesen Parameter finden.

- **Encoding**: Einer der beiden Werte **acUTF8** oder **acUTF16**.

- **OtherFlags**: Weitere Parameter – siehe unten.

Bild 3: Der Dialog **XML exportieren**

- **WhereCondition**: Nimmt ein Kriterium entgegen, um die auszugebenden Daten einzuschränken.

- **AdditionalData**: Legt fest, welche weiteren Tabellen berücksichtigt werden sollen. Dies ist beim Export über die Benutzeroberfläche im Dialog mit den erweiterten Einstellungen möglich (siehe Bild 4).

Der Parameter OtherFlags

Dieser Parameter nimmt eine Kombination aus weiteren Konstanten auf. Diese beschreiben wir in der folgenden Auflistung (hier nur die für unsere Einsatzzwecke sinnvollen Werte). Wenn Sie den Export über

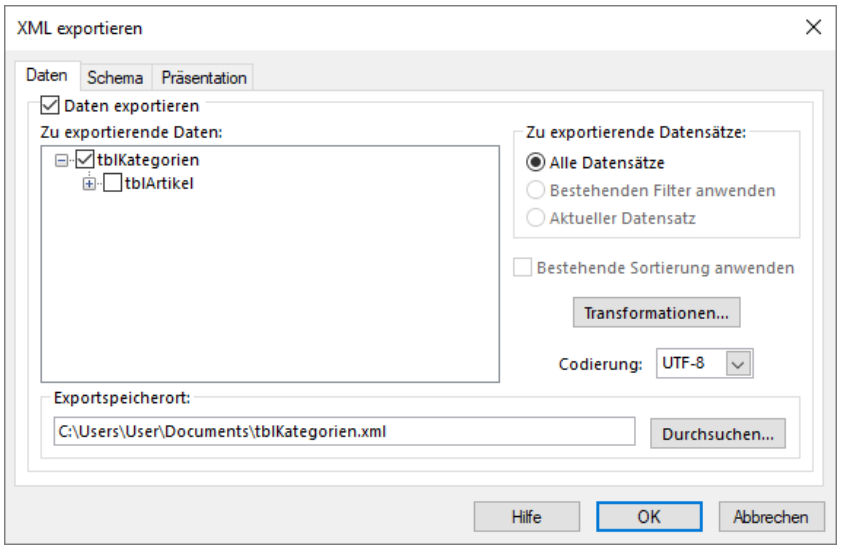


Bild 4: Weitere Einstellung des Dialogs **XML exportieren**

die Benutzeroberfläche durchführen, finden Sie die entsprechenden Optionen auf der Registerseite **Schema** der erweiterten Einstellungen des Dialogs **XML exportieren** (siehe Bild 5):

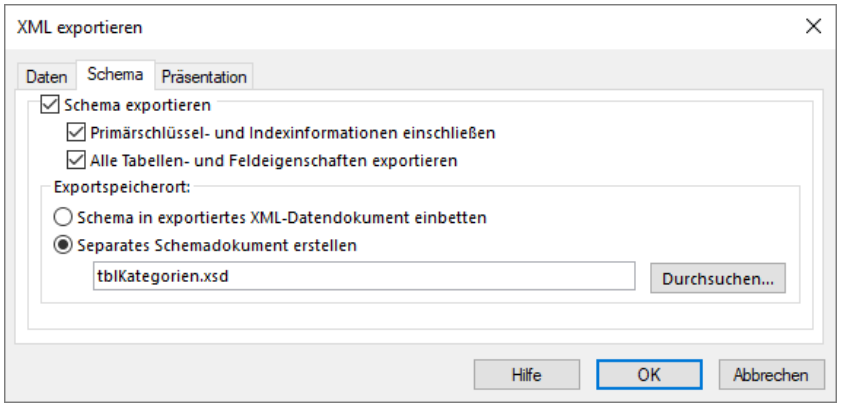


Bild 5: Optionen in den erweiterten Einstellungen

- **acEmbedSchema** (Zahlenwert **1**): Schreibt die Schema-Informationen direkt in das mit dem Parameter **DataTarget** angegebene zu erstellende Dokument. Der Inhalt, der sonst in die durch **SchemaTarget** angegebene Datei geschrieben wird, landet dann direkt in der unter **DataTarget** angegebenen Datei.

- **acExcludePrimaryKeyAndIndexes** (2): Legt fest, ob Primärschlüssel- und Indexinformationen vom Export ausgeschlossen werden sollen.

- **acExportAllTableAndFieldProperties** (32): Legt fest, ob Tabellen- und Feldeigenschaften exportiert werden sollen.

Daten einschränken mit dem Parameter WhereCondition

Wenn Sie nicht alle Daten einer Tabelle in das XML-Dokument exportieren wollen, haben Sie zwei Möglichkeiten: Sie können eine Abfrage erstellen, welche die Daten nach den gewünschten Kriterien filtert.

Oder Sie verwenden den Parameter **WhereCondition** der **ExportXML**-Methode.

Wenn Sie etwa nur die erste Kategorie exportieren wollen, verwenden Sie einen Aufruf wie den folgenden:

```
Application.ExportXML acExportTable, "tblKategorien",
CurrentProject.Path & "\tblKategorien.xml", , , , , ,
"KategorieID = 1"
```

Sollen Ihnen die ganzen Kommata zu unübersichtlich sein, können Sie auch mit benannten Parametern arbeiten – oder auch mit nur einem benannten Parameter für den letzten Parameter:

```
Application.ExportXML acExportTable, "tblKategorien",
CurrentProject.Path & "\tblKategorien.xml",
WhereCondition:="KategorieID = 1"
```

An dieser Stelle ein wichtiger Hinweis, falls Sie als **WhereCondition** einen Ausdruck verwenden wollen, der mit dem **LIKE**-Operator und Platzhaltern arbeitet – zum Beispiel diesen hier:

```
Application.ExportXML acExportTable, "tblKategorien",
CurrentProject.Path & "\tblKategorien.xml",
WhereCondition:="Kategoriename LIKE 'G*'"
```

Das Ergebnis ist ein XML-Dokument, das nur das Wurzelelement enthält:

```
<?xml version="1.0" encoding="UTF-8"?>
<dataroot xmlns:od="urn:schemas-microsoft-
com:officedata" generated="2019-07-31T15:45:01"/>
```

Der Grund ist, dass die **ExportXML**-Methode die Platzhalter verwendet, die vom SQL Server bekannt sind. Statt des Sternchens (*) verwenden Sie also das Prozentzeichen (%):

```
Application.ExportXML acExportTable, "tblKategorien",
CurrentProject.Path & "\tblKategorien.xml",
WhereCondition:="Kategoriename LIKE 'G%'"
```

Mit dieser Variante landen dann auch wie erwartet die drei Kategorien **Getränke**, **Gewürze** und **Getreide** im exportierten XML-Dokument.

Kategorie-XML-Export mit Transformation

Wie Sie die Daten aus Access-Tabellen mithilfe von Abfragen in die gewünschten Form bringen, wissen Sie bereits. Dazu eignen sich beispielsweise Auswahl- oder Kreuztabellenabfragen. Sie können aber auch die Inhalte von XML-Dokumenten ganz nach Ihren Wünschen strukturieren. Alles, was Sie benötigen, ist eine passende Transformation. Die notwendigen Informationen landen in einer XSL-Datei und beschreiben, wie die Daten des XML-Dokuments ausgegeben werden sollen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1903_KategorieXMLExportMitTransformation.zip**.

Einstiegsbeispiel

Im Artikel **XML: Austausch mit Access-Bordmitteln** haben wir einen Export in eine XML-Datei definiert, der die Daten der Tabellen **tblKategorien** und **tblArtikel** enthält. Das resultierende XML-Dokument enthält Elemente namens **tblKategorien**, die in Unterelementen die Feldinhalte der Felder dieser Tabelle enthalten sowie weitere Unterelemente namens **tblArtikel**, welche die Artikel der jeweiligen Kategorie inklusive aller Feldwerte auflistet. Das Ergebnis sieht in stark gekürzter Form etwa wie folgt aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<dataroot ...>
  <tblKategorien>
    <KategorieID>1</KategorieID>
    <Kategoriename>Getränke</Kategoriename>
    <Beschreibung>Alkoholfreie Ge ...</Beschreibung>
    <Abbildung>FRwvAAIAAAANAA4AFAAhAPAgS...</Abbildung>
    <tblArtikel>
      <ArtikelID>1</ArtikelID>
      <Artikelname>Chai</Artikelname>
      ...
    </tblArtikel>
    <tblArtikel>
      ...
    </tblArtikel>
    ...
  </tblKategorien>
  <tblKategorien>
    ...
  </tblKategorien>
</dataroot>
```

Wir möchten hier durch die Angabe einer XSL-Datei einige Änderungen an der Ausgabe erwirken. So soll etwa das Wurzelelement nicht **dataroot** heißen, sondern **Bestellverwaltung**. Die Elemente **tblKategorien** und **tblArtikel** sollen **Kategorie** und **Artikel** heißen. Außerdem soll der jeweilige Primärschlüsselwert nicht in einem eigenen Element namens **KategorieID** beziehungsweise **ArtikelID** landen, sondern in einem Attribut namens **ID** im jeweils übergeordneten Element **Kategorie** beziehungsweise **Artikel**. Außerdem wollen wir der Übersicht halber nur die Felder **Kategoriename** und **Beschreibung** der Tabelle **tblKategorien** sowie das Feld **Artikelname** der Tabelle **tblArtikel** als eigene Elemente in das XML-Dokument aufnehmen. Außerdem, das ist uns als Mensch im Gegensatz zur Maschine besonders wichtig, sollen die Elemente entsprechend der Hierarchie eingerückt dargestellt werden.

Das Ergebnis soll also etwa wie folgt aussehen:

```
<?xml version="1.0" encoding="UTF-16"?>
<Bestellverwaltung ...>
  <Kategorie ID="1">
    <Kategoriename>Getränke</Kategoriename>
    <Beschreibung>Alkoholfreie Getränke, Kaffee, Tee,
Bier</Beschreibung>
    <Artikel ID="1">
      <Artikelname>Chai</Artikelname>
    </Artikel>
    <Artikel ID="2">
```

```
      <Artikelname>Chang</Artikelname>
    </Artikel>
    ...
  </Kategorie>
  <Kategorie ID="2">
    ...
  </Kategorie>
  ...
</Bestellverwaltung>
```

Selektieren und Formatieren per XSL-Datei

Die XSL-Datei, die wir verwenden, sieht in der Übersicht wie in Listing 1 aus. Wir werden diese nun analysieren.

Es gibt grundsätzlich zwei verschiedene Typen von Zeilen: Diejenigen, die mit **xsl:** beginnen, enthaltenen Steueranweisungen. Diese werden ausgeführt, aber nicht in das Zieldokument übernommen. Die übrigen beginnen nicht mit **xsl:** und werden so, wie Sie in der XSL-Datei enthalten sind, in das Zieldokument geschrieben.

So landet die erste Zeile, die mit **<?xml** beginnt, direkt im Zieldokument. Die folgende Zeile enthält hingegen ähnlich wie die erste, für das Zieldokument bestimmte Zeile, Informationen wie etwa die XSL-Version.

Danach folgt die erste Anweisung, die eine Funktion ausführt. Sie enthält das Schlüsselwort **xsl:template**, was besagt, dass wir nun ein Template definieren, also eine Vorlage für den einzufügenden Inhalt. Das Attribut **match** gibt an, für welches Element unser Inhalt eingesetzt werden soll, in diesem Fall für das Element **dataroot**:

```
  <xsl:template match="dataroot">
```

Das heißt, wenn wir nun zwischen dem öffnenden und dem schließenden **xsl:template**-Element etwa nur das folgen-

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
Transform" xmlns="http://www.w3.org/TR/REC-html40">
  <xsl:template match="dataroot">
    <Bestellverwaltung>
      <xsl:text>
      </xsl:text>
      <xsl:for-each select="tblKategorien">
        <Kategorie>
          <xsl:attribute name="ID">
            <xsl:value-of select="KategorieID"/>
          </xsl:attribute>
          <xsl:text>
          </xsl:text>
          <Kategoriename>
            <xsl:value-of select="Kategoriename"/>
          </Kategoriename>
          <xsl:text>
          </xsl:text>
          <Beschreibung>
            <xsl:value-of select="Beschreibung"/>
          </Beschreibung>
          <xsl:text>
          </xsl:text>
          <xsl:for-each select="tblArtikel">
            <Artikel>
              <xsl:attribute name="ID">
                <xsl:value-of select="ArtikelID"/>
              </xsl:attribute>
              <xsl:text>
              </xsl:text>
              <Artikelname>
                <xsl:value-of select="Artikelname"/>
              </Artikelname>
              <xsl:text>
              </xsl:text>
            </Artikel>
            <xsl:text>
            </xsl:text>
          </xsl:for-each>
        </Kategorie>
        <xsl:text>
        </xsl:text>
      </xsl:for-each>
    </Bestellverwaltung>
  </xsl:template>
</xsl:stylesheet>
```

Listing 1: XSL-Datei zum Erreichen unserer Vorgaben

XML-Dokumente transformieren mit XSL

Im Artikel **Kategorie-XML-Export mit Transformation** haben wir an einem Praxisbeispiel gezeigt, wie Sie die Daten aus den beiden Tabellen Kategorien und Artikel exportieren und in eine andere Form bringen, als es der reine Export getan hätte. Im vorliegenden Artikel schauen wir uns nun an, welche Möglichkeiten die Transformation mit XSL grundsätzlich bietet und welche Rolle dabei die Sprache XPath spielt.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1903_XMLTransformieren.zip**.

Eine tragende Rolle für die Beispiele der folgenden Seiten spielt die Funktion **Transformieren** aus Listing 1. Diese Funktion erwartet drei per Parameter übergebene Werte: Den Pfad zur Quelldatei, den Pfad zur XSL-Datei und den Pfad der zu erstellenden Datei. Der vierte Parameter ist optional und liefert einen Text zu einem eventuell beim Parsen der beteiligten XML-Dokumente gefundenen Fehler zurück.

Mit dieser Funktion werden wir die folgenden Beispiele dieses Artikels durchführen. Sie lädt als erstes das XML-Dokument aus **strQuelle** mit der **Load**-Methode in das **DOMDocument**-Element **objQuelle**. Dann prüft sie, ob das Dokument einen Fehler enthält. Falls nicht, lädt sie das XSL-Dokument in die Variable **objXSL**.

Auch hier folgt eine Prüfung auf eventuelle Fehler beim Parsen der Datei. Erfolgt auch dies fehlerfrei, ruft die Prozedur die **transformNodeToObject**-Methode von **objQuelle** auf und übergibt **objXSL** als

```
Public Function Transformieren(strQuelle As String, strXSL As String, strZiel As String, Optional strFehler As String) As Long
    Dim objQuelle As MSXML2.DOMDocument
    Dim objXSL As MSXML2.DOMDocument
    Dim objZiel As MSXML2.DOMDocument
    Set objQuelle = New MSXML2.DOMDocument
    objQuelle.Load strQuelle
    If objQuelle.parseError = 0 Then
        Set objXSL = New MSXML2.DOMDocument
        objXSL.Load strXSL
        If objXSL.parseError = 0 Then
            Set objZiel = New MSXML2.DOMDocument
            objQuelle.transformNodeToObject objXSL, objZiel
            objZiel.Save strZiel
        Else
            Transformieren = objXSL.parseError.errorCode
            strFehler = ".xsl-datei: " & vbCrLf & strXSL & vbCrLf & objXSL.parseError.reason
        End If
    Else
        Transformieren = objQuelle.parseError.errorCode
        strFehler = "Quelldatei: " & vbCrLf & strQuelle & vbCrLf & objQuelle.parseError.reason
    End If
End Function
```

Listing 1: Die Prozedur **Transformieren**

ersten und **objZiel** als zweiten Parameter. Danach speichert sie das neu erstellte Objekt mit der **Save**-Methode des Objekts **objZiel**, und zwar unter dem Namen, der in der Variablen **strZiel** angegeben ist.

Sollte an irgendeiner Stelle ein Fehler beim Parsen aufgetreten sein, liefert

Für all dies ist ein Verweis auf die Bibliothek **Microsoft XML, v3.0** erforderlich. Es gibt zwar moderne Versionen dieser Bibliothek, aber wir verwenden diese, weil die XPath-Befehle damit besser funktionieren – mehr dazu im Artikel **XML-Dokumente mit XPath durchsuchen**.

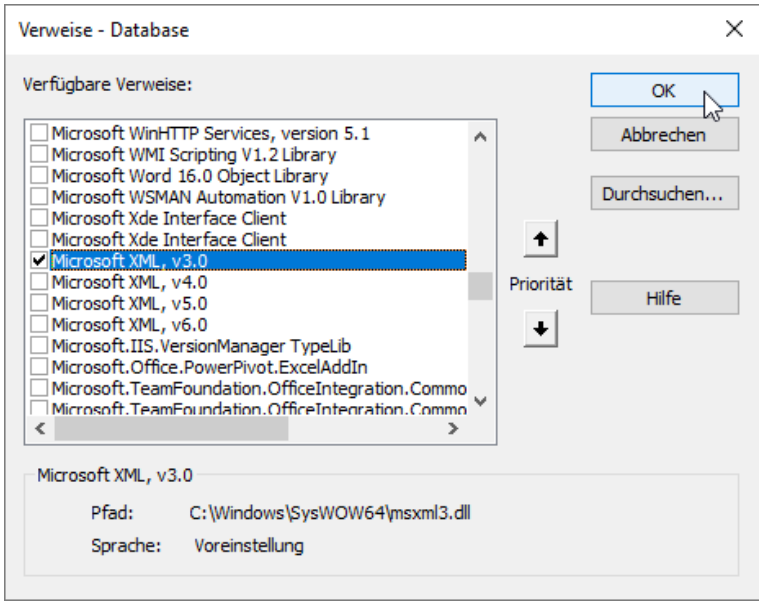


Bild 1: Hinzufügen eines Verweises auf die XML-Bibliothek

Diesen Verweis fügen wir über den **Verweise**-Dialog hinzu, den Sie im VBA-Editor mit dem Menüeintrag **Extras|Verweise** aktivieren (siehe Bild 1).

Wir gehen an dieser Stelle davon aus, dass Sie bereits eine Beispiel-XML-Datei erstellt haben, die im gleichen Verzeichnis wie die Datenbank liegt.

Fehler ausgeben

Um eventuell auftretende Fehler auszugeben, rufen wir die Prozedur **Transformieren** wie folgt auf und übergeben dabei noch die Variable **strFehler**:

```
Public Sub TransformierenMitMeldungen()
    Dim strFehler As String
    Transformieren CurrentProject.Path _
                   & "\KategorienUndArtikel.xml", _
                             CurrentProject.Path _
                   & "\KategorienUndArtikel.xsl", _
        CurrentProject.Path & "\Transformiert.xml", _
                                        strFehler
    Debug.Print strFehler
End Sub
```

Tritt hier ein Fehler auf, wird dieser mit dem Parameter **strFehler** an die aufrufende Prozedur zurückgegeben. Diese gibt ihn dann im Direktbereich des VBA-Editors aus.

Beispieldokument als Quelle

Als Beispiel verwenden wir wie in einigen anderen Artikeln den Export der Tabelle **tblKategorien**, für die wir zusätzlich die Tabelle **tblArtikel** hinzugefügt haben. Diese sieht in stark gekürzter Form wie folgt aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<dataroot xmlns:od="urn:schemas-microsoft-
com:officedata" generated="2019-08-01T21:45:19">
<tblKategorien>
<KategorieID>1</KategorieID>
<Kategoriename>Getränke</Kategoriename>
<Beschreibung>Alkoholfreie Getränke...</Beschreibung>
<Abbildung>FRwvAAIAAAANAA4AFAAhAP...</Abbildung>
<tblArtikel>
<ArtikelID>1</ArtikelID>
<Artikelname>Chai</Artikelname>
<LieferantID>1</LieferantID>
<KategorieID>1</KategorieID>
<Liefereinheit>10 Kartons x 20 Beutel</Liefereinheit>
<Einzelpreis>9</Einzelpreis>
<Lagerbestand>39</Lagerbestand>
<BestellteEinheiten>0</BestellteEinheiten>
<Mindestbestand>10</Mindestbestand>
<Auslaufartikel>0</Auslaufartikel>
</tblArtikel>
```

```
...
</tblKategorien>
...
</dataroot>
```

Im einführenden Beispiel des Artikels **Kategorie-XML-Export mit Transformation** haben wir aus diesem Dokument ein Dokument gemacht, das so aussieht, wie wir uns ein solches Dokument vorgestellt hätten, also etwa so:

```
<?xml version="1.0" encoding="UTF-16"?>
<Bestellverwaltung xmlns="http://www.w3.org/TR/REC-
html40">
  <Kategorien>
    <Kategorie ID="1">
      <Kategoriename>Getränke</Kategoriename>
      <Beschreibung>Alkoholfreie Getränke, Kaffee, 
                                Tee, Bier</Beschreibung>
      <Artikelliste>
        <Artikel ID="1">
          <Artikelname>Chai</Artikelname>
        </Artikel>
        ...
      </Artikelliste>
    </Kategorie>
    ...
  </Kategorien>
</Bestellverwaltung>
```

Für die komplette XSL-Datei, die wir hier verwendet haben, verweisen wir Sie auf den Artikel **Kategorie-XML-Export mit Transformation**.

Kopf einer XSL-Datei

Die erste Zeile einer XSL-Datei sieht immer wie folgt aus:

```
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns="http://www.w3.org/TR/REC-html40">
```

Dynamische und statische Anweisungen

Danach folgen einige weitere Anweisungen, die sich vor allem durch ein Merkmal unterscheiden:

Sie beginnen entweder mit **<xsl:** oder eben nicht. Wenn Sie schon einmal eine PHP- oder ASP-Seite programmiert haben, kennen Sie das Prinzip: Sie mischen dort HTML-Code und dynamische Skriptbestandteile. Der HTML-Code wird 1:1 in das resultierende HTML-Dokument übernommen, der PHP- oder ASP-Code fügt weitere, dynamisch ermittelte Inhalte etwa aus einer Datenbank hinzu oder strukturiert die Inhalte etwa durch den Einsatz von Schleifen oder Bedingungen. Genau das Gleiche finden Sie auch in XSL-Dokumenten vor. Der Unterschied ist, dass die Daten, die im Zieldokument abgebildet werden sollen, aus dem Quelldokument stammen und nicht etwa aus einer Datenbank.

Das template-Element als Wurzelelement

Genau wie ein XML-Dokument hat auch das XSL-Dokument ein Wurzelelement, nämlich das Element **template**. Dieses verlangt zwingend nach der Angabe des Parameters match. **match** gibt an, auf welches Element der Originaldatei sich die nachfolgenden Anweisungen beziehen beziehungsweise welches Element samt Unterelementen die Daten für das Zielelement liefern soll. Hier stellen wir **match** mit **"/"** auf das Wurzelelement ein. Außerdem wollen wir als Basis ein öffnendes und schließendes Element namens **Bestellverwaltung** anlegen:

```
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/">
      <Bestellverwaltung>
      </Bestellverwaltung>
   </xsl:template>
</xsl:stylesheet>
```

Das Ergebnis sieht dann so aus:

```
<?xml version="1.0" encoding="UTF-16"?>
<Bestellverwaltung></Bestellverwaltung>
```

Zeilenumbrüche

Hier wollen wir nun zunächst einen Zeilenumbruch einfügen, denn obwohl wir das öffnende und schließende **Bestellverwaltung**-Element in zwei Zeilen

geschrieben haben, wird der Zeilenumbruch nicht in das Ergebnis übernommen. Das gelingt nur, wenn das entsprechende Zeichen innerhalb eines öffnenden und schließenden Elements **xsl:text** eingefasst wird, also etwa so:

```
<Bestellverwaltung>
 <xsl:text>
 </xsl:text>
 </Bestellverwaltung>
```

Einzelne Elemente des Quelldokuments ausgeben

Sie können auf einzelne Elemente des Quelldocuments mit dem Element **xsl:value-of** zugreifen und dabei den Pfad zu dem Element mit dem Attribut **select** angeben.

Wenn sie etwa den Wert des ersten **KategorieID**-Elements unterhalb der Elemente **dataroot/tblKategorien** ausgeben wollen, verwenden Sie:

```
<xsl:value-of select="/dataroot/tblKategorien/Kategorie-
rieID"/>
```

Für das Feld **Kategoriename** nutzen Sie:

```
<xsl:value-of select="/dataroot/tblKategorien/Kategori-
ename"/>
```

Sie müssen also den kompletten Pfad vom Wurzelelement aus angeben.

```
<xsl:value-of select="/dataroot/tblKategorien/Kategorie-
rieID"/>
```

Wenn es Elemente des Typs **KategorieID** nur an einer Stelle gibt, können Sie auch gezielt auf das erste überhaupt verfügbare Element **KategorieID** zugreifen:

```
<xsl:value-of select="//KategorieID"/>
```

In diesem Fall bedeutet **//**, dass an beliebiger Stelle unabhängig von der Ebene gesucht wird.

Leerzeichen zwischen Elemente

Übrigens können Sie auch Leerzeichen zwischen zwei zusammengesetzte Textelemente nur mit **xsl:text** einfügen. Einfach ein Leerzeichen wie hier reicht nicht:

```
<xsl:value-of select="//KategorieID"/> <xsl:value-of
select="//Kategoriename"/>
```

Stattdessen fügen Sie das Leerzeichen in **xsl:text** eingefasst hinzu:

```
<xsl:value-of select="//KategorieID"/><xsl:text> </
xsl:text><xsl:value-of select="//Kategoriename"/>
```

Schleifen mit for-each

Mit **xsl:value-of** greifen Sie nur auf den ersten gefundenen Wert zu. Wenn Sie alle Werte mit diesem Pfad zugreifen wollen, etwa in einer Schleife, können Sie **xsl:for-each** verwenden.

Um beispielsweise alle Elemente des Typs **tblKategorie** des Beispieldokuments zu durchlaufen und wie oben einfach die **KategorieID** und den **Kategorienamen** auszugeben, verwenden Sie innerhalb des **template**-Elements zunächst ein Wurzelelement (**Kategorien** – ohne dieses Element hätte das Dokument mehrere Wurzelelemente, was nicht zulässig ist). Dann folgt ein Zeilenumbruch und das öffnende Element **xsl:for-each**. Der Wert für das Attribut **select** gibt an, welches Element durchlaufen werden soll, in diesem Fall die **dataroot/tblKategorien**-Elemente. Innerhalb der Schleife geben wir das öffnende **<Kategorie>**-Element, die Werte der Elemente **KategorieID** und **Kategoriename** und das schließende **</Kategorie>**-Element sowie einen Zeilenumbruch aus:

```
<Kategorien>
  <xsl:text>
  </xsl:text>
  <xsl:for-each select="dataroot/tblKategorien">
    <Kategorie>
      <xsl:value-of select="KategorieID"/>↵
                                <xsl:text> </xsl:text>↵
                <xsl:value-of select="Kategoriename"/>
```

```
    </Kategorie>
    <xsl:text>
    </xsl:text>
  </xsl:for-each>
</Kategorien>
```

Das ist das Ergebnis:

```
<?xml version="1.0" encoding="UTF-16"?>
<Kategorien>
  <Kategorie>1 Getränke</Kategorie>
  <Kategorie>2 Gewürze</Kategorie>
  <Kategorie>3 Süßwaren</Kategorie>
  ...
</Kategorien>
```

Wir haben vorher probiert, nur die Felder **KategorieID** und **Kategoriename** auszugeben, aber dann funktioniert der Zeilenumbruch nicht – dieser lässt sich offensichtlich nur zwischen zwei in spitze Klammern eingefasste Elemente einfügen.

Verschachtelte for-each-Schleifen

Innerhalb der **for-each**-Schleife können Sie beliebige Anweisungen einfügen, sogar weitere **for-each**-Schleifen. Im folgenden Beispiel haben wir zu jedem **Kategorie**-Element zunächst ein **Artikelliste**-Element hinzugefügt. Darin durchlaufen wir eine weitere **for-each**-Schleife, diesmal mit dem **select**-Attribut **tblArtikel**. Warum nicht **dataroot/tblKategorien/tblArtikel**?

Weil das Wurzelelement der **for-each**-Schleife immer das Element ist, in dem die Schleife ausgeführt wird – das ist in diesem Fall bereits das Element **tblKategorien**. Wir können uns also direkt auf die darin enthaltenen Elemente des Typs **tblArtikel** beziehen. Die Werte der darin enthaltenen Elemente ermitteln wir dann wie im vorherigen Beispiel mit der **value-of**-Funktion. Im folgenden Code haben wir aus Gründen der Übersicht die Code-Elemente für die Zeilenumbrüche entfernt:

```
<Kategorien>
  <xsl:for-each select="dataroot/tblKategorien">
    <Kategorie>
      <Kategoriedaten>
        <xsl:value-of select="KategorieID"/>
                              <xsl:text> </xsl:text>
              <xsl:value-of select="Kategoriename"/>
      </Kategoriedaten>
      <Artikelliste>
        <xsl:for-each select="tblArtikel">
          <Artikel><xsl:value-of select="ArtikelID"/>
                              <xsl:text> </xsl:text>
          <xsl:value-of select="Artikelname"/></Artikel>
        </xsl:for-each>
      </Artikelliste>
    </Kategorie>
  </xsl:for-each>
</Kategorien>
```

Das Ergebnis sieht dann etwa wie folgt aus:

```
<?xml version="1.0" encoding="UTF-16"?>
<Kategorien>
  <Kategorie>
    <Kategoriedaten>1 Getränke</Kategoriedaten>
    <Artikelliste>
      <Artikel>1 Chai</Artikel>
      <Artikel>2 Chang</Artikel>
      <Artikel>24 Guaraná Fantástica</Artikel>
      ...
    </Artikelliste>
  </Kategorie>
  ...
</Kategorien>
```

Die Informationen, die bisher im Element **Kategoriedaten** landen, sollen nun in zwei Elementen namens **KategorieID** und **Kategoriename** abgelegt werden. Dazu ersetzen wir das Element **Kategoriedaten** durch den folgenden Ausdruck:

```
<KategorieID><xsl:value-of select="KategorieID"/></KategorieID>
... Leerzeile ...
<Kategoriename><xsl:value-of select="Kategoriename"/></Kategoriename>
```

Das liefert dann diese Ausgabe:

```
<Kategorie>
  <KategorieID>2</KategorieID>
  <Kategoriename>Gewürze</Kategoriename>
</Kategorie>
```

Schleife über untergeordnete Elemente

Wir haben nun die **tblKategorie**-Elemente durchlaufen und die **tblKategorie**- und die **tblArtikel**-Elemente in zwei verschachtelten **for-each**-Schleifen. Sie können auch direkt nur die **tblArtikel**-Elemente durchlaufen. Das gelingt etwa so:

```
<xsl:template match="/">
  <Artikelliste>
    <xsl:text>
    </xsl:text>
    <xsl:for-each
            select="dataroot/tblKategorien/tblArtikel">
      <Artikel>
        <xsl:value-of select="ArtikelID"/>
        <xsl:text> </xsl:text>
        <xsl:value-of select="Artikelname"/>
      </Artikel>
      <xsl:text>
      </xsl:text>
    </xsl:for-each>
  </Artikelliste>
  <xsl:text>
  </xsl:text>
</xsl:template>
```

Wichtig hierbei ist, dass Sie im **select**-Attribut der **for-each**-Schleife nicht direkt **tblArtikel** referenzieren, sondern **dataroot/tblKategorien/tblArtikel**.

Das liegt daran, dass wir uns aktuell im Kontext des Wurzelelements befinden und nicht mehr, wie innerhalb der Schleife über die Kategorien aus dem vorherigen Beispiel, im Kontext eines **tblKategorie**-Elements.

Das Ergebnis ist dann eine solche Artikelliste:

```
<?xml version="1.0" encoding="UTF-16"?>
<Artikelliste>
  <Artikel>1 Chai</Artikel>
  <Artikel>2 Chang</Artikel>
  <Artikel>24 Guaraná Fantástica</Artikel>
  ...
  <Artikel>7 Uncle Bob's Organic Dried Pears</Artikel>
</Artikelliste>
```

Durch die Sortierung nach den übergeordneten Kategorien im Quelldokument sind die Artikel nicht nach der **ID** sortiert. Wie Sie das hinbekommen, sehen wir uns weitern unten an.

Kommentare

Kommentare im XSL-Code können Sie auf zwei Arten herstellen – der Unterschied ist, ob der Kommentar im XSL-Code gelten oder in der erstellten Datei landen soll. Die erste Methode ist die unter HTML übliche Vorgehensweise, bei der Sie die auszukommentierenden Elemente in **<!--** und **-->** einfassen. Damit sorgen Sie dafür, dass die so eingefassten Anweisungen nicht berücksichtigt werden.

Die zweite Methode ist die Verwendung des **xsl:comment**-Elements:

```
<xsl:comment>
  ... auskommentierter Bereich
</xsl:comment>
```

Dies sorgt dafür, dass die Zeichen **<!--** und **-->** im resultierenden Dokument landen.

Attribute hinzufügen

Grundsätzlich können Sie alle Daten in Form von Elementen zu einem übergeordneten Element hinzufügen. Es gibt aber auch die Gelegenheit, Attribute zu verwenden.

Zuletzt haben wir die beiden interessanten Felder der **Kategorie**-Elemente wie folgt abgebildet:

```
<Kategorie>
  <KategorieID>2</KategorieID>
  <Kategoriename>Gewürze</Kategoriename>
</Kategorie>
```

XML-Dokumente mit VBA lesen

Wir haben nun in einigen Artikeln beschrieben, welche Möglichkeiten die Bordmittel von Access für das Lesen und Schreiben von XML-Dokumenten bieten. Richtig flexibel ist das ganze natürlich nicht beziehungsweise nur, wenn Sie mit XSL nachhelfen. Wenn es beim Einlesen von XML-Dokumenten richtig individuell werden soll, können Sie aber immer noch mit VBA arbeiten. Dazu benötigen Sie nur einen Verweis auf die Objektbibliothek mit den Befehlen zum Bearbeiten von XML-Dokumenten – und das Know-how aus diesem Artikel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1904_XMLLesenVBA.accdb**.

Voraussetzungen

Um per VBA auf XML-Dokumente zuzugreifen, brauchen Sie imgrunde noch nicht einmal eine spezielle Bibliothek. XML-Dokumente sind ja auch nichts anderes als einfache Textdateien, nur das ihre Inhalte durch die XML-Auszeichnungen auf bestimmte Art strukturiert sind.

Im Notfall können Sie sich also auch mit der **Open**-Anweisung und den Zeichenkettenfunktionen von VBA behelfen. Aber wir wollen es natürlich schon etwas komfortabler haben und nutzen daher die Bibliothek **Microsoft XML, 6.0**.

Diese fügen Sie im VBA-Editor über den Menüeintrag **Extras|Verweise** und den dann erscheinenden Dialog **Verweise** wie in Bild 1 hinzu.

Wenn Sie nun mit der Schaltfläche **F2** den Objektkatalog öffnen, können Sie nach der Auswahl des Eintrags **MSXML2** oben im Kombinationsfeld alle Elemente dieser Bibliothek einsehen (siehe Bild 2).

Beispieldokument

Wir wollen die Daten aus einem XML-Dokument mit einer Bestellung als Beispiel verwenden, das Sie auch auf der folgenden Microsoftseite finden können – für den Fall, dass Sie mit weiteren Beispieldokumenten experimentieren wollen:

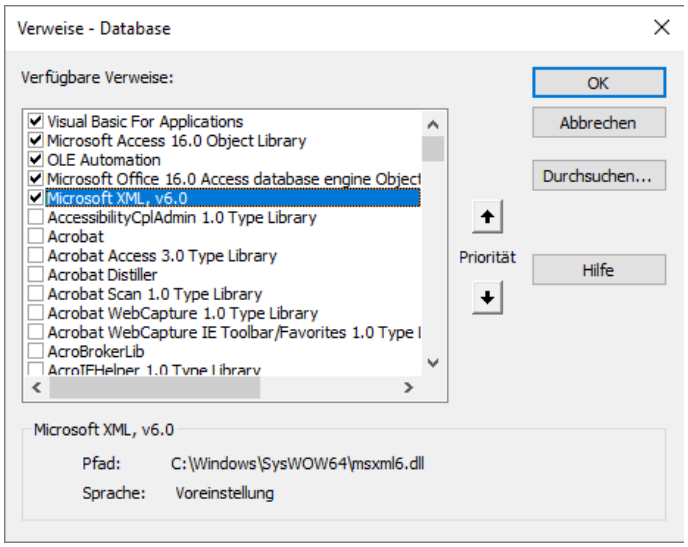


Bild 1: Verweis auf die XML-Bibliothek

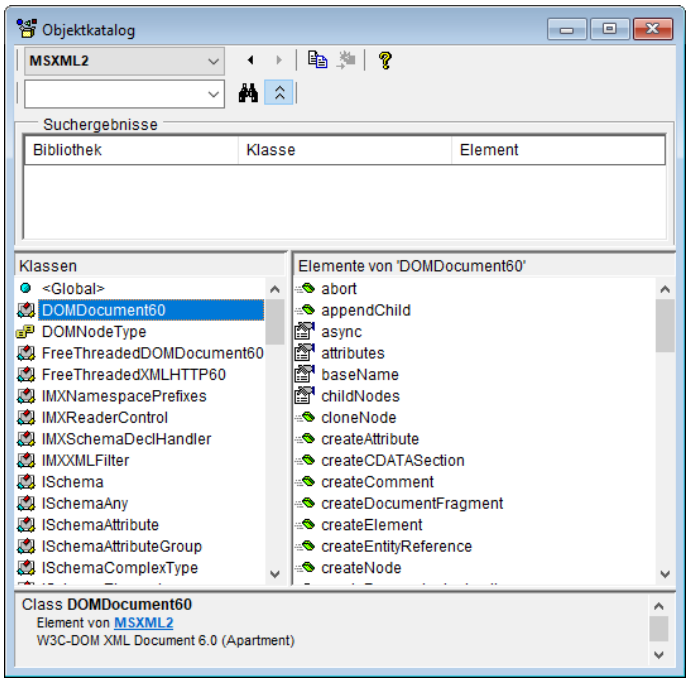


Bild 2: Objektkatalog von VBA

https://docs.microsoft.com/de-de/dotnet/visual-basic/programming-guide/concepts/linq/sample-xml-documents-linq-to-xml

Das Dokument sieht wie in Listing 1 aus. Es enthält genau eine Bestellung, die im Element **PurchaseOrder** enthalten ist. Das Hauptelement enthält in zwei Attributen die Bestellnummer und das Bestelldatum.

Darunter finden Sie die Liefer- und die Rechnungsadresse jeweils in **Address**-Elementen mit entsprechenden Unterelementen.

Ob es sich um eine Liefer- oder Bestelladresse handelt, entnehmen Sie dem Attribut **Type** des **Address**-Elements.

Darunter folgen **DeliveryNotes** als eigenes Element sowie eine Auflistung namens **Items** mit einigen **Item**-Elementen, welche die eigentlichen Bestellpositionen ausmachen. In den folgenden Abschnitten zeigen wir, wie Sie die dort enthaltenen Informationen per VBA und mithilfe der XML-Bibliothek auslesen können.

```
<?xml version="1.0"?>
<PurchaseOrder PurchaseOrderNumber="99503" OrderDate="1999-10-20">
  <Address Type="Shipping">
    <Name>Ellen Adams</Name>
    <Street>123 Maple Street</Street>
    <City>Mill Valley</City>
    <State>CA</State>
    <Zip>10999</Zip>
    <Country>USA</Country>
  </Address>
  <Address Type="Billing">
    <Name>Tai Yee</Name>
    <Street>8 Oak Avenue</Street>
    <City>Old Town</City>
    <State>PA</State>
    <Zip>95819</Zip>
    <Country>USA</Country>
  </Address>
  <DeliveryNotes>Please leave packages in shed by driveway.</DeliveryNotes>
  <Items>
    <Item PartNumber="872-AA">
      <ProductName>Lawnmower</ProductName>
      <Quantity>1</Quantity>
      <USPrice>148.95</USPrice>
      <Comment>Confirm this is electric</Comment>
    </Item>
    <Item PartNumber="926-AA">
      <ProductName>Baby Monitor</ProductName>
      <Quantity>2</Quantity>
      <USPrice>39.98</USPrice>
      <ShipDate>1999-05-21</ShipDate>
    </Item>
  </Items>
</PurchaseOrder>
```

Listing 1: Inhalt eines XML-Dokuments mit einer Bestellung

Die DOMDocument-Klasse

Um auf den Inhalt eines Dokuments zuzugreifen, wollen wir dieses zunächst referenzieren. Dazu gibt es genau eine geeignete Klasse, nämlich **DOMDocument60**.

Diese deklarieren wir mit der Variablen **objXML**, danach instanzieren wir ein neues Objekt auf Basis dieser Klasse mit dem **New**-Schlüsselwort.

Die **DOMDocument**-Klasse bietet unter anderem die beiden Methoden **Load** und **LoadXML** an:

- **Load**: Erwartet die Angabe des Pfades zum einzulesenden XML-Dokument als Parameter und liefert den Wert **True** zurück, wenn das Einlesen erfolgreich war.
- **LoadXML**: Erwartet ein XML-Dokument in Form einer Zeichenkette als Parameter. Auch hier liefert der Rückgabewert die Information, ob das Einlesen erfolgreich war.

Mit unserer Beispieldatei **PurchaseOrder.xml** sieht das Einlesen wie in der folgenden Prozedur aus. Hier prüfen wir direkt, ob das Einlesen erfolgreich war. Falls ja, geben wir den Inhalt des Dokuments im Direktbereich des VBA-Editors aus. Dazu verwenden wir die **Debug.Print**-Anweisung mit der **XML**-Eigenschaft von **objXML**.

```
Public Sub DokumentReferenzieren()
    Dim objXML As DOMDocument60
    Set objXML = New DOMDocument60
    If objXML.Load(CurrentProject.Path _
            & "\PurchaseOrder.xml") = True Then
        Debug.Print objXML.XML
    Else
        MsgBox "Fehler beim Einlesen"
    End If
End Sub
```

Fehler beim Einlesen

In unserem Beispiel gibt es keine Probleme beim Einlesen. Was aber könnte dabei nicht funktionieren? Es kann zum Beispiel vorkommen, dass das XML-Dokument nicht wohlgeformt ist.

Das bedeutet, dass zum Beispiel das öffnende Element **<PurchaseOrder>** vorhanden ist, aber das schließende Element fehlt. In diesem Fall liest **Load** das Dokument nicht ein, sondern liefert den Wert **False** zurück.

Um herauszufinden, welcher Fehler vorliegt, können Sie die Eigenschaften des Elements **ParseError** nutzen. Dazu fügen wir dem **Else**-Teil der **If...Then**-Bedingung aus der obigen Prozedur noch ein paar weitere Zeilen hinzu:

```
Debug.Print "Fehler bei Einlesen:"
Debug.Print "ErrorCode: " & objXML.parseError.errorCode
Debug.Print "Filepos:   " & objXML.parseError.filepos
Debug.Print "Line:      " & objXML.parseError.Line
Debug.Print "LinePos:   " & objXML.parseError.linepos
Debug.Print "Reason:    " & objXML.parseError.reason
Debug.Print "srcText:   " & objXML.parseError.srcText
Debug.Print "Url:       " & objXML.parseError.url
```

Wenn wir einmal das schließende **PurchaseOrder**-Element aus dem Dokument entfernen und die Prozedur erneut aufrufen, erhalten wir die Informationen aus Bild 3.

Die Meldung, welche die Eigenschaft **Reason** für uns bereithält, ist aussagekräftig genug, um den Fehler zu beheben.

Das Root-Element einlesen

Unser **objXML**-Objekt enthält nach dem Aufruf der **Load**-Methode, wie wir der **XML**-Eigenschaft entnehmen konnten, den kompletten Inhalt des XML-Dokuments. Wie können wir auf die einzelnen Elemente des Dokuments zugreifen und deren Inhalt ausgeben? Als Erstes wollen wir dabei das sogenannte Root-Element referenzieren, das in unserem Fall so aussieht:

```
<PurchaseOrder>
    ...
</PurchaseOrder>
```

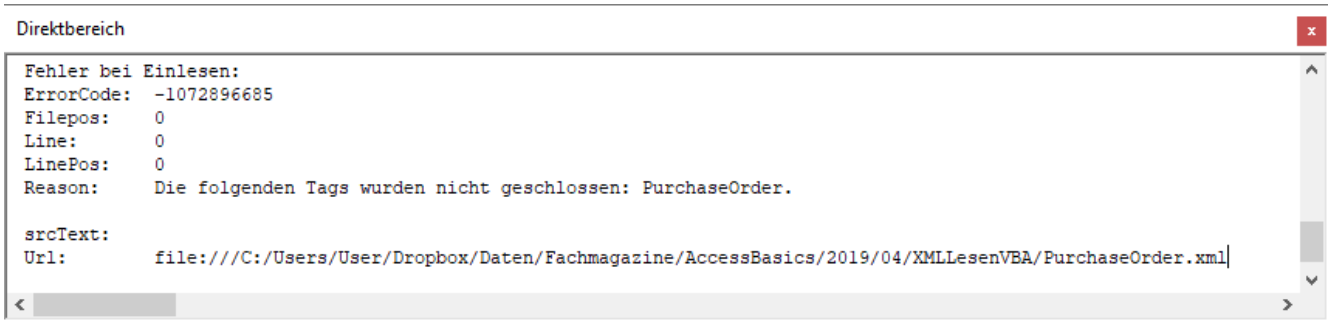


Bild 3: Fehlerinformationen des **parseError**-Objekts

Um auf ein einzelnes XML-Element innerhalb eines Dokuments zu verweisen, benötigen wir eine Objektvariable mit dem passenden Typ, in diesem Fall **IXMLDOMElement**:

```
Public Sub RootElementReferenzieren()
    Dim objXML As DOMDocument60
    Dim objRoot As IXMLDOMElement
    Set objXML = New DOMDocument60
```

Nach dem Laden des **documentElement**-Elements referenzieren wir das Root-Element dann über die **documentElement**-Eigenschaft:

```
    objXML.Load CurrentProject.Path ⤵
                                & "\PurchaseOrder.xml"
    Set objRoot = objXML.documentElement
```

Danach können wir beispielsweise mit **baseName** den Namen des Elements ermitteln, hier **PurchaseOrder**, oder mit der **XML**-Eigenschaft den Inhalt des XML-Elements ausgeben:

```
    Debug.Print objRoot.baseName
    Debug.Print objRoot.XML
End Sub
```

Für die **XML**-Eigenschaft des Root-Elements erhalten wir fast das gleiche Ergebnis wie für die des **DOMDocument**-Elements. Der einzige Unterschied in diesem Fall ist die fehlende **<?XML...>**-Zeile.

Untergeordnete Elemente durchlaufen

Das PurchaseOrder-Element enthält mehrere Unterelemente. Das ist eine gute Gelegenheit, einmal **IXMLDOMElement**-Objekte in einer Schleife zu durchlaufen. Dazu gibt es zumindest die beiden Möglichkeiten über eine **For...Next**- oder eine **For Each**-Schleife.

Die Prozedur zum Durchlaufen der direkten Unterelemente des **PurchaseOrder**-Elements sieht so aus:

```
Public Sub ElementePerForNext()
    Dim objXML As DOMDocument60
    Dim objRoot As IXMLDOMElement
    Dim objChild As IXMLDOMElement
    Dim i As Integer
    Dim intAnzahl As Integer
    Set objXML = New DOMDocument60
    objXML.Load CurrentProject.Path ⤵
                                & "\PurchaseOrder.xml"
    Set objRoot = objXML.documentElement
    intAnzahl = objRoot.childNodes.length
    For i = 0 To intAnzahl - 1
        Set objChild = objRoot.childNodes.Item(i)
        Debug.Print objChild.baseName
    Next i
End Sub
```

Wir ermitteln hier zunächst die Anzahl der direkten Unterelemente des **PurchaseOrder**-Elements. Dazu nutzen wir die **length**-Eigenschaft der **childNodes**-Auflistung des Elements. Der Index der mit **Item** referenzierbaren Elemente ist nullbasiert, sodass wir die **For...Next**-Schleife von **0** bis **intAnzahl - 1** laufen lassen.

Innerhalb der Schleife referenzieren wir dann das jeweilige Element über den Index mit der Objektvariablen **objChild** und geben dessen Eigenschaft **baseName** im Direktfenster aus. Das Ergebnis sieht dann wie folgt aus:

```
Address
Address
DeliveryNotes
Items
```

Das Gleiche können wir auch über eine **For Each**-Schleife realisieren. Dabei verwenden wir mit **objChild** eine Variable des Typs **IXMLDOMElement** als Laufvariable und geben auch hier den Inhalt der Eigenschaft **baseName** im Direktbereich aus, was zum gleichen Ergebnis führt:

```
Public Sub ElementePerForEach()
    Dim objXML As DOMDocument60
    Dim objRoot As IXMLDOMElement
    Dim objChild As IXMLDOMElement
```

Performance: Verbindung aktiv halten

Wenn Sie mit einer in Frontend- und Backenddatenbank aufgeteilten Anwendung arbeiten, kann es sein, dass die Performance bei verschiedenen Operationen schlecht ist – zum Beispiel beim Öffnen von Formularen, die an die Tabellen im Backend gebunden sind. Um dies zu ändern, gibt es einen Trick. Welcher Trick das ist, erläutern wir in diesem Artikel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in den Datenbanken **1904_Frontend.accdb** und **1904_Backend.accdb**.

Performancefresser

Wenn Sie eine in Frontend- und Backenddatenbank aufgeteilte Anwendung nutzen, befinden sich die Tabellen der Anwendung im Backend und die Abfragen, Formulare, Berichte und so weiter im Frontend.

Das Frontend greift auf die Tabellen im Backend über Verknüpfungen zu, die im Prinzip wie lokale Tabellen verwendet werden können.

Der erste wichtige Unterschied zwischen lokalen Tabellen (also Tabellen, die direkt im Frontend gespeichert sind – zum Beispiel, wenn kein Backend vorhanden ist) und der Verwendung von verknüpften Tabellen ist folgender: Eine Datenbank legt immer, wenn Sie geöffnet wird, eine **.ldb**- oder **.laccdb**-Datei an. Das geschieht bei der Datenbank, die ihre Tabelle lokal speichert, einmal beim Öffnen der Datenbank und beim Schließen der Datenbank verschwindet die Datei wieder.

Wenn eine Frontend-Datenbank über die Verknüpfung auf eine Backend-Datenbank zugreift, wird zusätzlich eine **.ldb**- oder **.laccdb**-Datei geöffnet. Diese wird gleich wieder geschlossen, wenn der Benutzer das Objekt in der Frontenddatenbank schließt, das mit der Tabelle im Backend verknüpft ist.

Welche Aufgabe erledigt diese **.ldb**- oder **.laccdb**-Datei aber nun? Wenn wir die Datei öffnen, finden wir darin eine Liste von Einträgen mit dem folgenden Format:

[Rechnername] [Benutzername]

Für jeden Benutzer mit einer aktuell geöffneten Verknüpfung auf die Backend-Datenbank wird ein Eintrag in dieser Datei angelegt.

Wenn nun mehrere Benutzer immer wieder Verknüpfungen zu den Tabellen öffnen und wieder schließen, erfolgen ständig Zugriffe auf die **.ldb**- beziehungsweise **.laccdb**-Datei.

Schnellerer Zugriff mit bestehender Verbindung

Der Zugriff auf die Daten einer Backenddatenbank sollte schneller erfolgen, wenn es bereits eine Verbindung zum Backend gibt, also wenn das Frontend bereits über eine Verknüpfung auf eine Tabelle im Backend zugreift. Wenn der Benutzer also ein Formular geöffnet hat und dann ein weiteres Formular öffnet, das Daten aus dem Backend anzeigt, sollte das Öffnen an dieser Stelle schneller erfolgen, als wenn es das erste Formular wäre, das geöffnet wird und eine Verbindung zum Backend herstellt.

Verbindung offenhalten

Nun könnte man dem Benutzer empfehlen, immer ein Formular offen zu halten, damit die anderen Formulare schneller geöffnet werden können. Aber so arbeiten wir nicht – eine solche Lösung sollte ohne Einbindung des Benutzers funktionieren. Außerdem kann es immer sein, dass der Benutzer dennoch alle Formulare der Anwendung schließt – und dann verzögert sich der Start des nächsten Formulars doch wieder.

Wie können wir aber eine Verbindung zum Backend offenhalten, wenn wir kein Formular dazu benutzen können? Eine Tabellenverknüpfung geöffnet halten?

Erweitern der VBA-IDE: Projekte, Module und Co.

Die Visual Basic-Entwicklungsumgebung ist definitiv in die Jahre gekommen. Wer schon einmal mit einer modernen Entwicklungsumgebung wie Eclipse oder dem Visual Studio .NET gearbeitet hat, wird bei der Rückkehr zur VBA-IDE möglicherweise einiges vermissen. Es gibt aber Möglichkeiten, die Entwicklungsumgebung zu erweitern – sogar mit den Hausmitteln von Access und mit nur einer zusätzlichen Bibliothek. Im ersten Teil der Artikelreihe schauen wir uns an, wie Sie auf die Objekte der VBA-Entwicklungsumgebung zugreifen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1904_EntwicklungsumgebungErweitern_Module.accdb**.

Einstieg

Genau wie auf die Objekte einer Access-Datenbank wie Tabellen, Abfragen, Formulare und Berichte sowie die darin enthaltenen Steuerelemente können Sie auch auf die Elemente des VBA-Editors zugreifen und diese einlesen, bearbeiten, anlegen, speichern oder löschen. In diesem ersten Teil der Artikelreihe zum Thema Erweitern der VBA-Entwicklungsumgebung schauen wir uns an, wie Sie mit VBA auf die Objekte der Entwicklungsumgebung zugreifen können.

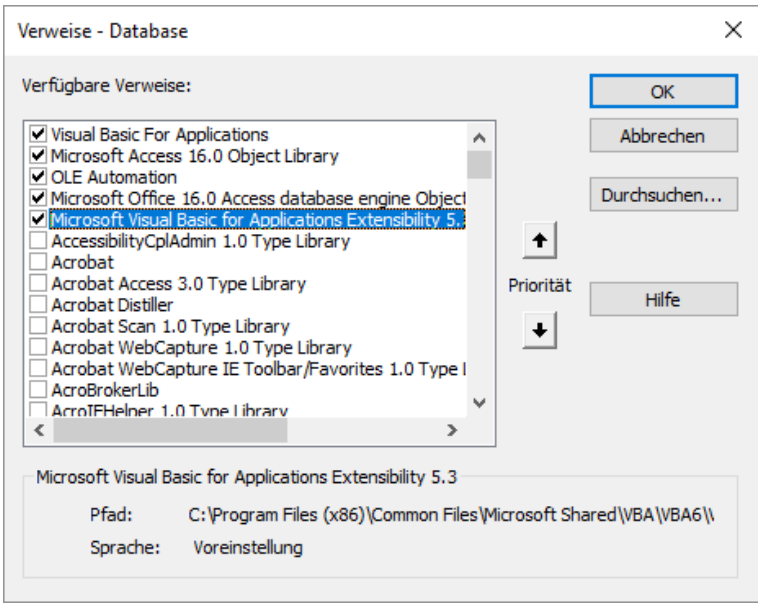


Bild 1: Der Verweis auf die Extensibility-Bibliothek für das Programmieren der VBA-Entwicklungsumgebung

Vorbereitung

Als Erstes benötigen Sie einen Verweis auf die Bibliothek, welche die Objekte, Methoden, Eigenschaften und Ereignisse für die Programmierung des VBA-Editors bereitstellt. Diesen fügen Sie im VBA-Editor hinzu, indem Sie den Menübefehl **Extras|Verweise** betätigen und im nun erscheinenden **Verweise**-Dialog den Eintrag **Microsoft Visual Basic for Applications Extensibility 5.3** hinzufügen (siehe Bild 1).

VBE: Das Hauptobjekt der Entwicklungsumgebung

In der soeben hinzugefügten Bibliothek finden Sie als Hauptelement die **VBE**-Klasse. Diese Klasse brauchen sie nicht zu instanzieren, Sie können direkt über den Ausdruck **VBE** darauf zugreifen. Das sehen Sie schon, wenn Sie im Direktbereich des VBA-Editors **VBE** und einen Punkt eingeben – IntelliSense zeigt dann direkt die Elemente dieser Klasse an (siehe Bild 2).

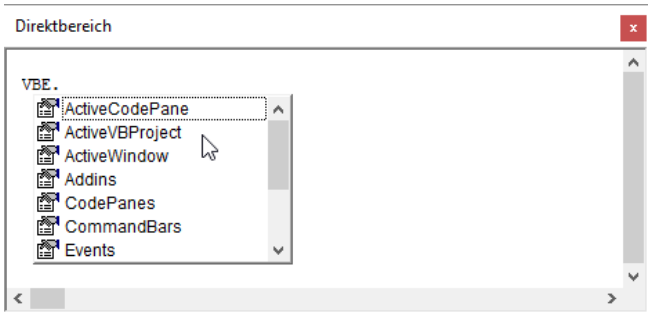


Bild 2: Zugriff auf die Eigenschaften und Methoden der **VBE**-Klasse

VBProject: Das VBA-Projekt

Damit können Sie dann auch direkt auf die Auflistung **VBProjects** zugreifen.

Warum aber gibt es eine Auflistung über die Projekte? Hat eine Access-Anwendung nicht immer nur ein VBA-Projekt, so wie in Bild 3 dargestellt?

Nein, das ist nicht der Fall: Sie können dort auch mehrere VBA-Projekte vorfinden. Das ist zum Beispiel der Fall, wenn Sie gerade ein Access-Add-In geladen haben. Dann sieht der Projekt-Explorer wie in Bild 4 aus.

Die einzelnen Projekte können Sie über die Auflistung VBProjects referenzieren, was etwa wie folgt aussieht:

```
Public Sub VBProjekteAusgeben()
    Dim objVBProject As VBProject
    Debug.Print "Anzahl VB-Projekte: " _
                                & VBE.VBProjects.Count
    For Each objVBProject In VBE.VBProjects
        Debug.Print objVBProject.Name
        Debug.Print objVBProject.FileName
        Debug.Print objVBProject.Type
    Next objVBProject
End Sub
```

Für das Beispiel mit dem zusätzlichen VBA-Projekt aus dem Add-In erhalten wir dann zum Beispiel die folgende Ausgabe:

```
Anzahl VB-Projekte: 2
prjVBEBeispiele
C:\...\1904_EntwicklungsumgebungErweitern_Module.accdb
 100
RA2010
C:\...\ADDINS\RIBBONADMIN2016.ACCDA
 100
```

Die Eigenschaftswerte finden Sie übrigens auch im Eigenschaftenfenster des Projekts wieder, das Sie über den Kontextmenü-Eintrag des Projekts öffnen können (siehe Bild 5).

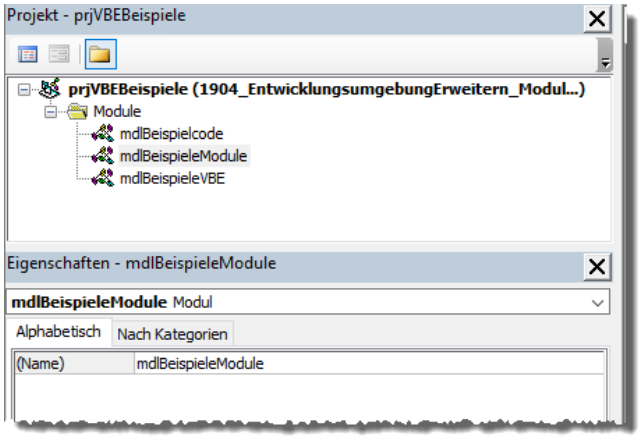


Bild 3: Das VBA-Projekt im Projekt-Explorer

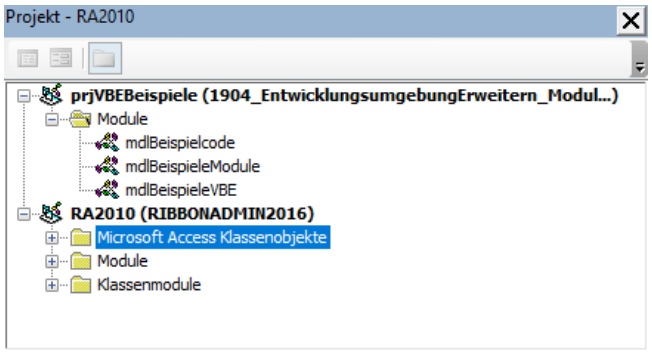


Bild 4: Mehrere VBA-Projekte im Projekt-Explorer

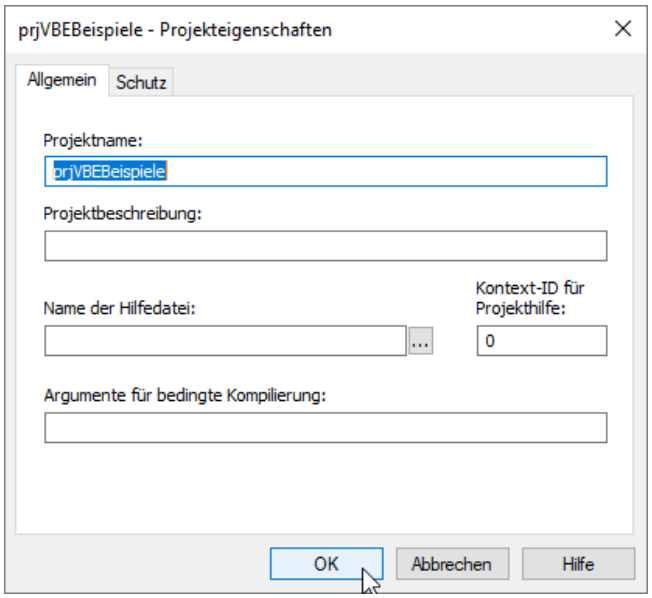


Bild 5: Eigenschaftsfenster des Projekts

Auf das aktive VBA-Projekt zugreifen

Wenn Sie das aktive VBA-Projekt referenzieren wollen, können Sie die Eigenschaft **ActiveVBProject** der **VBE**-Klasse verwenden. Wenn Sie die folgende Anweisung im Direktfenster des VBA-Editors ausgeben, erhalten Sie den Namen des VBE-Projekts, das gerade im Projekt-Explorer den Fokus hat:

```
Debug.Print VBE.ActiveVBProject.Name
```

Was geschieht, wenn Sie diese Anweisung in einer Prozedur unterbringen und diese dann aufrufen? Dann wird der Name des Projekts ausgegeben, in dem sich die Prozedur befindet:

```
Public Sub AktuellesProjektAusgeben()
    Debug.Print VBE.ActiveVBProject.Name
End Sub
```

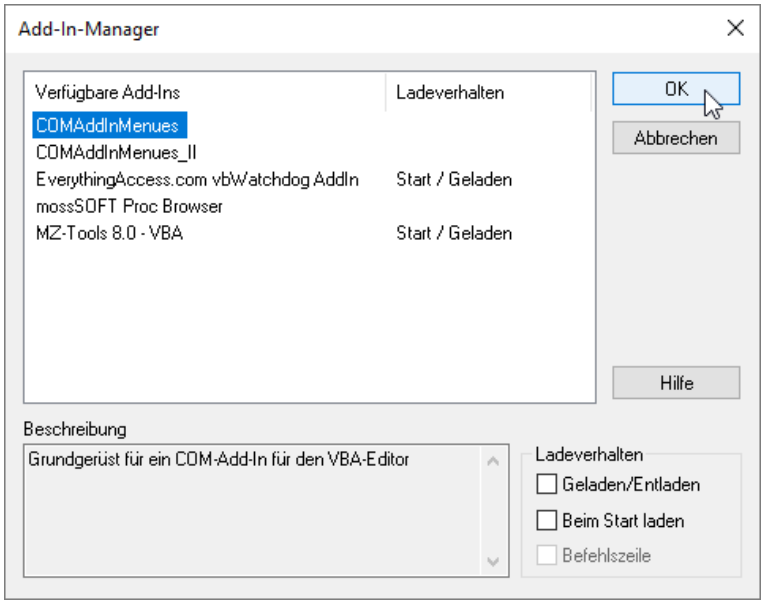


Bild 6: Der Add-In-Manager des VBA-Editors

Add-Ins ausgeben

Soeben haben wir schon angesprochen, dass das VBA-Projekt eines Add-Ins angezeigt wird, wenn dieses von der Benutzeroberfläche aus gestartet wurde beziehungsweise aktiv ist. Dabei handelt es sich um Access-Add-Ins. Es gibt aber auch noch Add-Ins für den Visual Basic Editor. Diese verwalten Sie über den Add-In-Manager des VBA-Editors, den Sie über den Eintrag **Add-Ins|Add-In-Manager...** öffnen.

Dieser zeigt eventuell vorhandene Add-Ins dann wie in Bild 6 in Listenform an. Sie können auch per VBA auf die vorhandenen Add-Ins zugreifen. Dazu verwenden Sie die Auflistung **Addins**, mit der Sie die Elemente des Typs **Addin** erhalten. Diese Auflistung können wir wie folgt per **For Each**-Schleife durchlaufen:

```
Public Sub AddInsAusgeben()
    Dim objAddIn As AddIn
    Debug.Print "Anzahl Addins: " _
                        & VBE.Addins.Count
    For Each objAddIn In VBE.Addins
        Debug.Print objAddIn.Description
    Next objAddIn
End Sub
```

Viele interessante Informationen liefert die **Addin**-Klasse jedoch nicht – wir erhalten noch nicht einmal den Namen, sondern nur die Beschreibung.

VBA-Projekt per Name ermitteln

Um auf Nummer sicher zu gehen, ob Sie mit dem richtigen **VBProject**-Element arbeiten, können Sie einen kleinen Trick anwenden. Es gibt zwar keine Möglichkeit, die Elemente der **VBProjects**-Auflistung über den Namen des Projekts anzusprechen, aber wir können diese ja durchlaufen und dabei prüfen, welchen Namen das aktuelle **VBProject**-Element hat. Um das VB-Projekt mit einem bestimmten Namen zu ermitteln, können wir die folgende Funktion verwenden:

```
Public Function GetVBProjectByName(strName As String) _
                                    As VBProject
    Dim objVBProject As VBProject
    For Each objVBProject In VBE.VBProjects
        If objVBProject.Name = strName Then
            Set GetVBProjectByName = objVBProject
            Exit Function
        End If
```

```
    Next objVBProject
End Function
```

Wenn Sie etwa eine Eigenschaft wie **FileName** des VBA-Projekts **prjVBEBeispiele** ermitteln wollen, können Sie das mit der obigen Funktion wie folgt erledigen:

```
Debug.Print GetVBProjectByName(_
                "prjVBEBeispiele").FileName
```

Verweise ausgeben

Auch die Verweise eines VBA-Projekts können Sie per VBA ausgeben lassen. Dazu nutzen Sie die **References**-Auflistung beziehungsweise die **Reference**-Klasse. Wichtig ist hierbei, in welchem Kontext Sie die **References**-Auflistung verwenden. Sie können diese beispielsweise im Direktfenster direkt aufrufen:

```
Debug.Print References.Count
```

Dann liefert das die Referenzen des aktuellen Projekts, wenn Sie nur ein Projekt geladen haben. Das aktuelle Projekt ist das Projekt, von dem das aktuelle Codefenster stammt. Wenn Sie mehr als ein Projekt geladen haben, sollten Sie dieses gezielt ansprechen. Wie das geht, haben Sie oben schon erfahren.

Wenn Sie wie in der Prozedur aus Listing 1 mit **ActiveVBProject** auf das aktuelle Projekt zugreifen, müssen Sie sicherstellen, dass eines der Elemente des gewünschten Projekts im Projekt-Explorer markiert ist – oder auch das Projekt selbst.

In der Prozedur durchlaufen wir in einer **For Each**-Schleife alle Elemente der **References**-Auflistung des aktuellen Projekts und geben verschiedene Eigenschaften des Verweises aus:

- **BuiltIn**: Gibt an, ob es sich um einen eingebauten Verweis handelt.
- **Description**: Beschreibung der Bibliothek

```
Public Sub VerweiseAusgeben()
    Dim objReference As vbide.Reference
    Dim objVBProject As VBProject
    Set objVBProject = VBE.ActiveVBProject
    For Each objReference In objVBProject.References
        Debug.Print "BuiltIn:     " & objReference.BuiltIn
        Debug.Print "Description: " & objReference.Description
        Debug.Print "FullPath:    " & objReference.FullPath
        Debug.Print "Guid:        " & objReference.Guid
        Debug.Print "IsBroken:    " & objReference.IsBroken
        Debug.Print "Major:       " & objReference.Major
        Debug.Print "Minor:       " & objReference.Minor
        Debug.Print "Name:        " & objReference.Name
        Debug.Print "Type:        " & objReference.Type
    Next objReference
End Sub
```

Listing 1: Ausgabe der Verweise und ihrer Eigenschaften

- **FullPath**: Pfad zur Datei, zum Beispiel **.DLL**
- **Guid**: Guid, unter der die Bibliothek registriert ist
- **IsBroken**: Gibt an, ob ein Verweis kaputt ist.
- **Major**: Hauptversionsnummer der Bibliothek
- **Minor**: Nebenversionsnummer der Bibliothek
- **Name**: Name der Bibliothek
- **Type**: Typ der Bibliothek (**vbext_rk_TypeLib** (**0**): TypeLibrary, **vbext_rk_Project** (**1**): Projekt, also zum Beispiel eine Access-Datenbank mit einem VBA-Projekt)

Verweise können Sie auch noch selbst hinzufügen und entfernen – das sprengt aber den Rahmen dieses Artikels und wir werden uns in einem weiteren Artikel um dieses Thema kümmern.

Wichtig ist noch, dass Sie immer **VBIDE** voranstellen sollten, also beispielsweise so:

```
Dim objReference As VBIDE.Reference
```

Berichte per VBA steuern

Berichte sind in vielerlei Hinsicht Formularen ähnlich, doch durch die fehlende Interaktion ist ihr Funktionsumfang auch geringer. Man sagt, die Steuerung per VBA biete daher bei Berichten auch nicht so viele Möglichkeiten. Das trifft nur begrenzt zu. Der vorliegende Artikel zeigt, welche Möglichkeiten die Steuerung von Berichten mit VBA offeriert.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1904_BerichtePerVBA.accdb**.

In der Beitragsreihe „Ereignisreiche Berichte" (ab Ausgabe 12/2005) haben Sie bereits erfahren, welche Ereignisse Berichte bereitstellen und für welche Zwecke sie sich einsetzen lassen. Neben diesen Ereignissen liefern Berichte einige Eigenschaften und Methoden, ohne die so manche Aufgabe nicht zu lösen wäre. Im vorliegenden Beitrag erfahren Sie, worum es sich dabei dreht und wie Sie die Eigenschaften und Methoden verwenden.

Steuerelemente und Berichtsbereiche referenzieren

Wenn Sie den Wert eines Steuerelements lesen oder schreiben möchten, greifen Sie wie in Formularen darauf zu: entweder über die entsprechende Auflistung, inklusive Objektname, oder, wenn die Referenz aus dem gleichen Objekt heraus erfolgt, mit dem Schlüsselwort **Me**.

Für die Variante mit dem Objektnamen gibt es mehrere Möglichkeiten, unter anderem die folgenden:

```
Reports.Item("rptBeispiel").Controls("ctlBeispiel")
Reports!rptBeispiel!ctlBeispiel
```

Mit dem Schlüsselwort **Me** sieht der Zugriff etwa folgendermaßen aus:

```
Me!ctlBeispiel
```

Auch die einzelnen Berichtsbereiche kann man referenzieren. Das macht Sinn, wenn man beispielsweise einen Bereich ein- oder ausblenden oder eine Eigenschaft ändern möchte. Der Verweis auf einen Berichtsbereich erfolgt über die **Section**-Auflistung eines Berichts. Dabei kommen für die Standardbereiche Zahlenwerte oder Konstanten in Frage, während man die Gruppenköpfe und -füße nur mit Zahlenwerten referenzieren kann. Den Detailbereich machen Sie etwa mit folgenden Anweisungen unsichtbar:

```
Me.Section(0).Visible = False
Me.Section(acDetail).Visible = False
```

Für die übrigen Bereiche verwenden Sie die folgenden Konstanten:

- Berichtskopf: **acHeader** (**1**)
- Berichtsfuß: **acFooter** (2)
- Seitenkopf: **acPageHeader** (3)
- Seitenfuß: **acPageFooter** (4)

Die Zahlenwerte für die einzelnen Gruppierungsbereiche sehen folgendermaßen aus: Der Kopfbereich der ersten Gruppierungsebene erhält den Zahlenwert **5**, der entsprechende Fußbereich den Zahlenwert **6**. Die Kopf- und Fußbereiche der übrigen Gruppierungsebenen werden entsprechend durchnummeriert.

Neben der Konstanten und dem Zahlenwert gibt es eine weitere Möglichkeit, einen Berichtsbereich zu referenzieren. Dabei handelt es sich um den Wert der Eigenschaft **Name** des jeweiligen Bereichs.

Den Namen können Sie nach Ihren eigenen Wünschen anpassen, die standardmäßig vergebenen Namen reichen allerdingsmeist aus (siehe Bild 1).

Über den Namen können Sie auf einen Bereich so zugreifen, als ob es sich dabei um ein Steuerelement des Berichts handelt. Die Bereichsnamen werden sogar von IntelliSense unterstützt (siehe Bild 2).

Die Routine aus Listing 1 zeigt, wie Sie die Namen und Höhen aller vorhandenen Berichtsbereiche im Direktfenster ausgeben können.

Das Ergebnis dieser Prozedur nach dem Öffnen eines Berichts sieht etwa wie folgt aus:

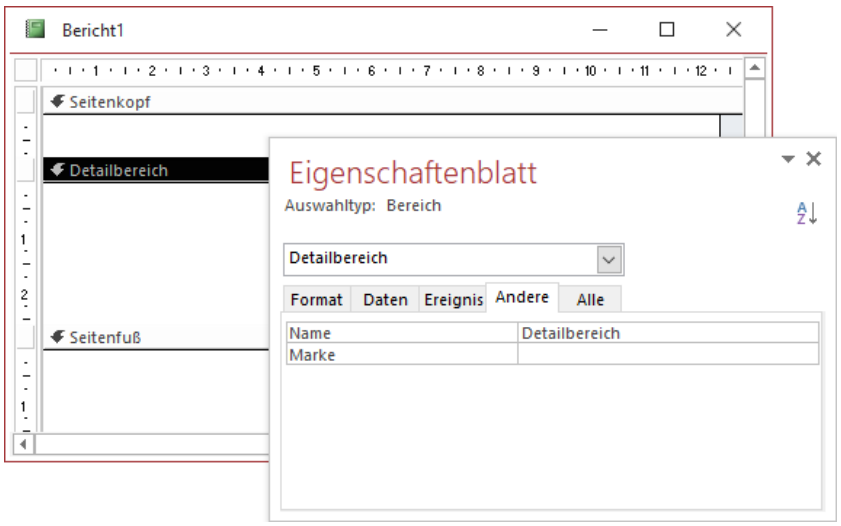


Bild 1: Der Name der einzelnen Bereiche eines Berichts kann angepasst werden.

```
Name des Bereichs: Detailbereich
Höhe des Bereichs: 1474
Name des Bereichs: Seitenkopfbereich
Höhe des Bereichs: 453
Name des Bereichs: Seitenfußbereich
Höhe des Bereichs: 1134
```

Auch die Gruppierungsebenen eines Berichts bieten eine weitere Möglichkeit des Zugriffs, nämlich über die **GroupLevel**-Auflistung.

Die folgende Anweisung gibt etwa den Namen des Feldes aus, nach dem die erste Gruppierungsebene gruppiert wird:

```
Debug.Print Me.GroupLevel(0).ControlSource
```

Doch Vorsicht: Die Gruppierungsebenen sind keinesfalls mit den Berichtsbereichen identisch und haben völlig andere Eigenschaften und Methoden.

Gruppierungen im Griff

Schauen Sie sich zunächst die Eigenschaften und Methoden der Gruppierungsebenen an.

Die Eigenschaften sind weitgehend mit denen aus dem Bereich **Gruppieren, Sortieren und Summe** identisch. Unter VBA

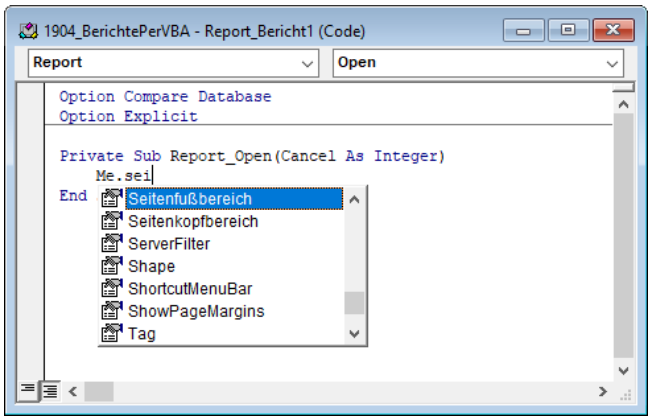


Bild 2: Referenzieren eines Berichtsbereichs über den Bereichsnamen per IntelliSense

stehen die folgenden Eigenschaften zur Verfügung, wobei einige im Anschluss ausführlicher besprochen werden müssen:

```
Private Sub Report_Open(Cancel As Integer)
    Dim i As Integer
    On Error Resume Next
    For i = 0 To 24
        Debug.Print "Name des Bereichs: " & Me.Section(i).Name
        Debug.Print "Höhe des Bereichs: " & Me.Section(i).Height
    Next i
End Sub
```

Listing 1: Ausgabe der Berichtsbereiche und deren Höhe

- **ControlSource**: Gibt das Feld an, nach dem sortiert oder gruppiert werden soll.
- **GroupHeader**: Schreibgeschützte Eigenschaft, die angibt, ob die Gruppierung einen Gruppenkopf enthält.
- **GroupFooter**: Schreibgeschützte Eigenschaft, die angibt, ob die Gruppierung einen Gruppenfuß enthält.
- **GroupOn**: Entspricht der Eigenschaft **Gruppieren nach**. Die Eigenschaft kann verschiedene Werte annehmen, die vom Datentyp des unter **ControlSource** angegebenen Feldes abhängen.
- **Groupinterval**: Entspricht der Eigenschaft **Intervall**. Die Eigenschaft kann verschiedene Zahlenwerte angeben, die das Aussehen des Intervalls bestimmen.
- **KeepTogether**: Legt fest, ob die Datensätze und Gruppenkopf/-fuß, wenn möglich, auf einer Seite gedruckt werden sollen.
- **SortOrder**: Entspricht der Eigenschaft **Sortierreihenfolge**. Enthält den Wert **False** (aufsteigende Sortierung) oder **True** (absteigende Sortierung).

Die Eigenschaften **GroupOn** und **Groupinterval** hängen vom Wert der Eigenschaft **ControlSource** beziehungsweise vom Datentyp des dort angegebenen Feldes ab.

Wenn Sie etwa nach einem Textfeld gruppieren, kann die Eigenschaft **GroupOn** die Werte **0** (gruppiert nach dem kompletten Feldwert) oder **1** (gruppiert nach den ersten x Zeichen) annehmen, wobei x dem Wert der Eigenschaft **GroupInterval** entspricht.

```
Private Sub Report_Open(Cancel As Integer)
    'Erste Gruppierungsebene referenzieren
    With Me.GroupLevel(0)
        'Zu gruppierendes Feld festlegen
        .ControlSource = "Artikelname"
        'Gruppieren nach einer bestimmten Anzahl von Zeichen...
        .GroupOn = 1
        '...die hier festgelegt wird:
        .GroupInterval = 1
        'Gruppenkopf und Detailbereich sollen zusammengehalten werden.
        .KeepTogether = 2
    End With
End Sub
```

Listing 2: Festlegen der Gruppierungseigenschaften eines Berichts

Wenn Sie also die Artikel-Tabelle der Südsturm-Datenbank so gruppieren möchten, dass alle Artikel mit dem gleichen Anfangsbuchstaben gruppiert und der Anfangsbuchstabe im Gruppenkopf angezeigt werden soll, realisieren Sie das unter VBA mit der Ereignisprozedur aus Listing 2.

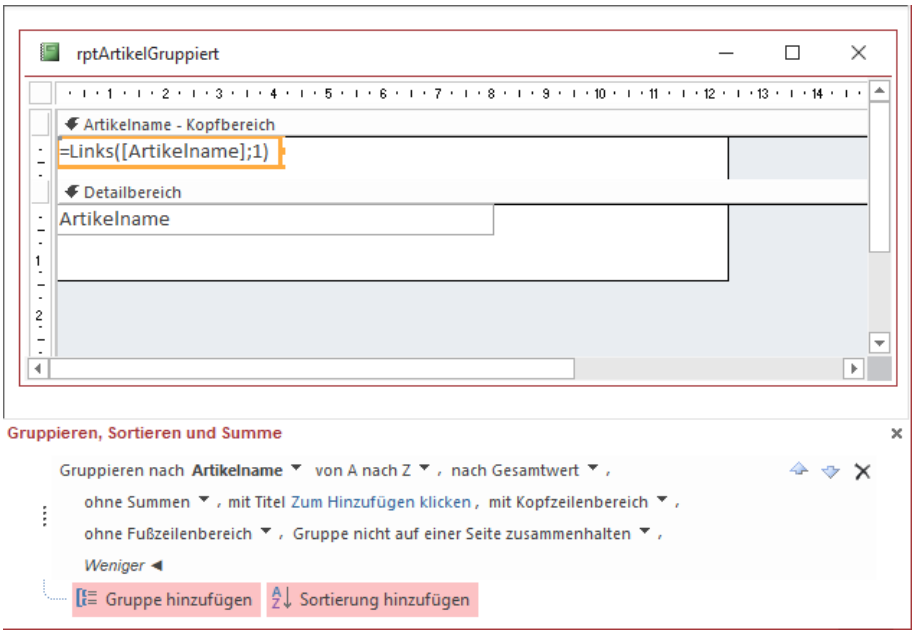


Bild 3: Entwurf eines Berichts zur Ausgabe eines Artikel-Katalogs.

Um das wie in der Entwurfsansicht der Abbildung zu realisieren, haben wir die Seitenkopf- und Seitenfuß-Bereiche ausgeblendet. Außerdem haben wir der Eigenschaft **Datensatzquelle** die Tabelle **tblArtikel** zugewiesen. Die Gruppierung legen wir im Bereich **Gruppieren, Sortieren und Summe** an, der unten im Access-Fenster erscheint, wenn wir im Ribbon die Schaltfläche **Entwurf|Gruppierung und Summen|Gruppieren und Sortieren** aktivieren. Hier klicken Sie auf **Gruppe hinzufügen** und wählen dann unter **Gruppieren nach** das Feld **Artikelname** aus. Dem Gruppenkopf weisen wir ein Textfeld mit dem Steuerelementinhalt **=Links([Artikelname];1)** zu, was jeweils den ersten Buchstaben des Artikelnamen liefert. Außerdem haben wir dem Detailbereich ein Textfeld zur Anzeige des Feldes **Artikelname** hinzugefügt. Der zugrunde liegende Bericht sieht dann im Entwurf wie in Bild 3 aus.

Das Ergebnis können Sie in Bild 4 betrachten. Was ist nun die Auswirkung der Ereignisprozedur **Beim Öffnen**, die wir weiter oben angelegt haben? Diese legt weitere Eigenschaften der Gruppierung fest. **GroupOn = 1** und **GroupInterval = 1** sorgen dafür, dass nach dem ersten Zeichen gruppiert wird, also nach dem Anfangsbuchstaben. KeepTogether = 2 sorgt dafür, dass Gruppenkopf und Detailbereich zusammengehalten werden, also dass immer mindestens ein Datensatz auf den Gruppenkopf folgt und diese nicht durch einen Seitenumbruch getrennt werden.

Für die anderen Datentypen sieht das im Prinzip ganz ähnlich aus. Enthält das unter **ControlSource** angegebene Feld einen Datums-/Zeitdatentyp, sorgt der Wert **0** für die Eigenschaft **GroupOn** ebenfalls dafür, dass die Gruppierung nach dem ganzen Wert erfolgt. Sie können aber auch nach den im Datum enthaltenen Bestandteilen sortieren, also etwa nach dem Jahr. Dazu verwenden Sie für die Eigenschaft **GroupOn** die folgenden Zahlenwerte:

- **2**: Jahr
- **3**: Quartal

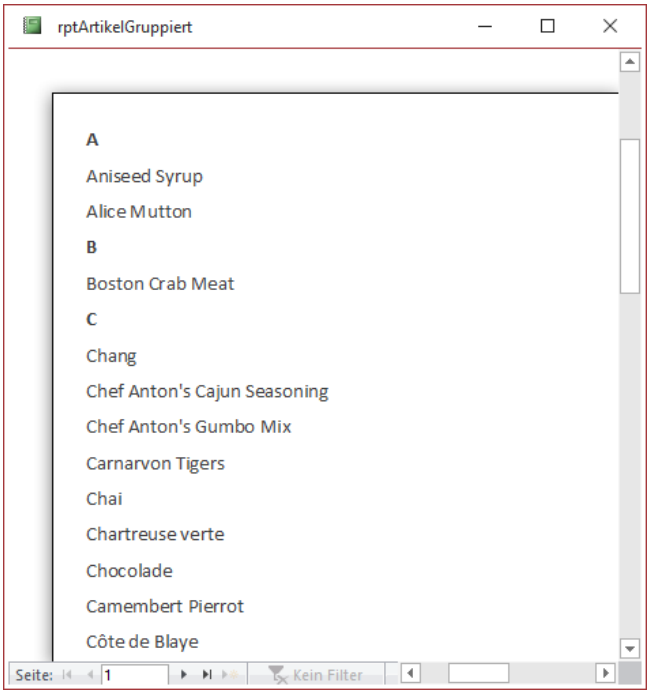


Bild 4: Bericht mit alphabetischer Artikeliste

- **4**: Monat
- **5**: Woche
- **6**: Tag
- **7**: Stunde
- **8**: Minute

Der Clou ist, dass Sie nicht nur nach den genannten Einheiten gruppieren können, sondern auch nach deren Vielfachen. Sie können also auch jeweils 30 Minuten zusammenfassen.

Dazu legen Sie einfach das gewünschte Vielfache mit der Eigenschaft **GroupInterval** fest.

Im folgenden Beispiel sorgen zwei Gruppierungen und eine Sortierung dafür, dass die Bestellungen der Südsturm-Datenbank nach Datum sortiert und nach Jahren und Monaten gruppiert werden – wobei jeweils zwei Monate zusammengefasst werden sollen.

Ligaverwaltung, Teil 1: Datenmodell

Für die meisten Sportarten gibt es Wettbewerbe in Ligen. Oder zumindest Turniere, die ja meist auch als kleine Liga angesehen werden können mit Spielpaarungen und Tabellen. Jedenfalls gibt eine Verwaltung solcher Ligen jede Menge Stoff für eine Datenbankanwendung – mit Datenmodell, Abfragen, Formularen zur Eingabe der Daten und Berichten, um Spielpläne und Tabellen auszugeben. Im ersten Teil der Artikelreihe schauen wir uns an, welche grundsätzlichen Anforderungen es gibt und wie das Datenmodell aussieht.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1904_Ligaverwaltung.accdb**.

Anforderungen

Was macht eigentlich eine Liga aus? Als Erstes haben wir eine Reihe von Mannschaften (wir gehen mal vom Mannschaftssport aus – natürlich kann eine Liga auch für Einzelsportler geschaffen werden). Diese Mannschaften spielen in der Regel jeweils zwei Mal gegeneinander, und zwar jeder gegen jeden. Das ist zumindest die am meisten verbreitete Form und wird etwa im Fußball so praktiziert. Beim Eishockey oder anderen Sportarten spielen die Mannschaften nicht nur zweimal gegeneinander, sondern mehrmals (zumindest in Deutschland).

Paarungen

Neben den Mannschaften brauchen wir also alle Paarungen der Mannschaften. Diese zusammenzustellen ist kein Problem. Mannschaft 1 spielt gegen Mannschaft 2, Mannschaft 3 und so weiter, Mannschaft 2 spielt gegen Mannschaft 3, Mannschaft 4 und so weiter und so fort, bis alle gegen alle auf dem Zettel stehen.

Das Problem ist, daraus einen Spielplan zu machen, bei dem an jedem Spieltag jede Mannschaft nur einmal spielt. Wenn man das etwa für 18 Mannschaften von Hand macht, kann das eine sehr zeitraubende Angelegenheit werden! Und wenn man dann noch in der Haut der Planer der Bundesliga steckt, die berücksichtigen müssen, dass Mannschaften maximal einmal pro Runde zwei Heim- oder zwei Auswärtsspiele hintereinander haben dürfen und Vereine, die nah beieinander liegen, nicht am gleichen Spieltag zuhause spielen dürfen, wird es richtig spannend.

Wir sollen aber maximal darauf achten, dass die Mannschaften so regelmäßig wie möglich abwechselnd Heim- und Auswärtsspiele haben. Was für unsere Spielpaarungen lediglich heißt: Die Mannschaften stehen abwechselnd an Position 1 und Position 2 einer Spielpaarung.

Spieltage

Neben den Mannschaften und Paarungen haben wir schon die dritte Einteilung erwähnt: den Spieltag.

Einen Spieltag definieren wir also als eine Einheit mit Spielpaarungen, an denen jede Mannschaft genau einmal vorkommt.

Ergebnisse

Meist endet eine Paarung mit einem Resultat, das in Toren oder Punkten angegeben wird, wobei die Anzahl dieser Tore oder Punkte auch noch über die Vergabe der eigentlich wichtigen Punkte entscheidet.

Jetzt wird es kompliziert, wenn man es Nicht-Sportlern erklären will. Beim Fußball (und bei anderen Sportarten, wo es um Tore geht – bei denen wir bleiben wollen) ist es einfach: Jede Mannschaft hat nach einem Spiel eine Anzahl von Toren erzielt. Die Mannschaft mit mehr Toren erhält drei Punkte, die andere keinen. Wenn beide gleichviel Tore erzielt haben, bekommt jede Mannschaft einen Punkt.

Tabellen

Die Punkte und die Tore geben an, wie die Reihenfolge der Mannschaften in der Tabelle aussieht. Das erste Kriterium sind die Punkte. Wenn zwei oder mehr Mannschaften gleichviele Punkte haben, wird die Tordifferenz herangezogen, also die Differenz der in allen bisherigen Spielen geschossenen Tore und der in allen Spielen erhaltenen Gegentore.

Ist auch die Tordifferenz gleich, wird die Anzahl der geschossenen Tore berücksichtigt. Ein Torverhältnis von 10:5 ist dann als besser als 9:4. Ist das Torverhältnis identisch, wird (meines Wissens) der direkte Vergleich zwischen den betroffenen Mannschaften herangezogen.

Wenn zwei Mannschaften die gleiche Anzahl Punkte haben und auch ihr Torverhältnis gleich ist, schaut man sich die beiden Spiele der Mannschaften gegeneinander an – sind auch die gleich, entscheiden Kriterien, die nicht mehr Bestandteil der Daten sind, die wir mit unserer Ligaverwaltung erfassen wollen.

Ohnehin werden nur am Ende der Saison Erbsen gezählt, wenn es um die Endplatzierung geht. Bis dahin belegen punkt- und torgleiche Mannschaften einfach den gleichen Platz.

Wollen wir die Tabellenstände eigentlich speichern oder berechnen wir diese immer zum aktuellen Zeitpunkt? Grundsätzlich wollen wir redundante Daten vermeiden.

Wenn wir also eine Tabelle, die sich allein aus den Ergebnissen der Mannschaften gegeneinander bis zum Zeitpunkt X errechnet, in einer Access-Tabelle speichern, und zwar beispielsweise nach jedem abgeschlossenen Spieltag, haben wir redundante Daten.

Allerdings könnte man aus den Tabellenständen an verschiedenen Spieltagen auch ableiten wollen, wie etwa die Entwicklung der Position einer Mannschaft über die Spieltage verläuft – und warum dann immer alle Spieltage neu berechnen?

Mehrere Ligen

Nun gibt es nicht nur eine Liga, sondern gleich eine ganze Menge – beim Fußball beginnend abwärts mit der zweiten Liga, der dritten Liga und dann geht es weiter mit jeweils mehreren Oberligen, Regionalligen und so weiter. Wir sollten also auch berücksichtigen, dass ein Benutzer nicht nur eine, sondern gleich mehrere Ligen in einer Datenbank verwalten möchte.

Mehrere Saisons

Die ganze Spielerei wäre nur halb so spannend, wenn nicht nach einem bestimmten Zeitraum der Beste der Liga gekürt würde – in der Bundesliga der Erste als Meister. Und auch einige der folgenden Positionen haben noch eine Bedeutung, denn damit kann man sich für internationale Wettbewerbe qualifizieren.

Und schließlich sollen nicht immer die gleichen Mannschaften in der gleiche Liga spielen, sondern die Mannschaften der unteren Ligen sollen auch die Gelegenheit erhalten, einmal in eine darüberliegende Liga aufzusteigen. Deshalb gibt es je nach Liga eine bestimmte Anzahl von Abstiegsplätzen und Aufstiegsplätzen.

Die Verwaltung mehrere Saisons hat übrigens auch noch den Charme, dass man Statistiken über mehrere Jahre aufstellen kann. Zum Beispiel könnte man sich ansehen, wie ein Verein in den letzten zehn Jahren abgeschnitten hat, wie die Paarungen zwischen zwei bestimmten Vereinen in der Historie ausgegangen sind et cetera.

Datenmodell

Aus diesen Informationen bauen wir nun ein Datenmodell. Die einfachste Tabelle ist die mit den Mannschaften. Diese braucht erst mal nur ein Primärschlüsselfeld sowie ein Feld, in dem man den Namen der Mannschaft speichern kann.

Allerdings kann eine solche Bezeichnung schon mal viele Zeichen enthalten. Um sicherzustellen, dass in einem Spielplan genügend Platz für eine Paarung zwischen zwei Mannschaften mit länglichen Namen ist, fügen wir noch ein Feld namens **MannschaftKurz**

hinzu, für das wir später noch eine maximale Zeichenanzahl festlegen wollen. Später deshalb, weil wir erst noch Formulare und Berichte erstellen müssen, um zu sehen, welche Zeichenanzahl angemessen ist.

Die Tabelle namens **tblMannschaften** sieht also in der Entwurfsansicht wie in Bild 1 aus. Wichtig ist hier, dass wir für die Felder **Mannschaft** und **MannschaftKurz** einen eindeutigen Index festlegen, damit es nicht zwei Mannschaften gleichen Namens gibt.

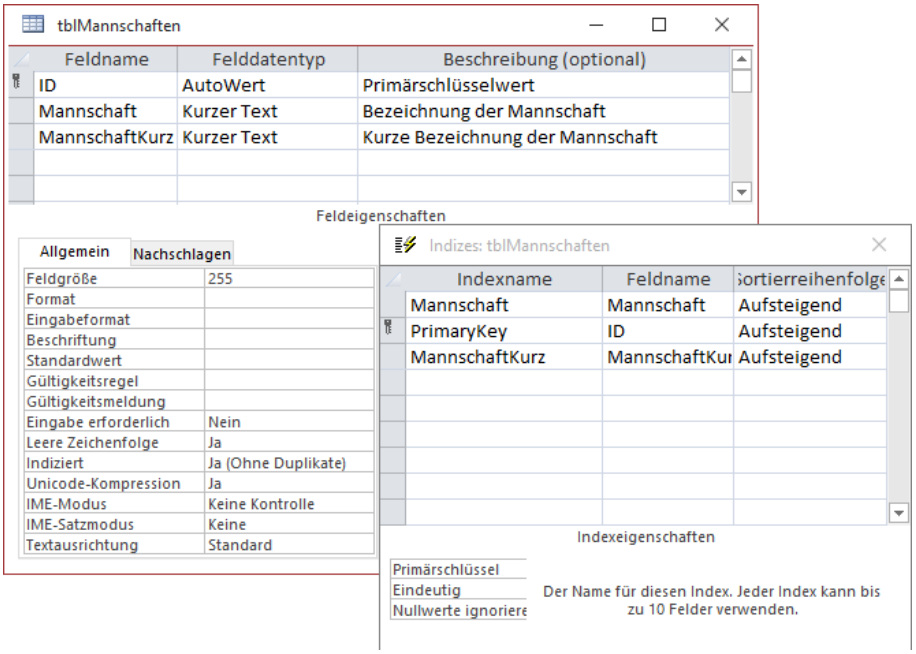


Bild 1: Entwurf der Tabelle **tblMannschaften**

Ligen und Spieljahre

Ligen wollen wir in einer Tabelle namens **tblLigen** verwalten. Diese soll drei Felder enthalten:

- **ID**: Primärschlüsselfeld
- **Liga**: Bezeichnung der Liga
- **Klasse**: Klasse der Liga als Zahlenwert

Für Liga soll dabei etwa **Bundesliga**, **2. Bundesliga** oder **3. Liga** angegeben werden, für Klasse der entsprechende Zahlenwert, also **1**, **2** und **3**. Auch hier soll das Feld **Liga** mit einem eindeutigen Index versehen werden.

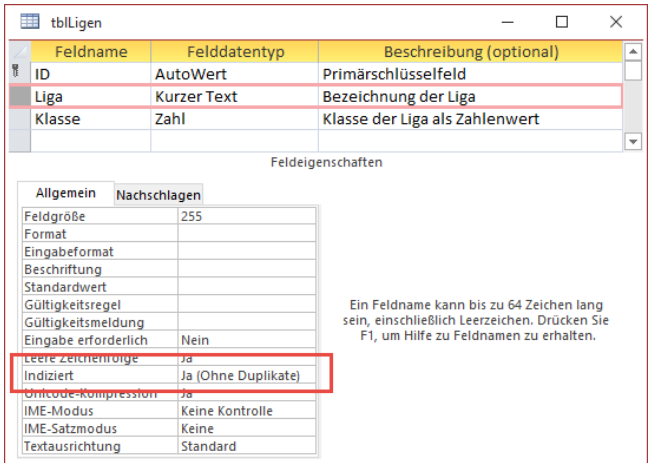


Bild 2: Entwurf der Tabelle **tblLigen**

Für das Feld **Klasse** benötigen wir keinen eindeutigen Index, da es hier sein kann, dass es in einer Klasse mehrere Ligen gibt (zum Beispiel **Regionalliga Nord**, **Regionalliga Nord-Ost** und so weiter). Der Entwurf dieser Tabelle sieht wie in Bild 2 aus.

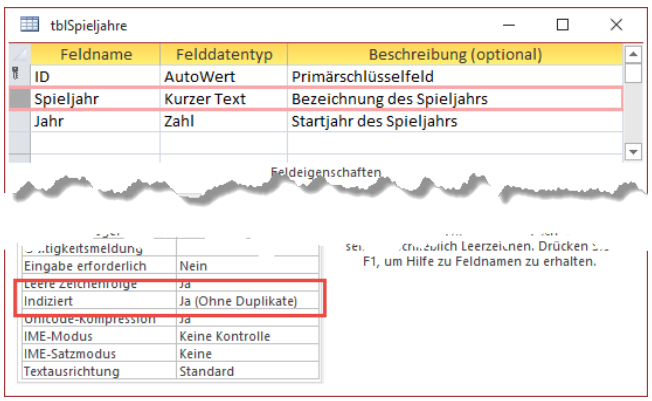


Bild 3: Entwurf der Tabelle **tblSpieljahre**

Zuordnung einer Mannschaft zu einer Liga?

Fehlt hier nicht schon eine Möglichkeit, eine Mannschaft einer Liga zuzuordnen? Zum Beispiel in Form eines Fremdschlüsselfeldes in der Tabelle **tblMannschaften**, mit dem wir die aktuelle Liga der Mann-

schaft auswählen können? Das wäre die einfache Lösung, aber sobald sie mehr als eine Saison betrachten wollen, gelingt das nicht mehr – wie legen Sie dann fest, zu welcher Liga eine Mannschaft in der vorherigen Saison gehört hat, wenn diese auf- oder abgestiegen ist? Für spätere statistische Auswertungen wäre dies ungünstig.

Wir brauchen also eine Zuordnung von Mannschaften zu Ligen, die auch das Spieljahr beziehungsweise die Saison berücksichtigt.

Und dazu müssen wir erst einmal die Spieljahre verwalten. Auch hier gibt es nämlich verschiedene Arten der Bezeichnung, zum Beispiel 2019/2020.

Um ein zuverlässiges Merkmal für die Sortierung zu haben, fügen wir noch ein Feld namens **Jahr** hinzu, in das das Jahr des Saisonstarts eingetragen werden soll, hier also **2019**. Dies spiegelt sich dann im Entwurf der Tabelle **tblSpieljahre** wie in Bild 3 wieder.

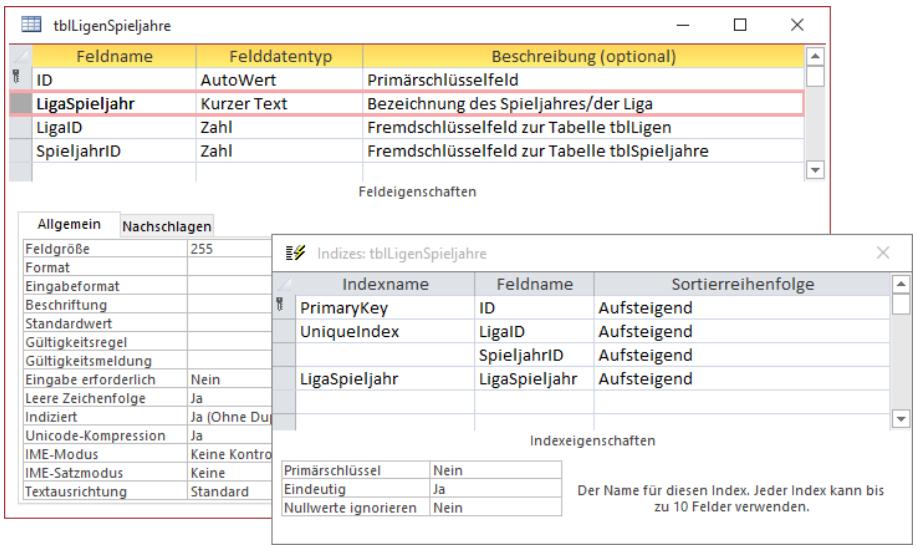


Bild 4: Entwurf der Tabelle **tblLigenSpieljahre**

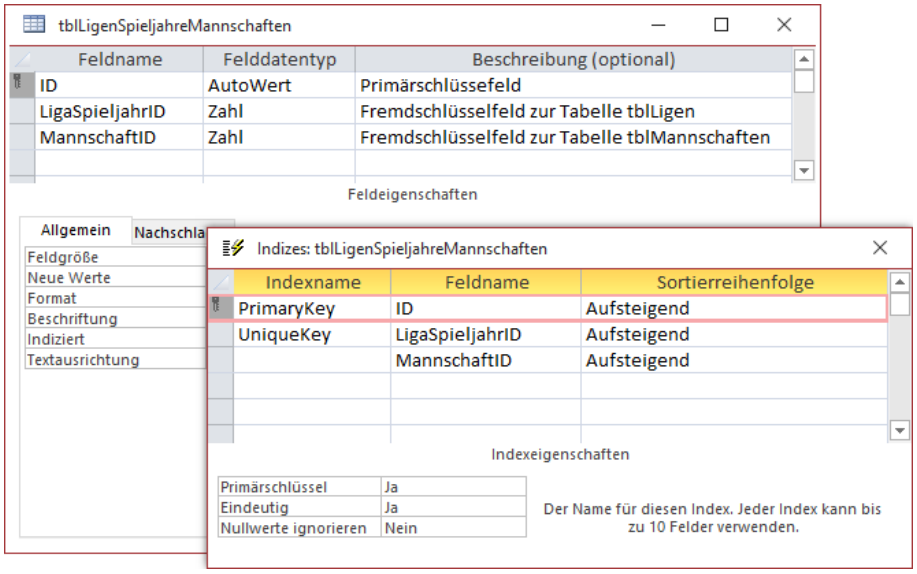


Bild 5: Entwurf der Tabelle **tblLigenSpieljahreMannschaften**

Nun können wir dann auch die Mannschaften mit den Ligen verheiraten und gleichzeitig die Spieljahre berücksichtigen – und zwar in einer Verknüpfungstabelle namens **tblLigenSpieljahreMannschaften**. Diese enthält ein Primärschlüsselfeld sowie drei Fremdschlüsselfelder zur Verknüpfung mit den Tabellen **tblLigen**, **tblSpieljahre** und **tblMannschaften**.

Nun stellt sich aber die Frage, welche Kombination diese Tabelle zulassen darf. Ich war erst versucht, alle drei Fremdschlüsselfelder zu einem eindeutigen Index zusammenzufassen. Aber dann könnte man eine Mannschaft beispielsweise innerhalb eines Spieljahres mit mehreren Ligen verknüpfen – und das ist rein aus Sicht des Datenmodells nicht optimal.

Besser wäre, erst einmal dafür zu sorgen, dass jede Kombination aus **LigaID** und **SpieljahrID** nur einmal vorkommt. Und dann wiederum jede Mannschaft nur mit einer diese Kombinationen zusammenführen zu können. Wie aber können wir das im Datenmodell umsetzen? Die einzige Möglichkeit ist es, eine Tabelle

mit den beiden Feldern **LigaID** und **SpieljahrID** zu erstellen, die einen eindeutigen Index für die Kombination dieser beiden Felder definiert.

Und dann fügen Sie die Mannschaften mit den Datensätzen dieser Tabelle zusammen, wobei es wieder einen eindeutigen Index über die beiden Fremdschlüsselfelder dieser Tabelle gibt.

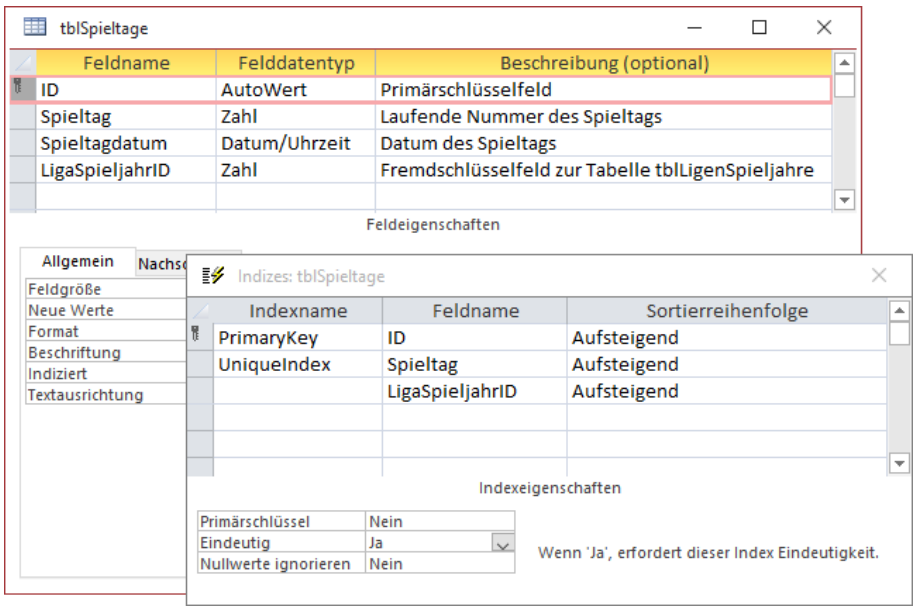


Bild 6: Entwurf der Tabelle **tblSpieltage**

Die erste Tabelle, die wir benötigen, heißt also **tblLigenSpieljahre**, enthält neben dem Primärschlüsselfeld **ID** noch die beiden Fremdschlüsselfelder **LigaID** und **SpieljahrID** und sieht im Entwurf wie in Bild 4 aus.

Außerdem haben wir dieser Tabelle noch ein Textfeld namens **LigaSpieljahr** hinzufügt, welches die Kombination aus Liga und Spieljahr beschreibt – also etwa **Bundesliga 2019/2020**. Für die beiden Felder **LigaID** und **SpieljahrID** legen wir einen zusammengesetzten, eindeutigen Schlüssel an.

Die zweite Tabelle heißt **tblLigenSpieljahreMannschaften** und enthält das Primärschlüsselfeld **ID** sowie die Fremdschlüsselfelder **LigaSpieljahrID** und **MannschaftID**, für die wiederum ein zusammengesetzter, eindeutiger Index definiert ist (siehe Bild 5).

Tabelle tblSpieltage

Damit kommen wir zur Tabelle **tblSpieltage**. Ein Spieltag soll mit einer Liga und mit einem Spieljahr verknüpft sein. Das ist praktisch, denn so können wir die Verknüpfung auch für die Spieltage gleich über die Tabelle **tblLigenSpieljahre** herstellen.

Außerdem erhält die Tabelle **tblSpieltage** natürlich eine laufende Nummer im Feld **Spieltag** sowie ein Datum im Feld **Spieltagdatum**. Benötigen wir wirklich ein Spieltagdatum? Finden nicht manchmal auch die Spiele eines Spieltags über mehrere Tage verteilt statt? Ja, und dennoch wollen wir das Datum des Hauptspieltags in dieser Tabelle festhalten.

Schließlich fügen wir auch dieser Tabelle noch einen zusammengesetzten Index hinzu, diesmal über die beiden Felder **Spieltag** und **LigaSpieljahrID**.

Denn: Jede Spieltagnummer darf pro Liga und Spieljahr nur einmal vorkommen (siehe Bild 6).

Tabelle tblSpielpaarungen

Nun kommt wieder eine interessante Stelle: Wir wollen eine Tabelle zum Speichern der Spielpaarungen hinzufügen. Die folgenden Felder sind uns klar:

- **Spieldatum**: Datum, an dem das Spiel stattfindet.
- **HeimmannschaftID**: Fremdschlüsselfeld zur Auswahl der Heimmannschaft aus der Tabelle **tblMannschaften**
- **AuswaertsmannschaftID**: Fremdschlüsselfeld zur Auswahl der Auswärtsmannschaft aus der Tabelle **tblMannschaften**

Zoomfenster: Einsatz und Neuerungen

Wenn Sie in der Benutzeroberfläche von Access mit Ausdrücken etwa in Eigenschaftsfeldern von Formularen und Steuerelementen oder auch mit berechneten Ausdrücken im Entwurfsraster des Abfrageentwurfs arbeiten, könnte es etwas unkomfortabel werden, wenn der Ausdruck nicht komplett im jeweiligen Textfeld zu sehen ist. Zur etwas komfortableren Bearbeitung hat Microsoft unter Access das Zoomfenster vorgesehen, das sie bei aktiviertem Steuerelement mit der Tastenkombination Umschalt + F2 öffnen können. Allerdings konnte man bisher seine Größe nicht ändern. Das hat sich mit dem aktuellen Update von Access geändert – das Zoomfenster kann nun auch in der Größe angepasst werden. Das und die Grundlagen zum Zoomfenster lesen Sie in diesem Artikel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1905_Zoomfenster.accdb**.

Einsatzbereiche des Zoomfensters

Das Zoomfenster können Sie prinzipiell überall dort aufrufen, wo Text in einem Textfeld angezeigt wird. Und das gilt sowohl für die Eigenschaften von Tabellen, Abfragen, Formularen und Berichten in der Entwurfsansicht als auch für die Eigenschaften von Steuerelementen. Alles, was Sie tun müssen, ist das Betätigen der Tastenkombination **Umschalt + F2**. Danach erscheint dann schon das Zoom-Fenster und zeigt den Inhalt des aktuell markierten Steuerelements an – in Bild 1 am Beispiel des Feldnamens einer Eigenschaft.

Wenn Sie den Inhalt im Zoom-Fenster ändern, wird dieser nach dem Schließen des Fensters mit der **OK**-Schaltfläche in das Textfeld übernommen, für das das Zoomfenster aufgerufen wurde.

Schriftart einstellen

Wenn Sie auf die Schaltfläche **Schriftart...** klicken, erscheint der Dialog zur Auswahl der Schriftart aus Bild 2. Hier können Sie die gewünschte Schriftart, Schriftgröße und weitere Eigenschaften einstellen. **Courier New** bietet sich als Schriftart an, weil man mit dem

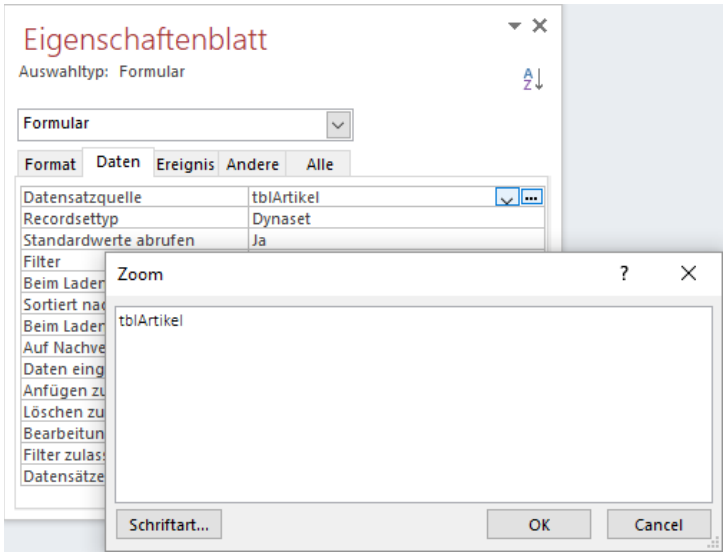


Bild 1: Formular-Entwurf unseres Beispielformulars

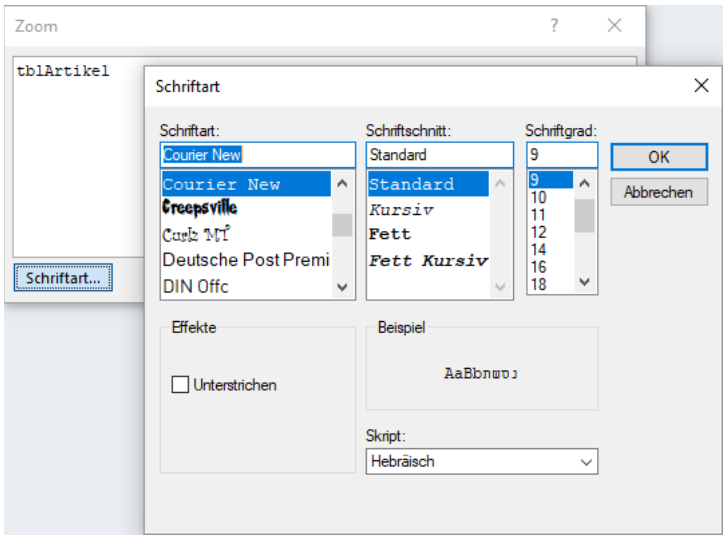


Bild 2: Einstellen der Schriftart des Zoomfensters

Feldinhalte aufteilen, Teil I

Wenn sie Daten etwa aus Excel-Tabellen oder anderen Quellen importieren, liegen diese nicht immer in der gewünschten Form vor. Dann sind Felder wie Vorname und Nachname in einem Feld zusammengefasst, oder Sie möchten vielleicht Straße und Hausnummer trennen und auf zwei Felder aufteilen. Wie das mit möglichst wenig manuellem Aufwand gelingt, zeigen wir in diesem Artikel am Beispiel von Straße und Hausnummer.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1905_FeldinhalteAufteilen.accdb**.

Wo und wie aufteilen?

Bevor wir uns überhaupt an eine Tabelle heranwagen, die Daten zum Aufteilen enthält, machen wir uns mit dem Handwerkzeug vertraut, das wir dazu benötigen – und das heißt in diesem Fall Zeichenkettenfunktionen. Schauen wir uns zunächst ein einfaches Beispiel an – eine Straße mit einer Hausnummer, wobei beide Informationen durch ein Leerzeichen getrennt sind:

```
Gerichtsstr. 27
```

Das ist ein einfacher Fall: Wir brauchen nur das Leerzeichen in der Zeichenkette zu suchen und den Teil davor als Straße und den dahinter als Hausnummer zu speichern.

Das erledigen wir mit folgendem Codeschnippsel (siehe Prozedur **StrasseUndHausnummer_1**). Wir deklarieren eine Variable für den Ausgangsausdruck sowie für die resultierenden Elemente:

```
Dim strStrasseUndHausnummer As String
Dim strStrasse As String
Dim strHausnummer As String
```

Dann legen wir den Startausdruck fest:

```
strStrasseUndHausnummer = "Gerichtsstr. 27"
```

Die Strasse ermitteln wir, indem wir das erste Auftreten eines Leerzeichens suchen und den Teil bis zu diese Leerzeichen extrahieren:

```
strStrasse = Left(strStrasseUndHausnummer, 
            InStr(1, strStrasseUndHausnummer, " ") - 1)
```

Bei der Hausnummer sieht es ähnlich aus, nur dass wir mit der **Mid**-Funktion hier den Teil nach dem Leerzeichen ermitteln:

```
strHausnummer = Mid(strStrasseUndHausnummer, InStr(1, 
                    strStrasseUndHausnummer, " ") + 1)
```

Danach geben wir beides aus:

```
Debug.Print "*" & strStrasse & "*"
Debug.Print "*" & strHausnummer & "*"
```

Die Sternchen geben wir hier aus, um sicherzustellen, dass das Leerzeichen nicht in einer der beiden Zeichenketten enthalten ist. Wir erhalten:

```
*Gerichtsstr.*
*27*
```

Das war der erste, naive Ansatz. Dieser funktioniert schon nicht mehr, wenn wir diese Straße untersuchen:

```
Borkhofer Str. 17
```

Hier erhalten wir dann:

```
*Borkhofer*
*Str. 17*
```

Hier haben wir schon ein Leerzeichen im Straßennamen. Wir müssen also nicht das erste, sondern das letzte Leerzeichen ermitteln. Dem werden wir in der nächsten Version gerecht (siehe Prozedur

StrasseUndHausnummer_2). Hier speichern wir die Position des letzten Leerzeichens auch direkt in einer Variablen:

```
Dim intLetztesLeerzeichen As Integer
strStrasseUndHausnummer = "Borkhofer Str. 17"
intLetztesLeerzeichen = InStrRev(7
                        strStrasseUndHausnummer, " ")
```

Damit ermitteln wir dann wie zuvor die beiden Teile des Strings:

```
strStrasse = Left(strStrasseUndHausnummer, 7
                          intLetztesLeerzeichen - 1)
strHausnummer = Mid(strStrasseUndHausnummer, _
                          intLetztesLeerzeichen + 1)
```

Das Ergebnis sieht nun wie gewünscht aus:

```
*Borkhofer Str.*
*17*
```

Testroutine schreiben

Aber klappt das auch noch für unser zuerst verwendetes Beispiel? Das können wir ja leicht und zuverlässig testen, indem wir aus unserer bisherigen Prozedur eine Funktion machen, diese mit verschiedenen Parameterwerten aufrufen und die Ergebnisse mit den erwarteten Ergebnissen abgleichen.

Wie das geht, haben wir im Detail im Artikel **VBA-Funktionen testen** beschrieben. Dort haben wir die Funktion **StrasseUndHausnummer** auch etwas umgestaltet, sodass diese nun drei Parameter enthält: den Eingangsparameter **strStrasseUndHausnummer** und die Ausgangsparameter **strStrasse** und **strHausnummer**. Wie unterscheiden sich Eingangs- und Ausgangsparameter? Auf den ersten Blick gibt es keinen sichtbaren Unterschied, was aber daran liegt, dass das Schlüsselwort ByRef, welches für die Verwendung von Ausgangsparametern zwingend erforderlich ist, standardmäßig verwendet wird. Die Funktion sieht nun im aktuellen Zustand wie in Listing 1 aus.

Hausnummer mit Leerzeichen

Wir sind nun soweit, dass wir Leerzeichen im Straßennamen nicht mit dem Leerzeichen zwischen Straße und Hausnummer verwechseln. Der nächste Schritt ist, dass wir Hausnummern erkennen, die selbst ein Leerzeichen enthalten. Das sollte normalerweise nicht der Fall sein, aber es kann sein, dass der Benutzer etwa folgende Zeichenkette eingibt:

```
Gerichtsst. 2 a
```

Mit der aktuellen Version unserer Funktion **StrasseUndHausnummer** erhalten wir für die Strasse nun den Wert **Gerichtsstr. 2** und als Hausnummer **a**. Wie können wir das noch korrekt behandeln? Wir könnten prüfen, ob die Zeichen vor dem letzten Leerzeichen numerisch sind und diese dann noch in die Hausnummer übernehmen. Wie würden wir das machen? Dieser Schritt würde nach den bisherigen Schritten stattfinden und wir würden den Inhalt von **strStrasse** separat untersuchen.

Wenn **strStrasse** auf numerische Zeichen endet, würden wir diese noch zur Hausnummer hinzufügen. Dazu fügen wir eine **Do While**-Schleife hinzu, die solange durchlaufen wird, bis das letzte Zeichen aus **strStrasse** kein numerisches Zeichen mehr ist. Damit das irgendwann geschieht, entfernen wir in jedem Schleifen-

```
Public Function StrasseUndHausnummer(strStrasseUndHausnummer As String, strStrasse As String, strHausnummer As String) As Boolean
    Dim intLetztesLeerzeichen As Integer
    intLetztesLeerzeichen = InStrRev(strStrasseUndHausnummer, " ")
    strStrasse = Left(strStrasseUndHausnummer, intLetztesLeerzeichen - 1)
    strHausnummer = Mid(strStrasseUndHausnummer, intLetztesLeerzeichen + 1)
    StrasseUndHausnummer = True
End Function
```

Listing 1: Funktion zum Ermitteln von Strasse und Hausnummer aus einer Zeichenkette

VBA-Funktionen testen

Wenn Sie VBA-Funktionen entwickeln, sollen diese reproduzierbare und für alle übergebenen Parameterwerte korrekte Ergebnisse liefern. Manch einer testet die Funktion dann für einen Wert, erhält das gewünschte Ergebnis, erhält für einen anderen Wert ein falsches Ergebnis, ändert die Funktion ... und: Irgendwann stellt sich heraus, dass diese nach den Änderungen nicht mehr das korrekte Ergebnis für den eingangs verwenden Parameter liefert. Hier hilft es, wenn Sie das Entwickeln von VBA-Funktionen etwas systematischer angehen, nämlich mit organisierten Tests. Dabei testen Sie die Funktion mit allen Parameterwerten, die Ihnen einfallen und wiederholen diese Tests auch nach Änderungen an der Funktion, um sicherzustellen, dass die Änderung der Funktion die Ergebnisse nicht beeinflusst. Wie das gelingt, zeigen wir in diesem Artikel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1905_VBAFunktionenTesten.accdb**.

Schnell mal programmiert

Eine VBA-Funktion ist mal mehr, mal weniger anspruchsvoll. Einfache Prozeduren, die vielleicht nur einen Datenbankaufruf kapseln sollen, brauchen kaum für viele verschiedene Fälle getestet zu werden. Aber je mehr verschiedene Eingabewerte eine Funktion hat und desto komplexer und umfangreicher die enthaltenen Anweisungen werden, umso wichtiger wäre es, während der Entwicklung der Funktion regelmäßig zu prüfen, ob diese für alle Fälle, für die sie vorgesehen ist, noch funktioniert.

Das ist insbesondere dann gefragt, wenn ein Eingabewert auftaucht, den die Funktion nicht in der gewünschten Weise verarbeitet – was bedeutet, dass sie nicht das erwartete Ergebnis liefert. Dann ändert man die Funktion so, dass sie auch auf die neuen Eingabewerte wie erwartet reagiert. Aber prüft man dann auch immer automatisch, ob die Eingabe- und Ausgabewerte, an die man die Funktion vorher angepasst hat, noch passen? Das ist vielleicht nicht der Fall, und dort liegt eine mögliche Fehlerquelle.

Anhand von Funktionen möchten wir in diesem Artikel aufzeigen, wie Sie automatisierte Tests von Code aufbauen können. Das erfordert zwar auf den ersten Blick zunächst mehr Programmieraufwand, aber wenn Sie den Code später anpassen wollen, sparen Sie viel Zeit, weil alle Tests, die Sie bis dahin durchgeführt haben, nun per Mausklick erledigt werden können.

Ein Beispiel ist eine Funktion aus dem Artikel **Feldinhalte aufteilen**. Hier haben wir eine Funktion programmiert, mit der wir die Bestandteile einer Adressangabe, in diesem Fall Straße und Hausnummer, auf zwei Zeichenketten mit Straße und Hausnummer aufteilen wollen. Die Funktion hat die folgende Kopfzeile:

```
Public Function StrasseUndHausnummer(_
        strStrasseUndHausnummer As String, strStrasse _
         As String, strHausnummer As String) As Boolean
```

Dabei erfasst der erste Parameter die zu verarbeitende Zeichenkette, der zweite und dritte liefert das Ergebnis der Funktion zurück. Der Rückgabewert gibt als **Boolean**-Wert an, ob die Funktion überhaupt ordnungsgemäß funktioniert hat.

Das ist eine Form, mit der wir beim automatisierten Testen gut arbeiten können. Wenn die Funktion nur einen Rückgabewert hat und nicht wie in diesem Fall gleich zwei beziehungsweise drei, können Sie den Rückgabewert auch direkt als Funktionswert zurückgeben.

Wir wollen die Funktion nun mit verschiedenen Aufrufen testen, wobei wir immer andere Kombinationen aus Straßenname und Hausnummer übergeben:

- Straße und Hausnummer mit Leerzeichen dazwischen,
- Straßen, die bereits Leerzeichen im Straßennamen enthalten,
- Straße und Hausnummer, bei denen zwischen dem Punkt von »Str.« und der Hausnummer kein Leerzeichen ist oder auch
- Straßen, bei denen die Hausnummer selbst Leerzeichen enthält (2 a).

Wenn wir eine Funktion erstellen wollen, die all diese Konstellationen wie gewünscht verarbeitet, müssen wir in der Funktion entsprechend viele Fälle berücksichtigen. Und um sicherzustellen, dass man beim Anpassen der Funktion auf einen neuen Fall auch die anderen Fälle noch berücksichtigt, sollte man alle bekannten Fälle nach jeder Änderung nochmals prüfen.

Ich schreibe bewusst von »allen bekannten Fällen«, denn der Benutzer ist gnadenlos und wird Möglichkeiten finden, um Ihre Funktion zum Versagen zu bringen. Doch das ist dann nur eine weitere Aufgabe, die es zu lösen gilt und das ist viel einfacher, wenn man wie nachfolgend beschrieben Testroutinen für die zu erstellende Funktion programmiert hat.

Einfach testen

Am einfachsten testet man, indem man die Funktion schlicht mit dem zu untersuchenden Parameterwert aufruft und die Rückgabewerte der Funktion auf Richtigkeit überprüft. Liefert die Funktion die erwarteten Werte zurück, braucht nichts weiter zu geschehen. Wenn diese jedoch andere Werte als erwartet liefert, sollte man durch eine Meldung oder durch eine Ausgabe im Direktbereich darauf hingewiesen werden.

Wie kann das aussehen? Im Fall der Funktion **StrasseUndHausnummer** sieht ein einfacher Test so aus wie in Listing 1. Hier prüfen wir zunächst, ob der Aufruf der Funktion den Wert **True** zurückliefert. Falls ja, prüfen wir, ob die beiden Rückgabeparameter **strStrasse** und **strHausnummer** die erwarteten Werte enthalten. Wenn dies zutrifft, geschieht nichts weiter – falls nein, erscheint eine Ausgabe, welche die erwarteten Werte sowie die tatsächlich gelieferten Werte ausgibt:

```
Aufruf mit Parameter 'Gerichtsstr. 27 a' fehlgeschlagen.
```

```
Public Sub Test_StrasseUndHausnummer()
    Dim strStrasseUndHausnummer As String
    Dim strStrasse As String
    Dim strHausnummer As String
    strStrasseUndHausnummer = "Gerichtsstr. 27"
    If StrasseUndHausnummer(strStrasseUndHausnummer, strStrasse, strHausnummer) = True Then
        If Not (strStrasse = "Gerichtsstr." And strHausnummer = "27") Then
            Debug.Print "Aufruf mit Parameter '" & strStrasseUndHausnummer & "' fehlgeschlagen."
            Debug.Print "Erwartet: 'Gerichtsstr.' Geliefert: '" & strStrasse & "'"
            Debug.Print "Erwartet: '27' Geliefert: '" & strHausnummer & "'"
        End If
    Else
        Debug.Print "Funktionsaufruf mit Parameter '" & strStrasseUndHausnummer & "' nicht erfolgreich."
    End If
End Sub
```

Listing 1: Erster Entwurf einer Prozedur zum Testen einer Funktion

MSForms: Formulare und Steuerelemente

Neben den eingebauten Formularen und Steuerelementen gibt es noch eine weitere Bibliothek, die einen eigenen Formulartyp und einige bereits bekannte sowie einige neue Steuerelemente liefert. Dabei handelt es sich um die MSForms-Bibliothek, welche das UserForm-Formular sowie besagte Steuerelemente bereithält. Die Elemente dieser Bibliothek bringen einige Vorteile gegenüber den eingebauten Elementen, aber es ist hier und da auch mehr Aufwand zu betreiben, um die gewohnten Funktionen nachzubilden. Deshalb wollen wir die Elemente für Aufgaben nutzen, wo wir mit Bordmitteln nicht weiterkommen. Welche das sind und wie die Umsetzung aussieht, zeigt der vorliegende Artikel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1905_MSForms.accdb**.

Die MSForms-Bibliothek

In einer Standard-Accessdatenbank finden Sie die Elemente der **MSForms**-Bibliothek am ehesten, wenn Sie ein ein Formular in der Entwurfsansicht öffnen und dann die Liste der ActiveX-Steuerelemente über den Ribbon-Eintrag **Entwurf|Steuerelemente|ActiveX-Steuerelemente** anzeigen (siehe Bild 1). Hier finden Sie dann viele Steuerelemente, die Ihnen bekannt vorkommen, weil Sie diese bereits als eingebaute Access-Steuerelemente erkennen – zum Beispiel **CheckBox**, **ComboBox**, **CommandButton** und so weiter.

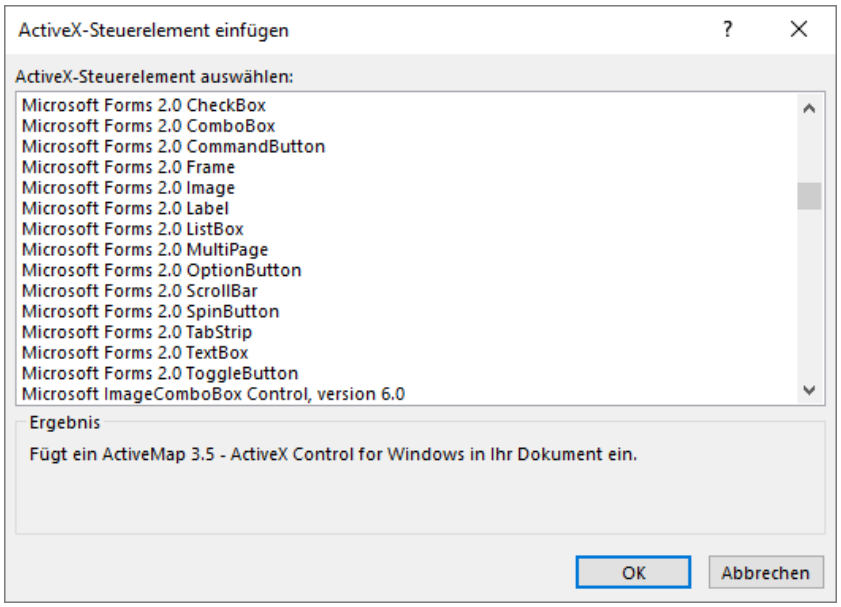


Bild 1: Liste der in Access verfügbaren **MSForms**-Steuerelemente

Möglichkeiten der MSForms-Steuerelemente

Die Steuerelemente haben teilweise andere Eigenschaften und Möglichkeiten als die üblichen Access-Steuerelemente. Welche das sind, schauen wir uns im Detail in eigenen Artikeln zu den einzelnen Steuerelementen an.

In diesem Artikel betrachten wir zunächst, welche Möglichkeiten diese Steuerelemente und das bisher noch nicht erwähnte **UserForm**-Element bieten.

Vorab noch die Information, dass diese Steuerelemente überall zur Verfügung stehen, wo Access oder die Access Runtime installiert sind. Sie brauchen sich also keine Sorgen zu machen, dass es zu Problemen mit Verweisen oder falschen Steuerelementversionen kommt – ähnlich wie bei Elementen der **MSComCtl**-Sammlung wie TreeView, ListView und so weiter.

Das UserForm-Element

Wer von Excel oder Word kommt und dort bereits Benutzeroberflächen programmiert hat, kennt das **UserForm**-Element bereits. Es handelt sich dabei um

eine Möglichkeit, Excel, Word und die anderen Office-Anwendungen mit Elementen für die Benutzeroberfläche auszustatten – beispielsweise, um Daten vom Benutzer abzufragen.

Dabei kann es sich um ein Eingabeformular für die Adressangabe eines Anschreibens handeln, mit dem der Benutzer die Daten komfortabel eingeben kann und sich nicht um die Platzierung der Informationen im zu erstellenden Word-Dokument kümmern muss. Oder Sie fügen einem Excel-Dokument ein **UserForm**-Element hinzu, mit dem der Benutzer Daten auswählen soll, die dem Excel-Dokument hinzugefügt werden sollen.

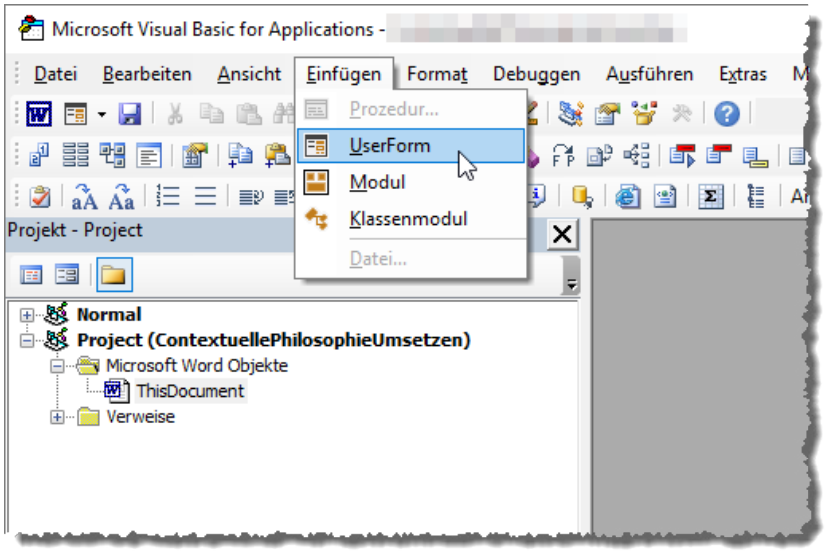


Bild 2: Einfügen eines **UserForm**-Elements im VBA-Editor von Word

UserForms in Word oder Excel

Natürlich können Sie ein solches Formular auch dazu nutzen, die Daten einer Access-Datenbank in einem Word- oder Excel-Dokument anzubieten – zum Beispiel, um in Word Adressdaten aus einer Kundendatenbank in ein Anschreiben zu übernehmen oder um in Excel eine Kalkulation auf Basis von Daten aus der Datenbank zu erstellen.

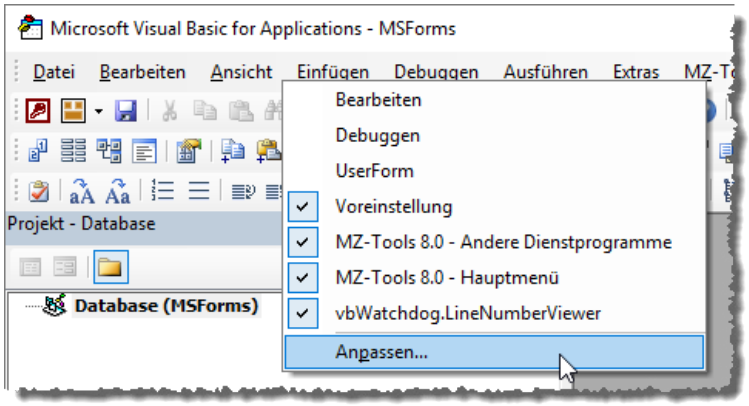


Bild 3: Aufrufen des **Anpassen**-Dialogs

UserForm-Elemente in Access

Und schließlich können Sie das **UserForm**-Element auch zu einer Access-Anwendung hinzufügen.

Warum sollten Sie das tun, wenn es doch Access-eigene Formulare gibt, die sogar sehr einfache Möglichkeiten der Datenbindung mitliefern und Sie die Steuerelemente der **MSForms**-Bibliothek offensichtlich auch in herkömmliche Access-Formulare integrieren können?

Warum UserForms in Access?

Access-Formulare sind relativ resourcenhungrig – wegen der Eigenschaften zur Anzeige von Daten et cetera. Eine UserForm ist standardmäßig gar nicht dafür vorgesehen, an die Daten einer Tabelle oder Abfrage gebunden zu werden und kommt entsprechend leichter daher.

Außerdem funktionieren die Steuerelemente der **MSForms**-Bibliothek zwar auch in Access-Formularen, zuverlässiger tun sie ihren Dienst jedoch in UserForms. Und schließlich können Sie in UserForms zur Laufzeit Steuerelemente anlegen – das ist in Access-Formularen nur zur Entwurfszeit möglich.

Auch wenn Sie ein Access-Formular per Code in den Entwurfsmodus versetzen und dann Steuerelemente hinzufügen können, stoßen Sie an die Grenzen, wenn

Sie die Datenbank als **.mde** oder **.accde** weitergeben wollen – dort gibt es nämlich keine Entwurfsansicht mehr.

Allerdings haben UserForms auch Nachteile gegenüber Access-Formularen: Diese bieten nämlich schon von Haus aus alle Möglichkeiten zur Datenbindung. UserForms bieten diese Möglichkeit gar nicht an, Steuerelemente müssten hier zur Laufzeit mit den gewünschten Daten gefüllt werden.

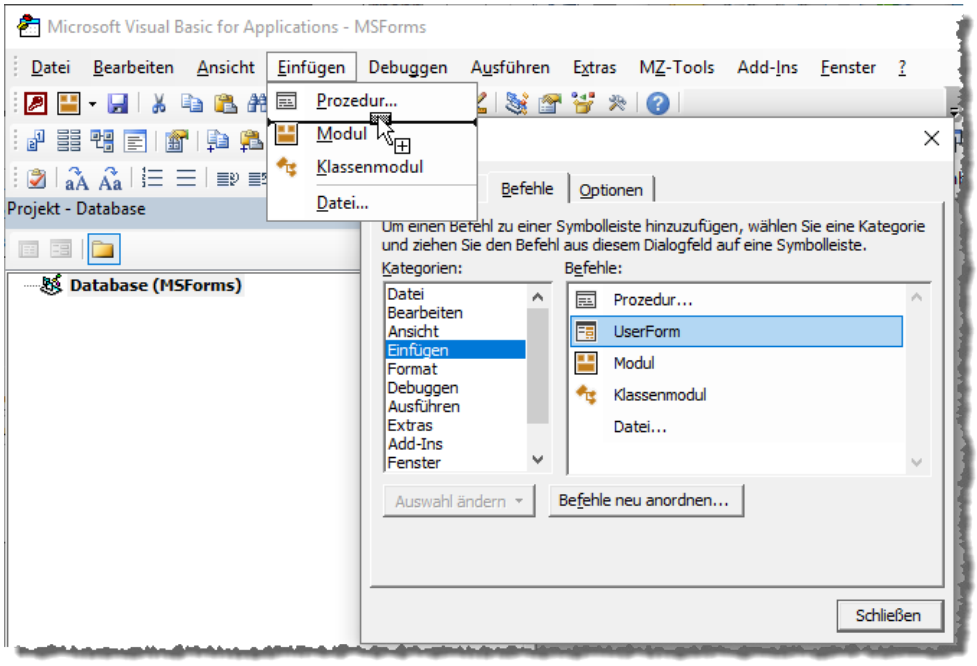


Bild 4: Hinzufügen des Eintrags **UserForm** zum **Einfügen**-Menü

UserForm in Access anlegen

Wenn Sie ein **UserForm**-Element in Access anlegen wollen, so wie Sie es vielleicht von Excel oder Word gewohnt sind, haben Sie zunächst keinen Erfolg.

Im VBA-Editor von Word etwa finden Sie den passenden Eintrag im Menü unter **Einfügen|UserForm** (siehe Bild 2).

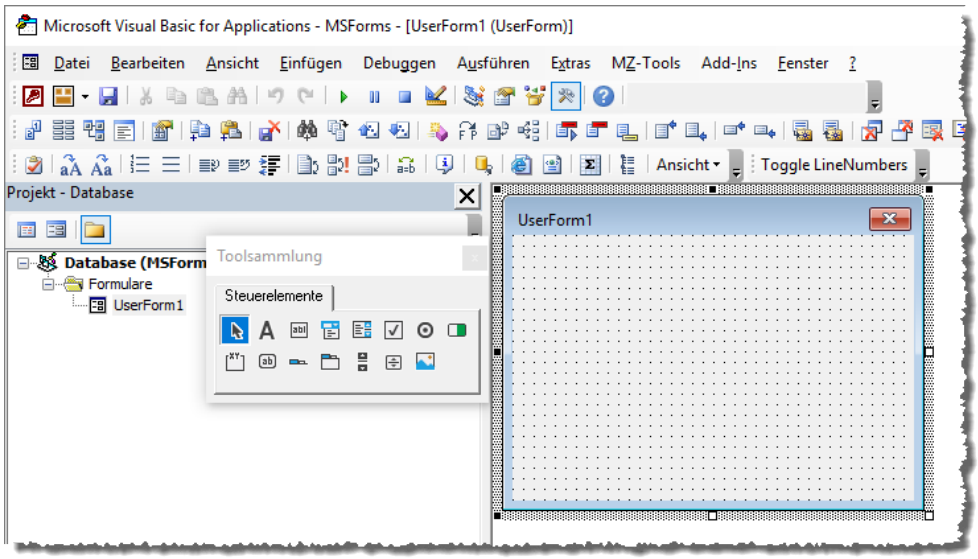


Bild 5: Anlegen einer UserForm

Das ist im VBA-Editor von Access nicht der Fall: Hier fehlt dieser Eintrag schlicht und einfach.

Wie aber sollen wir nun eine UserForm zu unserem Projekt hinzufügen? Dazu betätigen Sie zunächst den Eintrag **Anpassen** des Kontextmenüs der Menüleiste (siehe Bild 3).

Dort wechseln Sie dann zum Registerreiter **Befehle** und wählen im linken Listenfeld den Eintrag **Einfügen**. Rechts erscheinen dann alle für diese Kategorie verfügbaren Befehle, unter anderem auch **UserForm**.

Diesen ziehen wir dann per Drag and Drop erst auf den Menüeintrag **Einfügen** und dann im nun erschei-

nenden Untermenü an die gewünschte Stelle (siehe Bild 4). Fragen Sie nicht, warum dieser Eintrag nicht standardmäßig auch in Access vorhanden ist ...

Wenn wir diesen Menüeintrag danach betätigen, erscheint auch direkt das **UserForm**-Element mit dem Namen **UserForm1** in der Entwurfsansicht im VBA-Editor. Dazu erscheint eine Toolsammlung mit den Steuerelementen der **MSForms**-Bibliothek (siehe Bild 5). Im Projekt-Explorer erscheint der Eintrag für das neue **UserForm**-Element in einem neuen Unterbereich namens **Formulare**.

UserForm umbenennen

Um eine UserForm umzubenennen, können Sie das Eigenschaftenfenster des **UserForm**-Elements verwenden. Dieses bietet im Gegensatz zu den Eigenschaften von Access-Objekten sogar die Möglichkeit, die Eigenschaften nach Kategorien anzuzeigen (siehe Bild 6).

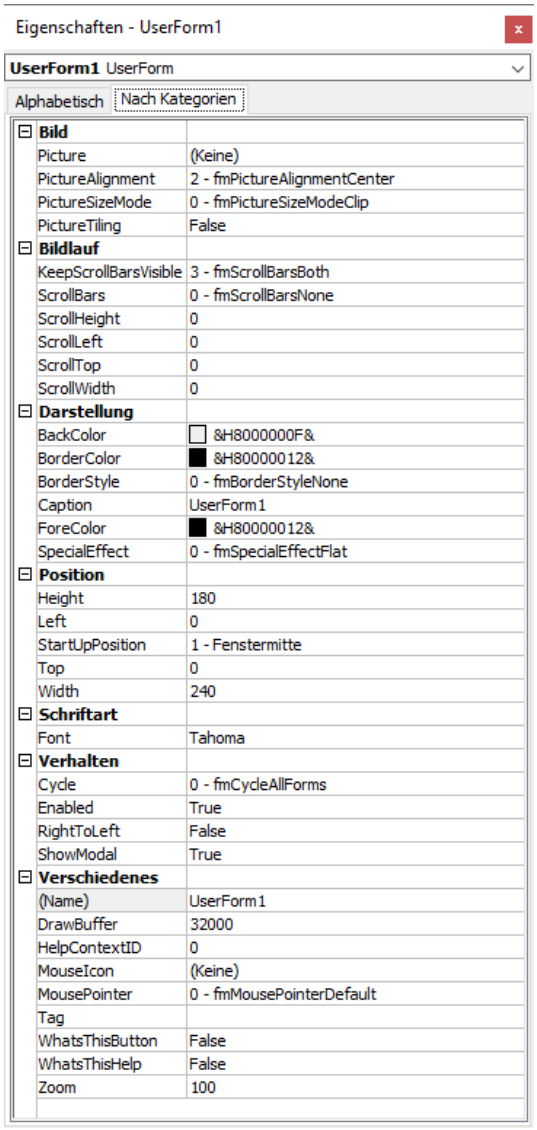


Bild 6: Eigenschaften einer UserForm

Hier finden wir dann auch schnell die Eigenschaft **Name**, mit der wir den Namen von **UserForm1** auf den gewünschten Namen ändern können. Gleichzeitig können wir uns um den Titel des UserForms kümmern, den wir mit der Eigenschaft **Caption** einstellen. Die übrigen Eigenschaften schauen wir uns später an.

Steuerelemente hinzufügen

Die Toolsammlung mit den Steuerelementen haben Sie ja schon gesehen. Damit können Sie das hinzuzufügende Steuerelement markieren und dann entweder einfach an die gewünschte Stelle im UserForm klicken, um das Steuerelement in einer Standardgröße hinzuzufügen oder Sie ziehen einen Rahmen in der entsprechenden Größe auf.

Danach erscheinen die Steuerlemente dann in der UserForm.

Steuerelemente ausrichten und Co.

Die Ausrichtung dieser Steuerelemente funktioniert ähnlich wie bei denen in Access-Formularen. Hilfestellung bieten die Anweisungen des Kontextmenüs eines oder mehrerer Steuerelemente (siehe Bild 7).

Hier finden Sie die Befehle zum Ausschneiden, Kopieren, Einfügen und Löschen, zum Anzeigen der Eigenschaften und auch zum Anzeigen des Codes. Die Ausrichten-Ein-

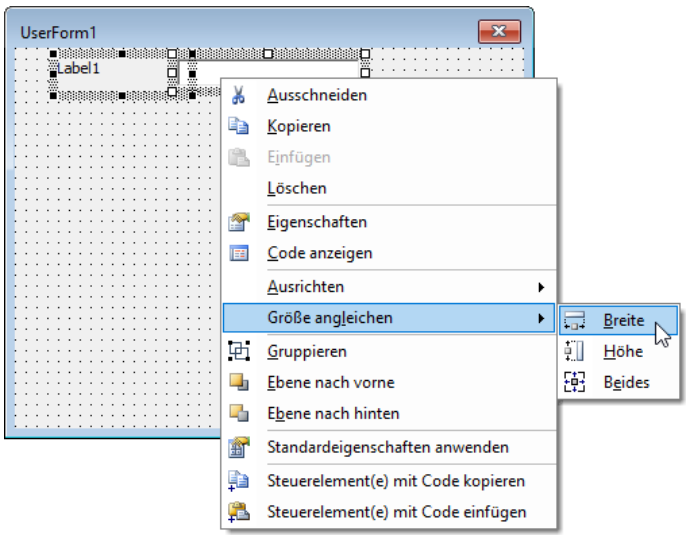


Bild 7: Einstellen von Steuerelementen per Kontextmenü

Ligaverwaltung, Teil 2: Formulare

Für die meisten Sportarten gibt es Wettbewerbe in Ligen. Oder zumindest Turniere, die ja meist auch als kleine Liga angesehen werden können mit Spielpaarungen und Tabellen. Jedenfalls gibt eine Verwaltung solcher Ligen jede Menge Stoff für eine Datenbankanwendung – mit Datenmodell, Abfragen, Formularen zur Eingabe der Daten und Berichten, um Spielpläne und Tabellen auszugeben. Im zweiten Teil der Artikelreihe programmieren wir die Formulare, die zur Eingabe der Daten der Ligaverwaltung nötig sind.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1905_Ligaverwaltung.accdb**.

Rapid Prototyping

Ich betreibe immer Rapid Prototyping. Das heißt, ich konzipiere nicht die komplette Anwendung durch, bevor ich sie programmiere, sondern ich fange einfach an zu programmieren. Dafür ist Microsoft Access auch das perfekte Tool, denn es gibt keine andere Entwicklungsumgebung für Datenbanken für Windows-Systeme, mit der Sie so schnell Ergebnisse erhalten.

Im ersten Teil dieser Artikelreihe haben wir ein Datenmodell für eine Ligaverwaltung erstellt, wie es uns sinnvoll erscheint und dabei die eine oder andere Abwägung getroffen. Beim Definieren des Datenmodells sollte man sorgfältig arbeiten, denn jede Entscheidung, die nachträglich revidiert werden muss, wirkt sich auf alle folgenden Ebenen aus – auf die Abfragen, Formulare und Berichte sowie auf den VBA-Code der Anwendung.

Wenn wir nun starten, die Benutzeroberfläche für die Ligaverwaltung zu programmieren, werden wir möglicherweise noch auf Probleme im Datenmodell stoßen, die wir dann direkt beheben werden.

Wie starten wir beim Prototyping? Wir wollen eine Anwendung erschaffen, die dem Benutzer eine Vorstellung von der Bedienung und den enthaltenen Funktionen gibt. Dazu gehört, dass der Benutzer damit auch alle Daten eingeben kann, die für den Betrieb der Anwendung nötig sind. Wir arbeiten uns deshalb am Datenmodell entlang und erstellen zuerst die Formulare, die für die Eingabe der grundlegenden Daten nötig sind – zum Beispiel die Ligen und die Spieljahre. Die Tabelle **tblLigenSpieljahre** kombiniert die Ligen und die Spieljahre. Als Nächstes benötigen wir die Mannschaften. Wenn wir diese eingegeben haben, können wir sie einer Kombination aus Liga und Spieljahr zuordnen. Erst dann können wir die Spieltage und somit den Spielplan für eine Liga festlegen. Dazu wollen wir später einen praktischen Editor bereitstellen. Stehen die Spieltage, soll der Benutzer in einem eigenen Formular die Ergebnisse von Spieltagen eingeben können. Aus den Ergebnissen eines oder mehrerer Spieltage resultieren dann die Tabellenstände, die nur noch ausgegeben werden sollen. Dazu stellen wir geeignete Berichte bereit – das allerdings erst in einem späteren Teil der Artikelreihe. Nun starten wir erst einmal mit den Tabellen für die Basisdaten.

Formular frmLigen

Das erste Formular soll der Eingabe und der Verwaltung der Ligen dienen. Dazu erstellen wir zunächst ein Unterformular namens **sfmLigenVerwalten**. Über die Eigenschaft **Datensatzquelle** weisen wir ihm die Tabelle **tblLigen** als Datenquelle zu. Damit es alle Daten dieser Tabelle in der Datenblattansicht ausgibt, ziehen wir alle drei Felder der Tabelle aus der Feldliste in den Detailbereich der Entwurfsansicht. Sie brauchen die Felder dort nicht ordentlich anzuordnen, denn wir stellen die Ansicht ohnehin über die Eigenschaft **Standardansicht** auf **Datenblatt** um. Das Ergebnis finden Sie in Bild 1.

Nun schließen wir das Unterformular und erstellen das Hauptformular namens **frmLigenVerwalten**. Diesem fügen wir per Drag and Drop den Eintrag für das Unterformular **sfmLigenVerwalten** zum Detailbereich hinzu. Darunter ordnen wir noch eine Schaltfläche zum Schließen des Formulars an. Außerdem entfernen wir das Beschriftungsfeld, das automatisch mit dem Unterformular zum Formular hinzugefügt wurde und ändern die Eigenschaften **Bildlaufleisten**, **Navigationsschaltflächen** und **Datensatzmarkierer** auf **Nein** sowie **Automatisch zentrieren** auf **Ja**, sodass dieses wie in Bild 2 aussieht.

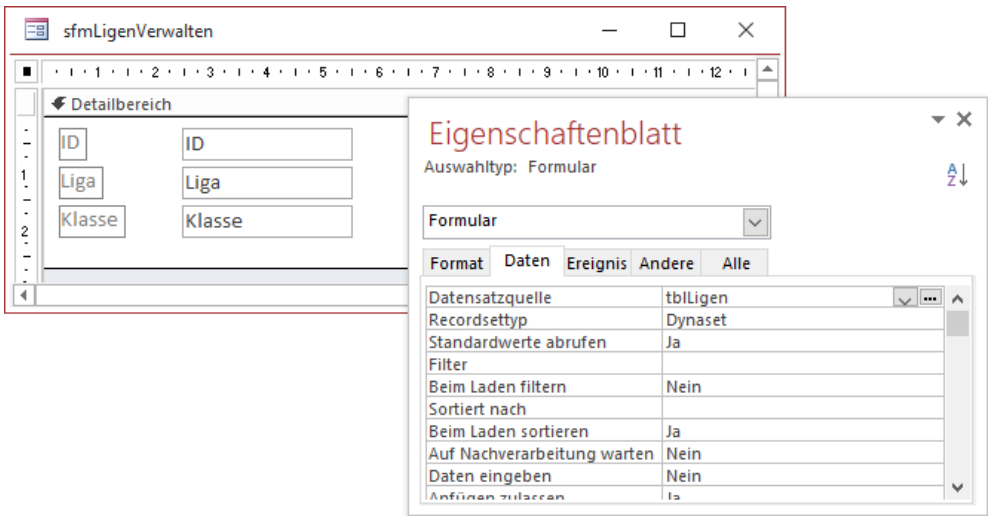


Bild 1: Entwurf des Unterformulars **sfmLigenVerwalten**

Für die Schaltfläche **cmdOK** hinterlegen wir die folgende Ereignisprozedur:

```
Private Sub cmdOK_Click()
    DoCmd.Close acForm, Me.Name
End Sub
```

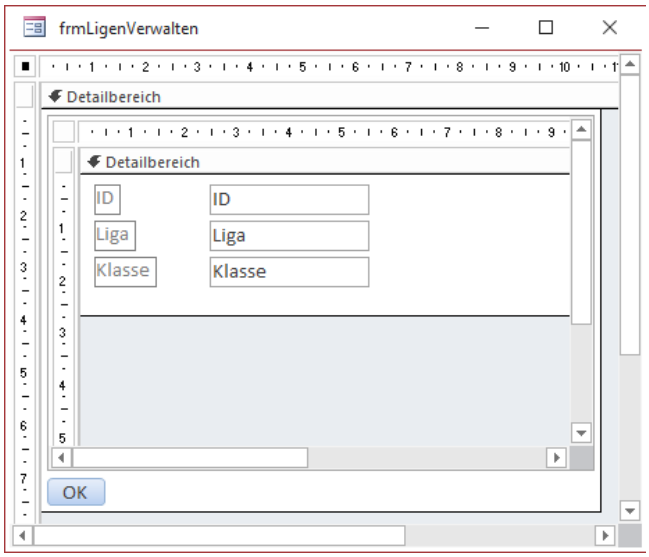


Bild 2: Entwurf des Formulars **frmLigenVerwalten**

Danach können wir schon die ersten Ligen eingeben, wie Bild 3 zeigt.

Spieljahre verwalten

Danach kümmern wir uns direkt um ein Formular beziehungsweise ein Formular samt Unterformular zur Eingabe von Spieljahren. Das Unterformular soll **sfmSpieljahreVerwalten** heißen und verwendet die Tabelle **tblSpieljahre** als Datensatzquelle. Wir fügen auch hier alle Felder der Tabelle zum Detailbereich des Formulars hinzu. Dann stellen wir die Eigenschaft **Standardansicht** auf **Datenblatt** ein.

Das Formular fügen wir nach dem Speichern und Schließen ebenfalls per Drag and Drop aus dem Navigationsbereich von Access in ein neues Formular

Bild 3: Formular zur Eingabe von Ligen

namens **frmSpieljahreVerwalten** ein. Hier führen wir die gleichen Schritte wie zuvor beim Hauptformular **frmLigenVerwalten** durch, sodass sich in der Entwurfsansicht die Ansicht aus Bild 4 ergibt.

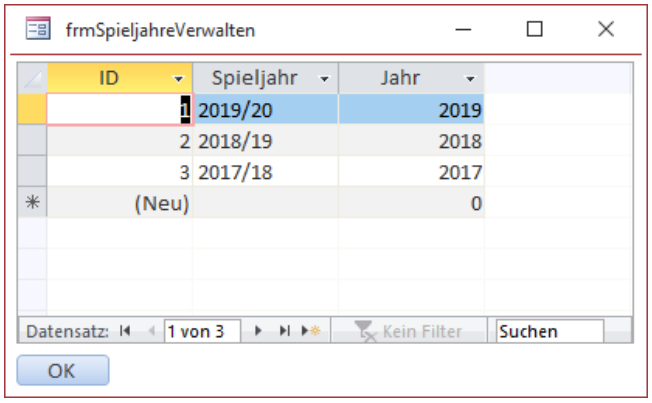


Bild 4: Formular zur Eingabe von Spieljahren

Spieljahre und Ligen zusammenführen

An dieser Stelle taucht die erste interessante Frage auf: Zu welchem Zeitpunkt erstellen wir die Kombinationen aus Liga und Spieljahr? Diese Kombination speichern wir ja in einer eigenen Tabelle namens **tblLigenSpieljahre**. Die Einträge sollen unter anderem eine Bezeichnung wie **Bundesliga 2019/2020** liefern. Die Idee ist: Wir legen immer neue Einträge an, wenn der Benutzer entweder eine neue Liga oder ein neues Spieljahr hinzufügt. Wir können in beiden Fällen die gleiche Prozedur aufrufen, die wie in Listing 1 aussieht. Diese Prozedur definiert zwei Recordsets auf Basis der Tabellen **tblLigen** und **tblSpieljahre**. Sie durchläuft die Datensätze der Tabelle **tblLigen** in einer äußeren **Do While**-Schleife und die der Tabelle **tblSpieljahre** in einer inneren **Do While**-Schleife. Damit die Einträge der inneren Schleife nach dem ersten Durchlauf der inneren und äußeren Schleife erneut durchlaufen werden, müssen wir den Datensatzzeiger der inneren Schleife danach mit **rstSpieljahre.MoveFirst** wieder auf den ersten Datensatz einstellen. In der inneren **Do While**-Schleife führen wir bei

```
Public Sub LigenUndSpieljahreAktualisieren()
    Dim db As DAO.Database
    Dim rstLigen As DAO.Recordset
    Dim rstSpieljahre As DAO.Recordset
    Set db = CurrentDb
    Set rstLigen = db.OpenRecordset("tblLigen", dbOpenDynaset)
    Set rstSpieljahre = db.OpenRecordset("tblSpieljahre", dbOpenDynaset)
    Do While Not rstLigen.EOF
        Do While Not rstSpieljahre.EOF
            On Error Resume Next
            db.Execute "INSERT INTO tblLigenSpieljahre(LigaSpieljahr, LigaID, SpieljahrID) VALUES('" & rstLigen!Liga & " " _
                & rstSpieljahre!Spieljahr & "', " & rstLigen!ID & ", " & rstSpieljahre!ID & ")", dbFailOnError
            If Err.Number = 3022 Then
                db.Execute "UPDATE tblLigenSpieljahre SET LigaSpieljahr = '" & rstLigen!Liga & " " & rstSpieljahre!Spieljahr _
                    & "' WHERE LigaID = " & rstLigen!ID & " AND SpieljahrID = " & rstSpieljahre!ID, dbFailOnError
            End If
            On Error GoTo 0
            rstSpieljahre.MoveNext
        Loop
        rstSpieljahre.MoveFirst
        rstLigen.MoveNext
    Loop
End Sub
```

Listing 1: Aktualisieren der Kombinationen aus Ligen und Spieljahren

deaktivierter Fehlerbehandlung eine **INSERT INTO**-Anweisung aus, die den Wert des Feldes **ID** der beiden Recordsets sowie die Kombination der beiden Felder **Liga** aus **rstLigen** und **Spieljahr** aus **rstSpieljahre** in die Felder der Tabelle **tblLigenSpieljahre** einträgt.

Es kann sein, dass dabei ein Fehler mit der Nummer **3022** auftritt, wenn die Kombination aus Liga und Spieljahr schon in der Tabelle angelegt wurde – und zwar deshalb, weil wir einen eindeutigen, zusammengesetzten Index für diese beiden Tabellen angelegt haben. Wenn dieser Fehler auftritt, brauchen wir den Datensatz natürlich nicht erneut anzulegen. Aber gegebenenfalls hat der Benutzer in der Zwischenzeit den Namen der Liga geändert.

tblLigenSpieljahre

ID	LigaSpieljahr	LigaID	SpieljahrID
1	1. Bundesliga 2019/20	Bundesliga	2019/20
2	1. Bundesliga 2018/19	Bundesliga	2018/19
3	1. Bundesliga 2017/18	Bundesliga	2017/18
5	2. Bundesliga 2019/20	2. Bundesliga	2019/20
6	2. Bundesliga 2018/19	2. Bundesliga	2018/19
7	2. Bundesliga 2017/18	2. Bundesliga	2017/18
8	3. Liga 2019/20	3. Liga	2019/20
9	3. Liga 2018/19	3. Liga	2018/19
10	3. Liga 2017/18	3. Liga	2017/18
11	Regionalliga Nord 2019/20	Regionalliga Nord	2019/20
12	Regionalliga Nord 2018/19	Regionalliga Nord	2018/19
13	Regionalliga Nord 2017/18	Regionalliga Nord	2017/18
14	Regionalliga Nord-Ost 2019/20	Regionalliga Nord-Ost	2019/20
15	Regionalliga Nord-Ost 2018/19	Regionalliga Nord-Ost	2018/19
16	Regionalliga Nord-Ost 2017/18	Regionalliga Nord-Ost	2017/18
	(Neu)		

Datensatz: 1 von 15 | Kein Filter | Suchen

Bild 5: Die Tabelle **tblLigenSpieljahre** mit einigen Beispieldatensätzen

Also prüfen wir nach der **INSERT INTO**-Anweisung, ob **Err.Number** den Wert **3022** enthält, und führen in diesem Fall noch eine **UPDATE**-Anweisung aus, welche die Kombination aus Liga und Spieljahr im Feld **LigaSpieljahr** der Zieltabelle **tblLigenSpieljahre** ergänzt. Nachdem wir in beiden Schleifen alle Kombinationen aus Liga und Spieljahr durchlaufen haben, sieht die Tabellen für unsere wenigen Beispieldatensätzen danach wie in Bild 5 aus.

Nun müssen wir nur noch den Aufruf dieser Prozedur überall dort einfügen, wo der Benutzer die Liga oder das Spieljahr ändern kann. Am einfachsten ist es, dies beim Schließen der beiden Formulare **frmLigenVerwalten** und **frmSpieljahreVerwalten** zu erledigen.

Dazu nutzen wir das Ereignis **Beim Entladen** der beiden Formulare. Die dadurch ausgelöste Ereignisprozedur sieht so aus:

```
Private Sub Form_Unload(Cancel As Integer)
    Call LigenUndSpieljahreAktualisieren
End Sub
```

Zur Sicherheit: Die Namen im Feld **LigaSpieljahr** werden dabei immer und bei jeder Änderung einer der beiden Tabellen an die Werte der dortigen Felder angepasst.

Nun haben wir die beiden Formulare zur Verwaltung von Ligen und Spieljahren so erstellt, dass der Access-kundige Benutzer diese verwenden kann – er muss wissen, dass man zum Löschen beispielsweise einen Datensatz über den Datensatzmarkierer markieren und dann die Entfernen-Taste betätigen muss und er sollte wissen, wie er einen neuen Datensatz anlegen kann.

Mannschaften verwalten

Auch die Mannschaften wollen erst einmal eingegeben werden, bevor man sich um die Zuteilung zu Ligen/Spieljahren und Spielplänen kümmern kann.

Also erstellen wir ein Hauptformular namens **frmMannschaftenVerwalten** und ein Unterformular namens **sfmMannschaftenVerwalten**, wobei das Unterformular wieder die Daten der Tabelle **tblMannschaften** in der Datenblattansicht anzeigt und das Hauptformular das Unterformular sowie die Schaltfläche **OK** enthält. Das erlaubt dann die Eingabe von Mannschaften wie in Bild 6.

Ligen, Spieljahre und Mannschaften zuordnen

Danach können wir uns an ein etwas aufwendigeres Formular begeben. Dieses soll dazu dienen, die in der Tabelle **tblLigenSpieljahre** definierten Kombinationen aus Ligen und Spieljahren mit den Mannschaften aus der Tabelle **tblMannschaften** zu verknüpfen.

Dieses Formular soll so aussehen, dass wir im Hauptformular mit einem Kombinationsfeld die Kombination aus Liga und Spieljahr auswählen und dann über ein Unterformular die Mannschaften zuordnen können. Dazu erstellen wir zunächst das Hauptformular und fügen diesem ein Kombinationsfeld namens **cboLigaSpieljahr** hinzu. Dieses Kombinationsfeld erhält als Datensatzherkunft die folgende Abfrage auf Basis der Tabelle **tblLigenSpieljahre**:

```
SELECT ID, LigaSpieljahr FROM tblLigen-
Spieljahre;
```

Die beiden Eigenschaften **Spaltenanzahl** und **Spaltenbreiten** stellen wir auf die Werte **2** und **0cm** ein. Der Entwurf sieht danach wie in Bild 7 aus.

Nun erstellen wir das Unterformular **sfmLigenSpieljahreMannschaften**. Es soll alle Mannschaften anzeigen, die bereits mithilfe des Kombinationsfeld für die Auswahl der Kombination aus Liga und Spieljahr zugeordnet wurden und außerdem das Hinzufügen und Entfernen dieser Einträge erlauben.

Das Unterformular verwendet also nicht etwa die Tabelle **tblMannschaften** als Datensatzquelle, sondern die Tabelle **tblLigenSpieljahreMannschaften** – und davon allein das Feld **MannschaftID**. Da wir dieses in der zugrundeliegenden Tabelle als Nachschlagefeld definiert haben, wird es im Formularentwurf beim Einfügen über die Feldliste direkt als Kombinationsfeld angelegt (siehe Bild 8). Für das Unterformular stellen wir die Eigenschaft **Standardansicht** wieder auf **Datenblatt** ein.

Bild 6: Mannschaften per Formular verwalten

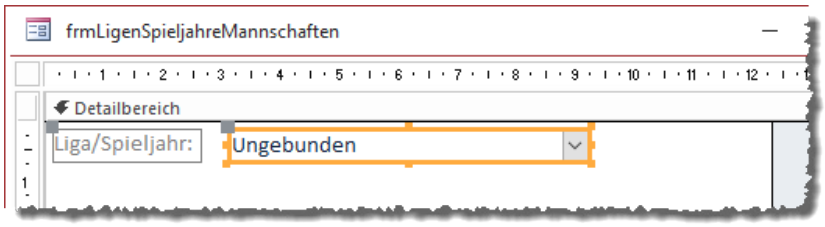


Bild 7: Auswählen von Liga/Spieljahr per Kombinationsfeld

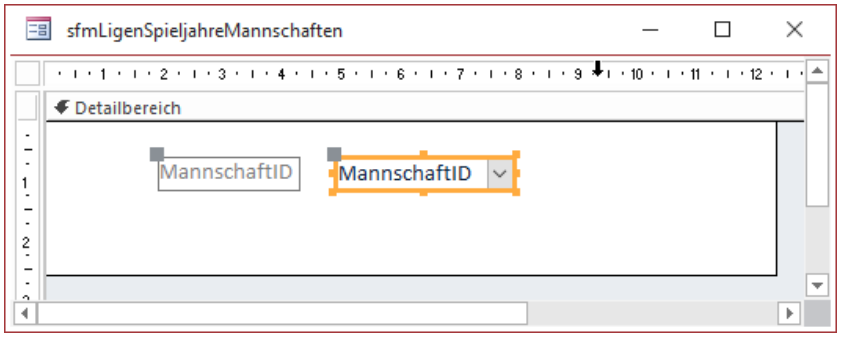


Bild 8: Unterformular zum Zuordnen von Mannschaften zu Liga/Spieljahr

Danach fügen wir das Formular **sfmLigenSpieljahreMannschaften** als Unterformular zum Formular **frmLigenSpieljahreMannschaften** hinzu (siehe Bild 9).

Mannschaften per Kombinationsfeld zuordnen

Nun sollte das Unterformular optimalerweise genau die Mannschaften anzeigen, die wir schon für eine Liga ausgewählt haben. Allerdings haben wir ja noch keine Einträge ausgewählt. Und wenn wir nun das Formular in der Formularansicht öffnen und eine Mannschaft auswählen, geschieht Folgendes: Es wird

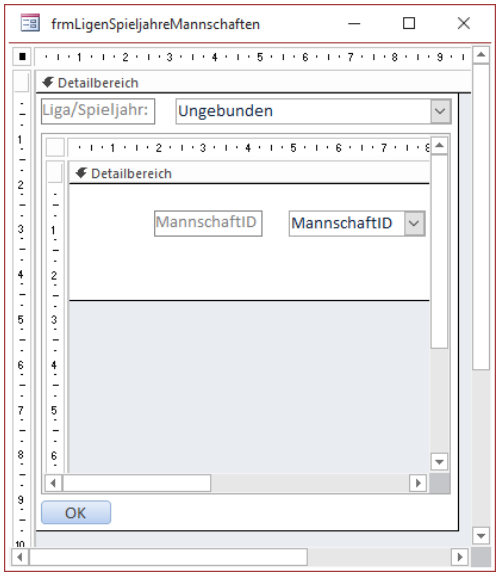


Bild 9: Haupt- und Unterformular zum Zuordnen von Mannschaften zu Liga/Spieljahr

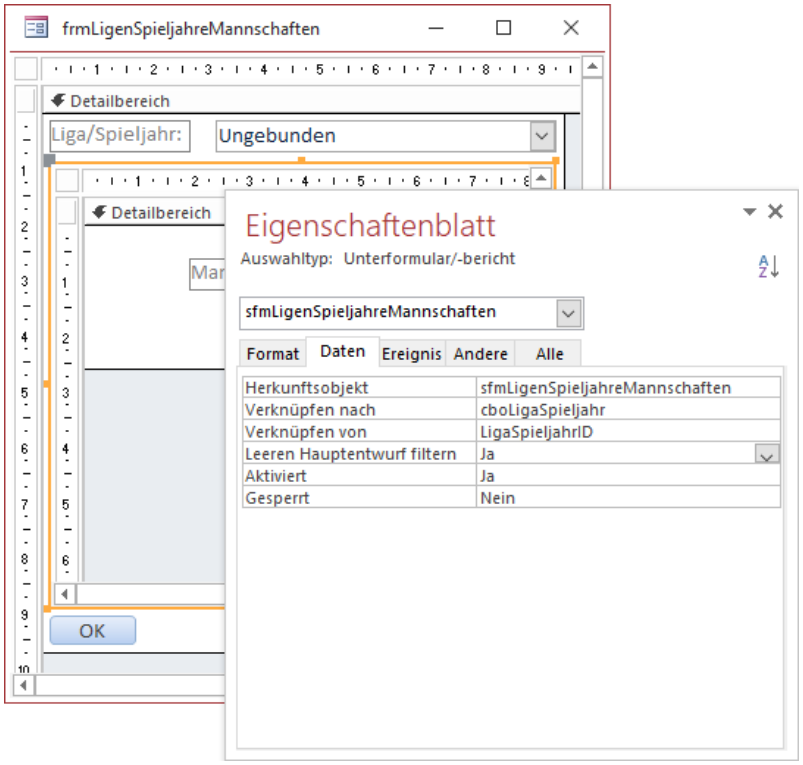


Bild 10: Einstellen der Verknüpfung zwischen Haupt- und Unterformular

ein Eintrag zur Tabelle **tblLigenSpieljahreMannschaften** hinzugefügt, der zwar einen Wert im Feld **MannschaftID** hat, aber keinen im Feld **LigaSpieljahrID**. Dabei erhalten wir das gleiche Ergebnis, egal, ob wir im Kombinationsfeld eine Saison ausgewählt haben oder nicht. Das Feld **LigaSpieljahrID** im Unterformular wird schlicht und einfach nicht gefüllt. Der Grund ist, dass wir das Feld nicht mit dem Wert des Kombinationsfeldes aus dem Hauptformular verknüpft haben.

Das holen wir nun nach, indem wir die beiden Eigenschaften **Verknüpfen von** und **Verknüpfen nach** des Unterformular-Steuerelements entsprechend einstellen. Dazu markieren Sie zunächst das Unterformular-Steuerelement, was am einfachsten gelingt, indem Sie seinen Namen aus dem Kombinationsfeld der Steuerelemente im Eigenschaftsfenster auswählen. Dann stellen Sie die Eigenschaft **Verknüpfen nach** auf das Kombinationsfeld **cboLigaSpieljahr** des Hauptformulars ein und die Eigenschaft **Verknüpfen von** auf das Feld **LigaSpieljahrID** des Unterformulars (siehe Bild 10).

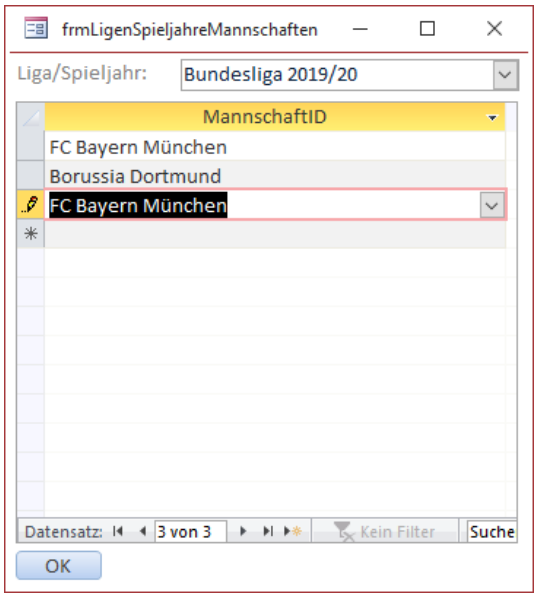


Bild 11: Mehrfache Auswahl der gleichen Mannschaft

Nun können Sie in die Formularansicht wechseln und einer Saison die Vereine zuweisen. Wenn Sie dabei versuchen, eine Mannschaft zweimal zur gleichen Saison hinzuzufügen, zeigt dies eine Fehlermeldung an, welcher dem Fehler **3022** entspricht (siehe Bild 11).

Diese eingebaute Fehlermeldung können wir auf zwei Arten vermeiden:

XML mit VBA lesen: Nodes selektieren

Im Artikel »XML-Dokumente mit VBA lesen« haben wir die grundlegenden Befehle erläutert, mit denen Sie per VBA auf den Inhalt von XML-Dokumenten zugreifen können. Wir sind dort bereits kurz auf die beiden Funktionen SelectNodes und SelectSingleNode eingegangen, mit denen Sie einen Verweis auf eine Auflistung mehrerer Node-Elemente oder eines einzelnen Elements holen können. Dabei geben Sie per Parameter den Pfad zu den gesuchten Node-Elementen oder die Attribute der Elemente auf diesem Pfad an. Im vorliegenden Artikel sehen wir uns im Detail an, wie Sie solche Suchkriterien formulieren können.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1905_XML-LesenVBA.accdb**.

Beispieldokument

Wir nutzen wieder das Beispieldokument aus dem eingangs erwähnten Artikel, das alle gängigen Eigenschaften und Pfade für Beispiele zum Auffinden von Elementen liefert (siehe Listing 1).

Voraussetzungen

Wenn Sie die Beispiele dieses Artikels nachvollziehen wollen, verwenden Sie entweder die Beispieldatenbank oder eine eigene Datenbank. Dieser fügen Sie dann einen Verweis auf die Bibliothek **Microsoft XML, v6.0** hinzu.

Schnelles Einlesen des XML-Dokuments

Da wir das XML-Dokument aus dem Listing in allen folgenden Beispielprozeduren einlesen müssen, lagern wir die dazu notwendigen Anweisungen in eine eigene Funktion namens **DokumentEinlesen** aus.

Diese finden Sie in Listing 2. Alle Aufgaben dieser Funktion haben wir bereits im Artikel **XML-Dokumente mit VBA lesen** erläutert.

```
<?xml version="1.0"?>
<PurchaseOrder PurchaseOrderNumber="99503" OrderDate="1999-10-20">
  <Address Type="Shipping">
    <Name>Ellen Adams</Name>
    <Street>123 Maple Street</Street>
    <City>Mill Valley</City>
    <State>CA</State>
    <Zip>10999</Zip>
    <Country>USA</Country>
  </Address>
  <Address Type="Billing">
    <Name>Tai Yee</Name>
    <Street>8 Oak Avenue</Street>
    <City>Old Town</City>
    <State>PA</State>
    <Zip>95819</Zip>
    <Country>USA</Country>
  </Address>
  <DeliveryNotes>Please leave packages in shed by driveway.</DeliveryNotes>
  <Items>
    <Item PartNumber="872-AA">
      <ProductName>Lawnmower</ProductName>
      <Quantity>1</Quantity>
      <USPrice>148.95</USPrice>
      <Comment>Confirm this is electric</Comment>
    </Item>
    <Item PartNumber="926-AA">
      <ProductName>Baby Monitor</ProductName>
      <Quantity>2</Quantity>
      <USPrice>39.98</USPrice>
      <ShipDate>1999-05-21</ShipDate>
    </Item>
  </Items>
</PurchaseOrder>
```

Listing 1: Inhalt eines XML-Dokuments mit einer Bestellung

In den folgenden Prozeduren füllen wir einfach eine **MSXML2.DOMDocument60**-Variable mit dem Ergebnis der Funktion **DokumentEinlesen**.

```
Public Function DokumentEinlesen() As MSXML2.DOMDocument60
    Dim strDatei As String
    Dim objXML As MSXML2.DOMDocument60
    strDatei = CurrentProject.Path & "\PurchaseOrder.xml"
    Set objXML = New MSXML2.DOMDocument60
    objXML.Load strDatei
    If Not Len(objXML.XML) = 0 Then
        Set DokumentEinlesen = objXML
    Else
        MsgBox "Fehler " & objXML.parseError.errorCode & ": " & objXML.parseError.reason
    End If
End Function
```

Listing 2: Funktion zum Einlesen des Inhalts von XML-Dokumenten

XPath

Die Parameter, die wir den beiden Funktionen **SelectNodes** oder **SelectSingleNode** übergeben, werden in einer eigenen Sprache formuliert – in **XPath**.

Wir schauen uns zunächst an, welche Daten uns die beiden Funktionen zurückliefern, wenn wir diesen gar keine Parameter übergeben.

selectSingleNode, einfache Variante

Die Funktion **selectSingleNode** funktioniert, wie wir schnell herausfinden, schon einmal nicht ohne Parameter. Sie erwartet mindestens die Angabe des Schrägstrichs als Zeichenkette:

```
Public Sub SelectNodesOhneParameter()
    Dim objXML As MSXML2.DOMDocument60
    Dim objNode As MSXML2.IXMLDOMNode
    Set objXML = DokumentEinlesen
    Set objNode = objXML.selectSingleNode("/")
    Debug.Print objNode.XML
End Sub
```

Wenn wir mit dieser Prozedur auf das mit dem Ausdruck **/** zu findende Element zugreifen, erhalten wir das vollständige XML-Dokument über die XML-Eigenschaft des gefundenen **IXMLDOMNode**-Elements.

Aber handelt es sich hierbei überhaupt um ein **IXMLDOMNode**-Element? Das finden wir heraus, wenn wir die Variable **objNode** als **Object** deklarieren und den Objekttyp dann mit der Funktion **Typename** ausgeben lassen:

```
...
Dim objNode As Object 'MSXML2.IXMLDOMNode
Set objXML = DokumentEinlesen
Set objNode = objXML.selectSingleNode("/")
Debug.Print TypeName(objNode)
...
```

Das Ergebnis lautet:

```
DOMDocument60
```

/ liefert also das vollständige Dokument zurück.

selectSingleNode mit Elementname

Wenn wir gezielt den Namen eines Elements angeben, in diesem Fall den des Root-Elements, dann liefert **selectSingleNode** ein Element des Typs **IXMLDOMElement** und gibt den vollständigen Dokumentinhalt mit Ausnahme der Processing Instruction (das ist die **<?xml...>**-Zeile) zurück:

```
Set objNode = objXML.selectSingleNode("PurchaseOrder")
Debug.Print TypeName(objNode)
Debug.Print objNode.XML
```

Anderes Element per Elementname

Wenn wir über den Elementnamen auf ein Element zugreifen wollen, das nicht auf der ersten Ebene liegt, könnten wir das so probieren (die beiden **Address**-Elemente liegen direkt unterhalb von **PurchaseOrder**):

```
Set objNode = objXML.selectSingleNode("Address")
```

Wenn wir danach mit **objNode.XML** auf den Inhalt zugreifen wollen, gelingt das nicht – wir erhalten dann die Fehlermeldung aus Bild 1.

Bild 1: Fehler beim Zugriff auf eine nicht gefüllte Objektvariable

Wie aber können wir auf ein Element mit einem bestimmten Namen zugreifen, dass sich unterhalb des Root-Elements befindet? Dazu gibt es verschiedene Möglichkeiten.

Die erste ist der Zugriff unter Angabe des kompletten Pfades zum gesuchten Element, also Root-Element/Unterelement, hier **PurchaseOrder/Address**:

```
Set objNode = objXML.selectSingleNode(_
                                "PurchaseOrder/Address")
```

Damit greifen Sie auf das erste der beiden **Address**-Elemente zu.

Das ist schon ein Fortschritt gegenüber der reinen Nutzung der übrigen Eigenschaften und Auflistungen.

Um an ein **Address**-Element zu kommen, müssten wir sonst erst das **documentElement** referenzieren und dann das erste Unterelement mit **childNodes(0)**:

```
Dim objXML As MSXML2.DOMDocument60
Dim objRoot As MSXML2.IXMLDOMNode
Dim objAddress As MSXML2.IXMLDOMNode
Set objXML = DokumentEinlesen
Set objRoot = objXML.documentElement
Set objAddress = objNode.childNodes(0)
Debug.Print objAddress.XML
```

Im ganzen Dokument per Elementname suchen

Wenn Sie ein **Address**-Element im ganzen Dokument suchen wollen und nicht wissen, dass es unterhalb des **PurchaseOrder**-Elements liegt, können Sie zwei Schrägstriche voranstellen:

```
Set objNode = objXML.selectSingleNode("//Address")
```

Das zweite Address-Element finden

Wenn Sie wissen, dass es zwei **Address**-Elemente im Dokument gibt und Sie möchten genau auf das zweite zugreifen, benötigen Sie eine zusätzliche Angabe, in diesem Fall die Angabe des Index in eckigen Klammern:

```
Set objNode = objXML.selectSingleNode("//Address[2]")
```

Diese Notation liefert das zweite Element mit dem Namen **Address** unter dem gleichen übergeordneten Element.

Sollten sich im Dokument weitere **Address**-Elemente befinden, die nicht unterhalb von **PurchaseOrder** liegen, müssen Sie **PurchaseOrder/Address[2]** verwenden, um auf die zweite Adresse zuzugreifen.

Sie sehen hier, dass der Index für Elemente von XPath-Anfragen bei **1** beginnt.

Ein Element nach Position finden

Neben der Angabe des Index können Sie noch weitere Methoden nutzen, um auf Elemente mit einer bestimmten Position zuzugreifen.

Das erste Element erhalten Sie automatisch, wenn Sie keine weiteren Angaben machen. Wenn Sie jedoch zum Beispiel das letzte Element einer Reihe von Elementen suchen, gelingt das mit dieser Abfrage und der **last()**-Funktion in eckigen Klammern:

```
Set objNode = objXML.selectSingleNode(_
                                    "//Address[last()]")
Debug.Print objNode.XML
```

Per Elementname und Attribut suchen

Praktischerweise haben die beiden **Address**-Elemente aber auch noch jeweils ein Attribut namens **Type**, das für die Rechnungsadresse den Wert **Billing** und für die Lieferadresse den Wert **Shipping** enthält.

Sie können auch gezielt nach **Address**-Elementen mit einem bestimmten Wert für das Attribut **Type** suchen. Um die Lieferadresse zu ermitteln, verwenden Sie den folgenden Ausdruck:

```
Set objNode = objXML.selectSingleNode("PurchaseOrder/
Address[@Type=""Shipping""]")
Debug.Print objNode.XML
```

Wir geben also in eckigen Klammern den Namen des Attributs mit führendem @-Zeichen an und dahinter den Wert, den das Attribut für das gelieferte Element aufweisen soll. Das Ergebnis sieht so aus:

```
<Address Type="Shipping">
    <Name>Ellen Adams</Name>
    <Street>123 Maple Street</Street>
    <City>Mill Valley</City>
    <State>CA</State>
    <Zip>10999</Zip>
    <Country>USA</Country>
</Address>
```

Wenn wir die **Billing**-Adresse holen wollen, gelingt das so:

```
Set objNode = objXML.selectSingleNode(_
            "PurchaseOrder/Address[@Type=""Billing""]")
Debug.Print objNode.XML
```

Das liefert dann:

```
<Address Type="Billing">
    <Name>Tai Yee</Name>
    <Street>8 Oak Avenue</Street>
    <City>Old Town</City>
    <State>PA</State>
    <Zip>95819</Zip>
    <Country>USA</Country>
</Address>
```

Werte von Elementen ausgeben

Bisher haben wir immer nur den Typ der gefundenen Elemente ausgegeben oder den XML-Code. Meist wollen Sie jedoch den Wert eines Elements ermitteln, beispielsweise für den Namen der Rechnungsadresse. Hier verwenden wir dann die Eigenschaft **NodeTypedValue**, um den Namen für die **Billing**-Adresse auszugeben:

```
Set objNode = objXML.selectSingleNode(_
      "PurchaseOrder/Address[@Type=""Shipping""]/Name")
Debug.Print objNode.nodeTypedValue
```

Das liefert den Wert **Ellen Adams**.

Namen von Elementen ausgeben

Wenn Sie ein Element ermittelt haben und dessen Namen ausgeben wollen, erledigen Sie das mit der Eigenschaft **baseName** – hier gleich in Kombination mit dem Wert:

```
Set objNode = objXML.selectSingleNode(_
      "PurchaseOrder/Address[@Type=""Shipping""]/Name")
Debug.Print objNode.baseName & ": " _
                              & objNode.nodeTypedValue
```

Das liefert nun:

```
Name: Ellen Adams
```

Element mit bestimmtem Wert ermitteln

Weiter oben haben wir bereits Elemente nach dem Wert eines Attributs ermittelt. Sie können auch Elemente ermitteln, die selbst einen bestimmten Wert aufweisen – zum Beispiel ein **Address**-Element mit dem Wert **Ellen Adams** im Element **Name**:

```
Set objNode = objXML.selectSingleNode(_
        "PurchaseOrder/Address[Name=""Ellen Adams""]")
```

Erweitern der VBA-IDE: Mit Code arbeiten

Die Visual Basic-Entwicklungsumgebung ist definitiv in die Jahre gekommen. Wer schon einmal mit einer modernen Entwicklungsumgebung wie Eclipse oder dem Visual Studio .NET gearbeitet hat, wird bei der Rückkehr zur VBA-IDE möglicherweise einiges vermissen. Es gibt aber Möglichkeiten, die Entwicklungsumgebung zu erweitern – sogar mit den Hausmitteln von Access und mit nur zwei zusätzlichen Bibliotheken. Im zweiten Teil dieser Artikelreihe schauen wir uns an, wie Sie per Code mit dem VBA-Code des VBA-Projekts arbeiten.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1906_Entwicklungsumgebung_Code.accdb**.

Einstieg

Was ließen sich nicht für tolle Funktionen für die VBA-Entwicklungsumgebung umsetzen: Automatisches Hinzufügen oder Entfernen von Code, Kommentarzeichen oder Nummerierungen, automatisches Anlegen von Klassen oder Modulen, Erweitern von Prozeduren um Fehlerbehandlungen und vieles mehr. Der Haken an der Sache ist: Wie programmiert man das, und wie soll man die Funktionen anschließend aufrufen?

Grundlage für all diese Techniken ist das Objektmodell der Visual Basic-Entwicklungsumgebung (kurz: VBE). Die Grundlagen dazu haben wir bereits im ersten Teil dieser Artikelreihe betrachtet (siehe **Erweitern der VBA-IDE: Mit Modulen**). Sie bietet noch eine ganze Reihe von Möglichkeiten mehr: Sie können damit den Quellcode direkt manipulieren, Codepassagen suchen und ersetzen, Zeilen von Modulen und Prozeduren zählen oder die Prozeduren eines Moduls ermitteln.

Das ist alles schön und gut, aber was hilft es, wenn man die notwendigen Routinen zur Codemanipulation nur auf herkömmlichem Wege, also etwa per Direktfenster oder über die Schaltflächen eines Formulars, aufrufen kann? Im letzteren Fall müsste man dazu sogar noch von der VBE zur Access-Oberfläche wechseln.

Und wer schon einmal versucht hat, über das VBE-Objektmodell einfach eine Menüleiste anzulegen

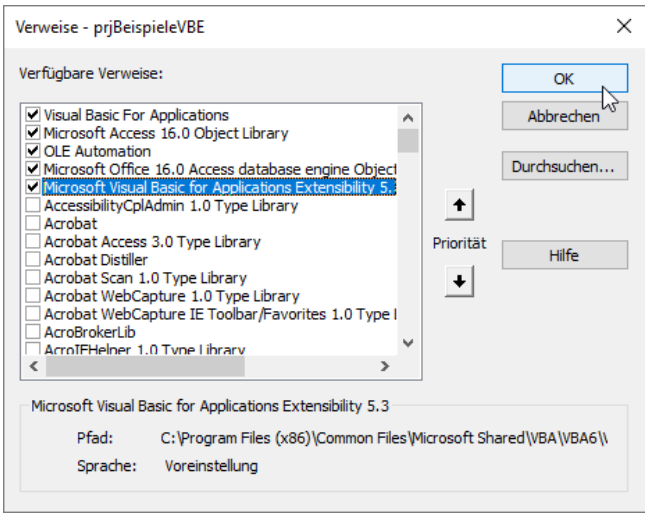


Bild 1: Notwendiger Verweis für das Programmieren der VBA-Entwicklungsumgebung

und über die **OnAction**-Methode eines **CommandBarButton**-Objekts eine entsprechende Funktion zu starten, musste feststellen, dass dies leider nicht funktioniert.

Der vorliegende Beitrag beschäftigt sich sowohl mit den zur Manipulation von VBA-Modulen notwendigen Objekten, Methoden und Eigenschaften des VBE-Objektmodells als auch mit einem Trick, mit dem Sie den benutzerdefinierten Menüleisten in der VBA-Entwicklungsumgebung doch noch etwas abgewinnen.

Voraussetzungen für den Zugriff auf VBA-Module

Bevor es an das Programmieren geht, müssen Sie noch zwei Verweise einstellen: Neben den üblichen Verdächtigen sind das die Bibliotheken **Microsoft**

Visual Basic forApplications Extensibility x.y und **Microsoft Office x.y Object Library** (siehe Bild 1). Erstere stellt das VBE-Objektmodell und Letztere die für das Anlegen von Menüs im VBA-Editor notwendigen Objekte bereit – dazu später mehr.

Code bearbeiten mit dem VBE-Objektmodell

Einige Informationen über die Handhabung von Modulen können Sie dem bereits weiter oben angegebenen Artikel entnehmen.

Im vorliegenden Artikel finden Sie Informationen über die Befehle, die sich mit dem Inhalt der Module beschäftigen – mit dem Code selbst. Dabei erfahren Sie, wie Sie die Methoden und Eigenschaften der VBE-Objektbibliothek verwenden, um den Code zu manipulieren.

Aus Platzgründen finden Sie hier keine komplette Vorstellung aller Objekte, Methoden und Eigenschaften. Wenn Sie weitere Informationen benötigen, schauen Sie sich einfach im Objektkatalog um und wählen dort die gewünschte Objektbibliothek aus (siehe Bild 2).

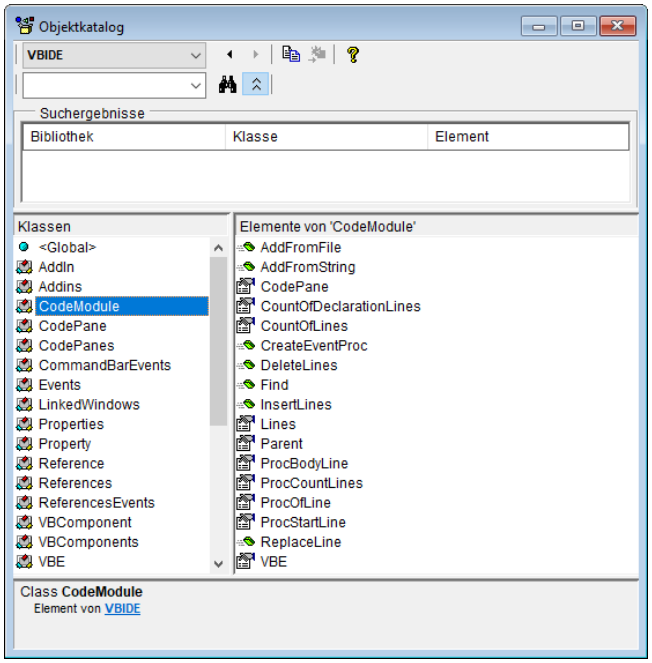


Bild 2: Überblick über die VBE-Objektbibliothek

Moduleigenschaften erkunden

Legen Sie doch einfach mit ein paar Fingerübungen los und erkunden Sie die Eigenschaften eines Moduls. Dabei lernen Sie direkt, welche davon sich dazu eignen, einzelne Prozeduren oder den Deklarationsbereich eines Moduls zu extrahieren – die einzelnen Elemente sind nämlich nicht über eine Auflistung oder Ähnliches auszuwählen, sondern müssen mühsam von Hand erfasst werden.

Zielmodul festlegen

Als Erstes müssen Sie im Code das Modul referenzieren, das Sie untersuchen möchten.

Zwei Möglichkeiten sind für die nachfolgenden Beispiele interessant: Das aktuelle Modul steht im Mittelpunkt des Interesses, oder Sie wählen das gewünschte Modul explizit aus.

In beiden Fällen deklarieren Sie zunächst eine Variable, mit der Sie das Zielmodul referenzieren:

```
Dim objCodeModule As CodeModule
```

Mit folgender Anweisung setzen Sie die Objektvariable auf das Codemodule im aktuellen Codepane:

```
Set objCodeModule = VBE.ActiveCodePane.CodeModule
```

```
Public Function AlleModuleAusgeben()
    Dim objVBComponent As VBComponent
    Dim objCodeModule As CodeModule
    For Each objVBComponent In VBE.ActiveVBProject.VBComponents
        Debug.Print "======================================"
        Debug.Print objVBComponent.Name
        Debug.Print "======================================"
        Set objCodeModule = objVBComponent.CodeModule
        Debug.Print objCodeModule.Lines(1, objVBComponent.CodeModule.CountOfLines)
    Next objVBComponent
End Function
```

Listing 1: Ausgabe aller Module inklusive Namen im Direktfenster

Für ein anderes, bestimmtes Modul verwenden Sie die folgende Zuweisung:

```
Set objCodemodule = VBE.ActiveVBProject._
    VBComponents.Item("mdlVBEBeispiele").
CodeModule
```

Zum Durchlaufen aller enthaltenen Codemodule verwenden Sie die Auflistung aller **VBComponents** in einer **For Each**-Schleife. Die Routine aus Listing 1 gibt den Inhalt aller vorhandenen Module im Direktfenster aus.

Zeilen zählen

Die oben beschriebene Routine verwendet bereits die Eigenschaft **CountOfLines** des **CodeModule**-Objekts.

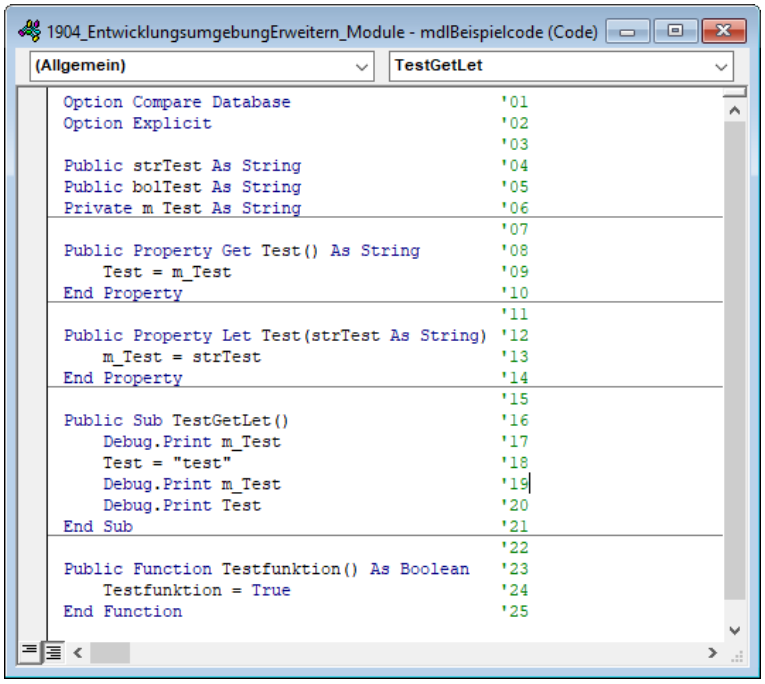


Bild 3: Beispielmodul zum Ausprobieren der **VBE**-Funktionen

Diese Eigenschaft gibt die Gesamtanzahl der Zeilen eines Moduls aus – so lässt sich die **Line**-Eigenschaft mit der Nummer der ersten Zeile und der Anzahl der Zeilen füttern, um alle enthaltenen Zeilen auszugeben.

Etwas weniger Zeilen gibt die Eigenschaft **CountOfDeclarationLines** aus. Sie zählt die Zeilen bis zur letzten Zeile des Deklarationsbereichs. Wenn der Deklarationsbereich eines Moduls nur aus den folgenden beiden Zeilen besteht, gibt die Funktion **CountOfDeclarationLines** den Wert **2** aus:

```
Option Compare Database
Option Explicit
```

Um diese und die nachfolgend vorgestellten Methoden und Eigenschaften zu testen, legen Sie am besten ein Beispielmodul wie in Bild 3 an. Wichtig sind dabei die per Kommentar angegebenen Zeilennummern.

Prozeduren untersuchen

Als Nächstes nehmen Sie sich den Bereich hinter dem Deklarationsteil eines Moduls vor. Dieser Bereich enthält die eigentlichen Prozeduren.

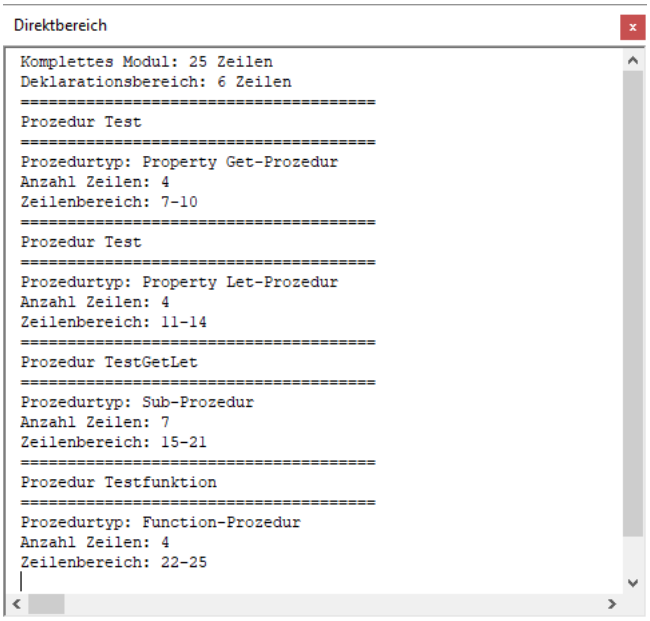


Bild 4: Ausgabe der Eigenschaften eines Moduls

Leider bietet das VBE-Objektmodell keinen Weg, um direkt Prozeduren aufzulisten, geschweige denn deren Inhalt zu ermitteln.

Stattdessen bieten sich allerdings einige Methoden an, mit denen Sie selbst eine Routine zur Ermittlung

Erweitern der VBA-IDE: CommandBars

Im zweiten Teil dieser Beitragsreihe haben Sie das Handwerkszeug für das Steuern der VBA-Entwicklungsumgebung und spezieil von Modulen und dem enthaltenen Code kennen gelernt. Im vorliegenden zweiten Teil geht es ans Eingemachte: Anhand praktischer Beispiele werden wir die VBA-Entwicklungsumgebung erweitern und dafür sorgen, dass die Erweiterungen von der Symbolleiste und dem Kontextmenü aus aufrufbar sind.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1906_Commandbars.accdb**.

Voraussetzungen

Für die Programmierung von Menüleisten mit Early Binding benötigen Sie einen Verweis auf die Bibliothek **Microsoft Office x.0 Object Library**. Diesen finden Sie in älteren Office-Versionen unter dieser Bezeichnung im **Verweise**-Dialog. In neueren Versionen heißt der Verweis einfach **Office** (siehe Bild 1). Nach dem Hinzufügen und dem erneuten Öffnen des Dialogs wird die Bibliothek jedoch unter dem standardmäßigen Namen angezeigt.

Einführung

Erweiterungen der VBA-Entwicklungsumgebung dienen häufig einer Vereinfachung der Quellcodeeingabe, um sich die Durchführung immer wiederkehrender oder komplizierter Aufgaben zu erleichtern.

Dazu gehört beispielsweise das Hinzufuügen einer Fehlerbehandlung zu einer Prozedur, einer Zeilennummerierung oder von **Property Get**- und **Property Let**- beziehungsweise **Property Set**-Prozeduren für bestimmte Variablen.

Eingabe von Klammern und Anführungszeichen

Eine zwar sehr klein erscheinende, aber dennoch auf Dauer nervende Aufgabe ist das Hinzufügen von Klammernpaaren sowie von einfachen oder doppelten Anführungszeichen. Damit ist nicht die Eingabe dieser Zeichen beim erstmaligen Anlegen einer Prozedur gemeint, sondern das nachträgliche Einfügen – etwa, wenn Sie eine **MsgBox**-Funktion nicht nur aufrufen, sondern auch den Rückgabewert ermitteln möchten. Die erste Variante sieht wie folgt aus:

```
MsgBox "Bitte klicken Sie auf Ja oder Nein.", vbYesNo
```

Variante mit Rückgabewert:

```
Debug.Print MsgBox("Bitte klicken Sie auf Ja oder
Nein.", vbYesNo)
```

Das Anlegen der Klammern ist zwar kein allzu großer Aufwand, aber man könnte es auch vereinfachen, indem man beispielsweise den einzuklammernden Bereich markiert und auf eine Schaltfläche der Symbolleiste oder des Kontextmenüs des Moduls klickt.

Die Aussicht auf eine kleine Vereinfachung des Programmieralltags müssen Sie nun allerdings noch ein paar Abschnitte lang zurückstellen: Zunächst erfahren Sie, wie sich Menüs in die VBA-Benutzeroberfläche

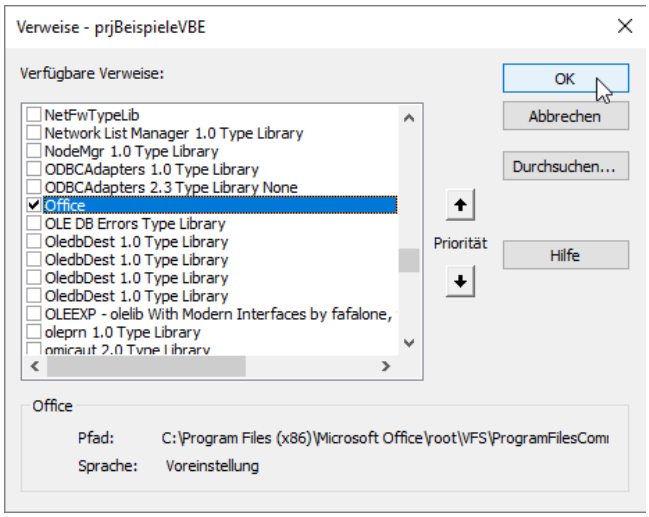


Bild 1: Verweis auf die Office-Bibliothek

integrieren lassen, mit denen die Erweiterung aufgerufen werden kann.

Menüpunkte in der VBA-Entwicklungsumgebung anlegen

Die Verwendung von Menübefehlen und deren Erstellung wäre an sich kein Problem – wenn es nicht einen unsäglichen Bug gäbe, der die **OnAction**-Eigenschaft der Menüleisten-Steuerelemente untauglich macht.

Man könnte dann über den Anpassen-Modus der Menüleisten die gewünschten Schaltflächen hinzufügen, per **OnAction**-Eigenschaft die gewünschten Funktionen festlegen und diese in einem öffentlichen Modul speichern. Das Gleiche funktioniert zum Glück auch per VBA, allerdings nur mit einer alternativen Programmiertechnik zu der einfachen Verwendung der für die Eigenschaft **OnAction** hinterlegten Prozedur.

```
Option Compare Database
Option Explicit

Dim colButtons As Collection

Public Function VBEMenueAnlegen()
    Dim cbr As CommandBar
    Dim objMenuEvents As clsMenuEvents
    Set colButtons = New Collection
    On Error Resume Next
    Application.VBE.CommandBars("VBEErweiterungen").Delete
    On Error GoTo 0
    Set cbr = Application.VBE.CommandBars.Add("VBEErweiterungen")
    cbr.Visible = True
    cbr.Position = msoBarTop
    Set objMenuEvents = New clsMenuEvents
    Set objMenuEvents.objButtonBrakes = cbr.Controls.Add(msoControlButton)
    With objMenuEvents.objButtonBrakes
        .Style = msoButtonCaption
        .Caption = "Klammern setzen()"
    End With
    colButtons.Add objMenuEvents
End Function
```

Listing 1: Standardmodul mit einer Funktion zum Anlegen einer neuen Symbolleiste mit einer Schaltfläche

So muss man sich eines trickreichen Umweges behelfen, der folgendermaßen aussieht (siehe auch Listing 1):

Deklarieren Sie in einem öffentlichen Modul ein **Collection**-Objekt namens **colButtons**. Dieses wird später die einzelnen Schaltflächen-Objekte aufnehmen.

Die Anweisungen zum Erstellen der Symbolleiste und Anlegen der gewünschten Schaltflächen fassen Sie in einer Funktion namens **VBEMenueAnlegen** zusammen. Dort deklarieren Sie zunächst ein **CommandBars**-Element namens **cbr** und eine Objektvariable, die auf eine Instanz der Klasse **clsMenuEvents** verweist – diese Klasse schauen wir uns gleich im Anschluss an.

Als Erstes initialisieren wir die Auflistung **colButtons** mit der **New**-Anweisung.

Das Erstellen der Symbolleiste verläuft dann genauso, als ob Sie die Access-Oberfläche mit zusätzlichen Symbolleisten bestücken wollten. Sie fügen zur **CommandBars**-Auflistung des **VBE**-Objekts ein neues Element namens **VBEErweiterungen** hinzu, positionieren es oben bei den anderen Symbolleisten (**cbr.Position = msoBarTop**) und machen es sichtbar (**cbr.Visible = True**).

Nachdem die Symbolleiste vorhanden ist, erstellen wir eine neue Instanz der Klasse **clsMenuEvents** und bestücken die Symbolleiste mit einer Schaltfläche.

Dazu ruft die Prozedur die **Add**-Methode der **Controls**-Auflistung des **CommandBar**-Objekts auf. Das neue **CommandBarButton**-Objekt weist sie der öffentlichen Objektvariablen **objButtonBrakes** der Klasse **clsMenuEvents** zu.

```
Public WithEvents objButtonBrakes As ConmandBarButton

Private Sub objButtonBrakes_Click(ByVal ctrl As Office.ConmandBarButton, CancelDefault As Boolean)
    MsgBox "Klammern anlegen"
End Sub
```

Listing 2: Die Klasse **clsMenuEvents** mit einem Schaltflächenobjekt und der entsprechenden Methode

Nach dem Einstellen der Eigenschaften **Style** und **Caption** landet die Instanz der Klasse **objMenuEvents** – und damit auch die referenzierte neue Schaltfläche – in der Auflistung **colButtons**.

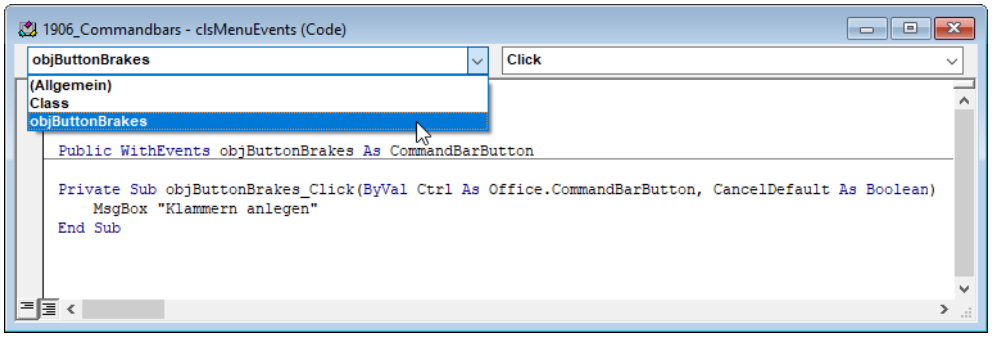


Bild 2: Hinzufügen einer Ereignisprozedur für die Schaltfläche

Klassenmodul mit Button-Ereignis

Danach legen Sie mit dem Menübefehl **Einfügen|Klassenmodul** ein neues Klassenmodul namens **clsMenuEvents** an. Diesem fügen wir eine Objektvariable namens **objButtonBrakes** hinzu und deklarieren sie mit dem Typ **CommandBarButton**.

Dabei verwenden wir das Schlüsselwort **WithEvents**, was uns erlaubt, die Ereignisprozeduren des Typs **CommandBarButton** in der gleichen Klasse zu implementieren. Um diese Ereignisprozedur anzulegen, wählen wir im linken Kombinationsfeld des Codefensters für das Klassenmodul einfach den Namen der Variablen objButtonBrakes aus. Da für dieses nur das Ereignis **Click** vorgesehen ist, wird die entsprechende Ereignisprozedur so direkt angelegt (siehe Bild 2).

Das Ergebnis sieht so aus wie in Listing 2. Sie brauchen nur noch eine **MsgBox**-Anweisung hinzuzufügen. Wenn Sie diese gleich im Anschluss erfolgreich getestet haben, füllen Sie die Prozedur mit der eigentlichen Funktion.

Menü mit Schaltfläche erstellen lassen

Rufen Sie dann die Funktion **VBEMenueAnlegen** auf und kontrollieren Sie, ob die VBA-Entwicklungsumgebung eine zusätzliche Symbolleiste wie in Bild 3 anzeigt.

Ein Klick auf die Schaltfläche in der Symbolleiste sorgt für einen Wechsel zur Access-Oberfläche und zeigt dort das erwartete Meldungsfenster an. Dessen Anzeige lässt sich leider nicht in die VBA-Entwicklungsumgebung verlegen.

Klammern um markierte Ausdrücke setzen

Sie haben nun den ersten benutzerdefinierten Symbolleisteneintrag zur VBA-Entwicklungsumgebung hinzugefügt. Es wird also Zeit, diesen mit einer sinnvollen Funktion zu füllen.

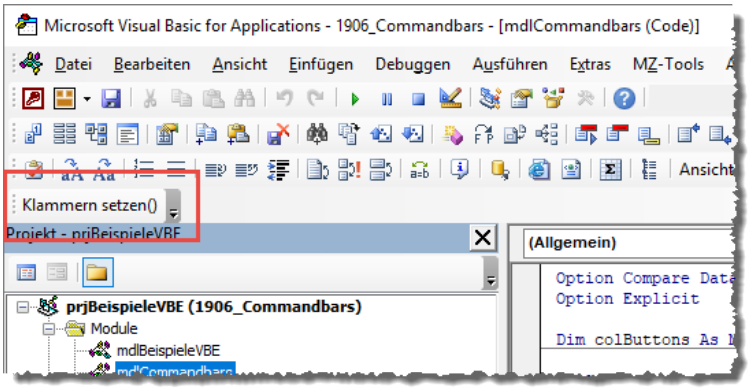


Bild 3: Neue Symbolleiste in Visual Studio

Die Funktion soll vor und hinter den aktuell markierten Bereich des aktuellen Codefensters eine öffnende und eine schließende Klammer hinzufügen.

In aller Kürze passiert dabei Folgendes:

- Die Funktion ermittelt die Koordinaten des einzuklammernden Ausdrucks.

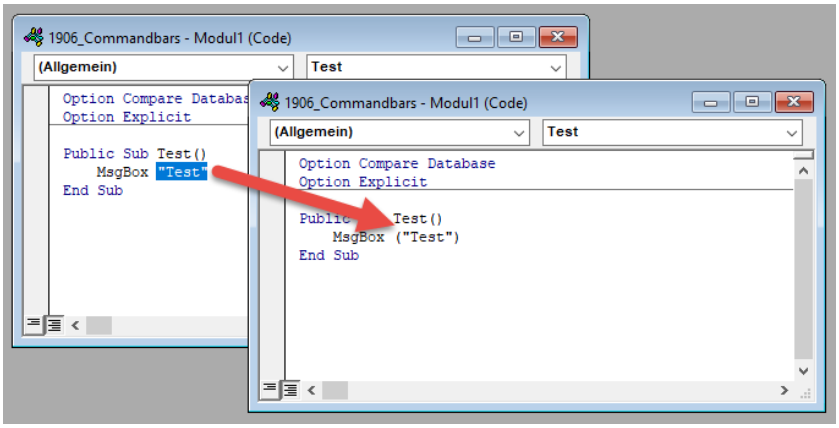


Bild 4: Einklammern der markierten Stelle

- Die Funktion liest den Ausdruck ein und fügt vorne und hinten eine Klammer ein.
- Die Funktion ersetzt den markierten Bereich durch den eingeklammerten Ausdruck (siehe Bild 4).

Schritt für Schritt

Was sich so einfach anhört, erfordert doch eine Menge VBA-Code, wie Listing 3 zeigt. Zu Beginn der Prozedur werden unter anderem zwei Objektvariablen für ein **CodeModule**-Objekt und für ein **CodePane**-Objekt deklariert.

Wie Sie im zweiten Teil der Beitragsreihe erfahren haben, sind die interessanten Methoden und Eigenschaften zur Manipulation des Quellcodes in diesen beiden Klassen verteilt. Die Stringvariablen benötigen Sie, um den markierten Bereich und die geänderte Zeichenkette aufzunehmen.

Wichtig ist die Deklaration der vier **Long**-Variablen **lngStartline**, **lngStartColumn**, **lngEndLine** und **lngEndColumn**: Bei diesen handelt es sich um Variablen, die als Parameter an die Funktion **GetSelection** übergeben und dort gefüllt werden.

Die vier Variablen enthalten anschließend die Koordinaten des markierten Bereichs. Nach der Ermittlung des Bereichs prüft die Prozedur zunächst, ob der Bereich mindestens ein Zeichen enthält. Das ist nur dann nicht der Fall, wenn Startzeile und Endzeile sowie Startspalte und Endspalte übereinstimmen.

Dann geht es an die Analyse der Markierung. Es gibt zwei Konstellationen, die eine unterschiedliche Bearbeitung erfordern:

- Die Markierung ist auf eine Zeile begrenzt.
- Die Markierung erstreckt sich über zwei oder mehr Zeilen.

In beiden Fällen speichern wir die Zeile, in der die Markierung beginnt, in der Variablen **strZeileStart**.

Im ersten Fall sind die Klammern schnell eingearbeitet: Die Prozedur setzt eine Zeichenkette zusammen, die aus dem Teil vor der Markierung, der öffnenden Klammer, dem markierten Teil, der schließenden Klammer und dem Teil hinter der Markierung besteht – hier übersichtlich dargestellt, wobei die **Left**-Funktion den Teil links von der Markierung liefert, die erste **Mid**-Funktion den Teil vom Start bis zum Ende der Markierung und die zweite **Mid**-Funktion den Teil hinter der Markierung:

```
Left(strZeileStart, lngStartColumn - 1)
& "("
& Mid(strZeileStart, lngStartColumn, lngEndColumn -
lngStartColumn)
& ")"
& Mid(strZeileStart, lngEndColumn)
```

Danach setzen wir noch eine neue Markierung, welche an der gleichen Position beginnt, aber um zwei

Ligaverwaltung, Teil 3: Spielpläne generieren

Für die meisten Sportarten gibt es Wettbewerbe in Ligen. Oder zumindest Turniere, die ja meist auch als kleine Liga angesehen werden können mit Spielpaarungen und Tabellen. Jedenfalls gibt eine Verwaltung solcher Ligen jede Menge Stoff für eine Datenbankanwendung – mit Datenmodell, Abfragen, Formularen zur Eingabe der Daten und Berichten, um Spielpläne und Tabellen auszugeben. Im dritten Teil der Artikelreihe entwickeln wir einen Algorithmus, der es uns erlaubt, Spielpläne für Ligen mit beliebig vielen Mannschaften zu generieren.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1906_Ligaverwaltung.accdb**.

Aufgabenstellung

Die Aufgabe lautet: Wir wollen für eine Anzahl von Mannschaften, egal ob 4, 6 oder 18, programmgesteuert einen Spielplan erstellen. Dieser Spielplan soll folgene Bedingungen erfüllen:

- Jede Mannschaft soll ein Heimspiel gegen jede andere Mannschaft haben. Für die Fußball-Bundesliga bedeutet das beispielsweise: Jede Mannschaft macht 17 Heimspiele plus, resultierend aus den 17 Spielen, in denen die anderen Mannschaften ein Heimspiel gegen diese Mannschaft haben, 17 Auswärtsspiele. Dementsprechend gibt es 2 mal n-1 Spieltage, wobei n der Anzahl der Mannschaften entspricht.

- Jede Mannschaft soll auch nur genau einmal in einem Heimspiel gegen jede andere Mannschaft spielen.

- Die Spielpaarungen sind auf Spieltage aufgeteilt, wobei an jedem Spieltag jede Mannschaft einmal spielt.

- Die Spiele sollen in zwei Runden ausgetragen werden, wobei die Hin- und die Rückrunde identisch zusammengestellt werden – nur mit umgedrehtem Heimrecht. Es reicht also prinzipiell, nur die Hinrunde zu generieren und daraus die Spielpaarungen der Rückrunde abzuleiten.

- Um es nicht zu einfach zu machen: Jede Mannschaft soll maximal einmal pro Spielrunde (also bezogen auf Hin- und Rückrunde) zwei Heim- oder Auswärtsspiele hintereinander bestreiten. Und das soll auch erst ab dem zweiten Spieltag und bis zum vorletzten Spieltag möglich sein (also keine zwei Heim- oder Auswärtsspiele am ersten und zweiten oder vorletzten und letzten Spieltag).

- Die Krönung wäre – und so ist es in den echten Spielplänen der Fußballbundesliga –, wenn die Abfolge der Gegner einer jeden Mannschaft ungefähr gleich wäre. Also wenn Mannschaft 1 nacheinander gegen Mannschaft 2, Mannschaft 3 und Mannschaft 4 spielt, dann wird Mannschaft 5 auch irgendwann in dieser Abfolge gegen diese Mannschaften spielen. Das funktioniert natürlich nicht durchgehend, denn wenn man die Abfolge der 17 Spiele der Hinrunde einer Mannschaft auf eine andere Mannschaft übertragen wollte, müsste diese ja gegen sich selbst spielen – und spätestens an dieser Stelle müssten wir die Mannschaft dann austauschen.

Ablauf

Wie könnte das programmatisch ablaufen? Man könnte auf verschiedene Arten vorgehen. Die erste Idee ist, zuerst alle Spiele für Mannschaft 1 festzulegen, dann die verbleibenden Spiele für Mannschaft 2 und so weiter.

Wir könnten auch einfach alle Spiele von Spieltag 1 an komplett durchbesetzen, indem wir immer die Mannschaft mit dem kleinsten Index eintragen, die

noch verfügbar ist. Am erste Spieltag sieht der Spielplan dann so aus:

```
1-2
3-4
5-6
...
```

Am zweiten Spieltag beginnen wir dann wieder mit Mannschaft 1 und suchen die Mannschaft mit dem kleinsten Index, gegen den diese Mannschaft noch nicht gespielt hat – in diesem Fall Mannschaft 3 –, und tauschen das Heimrecht um:

```
3-1
```

Dann hätte allerdings Mannschaft 3 direkt an den ersten beiden Spieltagen zwei Heimspiele, was frühestens ab dem zweiten Spieltag möglich sein soll. Mannschaft 1 müsste dann also bei Mannschaft 4 antreten:

```
4-1
```

Sie sehen: Es ist nicht einfach. Und das wollen wir gleich auch noch programmieren! Abzusehen ist, dass wir dann irgendwann bei der Ermittlung der Paarungen für die hinteren Spieltage auf das Problem stoßen werden, dass wir je Spieltag nicht mehr ausreichend Spiele zusammenstellen und gleichzeitig alle Bedingungen erfüllen können.

Das heißt, wir können keinen starren Ablauf programmieren, sondern müssen die Möglichkeit der Korrektur vorheriger Spieltage berücksichtigen.

In der Programmierung bekommt man das am besten mit rekursiven Funktionen hin. Das heißt, wir legen ein Spiel an, prüfen, ob wir das nächste Spiel nach anlegen können, und wenn nicht, löschen wir das zuvor angelegte Spiel wieder, um ein anderes Spiel anzulegen. Auf diese Weise arbeiten wir uns durch die Kombinationen, um am Ende einen Spielplan zu erhalten, der alle Bedingungen erfüllt. Gegebenenfalls müssen wir uns dabei bis zurück zum Anfang bewegen.

Alternative: Erst Spiele anlegen, dann auf Spieltage aufteilen

Es gibt allerdings noch andere Herangehensweisen. Eine, die vielleicht etwas anschaulicher ist, sieht so aus: Wir tragen erst einmal alle Paarungen in die Tabelle **tblSpielpaarungen** ein. Das ist einfach: Wir brauchen ja nur zwei Recordsets mit den Mannschaften der Liga/Spielzeit in zwei verschachtelten **Do While**-Schleifen zu durchlaufen und darin jeweils die Heim- und die Auswärtsmannschaft in die Felder der Tabelle **tblSpielpaarungen** einzutragen. Dabei müssen wir nur prüfen, ob die Heim- und die Auswärtsmannschaft identisch sind und in diesem Fall keinen Datensatz anlegen.

Programmierung der Spielplangenerierung

Wir beginnen mit dem Anlegen der Spiele einer Liga. Dazu verwenden wir die Prozedur aus Listing 1. Die Prozedur erwartet die **LigaSpieljahrID** für die Liga, für deren Mannschaften der Spielplan angelegt werden soll. Wir haben zunächst eine kleine Liga angelegt, die nur vier Mannschaften enthält.

Die Prozedur verwendet drei Recordset-Objekte: **rstSpielpaarungen** liefert die Daten der Tabelle **tblSpielpaarungen** und **rstHeimmannschaften** und **rstAuswaertsmannschaften** jeweils die Daten des Feldes **MannschaftID** der Tabelle **tblLigenSpieljahreMannschaften**. Dabei werden die Daten dieser beiden Recordsets nach dem Wert des Parameters **lngLigaSpieljahrID** gefiltert.

In einer äußeren Schleife durchlaufen wir die Elemente des Recordsets **rstHeimmannschaften**, in einer inneren die Elemente aus **rstAuswaertsmannschaften**. Dabei stellen wir den Datensatzzeiger der inneren Schleife immer wieder mit **rst.MoveFirst** auf den ersten Datensatz zurück, wenn wir alle Auswärtsmannschaften für die aktuelle Heimmannschaft durchlaufen haben. Innerhalb der Schleife prüfen wir dann, ob die aktuelle Heimmannschaft gleich der aktuellen Auswärtsmannschaft ist. In diesem Fall legen wir keine neue Spielpaarung an. Anderenfalls fügt die Prozedur einen neuen Datensatz zum Recordset rstSpielpaarungen hinzu und trägt die jeweiligen IDs

```
Public Sub PaarungenAnlegen(lngLigaSpieljahrID As Long)
    Dim db As DAO.Database
    Dim rstSpielpaarungen As DAO.Recordset
    Dim rstHeimmannschaften As DAO.Recordset
    Dim rstAuswaertsmannschaften As DAO.Recordset
    Set db = CurrentDb
    Set rstSpielpaarungen = db.OpenRecordset("SELECT * FROM tblSpielpaarungen", dbOpenDynaset)
    Set rstHeimmannschaften = db.OpenRecordset("SELECT MannschaftID FROM tblLigenSpieljahreMannschaften " _
        & "WHERE LigaSpieljahrID = " & lngLigaSpieljahrID, dbOpenDynaset)
    Set rstAuswaertsmannschaften = db.OpenRecordset("SELECT MannschaftID FROM tblLigenSpieljahreMannschaften " _
        & "WHERE LigaSpieljahrID = " & lngLigaSpieljahrID, dbOpenDynaset)
    Do While Not rstHeimmannschaften.EOF
        Do While Not rstAuswaertsmannschaften.EOF
            If Not rstHeimmannschaften!ID = rstAuswaertsmannschaften!ID Then
                rstSpielpaarungen.AddNew
                rstSpielpaarungen!HeimmannschaftID = rstHeimmannschaften!ID
                rstSpielpaarungen!AuswaertsmannschaftID = rstAuswaertsmannschaften!ID
                rstSpielpaarungen.Update
            End If
            rstAuswaertsmannschaften.MoveNext
        Loop
        rstAuswaertsmannschaften.MoveFirst
        rstHeimmannschaften.MoveNext
    Loop
End Sub
```

Listing 1: Anlegen von Spielpaarungen

für die Felder **HeimmannschaftID** und **AuswaertsmannschaftID** ein. Danach sieht die Tabelle **tblSpielpaarungen** für vier Mannschaften etwa wie in Bild 1 aus.

Nun brauchen wir hier nur noch die Spieltage einzutragen. Und das wird eine interessante Aufgabe. Die Grundidee ist: Wir beginnen mit dem ersten Spieltag und suchen uns für diesen Paarungen heraus. Für das erste Spiel am ersten Spieltag kein Problem: Hier können wir einfach das erste Spiel auswählen. Danach wollen wir das erste Spiel aus der Tabelle tblSpielpaarungen suchen, dessen Heimmannschaft und dessen Auswärtsmannschaft nicht am ersten

tblSpielpaarungen

ID	Spieldatum	HeimmannschaftID	AuswaertsmannschaftID	T	SpieltagID
39		FC Bayern München	Borussia Dortmund		
40		FC Bayern München	Eintracht Frankfurt		
41		FC Bayern München	TSG Hoffenheim 1899		
42		Borussia Dortmund	FC Bayern München		
43		Borussia Dortmund	Eintracht Frankfurt		
44		Borussia Dortmund	TSG Hoffenheim 1899		
45		Eintracht Frankfurt	FC Bayern München		
46		Eintracht Frankfurt	Borussia Dortmund		
47		Eintracht Frankfurt	TSG Hoffenheim 1899		
48		TSG Hoffenheim 1899	FC Bayern München		
49		TSG Hoffenheim 1899	Borussia Dortmund		
50		TSG Hoffenheim 1899	Eintracht Frankfurt		
(Neu)					

Datensatz: 1 von 12 | Kein Filter | Suchen

Bild 1: Spielpaarungen einer Vierer-Liga

Spiel beteiligt sind. Soweit kein Problem – der erste Spieltag könnte so aussehen:

```
FC Bayern München - Borussia Dortmund
Eintracht Frankfurt - TSG Hoffenheim 1899
```

Für den zweiten Spieltag können wir dann eines der verbleibenden Spiele als Startpunkt nehmen, und zwar das nächste verfügbare in der Tabelle. Hier also:

```
FC Bayern München - Eintracht Frankfurt
```

Dazu passt dann:

```
Borussia Dortmund - TSG Hoffenheim 1899
```

Damit hat Bayern München allerdings direkt zwei Heimspiele und Hoffenheim gleich zwei Auswärtsspiele. Aber gut, das lässt sich bei einer Vierergruppe nicht vermeiden. Würden wir eine der beiden Paarungen umdrehen, hätten die beiden anderen Mannschaften entweder zweimal zuhause oder auswärts gespielt. Für den dritten Spieltag erhalten wir dann, wenn wir wieder das nächste noch nicht an einen Spieltag vergebene Spiel holen und dann das nächste Spiel mit den übrigen Mannschaften:

```
FC Bayern München - TSG Hoffenheim 1899
Borussia Dortmund - Eintracht Frankfurt
```

Spätestens hier ist die Verteilung von Heim- und Auswärtsspielen auffällig, denn FC Bayern München hätte nun drei Heimspiele in Folge.

Uns wurde schnell klar, dass es ein sehr aufwendiges Unterfangen werden würde, selbst mit Verfahren wie Rekursion eine programmgesteuerte Methode zum Erstellen eines Spielplans mit beispielsweise 18 Mannschaften zu erstellen.

Algorithmus nachbauen

Also haben wir im Internet geschaut, ob es nicht Informationen gibt, wie so ein Bundesligaplan zusammengestellt wird. Und Wikipedia hat unter **https://de.wikipedia.org/wiki/Spielplan_(Sport)** eine passende Erläuterung präsentiert, die wir nun in Form einer Prozedur für unser Datenmodell umsetzen wollen.

Hier lautet es zunächst:

»Im ersten Schritt wird dazu jeder der 18 Mannschaften außer einer, dem „Joker“ (in diesem Fall dem Hamburger SV) eine zufällige Zahl zwischen 1 und 17 zugeordnet, woraufhin der „Joker“ selbst die Nummer 18 erhält.«

Das ergibt eine Liste mit 17 Mannschaften:

```
1 Bayern München
2 Borussia Dortmund
...
17 Union Berlin
```

Weiter lautet es:

»Am Spieltag n (hier: $1 \leq n \leq 17$) tritt der Verein mit der Nummer k gegen genau die gegnerische Mannschaft mit der Nummer l an, so dass der Rest der Summe beider Nummern k + l bei Division durch 17 genau den Rest n ergibt. Der „Joker“ mit der Nummer 18 (in diesem Fall also der Hamburger SV) tritt schließlich gegen den Verein an, für den sich nach obiger Regel an dem betreffenden Spieltag kein Partner findet.«

Und schließlich für die Entscheidung, ob eine Mannschaft daheim oder auswärts antritt:

»Um zu ermitteln, welche Mannschaft Heimrecht hat, gilt in der Hinrunde folgendes: Ist die Summe der beiden Nummern k + l eine gerade Zahl, hat die Mannschaft mit der höheren Nummer Heimrecht; ist die Summe dagegen eine ungerade Zahl, hat die Mannschaft mit der niedrigeren Nummer Heimrecht. Der „Joker“ mit der Nummer 18 schließlich hat nur Heimrecht gegen die Mannschaften mit den Nummern 8 bis 16, nicht aber 1 bis 7 und 17.«

Schauen wir, wie wir das in Programmform umsetzen können.

Als Erstes brauchen wir eine Liste der Mannschaften mit dem Index von 1 bis 18. Wir haben kein entsprechendes Feld in der Tabelle **tblMannschaften** vorgesehen, was auch keinen Sinn macht, da die Mannschaften ja in verschiedenen Spieljahren verschiedenen Ligen zugeordnet werden sollen. Viel praktischer wäre es, wenn wir in der Tabelle **tblLigenSpieljahreMannschaften** ein Feld hätten, wo wir jeder Mannschaft in der Zuordnung zu Liga/Spieljahr eine Nummer vergeben können.

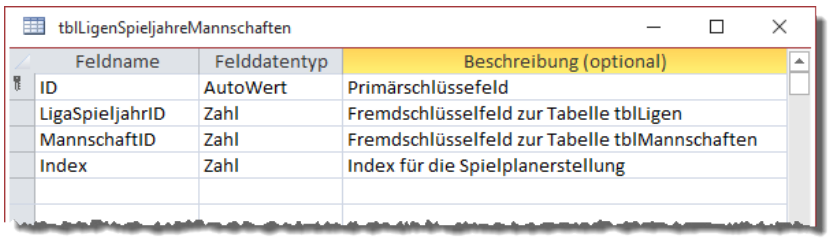


Bild 2: Neues Feld in der Tabelle **tblLigenSpieljahreMannschaften**

Wir könnten zwar auch einfach die Mannschaften einer Liga einlesen und diesen dann die Zahlen von 1 bis 18 zuweisen, etwa in einer Collection, aber wenn wir die Zahlen direkt in der Tabelle **tblLigenSpieljahreMannschaften** speichern, kann der Benutzer auch die Zahlenzuordnung ändern und den Spielplan neu generieren, wenn dieser nicht seinen Anforderungen entspricht. Die Tabelle sieht nach dieser Änderung in der Entwurfsansicht wie in Bild 2 aus.

Bild 3: Neues Feld im Formular **frmLigenSpieljahreMannschaften**

Das neue Feld fügen wir auch zum Formular **frmLigenSpieljahreMannschaften** hinzu (siehe Bild 3).

Damit können wir nun in die Prozedur für die Erstellung des Spielplans einsteigen. Den ersten Teil der Aufgabe haben wir schon gelöst: Wir haben jeder Mannschaft eine Zahl zwischen 1 und 17 zugeteilt, und auch die 18 wurde vergeben.

Damit beginnen wir mit der Prozedur zum Anlegen der Mannschaften (erster Teil siehe Listing 2). Dieser übergeben wir mit dem Parameter **lngLigaSpieljahrID** den Primärschlüsselwert der Tabelle **tblLigenSpieljahre** und somit einen Hinweis auf Liga/Spielzeit, für welche die Paarungen angelegt werden sollen. Nach der Deklaration einiger Variablen, die wir erläutern, wenn Sie zum Einsatz kommen, erstellt die Prozedur einen Verweis auf das **Database**-Objekt der aktuellen Datenbank.

Dann legt die Prozedur zwei Recordsets an – jeweils für die Mannschaften, die an der Saison beteiligt sind und die wir aus der Tabelle **tblLigenSpieljahreMannschaften** beziehen. Dabei verwenden wir **lngLigaSpieljahrID** als Filter. Wir sortieren die Recordsets jeweils nach dem frisch hinzugefügten Index-Feld, das Sie gegebenenfalls zuvor noch füllen müssen.

Dann ermitteln wir die Werte für ein paar Variablen. **lngSpielpaarungen** nimmt die Anzahl der Spielpaarungen je Spieltag auf, ermittelt durch die Anzahl der Mannschaften durch zwei. **lngSpieltage** soll die Anzahl der Spieltage enthalten. Diese ermitteln wir über die Anzahl der Mannschaften minus eins, in der Bundesliga also etwa 18. Die Anzahl der Mannschaften aus der Anzahl der Einträge des Recordsets **rstMannschaften** speichern wir in der Variablen **lng-**

```
Public Sub SpielplanErstellen(lngLigaSpieljahrID As Long)
    Dim db As dao.Database, i As Integer, lngMannschaften As Long
    Dim rstMannschaften1 As dao.Recordset. rstMannschaften2 As dao.Recordset, rstSpieltage As dao.Recordset
    Dim lngSpielpaarungen As Long, lngSpieltag As Long, bolPaarungGefunden As Boolean
    Dim colMannschaften As Collection
    Dim lngSpieltage As Long, bolHinzufuegen As Boolean, lngHeim As Long
    Dim lngAuswaerts As Long, bolRueckrunde As Boolean
    Set db = CurrentDb
    Set rstMannschaften1 = db.OpenRecordset("SELECT * FROM tblLigenSpieljahreMannschaften WHERE LigaSpieljahrID = " _
        & lngLigaSpieljahrID & " ORDER BY Index", dbOpenSnapshot)
    Set rstMannschaften2 = db.OpenRecordset("SELECT * FROM tblLigenSpieljahreMannschaften WHERE LigaSpieljahrID = " _
        & lngLigaSpieljahrID & " ORDER BY Index", dbOpenSnapshot)
    lngSpielpaarungen = rstMannschaften1.RecordCount / 2
    lngSpieltage = rstMannschaften1.RecordCount - 1
    lngMannschaften = rstMannschaften1.RecordCount
    Set rstSpieltage = db.OpenRecordset("SELECT * FROM tblSpieltage WHERE LigaSpieljahrID = " & lngLigaSpieljahrID, dbOpenDynaset)
    Do While Not rstSpieltage.EOF
        lngSpieltag = rstSpieltage!Spieltag
        If lngSpieltag > lngSpieltage Then
            bolRueckrunde = True
            lngSpieltag = 1
        End If
        db.Execute "DELETE FROM tblSpielpaarungen WHERE SpieltagID = " & rstSpieltage!ID, dbFailOnError
        rstMannschaften1.MoveFirst
        rstMannschaften2.MoveFirst
        Set colMannschaften = New Collection
        For i = 1 To lngSpielpaarungen
            Do While Not bolPaarungGefunden
                If ((rstMannschaften1!Index + rstMannschaften2!Index) Mod lngSpieltage = lngSpieltag Or (lngSpieltag = _
                        lngSpieltage And (rstMannschaften1!Index + rstMannschaften2!Index) Mod lngSpieltage = 0))
                        And rstMannschaften1!Index <> rstMannschaften2!Index Then
                    bolNochNichtHinzugefuegt = NochNichtHinzugefuegt(colMannschaften, rstMannschaften1!Index, _
                        rstMannschaften2!Index)
                    If bolNochNichtHinzugefuegt = True Then
                        HeimAuswaerts rstMannschaften1!Index, rstMannschaften2!Index, lngHeim, lngAuswaerts, bolRueckrunde
                        SpielpaarungAnlegen db, lngHeim, lngAuswaerts, rstSpieltage!ID, lngLigaSpieljahrID
                        colMannschaften.Add rstMannschaften1!Index.Value
                        colMannschaften.Add rstMannschaften2!Index.Value
                        rstMannschaften2.MoveFirst
                        rstMannschaften1.MoveNext
                        bolPaarungGefunden = True
                        Exit Do
                    End If
                    rstMannschaften1.MoveNext
                    rstMannschaften2.MoveFirst
                End If
```

Listing 2: Spielpaarungen ermitteln, Teil 1

Verweise beim Start prüfen

Beim Einsatz von Access-Anwendungen gibt es manchmal Probleme, weil die Verweise auf Bibliotheken nicht korrekt gesetzt sind. Dann werden plötzlich VBA-Funktionen wie »Left« oder »Right« als fehlerhaft bewertet. Der Benutzer steht dann dumm da, weil er nicht weiß, was zu tun ist. Dieser Beitrag zeigt, wie Sie das Problem nachstellen können und welche Lösungsmöglichkeiten es gibt – von der manuellen Variante bis zur automatischen Korrektur des Verweises beim Start der Anwendung.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1904_VerweiseBeimStart.accdb**.

Fehlerhafter Verweis

Probleme wie die oben genannten treten meist auf, wenn Sie eine Anwendung auf Ihrem Rechner entwickeln und diese dann an einen Benutzer weitergeben, auf dessen Rechner nicht alle per Verweis referenzierten Bibliotheken vorliegen – oder diese nicht in der richtigen Version vorliegen.

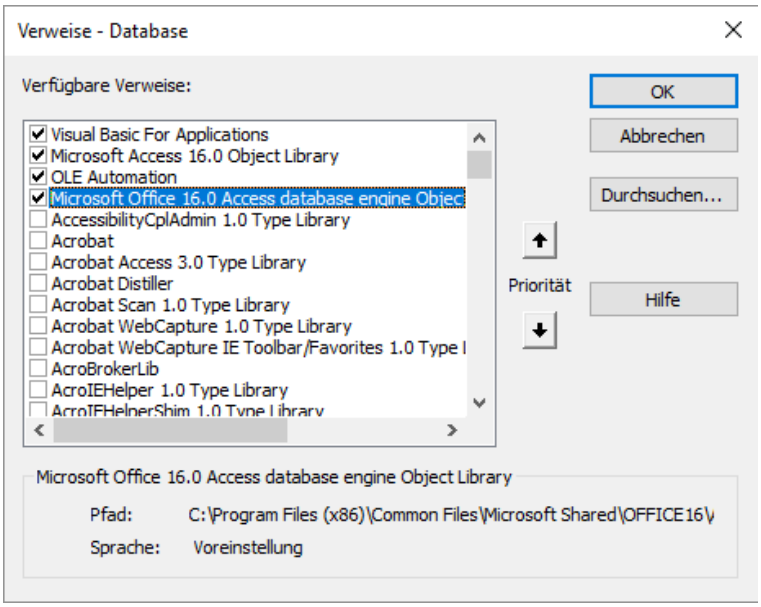


Bild 1: Der **Verweise**-Dialog von Access

Verweise finden Sie grundsätzlich im **Verweise**-Dialog, den Sie vom VBA-Editors aus (zu öffnen mit **Strg + G** oder **Alt + F11**) mit dem Menübefehl **Extras|Verweise** anzeigen. Dieser Dialog zeigt die Liste der eingebundenen Bibliotheken, die bei einer neuen, leeren Access-Anwendung auf Basis von Access 2016 etwa wie in Bild 1 aussieht. Die Bibliotheken hinter diesen vier Einträge sollten überall, wo Access 2016 installiert ist, auch vorhanden sein.

Bei umfangreicheren Anwendungen kommen meist noch Verweise auf weitere Bibliotheken hinzu, zum Beispiel um Word, Excel oder Outlook zu automatisieren, die Datenzugriffs-Bibliothek ADODB zu nutzen oder auch um die Befehle für die Verarbeitung von XML hinzuzufügen.

Wenn Sie nun einen Verweis auf Outlook hinzufügen, die Datenbank auf den Zielrechner übertragen und dann aber Outlook gar nicht auf dem Zielrechner vorliegt, gibt es Probleme wegen des fehlenden Verweises.

Manchmal gibt es auch Probleme, wenn Sie mit einem Verweis auf die Objektbibliothek von Outlook 2016 programmieren und auf dem Zielrechner ist eine ältere Version von Outlook vorhanden.

Manuelle Lösung

Vorausgesetzt, die Bibliothek, auf die der Verweis zeigen soll, ist auf dem Zielrechner installiert und registriert, können Sie wie folgt vorgehen, um den Verweis zu erneuern:

- Öffnen Sie den **Verweise**-Dialog.

- Entfernen Sie den mit **NICHT VORHANDEN** markierten Verweis.
- Fügen Sie den Verweis auf die gewünschte Bibliothek zur Liste der Verweise hinzu, indem Sie den entsprechenden Eintrag per Haken selektieren.
- Schließen Sie den **Verweise**-Dialog.

Danach sollte die Anwendung dann wie gewohnt funktionieren. Eine Ausnahme könnte sein, dass Sie in Ihrer Anwendung tatsächlich Funktionen einer neueren Bibliothek verwenden, die in der älteren, auf dem Zielrechner vorhandenen Version noch nicht verfügbar ist.

Wenn Sie eine Anwendung beim Kunden laufen haben, für die Sie regelmäßig neue Versionen liefern, dann ist es für den Kunden natürlich mühselig, immer die Verweise zu aktualisieren – daher stellen wir nun eine Alternative vor.

Verweise automatisch beim Start aktualisieren

Die Alternative lautet, beim Start einer Anwendung die Verweise zu prüfen und diese gegebenenfalls zu erneuern. Wenn ein Verweis nicht gesetzt werden kann, weil die entsprechende Bibliothek nicht verfügbar ist, soll eine Meldung erscheinen, die den Benutzer darüber informiert, damit dieser mit dem Entwickler der Software herausfinden kann, wo das Problem in diesem Fall liegt.

Zu Beispielzwecken nehmen wir an, dass wir Verweise auf die Bibliotheken für den Zugriff auf die Office-Anwendungen Excel, Outlook und Word hinzufügen sowie einen auf die Bibliothek **Microsoft XML, v6.0** (siehe Bild 2).

Bei den mit diesen Verweisen referenzierten Bibliotheken handelt es sich sämtlich um registrierte Bibliotheken, für die es eine GUID gibt. Diese wollen wir nun zunächst ermitteln, indem wir diese per VBA-Routine im Direktbereich des VBA-Editors ausgeben:

```
Public Sub VerweisePruefenUndErneuern()
    Dim objReference As Reference
    Dim strGUID As String
    For Each objReference In References
        Debug.Print objReference.Name
        Debug.Print objReference.FullPath
        Debug.Print objReference.Guid
    Next objReference
End Sub
```

Das Ergebnis sieht dann so aus:

```
VBA
C:\Program Files (x86)\...\VBA7.1\VBE7.DLL
{000204EF-0000-0000-C000-000000000046}
Access
C:\Program Files (x86)\...\MSACC.OLB
{4AFFC9A0-5F99-101B-AF4E-00AA003F0F07}
stdole
C:\Windows\SysWOW64\stdole2.tlb
{00020430-0000-0000-C000-000000000046}
DAO
C:\Program Files (x86)\...\OFFICE16\ACEDAO.DLL
```

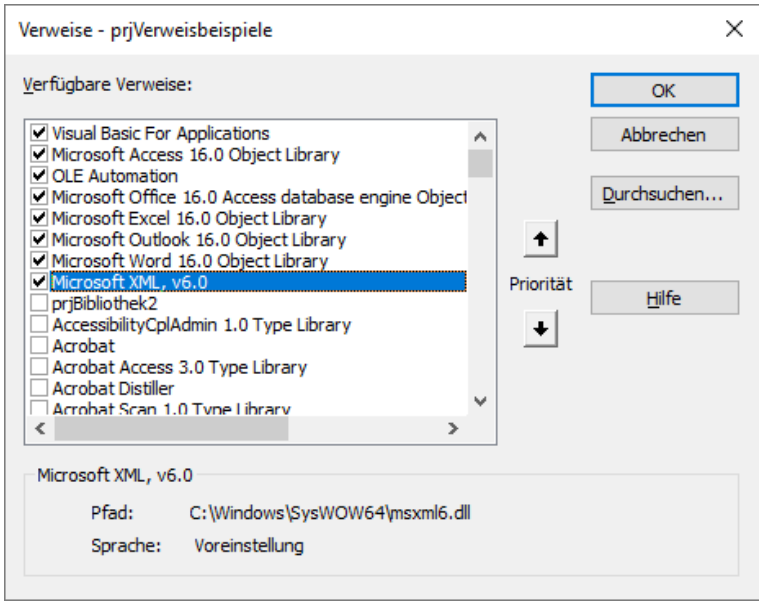


Bild 2: **Verweise**-Dialog mit zusätzlichen Verweisen

```
{4AC9E1DA-5BAD-4AC7-86E3-24F4CDCECA28}
Excel
C:\Program Files (x86)\...\EXCEL.EXE
{00020813-0000-0000-C000-000000000046}
Outlook
C:\Program Files (x86)\...\MSOUTL.OLB
{00062FFF-0000-0000-C000-000000000046}
Word
C:\Program Files (x86)\...\MSWORD.OLB
{00020905-0000-0000-C000-000000000046}
MSXML2
C:\Windows\SysWOW64\msxml6.dll
{F5078F18-C551-11D3-89B9-0000F81FE221}
```

Nun wollen wir ja gar nicht alle Einträge prüfen, sondern nur diejenigen, die nicht eingebaut sind. Das prüfen wir durch eine **If...Then**-Bedingung, mit der wir den Wert der Eigenschaft **BuiltIn** untersuchen. Ist diese **True**, handelt es sich um eine eingebaute Eigenschaft, die wir nicht ersetzen müssen, da diese meist vorhanden sind. Hat **BuiltIn** den Wert **False**, müssen wir handeln: Dann könnte es sein, dass ein falscher Verweis zu Fehlern führt. Mit der folgenden geänderten Version der Prozedur **VerweisePruefenUndErneuern** geben wir nur noch die Verweise aus, die wir erneuern können:

```
Public Sub VerweisePruefenUndErneuern()
    Dim objReference As Reference
    Dim strGUID As String
    For Each objReference In References
        If objReference.BuiltIn = True Then
            Debug.Print objReference.Name
            Debug.Print objReference.FullPath
            Debug.Print objReference.Guid
        End If
    Next objReference
End Sub
```

Damit erhalten wir dann ein ähnliches Ergebnis wie weiter oben, nur dass diesmal die eingebauten Verweise nicht berücksichtigt werden.

Was fangen wir nun damit an? Wir könnten direkt in dieser Prozedur ein paar Zeilen unterbringen, die dafür sorgen, dass die Verweise auf Gültigkeit überprüft und gegebenenfalls erneuert werden. Das würde dann etwa wie in Listing 1 aussehen. Hier durchlaufen wir wieder alle **Reference**-Objekte, schließen aber durch die ersten beiden **If...Then**-Bedingungen zuerst die eingebauten und dann die vorhandenen Verweise aus.

```
Public Sub VerweisePruefenUndErneuern()
    Dim objReference As Reference
    Dim objReferenceNew As Reference
    Dim strReference As String
    Dim strGUID As String
    For Each objReference In References
        If objReference.BuiltIn = False Then
            If objReference.IsBroken = True Then
                strGUID = objReference.Guid
                strReference = objReference.Name
                References.Remove objReference
                Set objReferenceNew = References.AddFromGuid(strGUID, 0, 0)
                If objReference Is Nothing Then
                    MsgBox "Der Verweis auf die Bibliothek '" & strReference & "' fehlt."
                End If
            End If
        End If
    Next objReference
End Sub
```

Listing 1: Fehlende Verweise werden direkt erneuert

Sind beide **If...Then**-Bedingungen wahr, merken wir uns in den Variablen **strGUID** und **strReference** die GUID und den Namen der referenzierten Bibliothek. Dann fügen wir die Referenz auf eine Bibliothek über die Methode **AddFromGUID** wieder zur Liste **References** hinzu. Dabei

übergeben wir als Parameter die GUID der Bibliothek sowie den Wert **0** für die beiden Parameter **Major** und **Minor**.

Eine fehlende Bibliothek sollte damit wieder referenziert werden können.

Diese Prozedur rufen wir am besten beim Start der Anwendung auf, damit keine Fehler durch fehlende Verweis auftreten. Wie das gelingt, zeigen wir am Ende des Beitrags.

Benötigte Verweise an anderer Stelle speichern

Das Problem ist nur: Wenn der Benutzer auf irgendeine Weise einen der Verweise entfernt, ohne diesen wieder hinzuzufügen, treten in der Folge Probleme auf an den Stellen, wo im Code Elemente dieser Bibliothek verwendet werden.

Wenn wir dann beim Starten der Anwendung die Prozedur aufrufen, durchläuft diese wieder alle Verweise und prüft die Verweise mit der Eigenschaft **IsBroken**. Wenn jedoch ein Verweis auf eine Bibliothek gar nicht mehr in der Liste der Verweise enthalten ist, kann er mit der hier vorliegenden Version der Prozedur auch nicht ersetzt werden – die Prozedur erfährt ja gar nicht, das ein Verweis fehlt. Das wird erst später offensichtlich, wenn die Anwendung das erste Mal auf Elemente der mit dem Verweis referenzierten Bibliothek zugreift und dadurch Laufzeitfehler ausgelöst werden.

Also wollen wir, bevor wir die Anwendung an den Benutzer weitergeben, die Verweise, die überprüft werden sollen, in irgendeiner Form speichern. Dabei gibt es zwei Möglichkeiten:

- Wir speichern die GUIDs fest im Code.
- Wir legen eine Tabelle an, der wir die Informationen über die Verweise hinzufügen. Diese Tabelle sollte dann im Frontend der Datenbankanwendung liegen, wenn diese in Frontend und Backend aufgeteilt ist.

Die Basis für beide Varianten sieht beispielsweise wie in Listing 2 aus. Diese Funktion nimmt den Parameter **strGUID** entgegen. Im Direktbereich könnten wir die Funktion wie folgt aufrufen:

```
? VerweiseErneuern("{00062FFF-0000-
0000-C000-000000000046}")
```

Die Funktion stellt die Variablen **bolReferenceBroken** auf **False** und **bolReferenceMissing** auf **True** ein. Dann versucht sie, einen Verweis auf die Bibliothek mit der mit **strGUID** übergebenen GUID zu finden. Dies geschieht in einer **For Each**-Schleife über alle Elemente der Auflistung **References**. Wenn der aktuell mit **objReference** referenzierte Verweis die gesuchte GUID enthält, können wir die Variable **bolReferenceMissing** schon einmal auf **False** einstellen, denn der Verweis ist ja zumindest vorhanden.

Dann prüft die Prozedur mit der Eigenschaft **IsBroken** noch, ob der Verweis als **NICHT VORHANDEN** markiert ist. Das ist etwas missverständlich, denn **NICHT VORHANDEN** gibt ja nicht an, dass der Verweis fehlt, sondern die referenzierte Bibliothek. Jedenfalls stellt die Funktion die Variable **bolReferenceBroken** auf **True** ein, wenn **IsBroken** den Wert **True** aufweist. Sobald der Verweis passend zu **strGUID** gefunden wurde, verlassen wir dann auch mit **Exit For** die Schleife.

objReference enthält somit auch einen Verweis auf das **Reference**-Objekt. Damit gehen wir in eine **If... Then**-Bedingung, in der wir prüfen, ob entweder **bolReferenceMissing** oder **bolReferenceBroken** den Wert **True** hat. Das ist nun entweder der Fall, wenn der Verweis gar nicht vorhanden ist oder dieser »broken« ist.

Dann versucht die Funktion, den Verweis mit der **Remove**-Methode der **References**-Auflistung zu entfernen und diesen dann über die Methode **AddFromGUID** erneut hinzuzufügen, wobei ein Verweis auf das neue **Reference**-Objekt in der Variablen **objReference** landet – aber auch nur, wenn das Hinzufügen erfolgreich war.