KOMPENDIUM

# ACCESS BASICS

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




# KOMPENDIUM

**ALLE AUSGABEN VON 2010-2017**

**2.000 SEITEN ACCESS-KNOW-HOW**

**280 BEISPIELDATENBANKEN**

www.access-basic.de

André Minhorst Verlag

**André Minhorst – Access [basics] Kompendium**

**ISBN 978-3-944216-05-8**
© 2018 André Minhorst Verlag, Borkhofer Straße 17, 47137 Duisburg/Deutschland
1. Auflage 2018
**Lektorat** 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 24
Kunden verwalten 49
Entwurf einer einfachen Kundentabelle 54
Formulare für einfache Tabellen 57
Datenbankfenster und Navigationsbereich im Detail 64
Kunden und Projekte verwalten 69
Einfache Auswahlabfragen kennenlernen 78
Darstellung von 1:n-Beziehungen 85
Formulare für Daten aus 1:n-Beziehungen 88
Anreden und Co. mit Wertlisten oder Lookup-Tabellen verwalten 95
Einfache Tabelle in der Detailansicht 99
Anfügeabfragen 104
Daten aus CSV-Dateien importieren 111
Mitarbeiter per m:n-Beziehung zu Projektteams zusammenstellen 119
m:n-Beziehung per Unterformular 125
Fremdschlüsselfelder mit Nachschlage-Funktion: Pro und Kontra 130
Vom Textfeld zur Lookuptabelle 134
Formularereignisse nutzen 139
Daten aus verknüpften Tabellen darstellen 144
Einführung in VBA 148
Formulare öffnen und schließen 154
Die Schaltfläche 161
Das Listenfeld 164
Löschabfragen 172
Eigene Routinen schreiben 175
If...Then- und Select Case-Bedingungen 178
Filtern von Textfeldern 184
Einsatz von Parametern 189
Filtern nach Zahlen 191
Variablen in VBA 196
m:n-Beziehungen in Listenfeldern darstellen 200
Felder und ihre Eigenschaften 208
Daten zusammenführen mit UNION-Abfragen 217
Das Textfeld 221
Das Kombinationsfeld 225
Datensatzauswahl per Kombinationsfeld 230
Eigenschaften von Tabellen 237

Der Ausdrucks-Generator 240
Berechnete Felder in Abfragen und Eigenschaften 245
Benutzerdefinierte Funktionen in Abfragen 249
Domänenfunktionen 251
Abfrageeigenschaften 258
Adressblöcke in Berichten 261
Bedingungen: Wenn, Schalter und Wahl 266
Abhängige Kombinationsfelder 268
Standardwerte statisch und dynamisch einstellen 272
Projektzeiterfassung - Aufgaben und Tätigkeiten verwalten 274
Indizierung von Feldern 283
Duplikate per Abfrage finden 288
Einfache Tabelle in der tabellarischen Ansicht darstellen 290
Darstellung von 1:n-Beziehungen per Gruppierung 295
Datum und Zeit, Teil I 299
Validieren von Benutzereingaben 305
Access-Funktionen: Datum und Zeit 313
Abfragen: Datum und Zeit 319
Schleifen in VBA 322
Werte zu Kombinationsfeldern hinzufügen 327
Aktionsabfragen per VBA ausführen 333
Daten per Kombinationsfeld auswählen und löschen 338
Meldungsfenster anzeigen und auswerten 340
Daten in der Endlosansicht von Formularen anzeigen 343
CSV-Export für Adressdaten 348
Im- und Export-Spezifikationen definieren und verwenden 354
Datenbanken aufteilen 357
Datenbanken im Mehrbenutzerbetrieb 361
Tabellenverknüpfungen pflegen 364
Aktion beim Start: Das Autoexec-Makro 370
Dialog zur Auswahl von Dateien anzeigen 372
Fehler unter VBA 375
Fehlerbehandlung unter VBA 379
Lookup-Daten per Formular verwalten 385
Recordsets in der Datenblattansicht 392
Formular- und Berichtsvorlagen 396
Datensätze in Unterformularen anlegen 400
Fehlerbehandlung in der Praxis 403
Fehlerbehandlung in Formularen 408
Kombinationsfeld als Eingabehilfe für Textfelder 414

Optionsgruppen und Optionsfelder 417
Berichtsbereiche 423
Sortieren in Berichten 426
Tipps und Tricks: Standardverzeichnis einstellen 429
Tipps und Tricks: Objekte ausblenden 429
Tipps und Tricks: Ausgeblendete Objekte anzeigen 430
Tipps und Tricks: Aktuelles Datum und Zeit blitzschnell 430
Tipps und Tricks: Wert aus vorherigem Datensatz übernehmen 431
Tipps und Tricks: Datum und Zeit im Formulartitel 431
Tipps und Tricks: Memofeld als Sortierungs- und Gruppierungsfeld 432
Tipps und Tricks: Datenbankkennwort vergeben 432
Tipps und Tricks: Textbausteine verwenden 433
Datenblattansicht im Griff 435
Aktuellen Datensatz kopieren 439
Formulare und Steuerelemente referenzieren 441
Berichte öffnen 445
DAO: Datenbank im Griff mit dem Database-Objekt 450
For Each-Schleifen 453
Bedingte Formatierung 457
Mehrspaltige Berichte 462
Zeichenketten-Funktionen 467
Zeilenumbruch und Tabulator in Access 472
DAO: Recordsets öffnen, lesen und bearbeiten 473
Mehrfachauswahl in Listenfeldern auslesen 479
DAO: Recordsets filtern, durchsuchen und sortieren 483
SQL: SELECT-Abfragen, Teil I 487
Kunden per Bezeichnung verwalten 493
SQL: SELECT-Abfragen, Teil II 500
Einfache Suchfunktion für Formulare 506
Arrays 517
Kommunikationsdaten im Griff 523
Bestellungen in Tabellen verwalten 528
Bestellungen in Formularen verwalten 533
Rechnen in Formularen 540
Rechnen in Berichten 545
Gruppierungen in Berichten 550
Bericht als PDF speichern 558
Tipps und Tricks zu Berichten: Duplikate ausblenden 561
Tipps und Tricks zu Berichten: Vergrößerbar 562
Tipps und Tricks zu Berichten: Verkleinerbar 562

Tipps und Tricks zu Berichten: Berichte zusammenfassen 564
Daten für die Ausgabe formatieren 566
Die Format-Funktion 570
Farbige und vom Wert abhängige Formatierung 575
Listenfeldauswahl speichern und wiederherstellen 577
Ereignisse in Formularen, Teil 1: Reihenfolge 582
Null-Werte und wie man damit umgeht 587
Gruppierungen in Abfragen 590
Kreuztabellenabfragen 595
Schnellsuche im Listenfeld 601
Ereignisse in Formularen: Öffnen und Schließen 605
Kreuztabellenabfragen nach Datum 610
Dateibefehle 613
Textdateien schreiben und lesen 617
Rechnungsbericht 623
Die Dir-Funktion 629
Binärzahlen in VBA-Eigenschaften 631
Kombinations- und Listenfelder mit Wertliste füllen 635
Kombinations- und Listenfelder per Callback füllen 638
Das ListView-Steuerelement - Grundlagen 644
Access und Excel: Import und Export 650
Die Umschaltfläche 657
ListView-Steuerelement mit Daten füllen 663
Zugriff auf Daten im ListView-Steuerelement 667
AutoKeys: Globale Tastenkombinationen 673
Access und Excel: Verknüpfungen 676
Beispieldaten-Assistent 679
Early Binding und Late Binding 684
ListView-Steuerelement: Daten bearbeiten 688
Datensätze individuell auswählen 693
Synchrone Unterformulare 698
Internetmarke mit Access nutzen 703
Das Registersteuerelement 714
ListView-Steuerelement: Kontrollkästchen nutzen 719
Aktionsabfragen per VBA 722
Dynamische Datenherkunft mit Formularbezug 725
Internetmarke mit Access nutzen, Teil II 729
Das Registersteuerelement in der Praxis 734
SQL-Grundlagen: Aktionsabfragen 741
DAO: QueryDefs und Parameterabfragen unter VBA 745

ListView-Steuerelement: Drag and Drop, Teil I 749
Programmieren mit Arrays 758
Reihenfolge individuell einstellen 763
Feiertage verwalten, Teil 1 771
Debugging im VBA-Editor, Teil 1 780
ListView-Steuerelement: Drag and Drop, Teil II 785
Feiertage verwalten, Teil 2 793
Tipps und Tricks: Ereignisprozeduren schneller anlegen 798
Tipps und Tricks: Monatsnamen und Wochentage 799
Tipps und Tricks: Texte in leeren Feldern vorblenden 799
Steuerelementlayouts mit Access 2007 und jünger 801
Balkendiagramme mit Bordmitteln 808
Haupt- und Unterformular synchron 814
Debugging im VBA-Editor, Teil 2 817
Filterkriterien für Formulare, Teil I 823
Textfelder: Value, OldValue und Text 831
Änderungen protokollieren, Teil I 835
Berichte per E-Mail verschicken, Teil I 840
Filterkriterien für Formulare, Teil II 846
E-Mails mit Outlook versenden 854
Berichte per E-Mail versenden, Teil II 860
E-Mails sicher versenden 869
Ereignisse abgreifen 874
Filterkriterien für Formulare, Teil III: Kombinationsfelder 878
Textfelder im Griff, Teil II: Markierungen 884
Rekursion mit VBA 891
Platzhalter in Texten ersetzen 898
Berichte per E-Mail verschicken, Teil III 903
Das TreeView-Steuerelement: Grundlagen, Teil I 906
Das TreeView-Steuerelement: Grundlagen, Teil II 913
Bilder in Access, Teil I 921
Das Anlage-Feld 927
Anlage-Felder und VBA 934
Das TreeView-Steuerelement: Verknüpfte Daten, Teil I 940
TreeView und Unterformulare 945
Bilder in Access, Teil II: Bilder in OLE-Feldern 950
Kontextmenüs 956
Das TreeView-Steuerelement: Verknüpfte Daten, Teil II 964
Das Bild-Steuerelement 970
Bilder in Access, Teil III: Bilder aus OLE-Feldern anzeigen 976

Das ImageList-Steuerelement 979
Kontextmenüs im TreeView, Teil I: Anlegen und Löschen 986
Das Kontrollkästchen 997
Anwendungsparts 1001
Autowerte 1007
Datensätze kopieren mit Bordmitteln 1012
Neue Autowerte ermitteln und RecordsAffected 1018
GUIDs 1021
Datensätze per VBA kopieren 1025
Verknüpfte Daten kopieren 1030
Programmieren mit Makros, Teil I 1038
Programmieren mit Makros, Teil II 1059
Fehlerbehandlung in Makros 1067
Datenmakros I: Grundlagen 1072
Vertrauenswürdige Datenbanken 1081
Vertrauenswürdigkeit per Makro prüfen 1087
Datenmakros II: Datenoperationen 1089
TempVars unter VBA 1101
Programmieren mit Collections 1106
Mehrere Formularinstanzen anzeigen 1110
Ereignisse in Formularen: Zeitgeber 1118
Detailformulare aufrufen 1121
Datenbankereignis beim Schließen 1126
Datenmakros III: Ereignisse 1130
Datensätze filtern per Kombinationsfeld 1142
Von Formular zu Formular mit OpenArgs 1148
Arrays mit Split und Join 1155
Finetuning für Steuerelemente 1157
Access-Berichte drucken per VBA 1161
Anwendungseinstellungen abspeichern 1168
Registry-Zugriff per VBA 1174
Filtern und Suchen 1181
Zufallszahlen generieren und verwenden 1189
Tabellen bearbeiten mit dem TableDef-Objekt 1196
Etiketten drucken per Assistent 1203
Ex- und Import zu Microsoft Word 1211
Rund um den VBA-Editor 1220
Adressen konsolidieren 1226
VBA-Funktionen in Abfragen 1233
Access-Systemtabellen 1240

Diagrammformulare 1248
Datenbanken und Objekte sichern 1259
Tabellen und Abfragen über DAO verwalten 1265
Die VBA-With-Anweisung 1275
Zippen mit Bordmitteln 1281
Navigationsbereiche gestalten 1289
Access als MP3-Player 1293
Textdateien exportieren und einlesen 1301
Kodierte Texte mit VBA umwandeln 1307
Navigieren über Formulartricks 1321
Nützliche Helferlein per VBScript 1331
Daten ins Web exportieren 1342
Reimen leicht gemacht 1353
Datenbankvorlagen und Designs 1362
Datentypen konvertieren unter VBA 1371
Mathematische VBA-Funktionen 1377
Arbeiten mit externen Bibliotheken 1382
Das Web im Formular 1386
Besseres Code-Layout 1394
ADODB als Alternative zu DAO 1402
Frontends aktualisieren 1413
Arrays als Funktionsergebnis 1418
Rekursive Tabellen 1423
XML-Import und -Export 1429
Recordset-Typen 1443
DAO-Objekte und -Auflistungen 1449
Datenbanken über Verknüpfung starten 1455
DoCmd- und RunCommand-Anweisungen 1464
Navigation in und zwischen Formularen 1474
DAO-Objekte und -Auflistungen, Teil II 1481
Kommunikationsdaten als Bericht 1488
Mehrspaltiger gruppierter Bericht 1494
Bericht als Unterformular 1500
Zusätzliche Steuerelemente mit MSForms 1504
Bubble Sort 1512
Klassenmodule und benutzerdefinierte Typen 1514
Horoskopgenerator 1524
Alles plausibel? 1538
SELECT-Abfragen in UNION 1547
Tastensteuerung abfangen 1554

Google- und Bing-Maps im Formular 1559
Elemente für die Bildanzeige 1570
Unterabfragen als Feldinhalt 1579
Textboxen im Detail 1591
Designs für Steuerelemente 1596
Währungen runden 1606
Daten sichern über XML 1613
Fixierte Formulare 1617
Seitennavigation 1623
Datenbanken von Altlasten säubern 1628
Flying Controls 1634
Zugriff auf Server über ODBC 1640
Ribbon-Programmierung, Teil I 1648
Zugriff auf Server über ODBC, Teil II 1654
Lokaler Webshop 1662
Steuerelemente im Eigenbau: Range Control 1675
Rezepte verwalten 1682
Nährwertdatenbank 1689
Lokaler Webshop, Teil II 1698
Datei- und Verzeichnisoperationen mit der Shell 1709
Erfahrungen mit LibreOffice Base 1719
Outlook-E-Mails importieren 1727
Ribbons, Teil II: Bilder im Ribbon 1738
Ereignisse anderer Objekte nutzen 1743
Formularpositionen lesen und setzen 1748
Mehrere Instanzen eines Formulars 1754
Anwendungs-Design mit positionierten Formularen 1765
Berichte zur Laufzeit filtern 1775
Tabelleninhalte verschlüsseln 1783
Ribbons, Teil III: Interaktion im Ribbon 1791
Platzhalter in Textdateien füllen 1799
Der Webbrowser als Bildanzeige 1809
Indizierung in der Praxis 1821
Sortieralgorithmen 1830
Lokaler Webshop, Teil III 1840
Mit Binärzahlen arbeiten 1852
Transaktionen in Access 1856
Selektion im Datenblatt 1862
Individuelle Auswahl mit eigener Tabelle 1872
Access-Add-Ins programmieren 1877

Access-Objekt-Assistenten programmieren, Teil I 1885
Tabellen erstellen mit SQL und DDL 1892
Alle Ereignisse im Hauptformular 1903
Text- und Bezeichnungsfelder 1908
Handyverwaltung I: Datenmodell 1913
Handyverwaltung II: Personen verwalten 1922
Meldungen bei Datensatzänderungen 1932
Handyverwaltung III: Provider verwalten 1937
Steuerelemente verankern 1946
Handyverwaltung IV: Verträge und SIM-Karten 1950
Detailformular öffnen und abfragen 1963
Tipps und Tricks für Memofelder 1969
Suche in Lookupfeldern 1972
Zeilenvorschub in Berichten steuern 1979
Mehrere Datensätze pro Zeile im Bericht 1985
Haupt- und Unterformular synchron 1994
Handyverwaltung IV: Mobilfunkgeräte verwalten 2002

# Inhalt nach Kategorie

## Grundlagen

Access - eine Einführung 24
Kunden verwalten 49
Datum und Zeit, Teil I 299
Datenbanken aufteilen 357
Datenbanken im Mehrbenutzerbetrieb 361
Tabellenverknüpfungen pflegen 364
Dialog zur Auswahl von Dateien anzeigen 372
Recordsets in der Datenblattansicht 392
Daten für die Ausgabe formatieren 566
Null-Werte und wie man damit umgeht 587
Datenbankvorlagen und Designs 1362
Etiketten drucken per Assistent 1203
Anwendungsparts 1001
Datensätze kopieren mit Bordmitteln 1012
Vertrauenswürdige Datenbanken 1081
Datenbanken und Objekte sichern 1259
Datenbanken über Verknüpfung starten 1455
Elemente für die Bildanzeige 1570
Währungen runden 1606
Daten sichern über XML 1613
Datei- und Verzeichnisoperationen mit der Shell 1709
Meldungen bei Datensatzänderungen 1932

## Tabellen entwerfen

Entwurf einer einfachen Kundentabelle 54
Kunden und Projekte verwalten 69
Anreden und Co. mit Wertlisten oder Lookup-Tabellen verwalten 95
Mitarbeiter per m:n-Beziehung zu Projektteams zusammenstellen 119
Fremdschlüsselfelder mit Nachschlage-Funktion: Pro und Kontra 130
Felder und ihre Eigenschaften 208
Eigenschaften von Tabellen 237
Indizierung von Feldern 283
Bestellungen in Tabellen verwalten 528
Autowerte 1007

Neue Autowerte ermitteln und RecordsAffected 1018
GUIDs 1021
Adressen konsolidieren 1226
Indizierung in der Praxis 1821
Tabellen erstellen mit SQL und DDL 1892

## Abfragen für die Datenauswahl

Einfache Auswahlabfragen kennenlernen 78
Darstellung von 1:n-Beziehungen 85
Daten aus verknüpften Tabellen darstellen 144
Filtern von Textfeldern 184
Einsatz von Parametern 189
Filtern nach Zahlen 191
Daten zusammenführen mit UNION-Abfragen 217
Berechnete Felder in Abfragen und Eigenschaften 245
Benutzerdefinierte Funktionen in Abfragen 249
Abfrageeigenschaften 258
Duplikate per Abfrage finden 288
Abfragen: Datum und Zeit 319
Gruppierungen in Abfragen 590
Kreuztabellenabfragen 595
Kreuztabellenabfragen nach Datum 610
Balkendiagramme mit Bordmitteln 808
Filterkriterien für Formulare, Teil I 823
VBA-Funktionen in Abfragen 1233
Rekursive Tabellen 1423
SELECT-Abfragen in Union 1547
Unterabfragen als Feldinhalt 1579

## Aktionsabfragen

Anfügeabfragen 104
Löschabfragen 172
Aktionsabfragen per VBA ausführen 333
Aktionsabfragen per VBA 722

## SQL-Grundlagen

SQL: SELECT-Abfragen, Teil I 487
SQL: SELECT-Abfragen, Teil II 500
SQL-Grundlagen: Aktionsabfragen 741

## Formulare für die Dateneingabe

Formulare für einfache Tabellen 57
Formulare für Daten aus 1:n-Beziehungen 88
m:n-Beziehung per Unterformular 125
Formulare öffnen und schließen 154
m:n-Beziehungen in Listenfeldern darstellen 200
Datensatzauswahl per Kombinationsfeld 230
Abhängige Kombinationsfelder 268
Standardwerte statisch und dynamisch einstellen 272
Validieren von Benutzereingaben 305
Werte zu Kombinationsfeldern hinzufügen 327
Daten per Kombinationsfeld auswählen und löschen 338
Daten in der Endlosansicht von Formularen anzeigen 343
Lookup-Daten per Formular verwalten 385
Datensätze in Unterformularen anlegen 400
Kombinationsfeld als Eingabehilfe für Textfelder 414
Bedingte Formatierung 457
Mehrfachauswahl in Listenfeldern auslesen 479
Bestellungen in Formularen verwalten 533
Rechnen in Formularen 540
Listenfeldauswahl speichern und wiederherstellen 577
Schnellsuche im Listenfeld 601
Synchrone Unterformulare 698
Datensätze individuell auswählen 693
Dynamische Datenherkunft mit Formularbezug 725
Haupt- und Unterformular synchron 1994
Filterkriterien für Formulare, Teil II 846
Filterkriterien für Formulare, Teil III: Kombinationsfelder 878
Detailformulare aufrufen 1121
Navigation in und zwischen Formularen 1474
Zusätzliche Steuerelemente mit MSForms 1504
Alles plausibel? 1538
Berichte zur Laufzeit filtern 1775

Tabelleninhalte verschlüsseln 1783
Text- und Bezeichnungsfelder 1908
Steuerelemente verankern 1946

## Steuerelemente

Die Schaltfläche 161
Das Listenfeld 164
Das Textfeld 221
Das Kombinationsfeld 225
Optionsgruppen und Optionsfelder 417
Das Registersteuerelement 714
Das Bild-Steuerelement 970
Die Umschaltfläche 657
Das Kontrollkästchen 997
Finetuning für Steuerelemente 1157
Kombinations- und Listenfelder mit Wertliste füllen 635
Kombinations- und Listenfelder per Callback füllen 638
ListView-Steuerelement mit Daten füllen 663
ListView-Steuerelement: Daten bearbeiten 688
ListView-Steuerelement: Kontrollkästchen nutzen 719
Das Registersteuerelement in der Praxis 734
ListView-Steuerelement: Drag and Drop, Teil I 749
ListView-Steuerelement: Drag and Drop, Teil II 785
Steuerelementlayouts mit Access 2007 und jünger 801
Textfelder: Value, OldValue und Text 831
Textfelder im Griff, Teil II: Markierungen 884
Das TreeView-Steuerelement: Grundlagen, Teil I 906
Das TreeView-Steuerelement: Grundlagen, Teil II 913
Das Anlage-Feld 927
Anlage-Felder und VBA 934
Das TreeView-Steuerelement: Verknüpfte Daten, Teil I 940
TreeView und Unterformulare 945
Das TreeView-Steuerelement: Verknüpfte Daten, Teil II 964
Das ImageList-Steuerelement 979
Kontextmenüs im TreeView, Teil I: Anlegen und Löschen 986

Das Web im Formular 1386
Designs für Steuerelemente 1596
Textboxen im Detail 1591
Seitennavigation 1623
Steuerelemente im Eigenbau: Range Control 1675

## Reporting mit Berichten

Einfache Tabelle in der Detailansicht 99
Adressblöcke in Berichten 261
Einfache Tabelle in der tabellarischen Ansicht darstellen 290
Darstellung von 1:n-Beziehungen per Gruppierung 295
Berichtsbereiche 423
Sortieren in Berichten 426
Berichte öffnen 445
Mehrspaltige Berichte 462
Rechnen in Berichten 545
Gruppierungen in Berichten 550
Bericht als PDF speichern 558
Rechnungsbericht 623
Access-Berichte drucken per VBA 1161
Kommunikationsdaten als Bericht 1488
Mehrspaltiger gruppierter Bericht 1494
Bericht als Unterformular 1500
Zeilenvorschub in Berichten steuern 1979
Mehrere Datensätze pro Zeile im Bericht 1985
Tipps und Tricks zu Berichten: Duplikate ausblenden 561
Tipps und Tricks zu Berichten: Vergrößerbar 562
Tipps und Tricks zu Berichten: Verkleinerbar 562
Tipps und Tricks zu Berichten: Berichte zusammenfassen 564

## Makros

Aktion beim Start: Das Autoexec-Makro 370
AutoKeys: Globale Tastenkombinationen 673
Programmieren mit Makros, Teil I 1038
Programmieren mit Makros, Teil II 1059

Datenmakros I: Grundlagen 1072
Vertrauenswürdigkeit per Makro prüfen 1087
Datenmakros II: Datenoperationen 1089
Datenmakros III: Ereignisse 1130

## VBA-Grundlagen

Einführung in VBA 148
Eigene Routinen schreiben 175
If...Then- und Select Case-Bedingungen 178
Variablen in VBA 196
Schleifen in VBA 322
Fehler unter VBA 375
Fehlerbehandlung unter VBA 379
Fehlerbehandlung in der Praxis 403
For Each-Schleifen 453
Zeichenketten-Funktionen 467
Zeilenumbruch und Tabulator in Access 472
Textdateien schreiben und lesen 617
Zufallszahlen generieren und verwenden 1189
Diagrammformulare 1248
Dateibefehle 613
Die Dir-Funktion 629
Binärzahlen in VBA-Eigenschaften 631
Early Binding und Late Binding 684
Programmieren mit Arrays 758
Debugging im VBA-Editor, Teil 1 780
Debugging im VBA-Editor, Teil 2 817
Ereignisse abgreifen 874
Rekursion mit VBA 891
TempVars unter VBA 1101
Programmieren mit Collections 1106
Arrays mit Split und Join 1155
Anwendungseinstellungen abspeichern 1168
Registry-Zugriff per VBA 1174
Rund um den VBA-Editor 1220
Die VBA-With-Anweisung 1275
Nützliche Helferlein per VBScript 1331
Arbeiten mit externen Bibliotheken 1382
Besseres Code-Layout 1394

Arrays als Funktionsergebnis 1418
Bubble Sort 1512
Klassenmodule und benutzerdefinierte Typen 1514
Ereignisse anderer Objekte nutzen 1743
Sortieralgorithmen 1830
Mit Binärzahlen arbeiten 1852

## Access-Funktionen

Domänenfunktionen 251
Bedingungen: Wenn, Schalter und Wahl 266
Access-Funktionen: Datum und Zeit 313
Meldungsfenster anzeigen und auswerten 340
Die Format-Funktion 570
Datentypen konvertieren unter VBA 1371
Mathematische VBA-Funktionen 1377
DoCmd- und RunCommand-Anweisungen 1464

## Formulare mit VBA programmieren

Formularereignisse nutzen 139
Fehlerbehandlung in Formularen 408
Aktuellen Datensatz kopieren 439
Formulare und Steuerelemente referenzieren 441
Einfache Suchfunktion für Formulare 506
Farbige und vom Wert abhängige Formatierung 575
Ereignisse in Formularen: Öffnen und Schließen 605
Mehrere Formularinstanzen anzeigen 1110
Ereignisse in Formularen: Zeitgeber 1118
Datenbankereignis beim Schließen 1126
Datensätze filtern per Kombinationsfeld 1142
Von Formular zu Formular mit OpenArgs 1148
Navigieren über Formulartricks 1321
Tastensteuerung abfangen 1554
Flying Controls 1634
Formularpositionen lesen und setzen 1748
Mehrere Instanzen eines Formulars 1754
Anwendungs-Design mit positionierten Formularen 1765
Selektion im Datenblatt 1862
Individuelle Auswahl mit eigener Tabelle 1872

Alle Ereignisse im Hauptformular 1903
Detailformular öffnen und abfragen 1963
Suche in Lookupfeldern 1972

## Datenzugriff programmieren

DAO: Datenbank im Griff mit dem Database-Objekt 450
DAO: Recordsets öffnen, lesen und bearbeiten 473
DAO: Recordsets filtern, durchsuchen und sortieren 483
DAO: Querydefs und Parameterabfragen unter VBA 745
Tabellen bearbeiten mit dem TableDef-Objekt 1196
Bilder in Access, Teil II: Bilder in OLE-Feldern 950
Datensätze per VBA kopieren 1025
Verknüpfte Daten kopieren 1030
Tabellen und Abfragen über DAO verwalten 1265
Kodierte Texte mit VBA umwandeln 1307
ADODB als Alternative zu DAO 1402
Recordset-Typen 1443
DAO-Objekte und -Auflistungen 1449
DAO-Objekte und -Auflistungen, Teil II 1481
Zugriff auf Server über ODBC, Teil II 1654
Transaktionen in Access 1856

## Access-Benutzeroberfläche

Datenbankfenster und Navigationsbereich im Detail 64
Der Ausdrucks-Generator 240
Formular- und Berichtsvorlagen 396
Datenblattansicht im Griff 435
Bilder in Access, Teil I 921
Kontextmenüs 956
Filtern und Suchen 1181
Navigationsbereiche gestalten 1289
Ribbon-Programmierung, Teil I 1648
Ribbons, Teil III: Interaktion im Ribbon 1791
Access-Add-Ins programmieren 1877
Access-Objekt-Assistenten programmieren, Teil I 1885

## Daten importieren

Daten aus CSV-Dateien importieren 111
Vom Textfeld zur Lookuptabelle 134
Ex- und Import zu Microsoft Word 1211
Textdateien exportieren und einlesen 1301
XML-Import und -Export 1429

## Interaktiv

CSV-Export für Adressdaten 348
Im- und Export-Spezifikationen definieren und verwenden 354
Access und Excel: Import und Export 650
Access und Excel: Verknüpfungen 676
E-Mails mit Outlook versenden 854
Berichte per E-Mail versenden, Teil II 860
E-Mails sicher versenden 869
Daten ins Web exportieren 1342
Google- und Bing-Maps im Formular 1559
Zugriff auf Server über ODBC 1640
Erfahrungen mit LibreOffice Base 1719
Outlook-E-Mails importieren 1727
Ribbons, Teil II: Bilder im Ribbon 1738
Der Webbrowser als Bildanzeige 1809

## Lösungen

Projektzeiterfassung - Aufgaben und Tätigkeiten verwalten 274
Kunden per Bezeichnung verwalten 493
Kommunikationsdaten im Griff 523
Internetmarke mit Access nutzen 703
Internetmarke mit Access nutzen, Teil II 729
Reihenfolge individuell einstellen 763
Feiertage verwalten, Teil 1 771
Feiertage verwalten, Teil 2 793
Änderungen protokollieren, Teil I 835
Berichte per E-Mail verschicken, Teil I 840
Platzhalter in Texten ersetzen 898
Berichte per E-Mail verschicken, Teil III 903
Zippen mit Bordmitteln 1281
Access als MP3-Player 1293

Reimen leicht gemacht 1353
Frontends aktualisieren 1413
Horoskopgenerator 1524
Lokaler Webshop 1662
Rezepte verwalten 1682
Nährwertdatenbank 1689
Lokaler Webshop, Teil II 1698
Platzhalter in Textdateien füllen 1799
Lokaler Webshop, Teil III 1840
Handyverwaltung I: Datenmodell 1913
Handyverwaltung II: Personen verwalten 1922
Handyverwaltung III: Provider verwalten 1937
Handyverwaltung IV: Verträge und SIM-Karten 1950

## Tools

Beispieldaten-Assistent 679

## Tipps und Tricks

Fixierte Formulare 1617
Access-Systemtabellen 1240
Datenbanken von Altlasten säubern 1628
Tipps und Tricks für Memofelder 1969
Tipps und Tricks: Standardverzeichnis einstellen 429
Tipps und Tricks: Objekte ausblenden 429
Tipps und Tricks: Ausgeblendete Objekte anzeigen 430
Tipps und Tricks: Aktuelles Datum und Zeit blitzschnell 430
Tipps und Tricks: Wert aus vorherigem Datensatz übernehmen 431
Tipps und Tricks: Datum und Zeit im Formulartitel 431
Tipps und Tricks: Memofeld als Sortierungs- und Gruppierungsfeld 432
Tipps und Tricks: Datenbankkennwort vergeben 432
Tipps und Tricks: Textbausteine verwenden 433
Tipps und Tricks: Ereignisprozeduren schneller anlegen 798
Tipps und Tricks: Monatsnamen und Wochentage 799
Tipps und Tricks: Texte in leeren Feldern vorblenden 799

# 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 2017 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 2017. 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 April 2018



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.

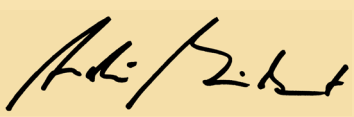
Warum nun ein neues Access-Magazin? Die Antwort ist einfach: Weil es einfach noch so viele Lücken zu schließen gibt, und weil vieles anderswo viel zu oberflächlich und lustlos beschrieben wird. Und weil ich glaube, dass das Internet Ihnen auch jetzt noch immer nicht alle Lösungen so präsentiert, dass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

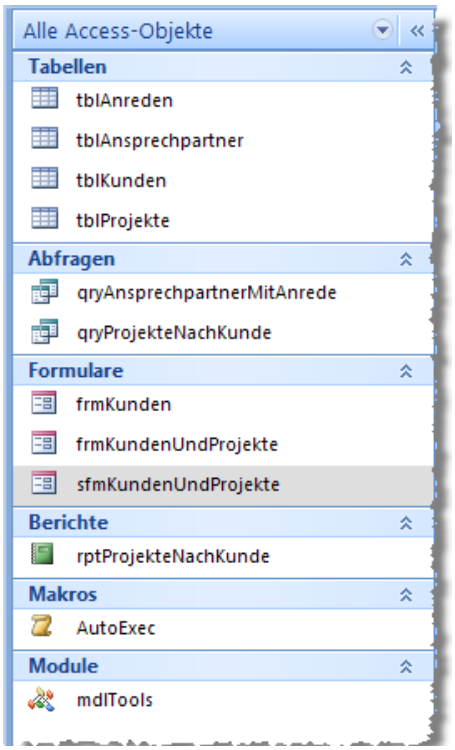
Aus welche Elementen besteht eine Access-Datenbank eigentlich? Hier erfahren Sie mehr über den Aufbau einer Access-Datenbank und lernen die verschiedenen Objekte kennen.

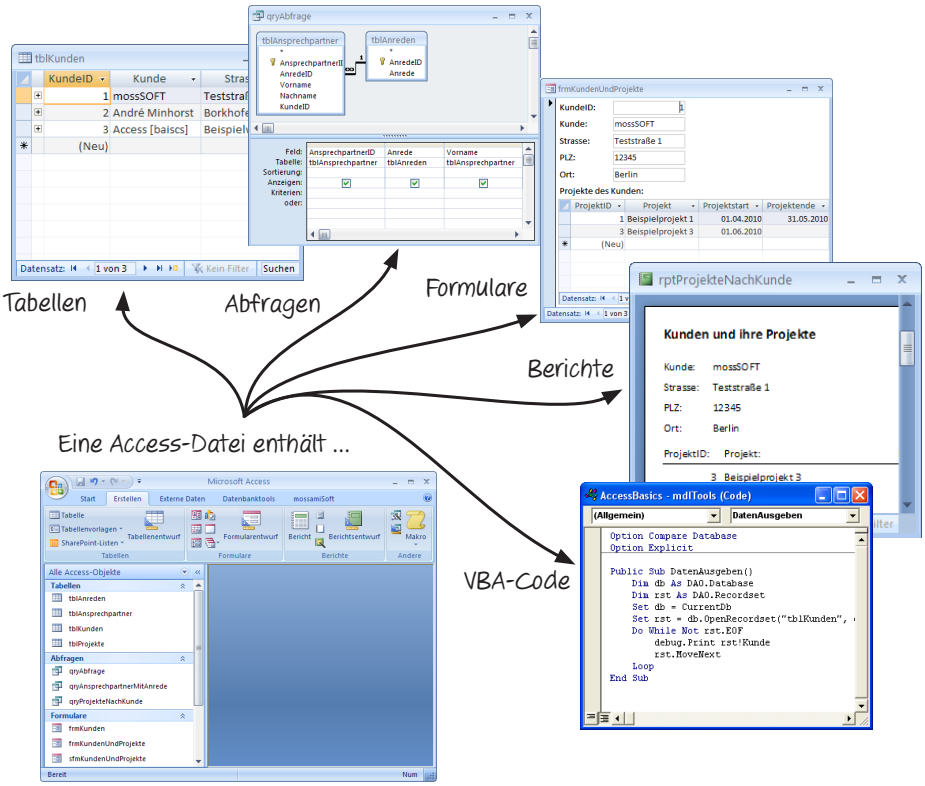
Während der Benutzer einer Datenbankanwendung nur Menüs, Formular und Berichte zu sehen bekommt, muss sich der Entwickler um einiges mehr kümmern. Für einen ersten Überblick reicht eine Auflistung wichtiger Begriffe, die wir später genauer erläutern werden:

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

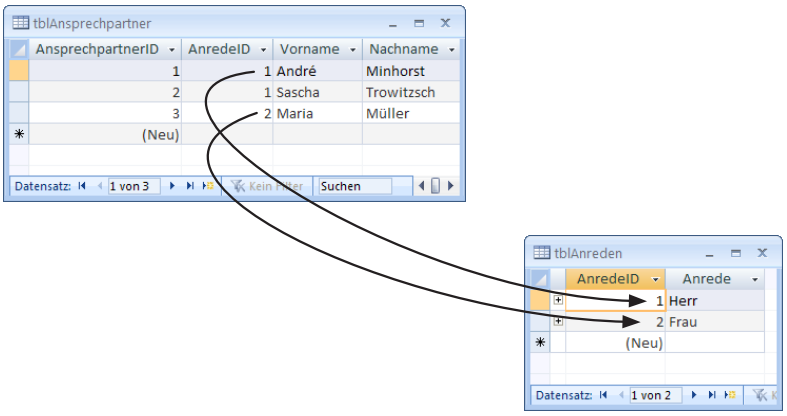
| KundeID | Kunde | Strasse | PLZ | Ort |
|---|---|---|---|---|
| 1 | mossSOFT | Teststraße 1 | 12345 | Berlin |
| 2 | André Minhorst | Borkhofer Str. 17 | 47137 | Duisburg |
| (Neu) | | | | |

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.

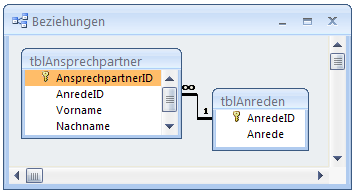
## Beziehungen, Primärschlüssel und Fremdschlüssel

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.

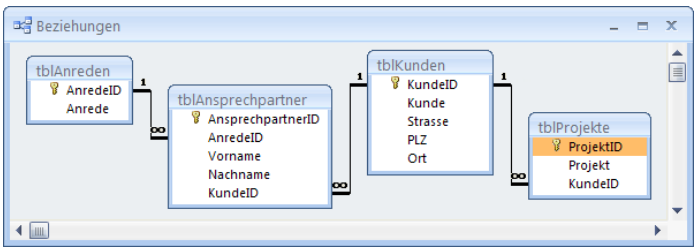
Eine Beziehung wie diese hier legt man im Beziehungen-Fenster von Access fest. Dazu fügt man dort die betroffenen Tabellen hinzu und verbindet die an der Beziehung beteiligten Felder miteinander:




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.



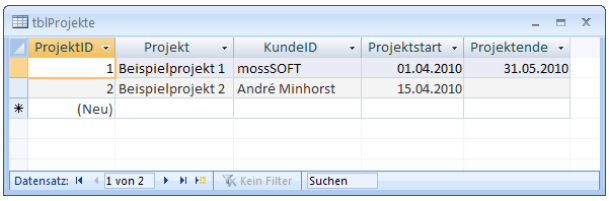
Sie können hier gut weitere Beispiele für Beziehungen zwischen Tabellen erkennen: So lassen sich jedem Kunden beliebig viele Ansprechpartner zuordnen, gleiches gilt für die Beziehung zwischen Projekten und Kunden.

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) | | | | |

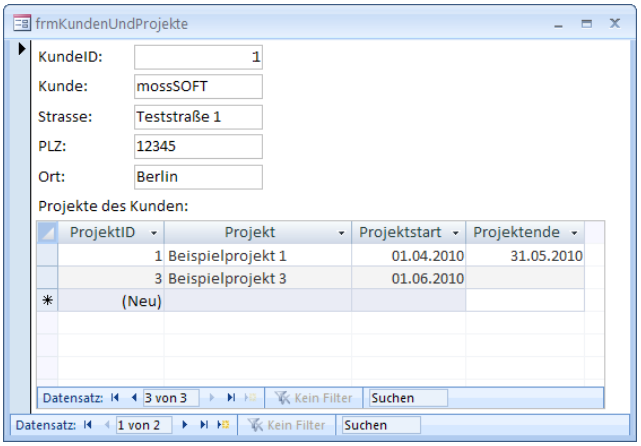
Datensatz: 1 von 2 | Kein Filter | Suchen

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

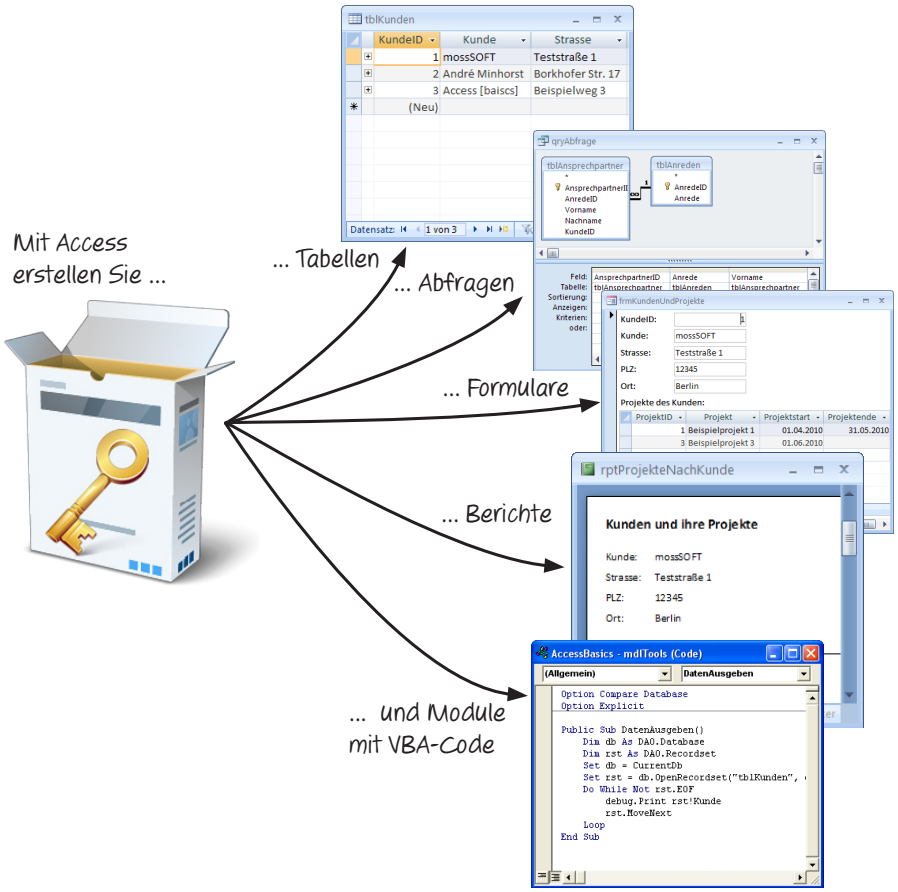
Ein **Makro** bietet die Möglichkeit, einfache Vorgänge schnell zu automatisieren. So einfach dies ist, so eingeschränkt ist der Leistungsumfang eines Makros.

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.

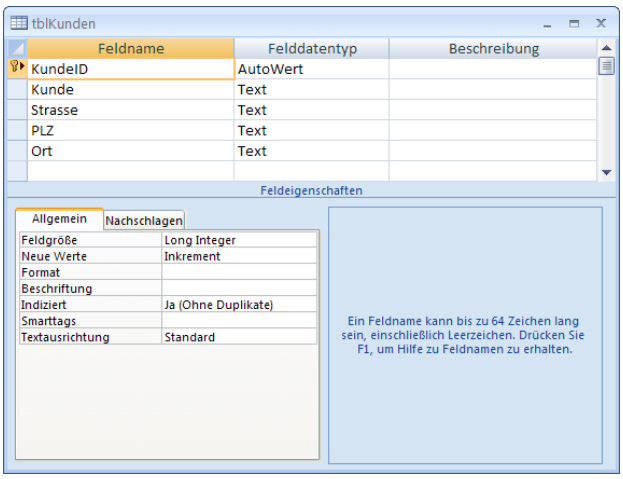
Access ist nur das Rahmenwerk für Ihre Anwendung. Tabellen, Abfragen, Formulare, Berichte, Makros und Module müssen Sie komplett selbst definieren (oder aus bestehenden Datenbanken zusammenkopieren - das heißt dann Wiederverwendung).




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.

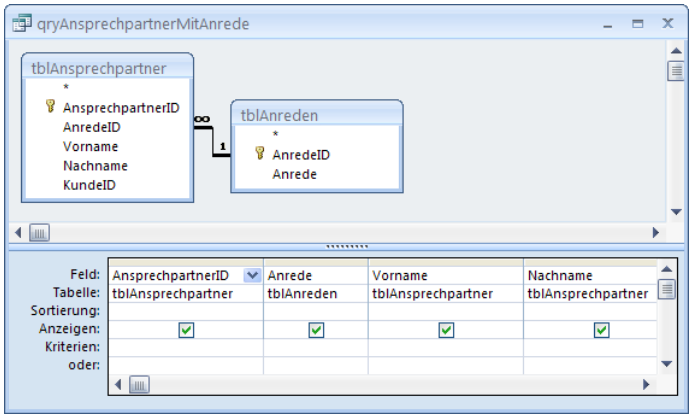
Access stellt Entwurfsansichten für alle wichtigen Objekte zur Verfügung:

- 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.



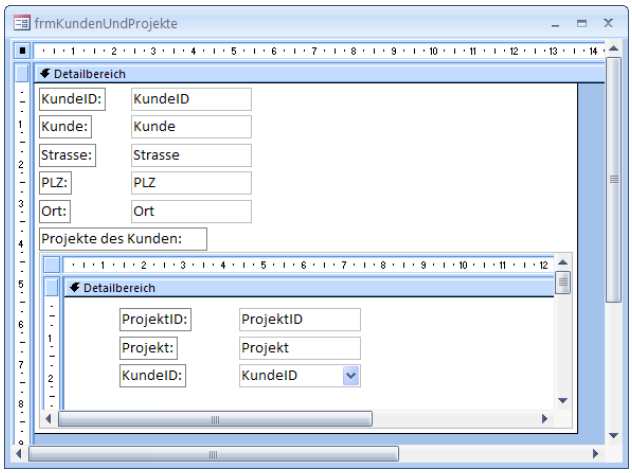

Der Datentyp Autowert beim Feld KundeID sorgt dafür, dass für jeden neuen Datensatz ein eindeutiger Wert in dieses Feld geschrieben wird. So können Sie jeden Datensatz immer eindeutig identifizieren - das ist zum Beispiel beim Verknüpfen von Datensätzen zweier Tabellen hilfreich.

- 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.



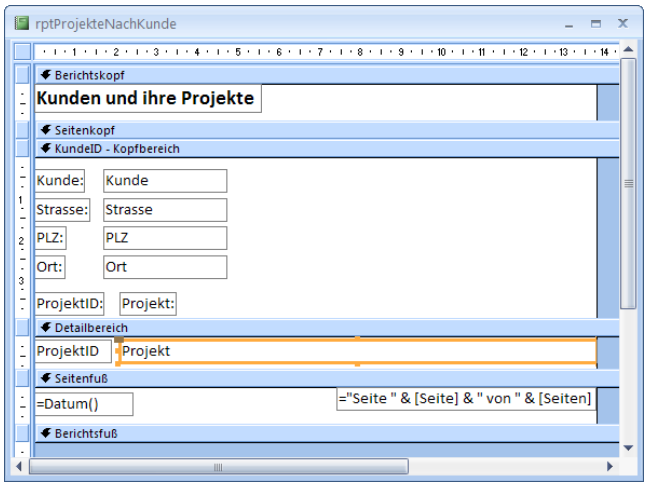

Der Abfrageentwurf sieht auf den ersten Blick verwirrend aus. Sie werden aber bald erfahren, wie einfach es ist, die Daten aus verschiedenen Tabellen zusammenzuführen oder einfach nur nach bestimmten Kriterien auszuwählen.

- 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.



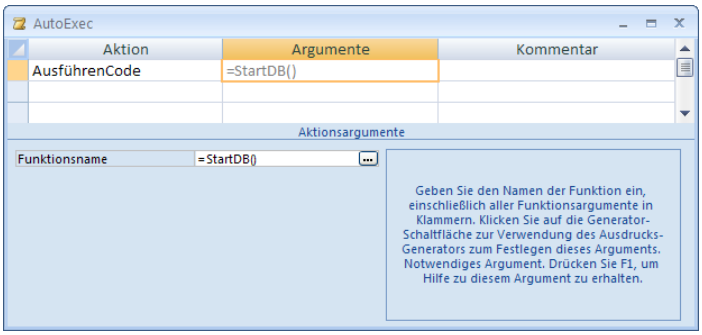

Es gibt kaum leichtere Tätigkeiten, als das Grundgerüst eines Formulars mit der entsprechenden Entwurfsansicht von Access zu erstellen. Der Teufel steckt allerdings im Detail - bis erstmal alles ordentlich ausgerichtet und mit den nötigen Funktionen versehen ist, kann es schon etwas dauern.

- 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.



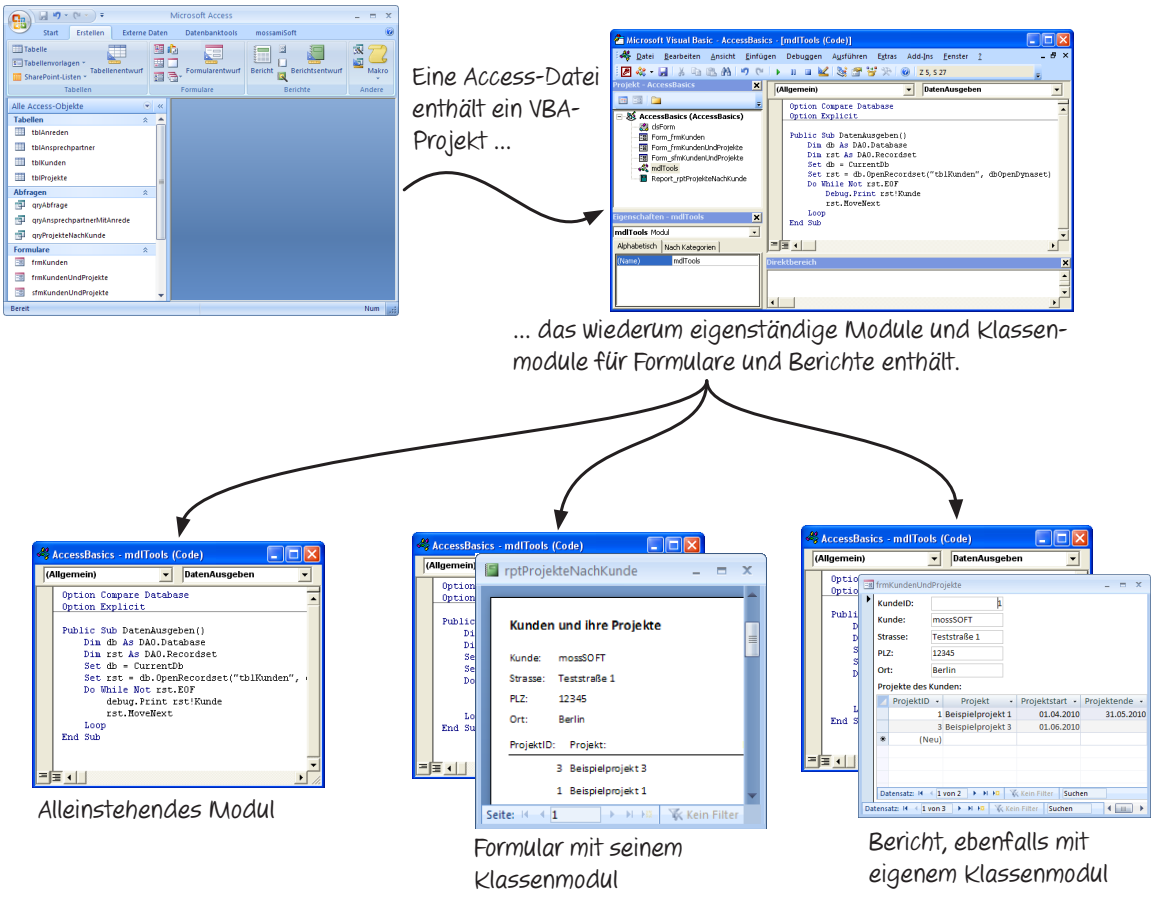

Das Makro namens AutoExec wird automatisch beim Öffnen einer Datenbankdatei ausgeführt. Das ist praktisch, wenn Sie beispielsweise gleich zu Beginn eine VBA-Prozedur ausführen möchten oder etwas Ähnliches erledigen wollen.

## 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.

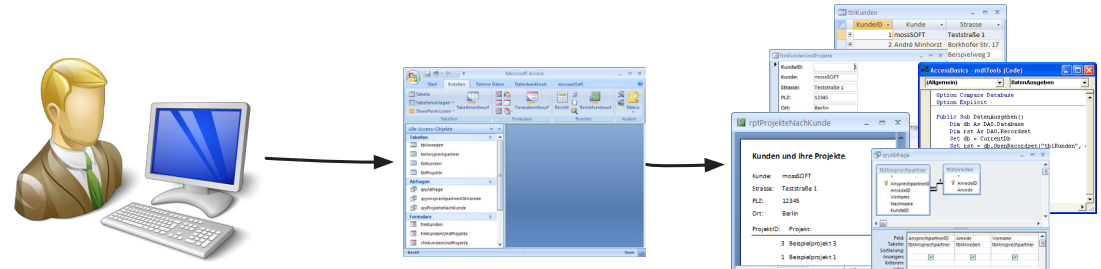
- Sie können Word für die Ausgabe von Daten Ihrer Datenbankanwendung einsetzen. Das populärste Beispiel hierfür ist die Erstellung von Serienbriefen.
- Sie importieren Daten aus Excel-Tabellen oder schreiben Access-Daten dorthin, damit beispielsweise ein Controller die Daten weiterverarbeiten kann.
- Sie füllen eine PowerPoint-Präsentation mit den Daten einer Produkt-Tabelle, um einem Kunden einen individuellen Katalog zu präsentieren.

Wenn Berichte mal nicht das Mittel der Wahl für die Ausgabe von Access-Daten sind, können Sie auch prima Word oder Excel als Reporting-Tools verwenden.

Aber nicht nur die Office-Produkte, sondern auch eine ganze Reihe weiterer Anwendungen lassen sich mit VBA automatisieren.

## Access im Netzwerk-/Mehrbenutzerbetrieb

Der wohl am weitesten verbreitete Anwendungsfall von Access-Anwendungen ist die Einzelplatzanwendung. Es greift nur ein einziger Benutzer auf die Daten zu, der meist auch der Entwickler dieser Anwendung ist. Deshalb braucht er auch nur eine einzige **.mdb**- oder **.accdb**-Datei mit allen Tabellen, Abfragen, Formularen, Berichten, Makros und Modulen.



Ein einziger Benutzer ...

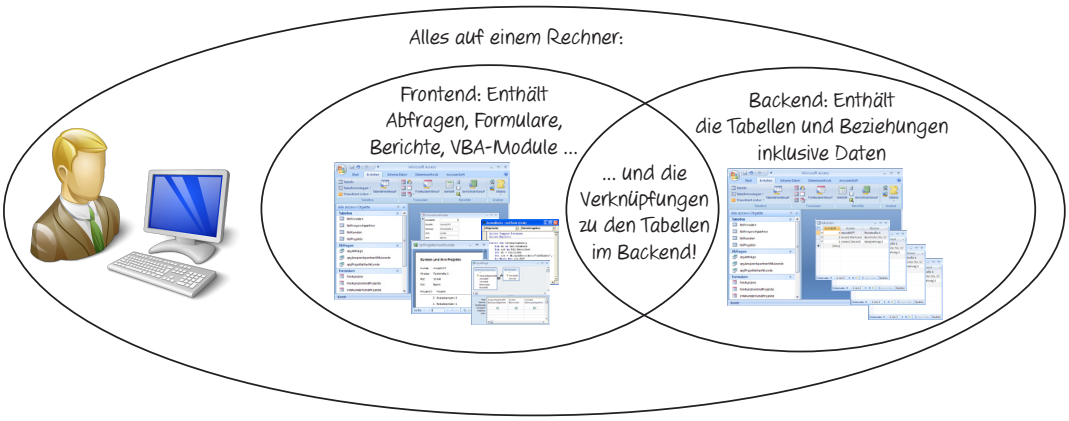
... verwendet eine einzige .mdb- oder accdb-Datei, ...

... die alle Tabellen, Abfragen, Formulare et cetera enthält.

Es gibt aber Gründe für andere Konfigurationen. Der erste unterliegt noch nicht einmal den Anforderungen der Mehrbenutzerumgebung, sondern wird durch eine einfachere Wartbarkeit bedingt. Stellen Sie sich einmal vor, mehrere Arbeitsplätze würden mit einer von Ihnen entwickelten Einzelplatzanwendung ausgestattet. Alles läuft prima, bis eine Änderung etwa an einem Bericht notwendig wird. Da jede Anwendung mittlerweile durch die Benutzer mit individuellen Daten gefüllt wurde, können Sie diese nicht einfach gegen eine neue Version der Anwendung mit geändertem Bericht austauschen.

Also die Änderungen an allen Arbeitsplätzen mehr oder weniger von Hand ausführen? Nein, das wäre je nach Anzahl der Arbeitsplätze wohl kaum erquickend. Stattdessen teilen Sie die Datenbankanwendung bereits vor der ersten Auslieferung auf, und zwar in zwei Elemente namens Frontend und Backend. Das Frontend enthält alle Objekte der Benutzeroberfläche und die Anwendungslogik, und das Backend enthält alle Tabellen mit den Daten.

Damit der Benutzer die Daten im Backend beispielsweise mit den Formularen im Frontend bearbeiten kann, muss er lediglich das Frontend mit Verknüpfungen zu den Tabellen im Backend ausstatten – schon kann er über Abfragen, Formulare und Berichte genauso auf diese zugreifen, als wenn alle Objekte sich in der gleichen Datenbank befänden.

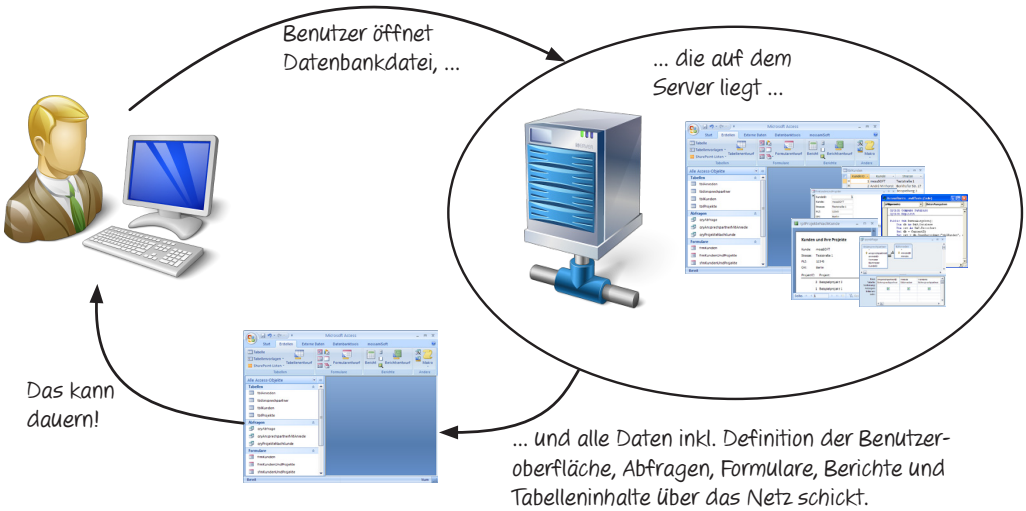



Der entscheidende Unterschied jedoch ist, dass Sie das Frontend weiterentwicklen können und zum gewünschten Zeitpunkt alle bereits verteilten Frontends an die Benutzer der Anwendung verteilen. Diese brauchen dann nur das alte Frontend gegen das neue auszutauschen und können dann problemlos mit ihren eigenen Daten weiterarbeiten.

## Mehrbenutzerbetrieb

Wenn eine oder mehrere Personen über das Netzwerk auf die Daten einer Access-Datenbankanwendung zugreifen sollen, ist das auch kein Problem, solange Sie einige Regeln beachten und sich die Anzahl der Benutzer und der Zugriffe sowie die Menge der Daten in gewissen Grenzen bewegen.

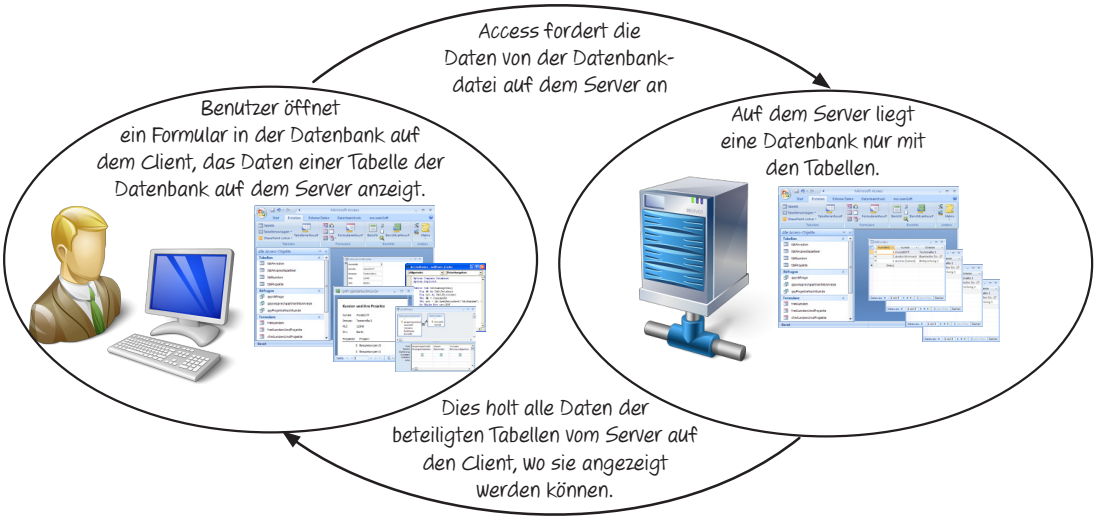
Die erste Regel lautet: Sie müssen die Datenbankanwendung, auf die Benutzer von mehreren Arbeitsplätzen aus zugreifen sollen, wie oben beschrieben aufteilen. Der Grund ist folgender: Wenn Sie die Datenbankdatei einfach auf den Server legen und diese von einem vernetzten Rechner aus öffnen, muss Access nicht nur die in den Tabellen der Datenbank gespeicherten Daten über das Netzwerk transportieren, sondern auch jegliche andere Information bezüglich der Datenbank – also auch alle Objekte wie Formulare oder Berichte.




Um die Netzlast zu verringern, teilen Sie die Datenbank genau wie im obigen Fall in **Frontend** und **Backend** auf. Das Backend enthält fortan lediglich die Tabellen mit den Daten, das Frontend alle übrigen Elemente und außerdem einige weitere. Dabei handelt es sich um Verknüpfungen zu den im Backend gespeicherten Tabellen.

Eine Verknüpfung erscheint im Datenbankfenster beziehungsweise im Navigationsbereich (ab Access 2007) beinahe wie eine richtige Tabelle, wird aber durch ein anderes Symbol gekennzeichnet, das auf die Verknüpfung hinweist. Einfach gesagt sorgt die Verknüpfung dafür, dass die übrigen Objekte wie Abfragen, Formulare und Berichte genauso auf die Tabellen zugreifen können, als wenn diese in der gleichen Datenbank gespeichert wären – mit dem kleinen Unterschied, dass die benötigten Daten erst über das Netzwerk aus den Tabellen des Backends zum Frontend transferiert werden müssen.

Access ist in einem wichtigen Punkt nicht für den Netzwerkbetrieb optimiert: Wenn Sie mit einer Abfrage einige wenige Daten aus einer Tabelle anfordern, um diese in einem Formular oder Bericht anzuzeigen, muss Access vereinfacht gesagt dennoch den kompletten Inhalt der Tabelle über das Netzwerk zum Frontend schaffen. Erst dort erfolgt dann die Auswertung der Abfrage. Dies wirkt sich je nach Datenmenge natürlich auf die Performance der Abfrage aus.




Aber, und das ist die gute Nachricht: Ein paar Zeilen weiter unten erfahren Sie, wie Sie auch bei größeren oder komplexeren Daten nicht hinterherhinken – Sie brauchen nur das Access-Backend durch eines mit Gehirn auszutauschen.

## Access ist was für Kinder – oder doch nicht?

Im Grunde ist Access kein vollwertiges Datenbank-Management-System. Erstens, weil die Anzahl der Benutzer und der gleichzeitigen Zugriffe eingeschränkt ist und weil Abfragen nicht nur die benötigten, sondern alle der Abfrage zugrunde liegenden Daten über das Netzwerk transportieren müssen.

Vorsicht!
Access ist gar kein richtiges Datenbanksystem! Aber erzählen Sie das bloß nicht weiter ... ;-)

Es gibt noch weitere Gründe, die schlicht in der Definition von Datenbank-Management-Systemen liegen. Demnach unterstützt ein Datenbank-Management-System sogenannte Trigger und Stored Procedures – beides Dinge, die Access nicht beherrscht.

Programmierer, die mit »richtigen« Datenbank-Management-Systemen wie Microsoft SQL Server, Oracle oder MySQL arbeiten, nehmen Access nicht ernst, was Sie allerdings bei der Wahl des richtigen Entwicklungswerkzeugs für Ihre Datenbankanwendung nicht verunsichern sollte. Wenn Access grundsätzlich als Werkzeug für die Erstellung Ihrer Anwendung in Frage kommt, bringt seine leicht zu bedienende Entwicklungsumgebung viele Vorteile mit, allen voran die Geschwindigkeit, mit der sich Datenbank-Anwendungen entwickeln lassen.

Und da man ein Ziel bei hoher Geschwindigkeit in kürzerer Zeit erreicht und Zeit bei der Software-Entwicklung quasi proportional zu den Kosten ist, sollte Access immer als Alternative in Erwägung gezogen werden.

Hinzu kommt, dass die scheinbar eingeschränkten Möglichkeiten von Access im Mehrbenutzerbetrieb oder bei der Datensicherheit noch lange kein Grund sind, auf Access zu verzichten – und wenn auch nur als Benutzeroberfläche Ihrer Anwendung.

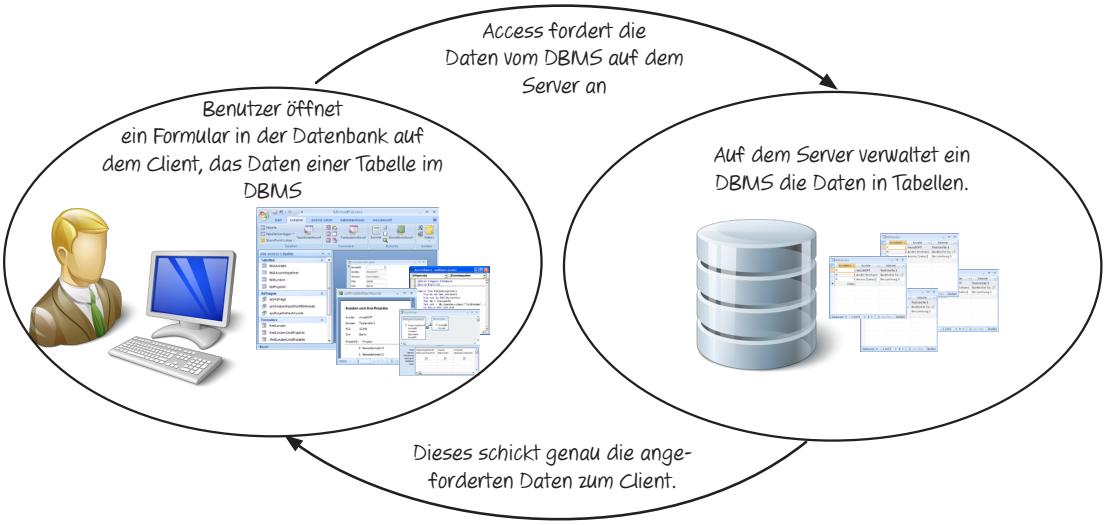
## Access als Frontend für DBMS

Es gibt nämlich Schnittstellen (zum Beispiel ODBC), mit denen Sie von einer Access-Anwendung aus auf die in einem richtigen DBMS gespeicherten Daten zugreifen können. Sie können sich dabei all die Vorteile des DBMS wie die integrierte Benutzerverwaltung oder Features wie Trigger oder Stored Procedures zu Nutze machen.

Und über das Thema Benutzerverwaltung gelangen wir auch gleich zum Thema Sicherheit: Die ist bei Einsatz eines richtigen DBMS nämlich ungleich höher als bei einer Kombination aus Access-Frontend/Backend. Das Frontend basiert natürlich nach wie vor auf Access – einfacher als damit stellen Sie Ihre Benutzeroberfläche nämlich nicht zusammen.

Und das Beste: Auch über die Performance brauchen Sie sich keine Sorgen mehr zu machen, denn ein DBMS hat als Backend gegenüber einem Access-Backend den Vorteil, dass es Abfragen gleich auf dem Server verarbeitet und nur die angefragten Daten an das aufrufende Backend schickt. Wenn Sie also aus tausend Artikel-Datensätzen die drei teuerste ansehen möchten, schickt das DBMS auch nur die Daten dreier Artikel über das Netzwerk zu Ihnen – und nicht alle tausend, wie es bei einer reinen Access-Frontend-Backend-Lösung der Fall wäre.

ODBC ist übrigens eine standardisierte Datenbankschnittstelle. Diese gibt es für verschiedene Datenbanksysteme wie MySQL, MS SQL Server oder Oracle. Für Access bedeutet dies: Sie können damit die Tabellen anderer Datenbanken in Ihre eigene Datenbank einbinden und so auf diese zugreifen, als wenn sie sich direkt in Ihrer Datenbank befinden würden.




## Access und das Internet

Eine der meistgefragten Anforderungen an Access-Datenbanken ist die Internet-Fähigkeit. Unter Access 2000 gab es einen ersten Versuch, Access-Datenbanken mit sogenannten Datenzugriffsseiten internetfähig zu machen, der jedoch mangels Akzeptanz kläglich scheiterte. Mit Access 2010 gibt es einen weiteren Versuch, der auf Microsoft Sharepoint aufsetzt. Nach den Erfahrungen der Beta-Phase scheint auch dies zumindest mit der Version 2010 noch nicht der Stein der Weisen zu sein.

Wer seine in Access gespeicherten Daten über das Internet verfügbar machen und den Komfort von Formularen, Berichten und VBA nicht missen möchte, geht aktuell den folgenden Weg, wobei das Vorhandensein eines Webservers beziehungsweise von Webspace mit installiertem DBMS Voraussetzung ist: Er migriert die Tabellen seiner Datenbank beispielsweise in eine Microsoft SQL Server- oder MySQL-Datenbank (je nachdem, welche Software der verwendete Webserver bietet).

Dann kopiert man die Datenbank auf den Webserver und greift schließlich von der nun um die Tabellen erleichterten Datenbank über entsprechende Tabellenverknüpfungen genauso auf die Datenbanken zu, als ob diese in einem Access-Backend liegen würden.

Wenn Sie hingegen die Daten auf einer Webseite bereitstellen möchten, damit Benutzer über den Internetbrowser statt über ein Access-Frontend auf die Daten zugreifen können, werden Sie um die Programmierung einer entsprechenden Webseite nicht herumkommen – und die Daten Ihrer Access-Datenbank landen dann ebenfalls in einem DBMS auf dem Webserver. Sie ahnen es bereits: Mit einer Access-Anwendung hat diese aus Internet-Frontend und DBMS-Backend dann nichts mehr gemein.

## Access-Versionen

Microsoft veröffentlicht mit jedem Office-Paket auch eine neue Access-Version. Access ist mit Erscheinen dieser Ausgabe etwa 18 Jahre alt und mit Access 2010 erscheint die zehnte Version (1.0, 1.1, 2.0 sind 16-bit-Versionen, 95, 97, 2000, 2002, 2003, 2007 und 2010 sind 32-bit-Versionen. Datenbankanwendungen, die mit einer älteren Version erstellt wurden, lassen sich meist problemlos in eine Access-Datenbank höherer Version umwandeln. Die aktuelle Version heißt Access 2007, die folgende Version Access 2010 erscheint voraussichtlich im ersten Halbjahr 2010. Es gibt aber noch eine Menge Anwendungen, die mit älteren Versionen von Access betrieben werden, und es gibt ebenfalls noch eine Menge Entwickler, die Anwendungen mit älteren Access-Versionen erstellen. Es gibt beispielsweise eine Menge Entwickler, die den Schritt von Access 2003 oder älter nach Access 2007 und höher nicht wagen, weil es mit Access 2007 einige einschneidende Änderungen gab: Das Sicherheitssystem etwa entfällt weitgehend, und außerdem wurden die bis Access 2003 verwendeten Menü- und Symbolleisten durch das sogenannte Ribbon (zu deutsch: Multifunktionsleiste) ersetzt. Letzteres wirkt moderner und bietet mehr Steuerelemente, ist aber aufwendiger in der Erstellung.

Grrr!!! Mit Access 2007 hat Microsoft das ganze Bedienkonzept über den Haufen geworfen. Für den Benutzer ist das nach einiger Eingewöhnung gut, aber für Buch- und Fachautoren, die für Anwender der neuen und alten Version schreiben, wird es schwierig ... naja, wenn hier und da große Unterschiede auftauchen, beschreiben wir halt die Vorgehensweise für die neuen Versionen und weisen an den entsprechenden Stellen auf abweichende Schritte für die alten Versionen hin ...

## Datensicherheit

Ihre Ansprüche bezüglich der Sicherheit Ihrer Daten ist einer der wesentlichen Faktoren bei der Wahl eines Entwicklungswerkzeugs. Wie oben beschrieben, entfällt das eingebaute Sicherheitssystem von Access mit der Version 2007. Mit diesem Sicherheitssystem konnten Sie Benutzer und Benutzergruppen definieren und Berechtigungen für die einzelnen Benutzer und Benutzergruppen für die Objekte der Datenbank festlegen. So ließ sich etwa bestimmen, das Benutzer A die Daten einer Tabelle T nur lesen, aber nicht ändern darf, und er durfte auch nur das Formular F öffnen, aber nicht das Formular G, denn dies war der Geschäftsleitung vorbehalten.

Dieses Sicherheitssystem war aber nach Expertenmeinung ohnehin nie sicher, sodass es zwar Otto Normalbenutzer davon abhielt, geheime Daten einzusehen, aber mit etwas krimineller Energie geknackt werden konnte.

Wenn Sie Ihre Daten sicher verwahren und definieren möchten, wer auf welche Daten zugreifen darf, liegen Sie mit Access aber nicht völlig falsch: Immerhin können Sie, wie oben schon für Datenbankanwendungen mit vielen

Für die Panzerknacker wäre eine Access-Datenbank keine Herausforderung. Aber Otto Normalverbraucher können Sie mit ein paar Tricks schon davon abhalten, unerwünschte Dinge mit Ihren Daten zu tun. Und für sensible Daten gibt es ja andere Backends wie MySQL, MS SQL Server und Co.

Benutzern und Zugriffen vorschlagen, die Daten mit einem richtigen DBMS verwalten und über Access als Frontend komfortabel darauf zugreifen.

Da DBMS eine viel größere Sicherheit als reine Access-Datenbanken bieten und auch Benutzer und Benutzergruppen verwalten, wäre hier die Kombination aus einem Access-Frontend und einem DBMS-Backend eine gute Wahl. Es gibt natürlich eine Reihe Alternativen zur Erstellung eines Frontends wie etwa VB.NET, Delphi und viele andere, aber der Aufwand zur Erstellung einer Anwendung mit den Funktionen einer Access-Anwendung dürfte ungleich höher liegen.

## Schutz des geistigen Eigentums

Während die Sicherheit der Daten einer Datenbankanwendung eher den Benutzer interessiert, ist der Hersteller oftmals darauf bedacht, sein geistiges Eigentum zu schützen. In der Regel steckt in den Funktionen einer Datenbank eine Menge Know-how und Arbeit, und die möchte man normalerweise untern veröffentlichen, sodass beliebige Personen dieses weiterverwenden oder ändern können.

Access erlaubt das Erstellen einer speziellen Version einer Access-Datenbankanwendung, die den Zugriff auf die Entwurfsansichten von Formularen, Berichten und VBA-Modulen sperrt – einzig Tabellen, Abfragen und Makros sind von dieser Sperrung ausgeschlossen.

Es gibt aktuell nur eine einzige Firma, die als Serviceleistung das Wiederherstellen einer Datenbank aus einer sogenannten **.mde**- oder **.accde**-Datenbank anbietet, und dies kostet ersten Geld und erfordert zweitens den Nachweis, dass man auch wirklich der Besitzer dieser Datenbankanwendung ist.

## Verwendung auf Arbeitsplätzen ohne Access?

Genauso wie Sie für das Bearbeiten eines Word-Dokuments die Word-Anwendung benötigen, muss für die Ausführung einer Access-Datenbank die Access-Anwendung auf dem Rechner installiert sein. Das bedeutet allerdings nicht unbedingt, dass jeder Arbeitsplatz mit einer Vollversion ausgestattet sein muss, was natürlich eine Menge Geld kostet. Stattdessen bietet Microsoft – wie weiter oben bereits angedeutet – eine spezielle, eingeschränkte Version von Access an, die sogenannte Runtime-Version.

Gute Nachrichten für schmale Budgets: Access gibt es für Endanwender auch in einer kostenlosen Runtime-Version. Einfach runterladen, installieren und loslegen - Ihre Anwender können dann alles durchführen, wofür Sie Formulare, Berichte, Menüs und Co. bereitstellen.

Mit dieser kann man eine Access-Datei starten und die vom Entwickler angelegten Menüs, Formulare und Berichte öffnen und damit arbeiten. Der Unterschied zur Vollversion von Access ist der, dass die Runtime-Version von Access keine der Entwurfsansichten zur Verfügung stellt und auch nicht die VBA-Entwicklungsumgebung. Genau genommen beschränkt sich die Grundfunktion der Access-Runtime darauf, eine Datenbankdatei zu öffnen und dem Benutzer die Möglichkeit zu geben, die vom Entwickler dafür vorgesehenen Objekte wie Formulare und Berichte zu verwenden.

## Access im Office-Umfeld

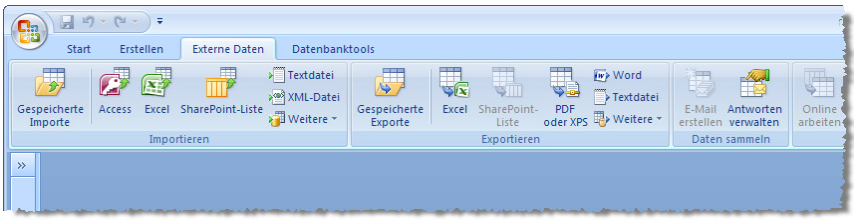
Microsoft Access ist Bestandteil einiger Versionen von Microsoft Office. Dies merkt man auch bei der Arbeit mit Access: Es gibt einige Import- und Export-Möglichkeiten für den Austausch von Daten zwischen den verschiedenen Office-Anwendungen, und die Programmiersprache VBA ermöglicht gar das Automatisieren der übrigen Office-Anwendungen.

Viele Mitglieder der Office-Familie sind der jeweilige Platzhirsch auf ihrem Gebiet – es gibt keine Textverarbeitung, die weiter verbreitet ist als Microsoft Word, keine Tabellenkalkulation kann Excel bezüglich des Funktionsumfangs das Wasser reichen, und für die Verwaltung von E-Mails, Adressen und Termi-

Unter Excel gibt es die sogenannten Userforms. Mit denen können Sie zwar auch Daten in Excel-Tabellen eingeben, aber eine Userform ist nicht an eine Tabelle gebunden - deshalb muss man das Aktualisieren jedes einzelnen Wertes selbst programmieren ...

nen ist Outlook oft erste Wahl. Da trifft es sich gut, dass auch Access die weitverbreiteste Desktop-Datenbank ist.

Um sich ein Bild von den Im- und Export-Möglichkeiten zu machen, brauchen Sie nur den Ribbon-Tab **Externe Daten** etwa unter Access 2007 zu betrachten. Importieren können Sie etwa von anderen Access-Datenbanken, aus Excel-Dateien oder aus Text- und XML-Dateien, umgekehrt geht es zum Beispiel in Richtung Excel, PDF oder Word.




Die Import- und Export-Möglichkeiten von Access.

Daten lassen sich aber auch aus diversen anderen Datenbankformaten importieren, zum Beispiel **dBase** oder **Paradox**, aber auch aus alle Datenbanken, die sich via ODBC ansprechen lassen.

## Daten importieren

Das Importieren können Sie sich so vorstellen, dass Access die Datenquelle analysiert und auf Basis der gefundenen Informationen eine neue Tabelle mit den zu importierenden Daten erstellt – Sie können dann also in der Access-Datenbank mit einer eigenen Kopie der Daten arbeiten.

Natürlich können Sie die gewünschten Daten auch in bestehende Tabellen einfügen – Sie müssen dann nur angeben, welche Tabelle Sie als Zieltabelle verwenden möchten und welchen Feldern dieser Tabelle Sie die neuen Daten hinzufügen möchten.

## Daten verknüpfen

Das Importieren ist nicht der einzige Weg, Daten aus anderen Dateien oder Datenbanken verfügbar zu machen: Sie können auch eine Verknüpfung erstellen. Das bedeutet, dass die Daten genauso dargestellt werden wie die aus den lokalen Tabellen. Der Unterschied ist aber, dass es sich dabei nur um eine Sicht auf diese Daten handelt – tatsächlich befinden sich diese noch in der ursprünglichen Datei.

Weiter oben haben wir von der Aufteilung von Datenbanken in Frontend und Backend gesprochen: Dort besitzt das Frontend beispielsweise Verknüpfungen mit allen Tabellen des Backends. Das Gleiche können Sie auch mit beliebigen anderen Daten durchführen: Sie können die Inhalte einer Excel-Datei in einer verknüpften Tabelle genauso darstellen wie die Daten einer Textdatei.

Diese Technik erlaubt es dann auch, Access als Frontend für Datenbanken auf Basis von MySQL, Microsoft SQL Server oder Oracle zu verwenden – oder andersherum gesagt: Sie können Daten aus diesen Datenbanksystemen in Access verfügbar machen.

Wenn Sie Daten aus einer Textdatei über eine Verknüpfung in einer Access-Datenbank verfügbar machen möchten, müssen diese Daten natürlich einigermaßen strukturiert vorliegen - also in Zeilen und Spalten, wobei letztere durch spezielle Trennzeichen wie das Tab-Zeichen oder das Semikolon dargestellt werden.

## Daten exportieren

Genauso wie Sie andere Datenquellen anzapfen, können Sie auch Daten von Access in Dateien anderer Formate exportieren. So fungieren beispielsweise Excel- oder Word-Dateien als Ziel von Exportvorgängen. Warum aber sollten Sie Daten in eines dieser Formate exportieren wollen? Dafür gibt es eine Menge Gründe, zum Beispiel diese:

- Sie möchten die Daten in Excel analysieren oder anderweitig weiterverarbeiten.
- Rechnungsdaten sollen in Word landen, um eine Rechnung zu erstellen.

- Ein kompletter Bericht soll in ein Word-Dokument exportiert werden, damit jemand diesen für spezielle Anforderungen anpassen kann.
- Ein Bericht soll in das PDF-Format exportiert werden, um diesen an beliebige Empfänger weiterleiten zu können, die nicht über Microsoft Access verfügen.

## Access-Daten von anderen Anwendungen aus importieren

Die Daten aus Access-Tabellen können Sie natürlich auch gleich von der jeweiligen Zielanwendung aus abfragen oder importieren. Excel etwa bietet ähnlich viele Varianten zum Importieren von Daten, und raten Sie mal, welche Datenquelle an erster Stelle steht ...

Während beim Transfer von Daten zwischen Access und Excel wenig zu tun ist, weil die Daten meist auf beiden Seiten in tabellarischer Form vorliegen, müssen die Daten in anderen Fällen erst ausgewählt und aufbereitet werden.

## Word-Seriendruck mit Access-Daten füttern

Ein gutes Beispiel hierfür ist der Word-Seriendruck: Für diesen können Sie neben anderen Datenquellen auch eine Access-Tabelle oder -Abfrage auswählen. Danach stellt Word eine Liste aller in der Datenherkunft enthaltenen Felder zur Verfügung, deren Einträge Sie als Platzhalter in das Seriendruck-Dokument einfügen können. Ein paar Schritte später erstellt Word dann ein neues Dokument, dass die Daten der Access-Tabelle und die Word-Dokumentvorlage verknüpft.

## Automatisierung per VBA

Während die standardmäßig angebotenen Features für den Datenaustausch schon die Lösung für viele Aufgaben darstellen, lässt sich durch VBA-Programmierung noch viel mehr erreichen – genau genommen können Sie damit nicht nur Access und somit Ihre eigene Anwendung, sondern auch die übrigen Mitglieder des Office-Pakets automatisieren.

Einfach gesagt besitzt jede Anwendung wie Word, Excel oder Outlook eine Bibliothek mit einer Reihe von Befehlen, mit denen man diese Anwendung „fernsteuern" kann. Ein vereinfachtes Beispiel sieht so aus:

- Sie öffnen Outlook per VBA.
- Danach erstellen Sie per VBA eine neue E-Mail.
- Dieses füllen Sie mit den gewünschten Informationen, also etwa der E-Mail-Adresse des Empfängers, dem Betreff und dem eigentlichen Inhalt.
- Schließlich schicken Sie die Mail mit einem weiteren Befehl ab.

Auf die gleiche Weise können Sie alles mit den Office-Anwendungen anstellen, was Sie auch über die Benutzeroberfläche erledigen können –also beispielsweise eine Excel-Tabelle öffnen, ihren Inhalt auslesen und in eine Access-Tabelle kopieren, ein Word-Dokument erstellen und dieses mit beliebigen Inhalten füllen oder diverse Tätigkeiten in Zusammenhang mit den E-Mails, Terminen, Kontakten oder Aufgaben von Outlook durchführen.

So können Sie beispielsweise die in einer Datenbank gespeicherten Adressen mit den Kontakten von Outlook abgleichen, oder Aufgaben aus einer Access-Datenbank direkt in eine Outlook-Aufgabe schreiben.

Bei den VBA-Profis heißt es: „Alles, was nicht bei drei auf den Bäumen ist, wird automatisiert! " Das gilt natürlich nur für solche Anwendungen, die auch eine VBA-Schnittstelle anbieten. Neben den Office-Anwendungen sind das noch einige mehr, wie Visio, einige Corel-Produkte oder auch Adobe Indesign.

## Excel oder Access?

Beide bieten die Möglichkeit, Daten in Tabellen einzugeben. Damit enden die Gemeinsamkeiten aber auch schon: Immerhin ist Excel eigentlich eine Tabellenkalkulation und Access ein Datenbanksystem. Das hält Unwissende aber nicht davon ab, Daten, die eigentlich in eine referentielle Datenbank gehören, in Excel-Tabellen zu speichern und diese auch dort zu pflegen. Andersherum leistet Excel dem Access-Anwender in vielen Fällen gute Dienste.

Wenn Sie sich gleich dabei ertappen, dass Sie die Daten in Ihren Excel-Dateien vielleicht besser in eine Access-Datenbank packen, brauchen Sie sich nicht zu schämen. Immerhin gibt es erwiesenermaßen auch Leute, die Ihren Briefverkehr komplett mit Excel erledigen - weil sich Adresse, Briefkopf und so weiter viel besser ausrichten lassen ...

Woran erkennt man denn eigentlich, ob Excel oder Access die richtige Wahl für einen Anwendungsfall ist? Normalerweise ist das ganz einfach: Wenn in einer einzigen Zeile einer Excel-Tabelle Daten stecken, die Informationen über grundverschiedene Dinge enthalten – also etwa einen Kunden und ein Projekt –, dann sollten Sie ganz schnell über einen Transfer der Daten von Excel nach Access nachdenken. Grundsätzlich ist der Einsatz von Access fast immer dann angezeigt, wenn die Daten sauber strukturiert in der Exceltabelle liegen.

Ein anderer Aspekt findet sich in den Tätigkeitsbeschreibungen von Excel und Access: Excel als Tabellenkalkulation stellt Berechnungen mit den Daten an und liefert gegebenenfalls Auswertungen, Access als Datenbank verwaltet die Daten.

Was tun Sie also mit den Daten, die da in Ihrer Excel-Tabelle stecken? Führen Sie aufwendige Berechnungen damit durch? Erstellen Sie Diagramme auf Basis dieser Daten? Dann scheinen Sie mit Excel gar nicht so falsch zu liegen. Oder beschäftigen Sie sich eher mit der Eingabe oder Bearbeitung der Daten, verwalten Sie diese also? Dann kann Access wohl eher seine Stärken ausspielen als Excel.

Haben Sie erst einmal erkannt, dass Sie mit Excel Daten verwalten, die eigentlich in eine relationale Datenbank gehören, beginnen Sie am besten gleich mit dem Erstellen der Datenbank.

Wollen Sie auch wissen, warum relationale Daten in Access besser aufgehoben sind als in Excel? Dann lesen Sie aufmerksam weiter ...

## Tabelle ist nicht gleich Tabelle

Einer der wichtigsten Unterschiede zwischen Excel und Access ist die Arbeit mit Tabellen. Unter Excel finden Sie gleich nach dem Erstellen einer neuen Datei ein Tabellenblatt mit einer Menge Zeilen und Spalten vor.

Und Sie können jede Zelle mit beliebigem Inhalt füllen – egal, ob Text, Datumsangaben, Zahlen oder Geldbeträgen. Sie brauchen nur die Formatierungseigenschaften der jeweiligen Zelle (oder auch eines Bereichs wie einer oder mehrerer Spalten oder Zeilen) so einzustellen, damit diese die enthaltenen Werte wie gewünscht darstellen.

Wenn Sie etwa ein Geburtsdatum in einer Excel-Tabelle mit einem Hinweis wie „Auf keinen Fall vergessen" versehen möchten, schreiben Sie das einfach direkt in die Zelle - ist das nicht wunderbar?

Auf diese Weise brauchen Sie etwa bei Geldbeträgen nicht jedes Mal das Währungssymbol hinzuzufügen – dieses wird automatisch mit angezeigt, wenn die Zelle als Währung formatiert ist.

Auch die Spaltenüberschriften können Sie unter Excel komplett selbst definieren und dabei auch mehrere Zeilen in Beschlag nehmen oder auch Zellen verbinden, um die Überschriften mehrerer Spalten nochmals zusammenfassend zu dokumentieren.

Unter Access herrscht ein viel strengeres Regiment: Statt Zeilen gibt es hier Datensätze, und die einzelnen Spalten der Tabelle heißen Felder. Bevor Sie überhaupt einen Wert in eine Tabelle schreiben können, legen Sie die Definition der Felder der Tabelle fest. Ein Feld hat dabei mindestens die folgenden Eigenschaften:

- einen Feldnamen (beispielsweise **Vorname** oder **Nachname**),

- einen Felddatentyp (etwa **Text**, **Zahl**, **Datum** oder **Währung**) und
- gegebenenfalls eine Feldgröße (etwa zur Festlegung der maximalen Anzahl Zeichen in einem Textfeld).

Die Definition der Felder erfolgt in der Entwurfsansicht einer Tabelle. Dort entspricht jede Zeile einem Feld der Tabelle. Die wichtigsten Eigenschaften stellen Sie gleich in der jeweiligen Zeile ein, die übrigen erschienen nach der Auswahl des betroffenen Feldes im Eigenschaften-Bereich weiter unten.

Genau genommen können Sie den Entwurf ab Access 2007 auch gleich in der Datenblattansicht definieren, indem Sie einfach ein paar Werte eingeben und Access den Datentypen „erraten" lassen. Profis machen das aber nicht ...

Nach dem Speichern der Tabelle unter einen Namen wie **tblPersonen** können Sie in die Datenblattansicht wechseln und dort Daten eingeben. Und hier kommt gleich einer der entscheidenden Unterschiede im Vergleich zu Excel: Sie können tatsächlich nur Daten eingeben, die auch zum Datentyp des jeweiligen Feldes passen. Wenn Sie also beispielsweise einen Text in ein Zahlenfeld eingeben, wird Access genauso meckern wie beim Eintragen eines nicht korrekten Datums wie dem 30.2.2010 in ein Datumsfeld.

## Das Speichern von Daten

Beim Speichern von Daten gibt es wichtige Unterschiede zwischen Excel und Access. Unter Excel speichern Sie immer ein Dokument. Dieses kann zwar auch mehrere Tabellenblätter mit Daten enthalten, aber wenn Sie auf die Speichern-Schaltfläche klicken, dann wird der aktuelle Zustand des kompletten Dokuments mit allen Tabellenblättern gespeichert.

Das hat Vor- und Nachteile: Wenn Sie sich nach einigen Änderungen entschließen, diese zu verwerfen, speichern Sie den neuen Stand einfach nicht und greifen beim nächsten Öffnen wieder auf den alten Stand zu. Wenn Excel allerdings mal abstürzt, dann können Sie nur hoffen, dass das letzte Anlegen der AutoWiederherstellen-Informationen noch nicht allzu lang her ist.

Unter Access steht beim Speichern nicht das Dokument, sondern der aktuell bearbeitete Datensatz im Mittelpunkt. Wenn Sie beispielsweise einen Adressdatensatz bearbeiten und diesen verlassen, um sich dem folgenden Datensatz zuzuwenden, wird der soeben geänderte Datensatz automatisch gespeichert.

Es gibt einen Weg, um die Änderung eines Datensatzes nicht gleich zu speichern. Dabei starten Sie eine sogenannte Transaktion, ändern die Daten und speichern die Änderungen erst, wenn Sie sagen: „Transaktion durchführen!" Dazu kommen wir aber viel später ...

Es gibt auch keinen Weg, diese Änderungen rückgängig zu machen, wenn der Datensatz erst gespeichert ist. Ein Undo ist nur während der Bearbeitung eines Datensatzes möglich. Dabei betätigen Sie einmal die Escape-Taste, um die Änderungen im aktuellen Feld zu verwerfen und zweimal, um alle Änderungen seit dem letzten Speichern des Datensatzes rückgängig zu machen.

Dies ist manchmal störend, bringt aber auch Vorteile: Wenn der Computer mal abstürzt, verlieren Sie in der Regel maximal die Änderungen an einem einzigen Datensatz – und das lässt sich vermutlich verkraften.

Etwas anders sieht das bei den Objekten von Access aus, also den Tabellen, Abfragen, Formularen, Berichten, Makros und VBA-Modulen. Diese verhalten sich jedes für sich wie ein Excel-Dokument unter Excel: Sie können Änderungen durchführen und diese rückgängig machen, und die Änderungen werden erst übernommen, wenn Sie den Speichern-Befehl ausführen.

Dummerweise ist die Anzahl der Undo-Schritte unter Access (und auch im VBA-Editor) auf 20 Aktionen begrenzt. Aber man gewöhnt sich dran ...

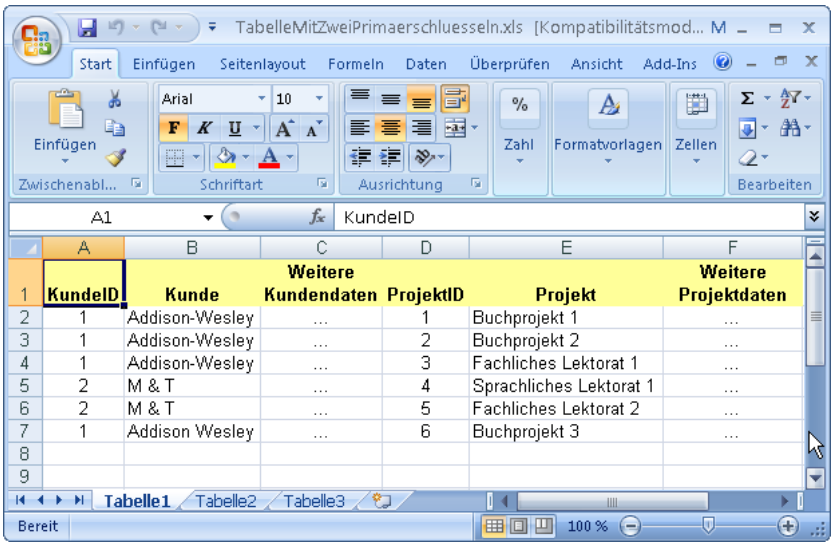
## Restriktionen

Nicht nur der Datentyp legt in Access fest, wie die in einem Feld eines Datensatzes enthaltenen Werte aussehen müssen. Es gibt noch weitere Eigenschaften, mit denen Sie festlegen können, wie ein Datensatz aussehen soll, bevor er gespeichert wird:

- Sie können Felder als Pflichtfeld deklarieren, damit der Benutzer immer einen Wert eingibt.
- Sie können Wertebereiche für ein Feld festlegen (Beispiel: die prozentuale Auslastung darf nicht größer als 100 % werden).
- Die Feldwerte können abhängig vom Inhalt anderer Felder sein (Beispiel: Das Austrittsdatum darf nicht kleiner als das Eintrittsdatum sein.)
- Ein Feld muss einen eindeutigen Wert aufweisen (dies ist sehr wichtig bei Primärschlüsselfeldern – dazu später mehr).

## Beziehungen zwischen Tabellen

Natürlich gibt es auch Excel-Anwendungen, bei denen die Daten so gepflegt werden, wie es auch in einer relationalen Datenbank der Fall wäre. Das geht so weit, dass der Benutzer verschiedene Tabellenblätter zum Speichern von Daten über unterschiedliche Objekte wie beispielsweise Kunden oder Projekte verwendet. Es kann aber auch so aussehen wie im folgenden Beispiel einer Excel-Tabelle – dort finden sich Kunden und Projekte in jeweils einer einzigen Zeile, wobei die Kundendaten natürlich redundant gespeichert werden:




| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | KundeID | Kunde | Weitere Kundendaten | ProjektID | Projekt | Weitere Projektdaten |
| 2 | 1 | Addison-Wesley | ... | 1 | Buchprojekt 1 | ... |
| 3 | 1 | Addison-Wesley | ... | 2 | Buchprojekt 2 | ... |
| 4 | 1 | Addison-Wesley | ... | 3 | Fachliches Lektorat 1 | ... |
| 5 | 2 | M & T | ... | 4 | Sprachliches Lektorat 1 | ... |
| 6 | 2 | M & T | ... | 5 | Fachliches Lektorat 2 | ... |
| 7 | 1 | Addison Wesley | ... | 6 | Buchprojekt 3 | ... |
| 8 | | | | | | |
| 9 | | | | | | |

Auch wenn dieses Beispiel gestellt ist: Solche oder ähnliche Tabellen gibt es in vielen Unternehmen wie Sand am Meer. Der große Nachteil ist die redundante Datenspeicherung, hier bei den Kunden. Wenn sich die Kundendaten einmal ändern, müssen Sie diese Änderungen überall durchführen ... oder es entstehen Inkonsistenzen im Datenbestand.

Wenn der Benutzer es wirklich ernst meint, teilt er diese Daten jedoch auf zwei Tabellen auf und vergibt er für jeden Kunden und jedes Projekt eine eindeutige ID. So kann er dann in einer Spalte der Projekte-Tabelle durch die Angabe der betreffenden Kunden-ID festlegen, für welchen Kunden ein Projekt durchgeführt wird.

Für einen solchen Fall bietet Access jedoch deutlich geeignetere Möglichkeiten. Die wichtigsten Werkzeuge dazu sind die folgenden:

- Sie können in jeder Tabelle (mindestens) ein Feld anlegen, dass eine ID zur eindeutigen Identifizierung eines Datensatzes bietet. Access sorgt automatisch dafür, dass kein Wert doppelt vorkommt und auf Wunsch legt es sogar automatisch neue, eindeutige Werte für dieses Feld an.
- Sie können eine Beziehung zwischen zwei Tabellen definieren, bei der eine Tabelle ein Feld enthält, mit dem Sie auf das eindeutige Feld einer anderen Tabelle verweisen – so können Sie das oben beschriebene Beispiel mit der Kunden- und der Projekte-Tabelle nachbilden. Access stellt im Gegensatz zu Excel allerdings automatisch sicher, dass dieses Feld auch tat-

Das soll nur ein kurzer Einblick in die Vorteile von Access bei der Verwaltung verwandter Daten in verschiedenen Tabellen sein - im Detail gehen wir später darauf ein.

sächlich nur Werte enthält, die auch im eindeutigen Feld der verknüpften Tabelle vorkommen.

## Formulare in Excel und Access

Prinzipiell können Sie unter Excel fast alle Funktionen nachbilden, die Sie auch mit Access bereitstellen können – der Aufwand dafür ist allerdings in den meisten Fällen viel höher. Ein gutes Beispiel sind Formulare zur einfachen Bearbeitung von Daten.

Otto Normalverbraucher wird unter Excel eher selten Formulare entwerfen, um die in den Tabellen seiner Excel-Datei enthaltenen Daten komfortabel bearbeiten zu können – warum auch? Excel ist ja schließlich ein Programm für die Tabellenkalkulation und keine Datenbank.

Access ist im Gegensatz dazu spezialisiert, die in den Tabellen einer Datenbank enthaltenen Daten in Formularen anzuzeigen und diese zum Bearbeiten zur Verfügung zu stellen. Ohne auf die folgenden Ausgaben vorgreifen zu wollen, zeigen wir ganz kurz, wie so etwas aussieht – und zwar am Beispiel der oben vorgestellten Tabelle **tblKunden**.

Wenn Sie ein Formular erstellen möchten, mit dem Sie die einzelnen Datensätze dieser Tabelle für die Bearbeitung übersichtlich darstellen möchten, reichen die folgenden Schritte aus:

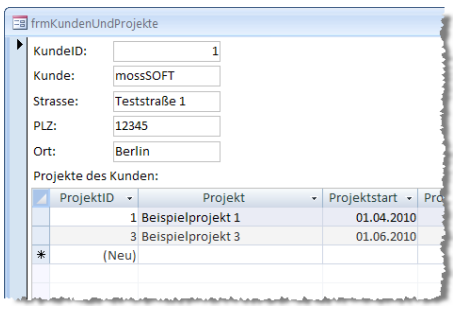
- Erstellen eines neuen Formulas
- Festlegen der Tabelle, die als Datenherkunft fungiert
- Platzieren der Felder der Tabelle im Formular

Danach brauchen Sie nur noch das Formular zu speichern und in die Formularansicht zu wechseln – schon können Sie bequem Daten in die Tabelle eingeben, diese löschen oder bearbeiten oder einfach nur betrachten.

Wir wollen in dieser Einführung erstmal nur anreißen, wie das eine oder andere unter Access funktioniert. Beispiele zum Mitmachen und Ausprobieren gibt es ab Ausgabe 2.

## In Beziehung stehende Daten anzeigen

Auch die Anzeige von Daten, die aus zwei in Beziehung stehenden Tabellen stammen, ist mit Access ganz schnell erledigt. Wir bleiben beim Beispiel von oben und nehmen zu den Kunden noch die Projekte hinzu. Wenn Sie nun etwa das Formular von oben weiterentwickeln und noch die Projekte zum jeweils angezeigten Kunden anzeigen möchten, benötigen Sie ein Unterformular, also ein Formular im Formular. Während das Hauptformular die Kundendaten übersichtlich in Form eines einzigen Datensatzes anzeigt, soll das Unterformular die Projekte des Kunden in tabellarischer Ansicht liefern.




Formulare wie dieses erstellen Sie unter Access in wenigen Minuten ohne eine Zeile Code.

Das Unterformular erstellen Sie dann genauso einfach wie das Hauptformular: Sie legen ein neues Formular an, fügen die Datenherkunft hinzu und ziehen die anzuzeigenden Felder in das Formular. Der einzige Unterschied zum Hauptformular ist, dass Sie dem Unterformular noch mitteilen müssen, dass es die Daten in der tabellarischen Datenblattansicht anzeigen soll. Nach dem Speichern und Schließen brauchen Sie nur noch das Unterformular zum Hauptformular hinzuzufügen - den Rest übernimmt Access für Sie! Mit diesem Formular können Sie nun problemlos die Kunden und ihre Projekte verwalten.

## PDF- oder Druckausgabe per Bericht

Auch dies unterscheidet Access und Excel: Unter Excel müssen Sie die Arbeitsansicht eines zu druckenden Tabellenblatts gleich so gestalten, dass es sich auch für den Ausdruck eignet (sofern Sie es überhaupt drucken wollen).

Unter Access sind Bearbeiten und Ausdrucken zwei Paar Schuhe: Sie bearbeiten die Daten in Formularen, und Sie bereiten die Daten mit Hilfe von Berichten für den Druck auf.

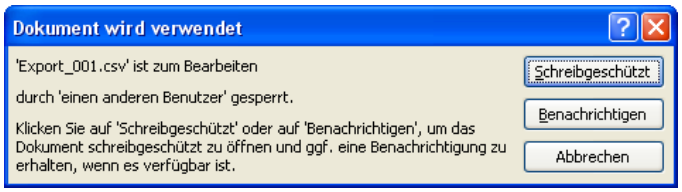
Einen Bericht bestücken Sie prinzipiell genauso wie ein Formular – mit den Unterschied, dass Sie hier erweiterte Möglichkeiten für die Gruppierung von Daten haben.

## Weitere Entscheidungshilfen im Fall Access vs. Excel

Weiter oben haben wir es bereits erwähnt: Access liefert zwar ohne ein richtiges DBMS als Backend keine eingebaute Benutzer- und Rechteverwaltung, aber es erlaubt immerhin den Mehrbenutzerzugriff auf die Daten einer Datenbank. Wenn mehrere Benutzer gleichzeitig einen Datensatz bearbeiten, können Sie festlegen, wie dies geschehen soll:

- Alle können gleichzeitig auf den Datensatz zugreifen und die Änderungen werden jeweils überschrieben.
- Ein Benutzer kann einen Datensatz nur ändern, wenn der andere Benutzer die Bearbeitung beendet hat.

Unter Excel ist jeweils die komplette Datei betroffen: Wenn Sie dort eine Datei öffnen und diese bearbeiten und dann ein weiterer Benutzer versucht, auf diese Datei zuzugreifen, erscheint eine entsprechende Meldung. An dieser Stelle ist es Zeit, über den Einsatz von Access nachzudenken.




Ist eine Excel-Datei in Bearbeitung, müssen sich weitere Benutzer in der Warteschlange einreihen.

# 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

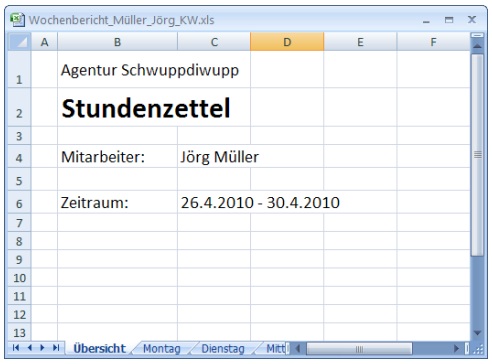
**Der Chef der kleinen Agentur Schwuppdiwupp in Duisburg-Meiderich ist ein richtiger Erbsenzähler. Jeder Mitarbeiter muss penibel in einer Excel-Tabelle aufschreiben, wann er was für welches Projekt für welchen Kunden getan hat. Und an jedem Wochenende sitzt der Chef im Büro und wertet die Daten aus, 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:

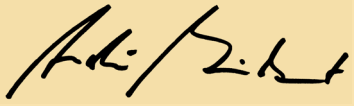

S. 1: Vom Excel-Chaos zur Access-Projektzeiterfassung
S. 5: Tabellen entwerfen
Teil I: Entwurf einer einfachen Kundentabelle
S. 9: Formulare für die Dateneingabe
Teil I: Formulare für einfache Tabellen
S. 16: Die Schaltzentrale von Access
Datenbankfenster und Navigationsbereich im Detail


## Im Großen und im Kleinen ...



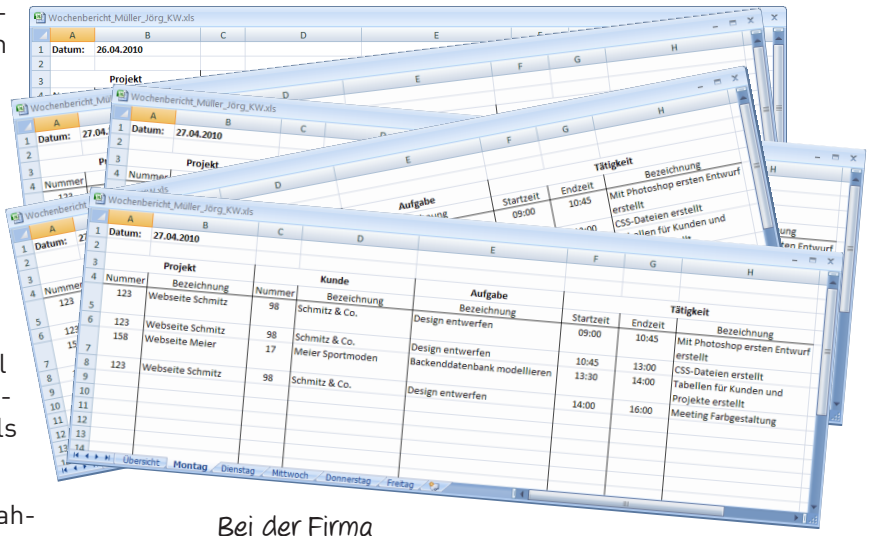
Es gibt eine große Menge Einsteigerbücher für angehende Access-Entwickler. Ein paar davon sind bestimmt gut geeignet, um die ersten Schritte zu machen, andere weniger. Tatsache ist: Die meisten verwenden irgendein Beispiel, um dem Leser zu zeigen, wie er mit Access arbeiten soll. In das Beispiel flechten 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**.

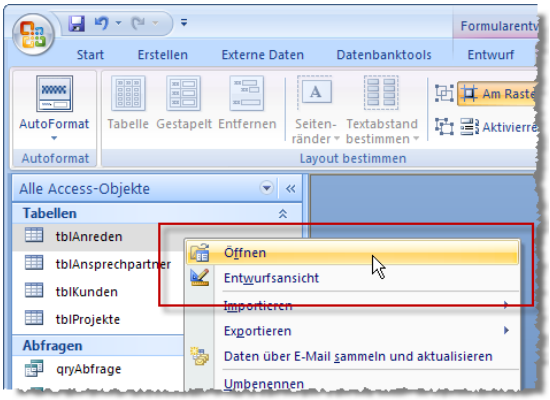
## Tabellenansichten

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:

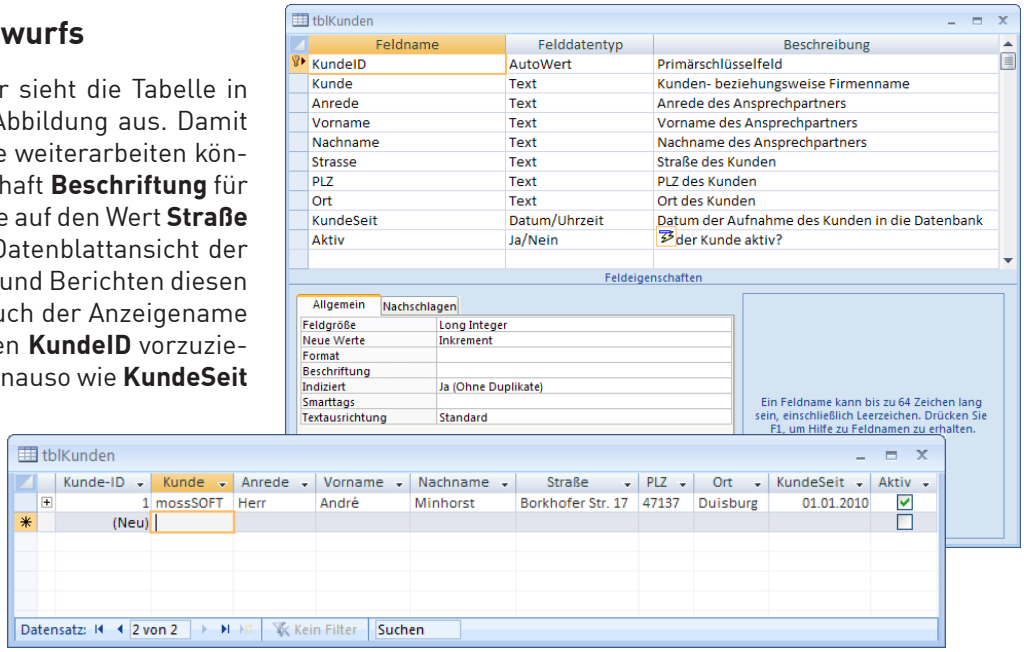
- **KundeID**: Das Feld enthält einen eindeutigen Wert, damit jeder Kunde ein eindeutiges Merkmal besitzt. Wenn Sie den Datentyp auf Autowert einstellen, fügt Access für jeden neuen Datensatz automatisch einen Zahlenwert ein, der noch nicht verwendet wurde (→*[basics]* **Autowertfeld anlegen**). Damit 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)

- **Vorname**: Textfeld zum Speichern des Vornamens des Kunden
- **Nachname**: Textfeld für den Nachnamen des Kunden
- **Strasse**: Textfeld zum Speichern der Straße. In Feldnamen sollten Sie keine Sonderzeichen verwenden, daher nennen wir das Feld **Strasse** statt **Straße**.
- **PLZ**: Textfeld zum Speichern der Postleitzahl. Das Speichern der PLZ ist ein Spezialfall: Eigentlich könnte man diese in einem Zahlenfeld speichern. Allerdings gibt es ja auch Postleitzahlen mit führender **0** – und die erste Stelle würde dann nicht standardmäßig angezeigt werden (außer, man sagt Access mit einer speziellen Einstellung, dass es immer fünf Stellen anzeigen soll – aber dies sind wiederum für Länder mit vierstelligen Postleitzahlbereichen wie Österreich oder der Schweiz zu viele Stellen). Also speichern wir die Postleitzahl schlicht in einem Textfeld.
- **Ort**: Textfeld zum Speichern des Ortes
- **Land**: Textfeld zum Speichern des Landes
- **KundeSeit**: Datumsfeld, welches das Aufnahmedatum des Kunden in die Datenbank kennzeichnet (→*[basics]* **Datum/Zeit-Feld anlegen**)
- **Aktiv**: **Ja/Nein**-Feld, das angibt, ob es sich um einen aktiven Kunden handelt (→*[basics]* **Ja/Nein-Feld anlegen**).

Ja/Nein-Felder nehmen die Werte 0 (False, Nein, nicht angehakt) und -1 (True, Ja, angehakt) an. Wenn Sie statt des Kontrollkästchens in der Tabelle den tatsächlichen Wert sehen möchten, stellen Sie die Eigenschaft Nachschlagen/Steuerelement anzeigen für dieses Feld auf Textfeld ein.

## Feinschliff des Tabellenentwurfs

Nach dem Anlegen dieser Felder sieht die Tabelle in der Entwurfsansicht wie in der Abbildung aus. Damit Sie gleich sinnvoll mit der Tabelle weiterarbeiten können, ändern Sie noch die Eigenschaft **Beschriftung** für das Feld **Strasse**. Stellen Sie diese auf den Wert **Straße** ein, damit Access später in der Datenblattansicht der Tabelle, in Abfragen, Formularen und Berichten diesen Text statt **Strasse** anzeigt. Und auch der Anzeigename **Kunde-ID** ist dem etwas sperrigen **KundeID** vorzuziehen (→*[basics]* **Bericht erstellen**), genauso wie **KundeSeit** zwar einen guten Feldnamen, aber keine gute Beschriftung abgibt – also tragen wir dort **Kunde seit** als **Beschriftung** ein.




Wenn Sie danach in die Datenblattansicht der Tabelle wechseln, finden Sie diese gleich mit ordentlich benannten Spalten vor. Hier können Sie direkt testweise einen ersten Datensatz eintragen.

Und beim Eingeben der Daten fällt Ihnen vielleicht gleich noch mehr auf, was wir am Entwurf der Tabelle optimieren können: Zum Beispiel können wir dafür sorgen, dass Access für das Feld **KundeSeit** immer automatisch das aktuelle Datum einträgt. Außerdem dürften neu angelegte Kunden gleichzeitig immer aktive Kunden sein, weshalb eine Voreinstellung des Feldes **Aktiv** ebenfalls sinnvoll wäre.

# 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

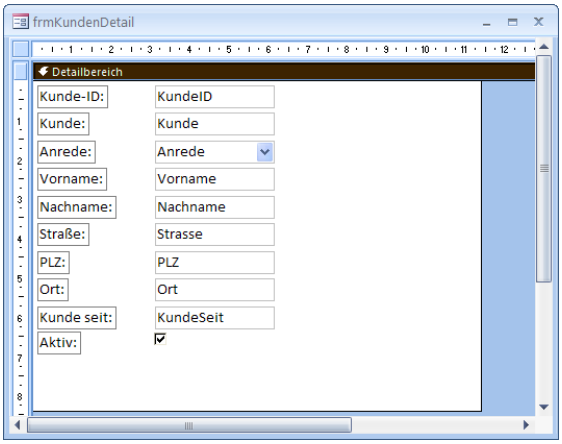
Die Beispieldatenbank zu diesem Artikel finden Sie im Onlinearchiv unter

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

### Formular erstellen

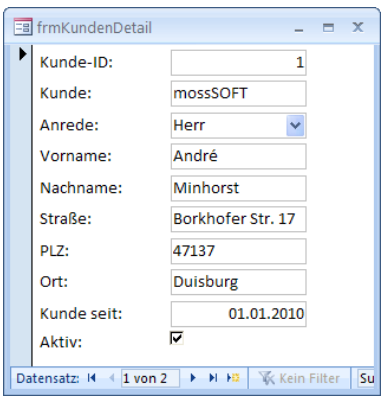
Unser Formular soll das Anlegen neuer Kunden, das Bearbeiten vorhandener Kunden und das Löschen von Kunden erlauben. Dafür brauchen Sie gar nicht so viel zu tun: Sie müssen einfach nur ein neues, leeres Formular erstellen, seine Eigenschaft **Datenherkunft** auf den Namen der Tabelle einzustellen, deren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



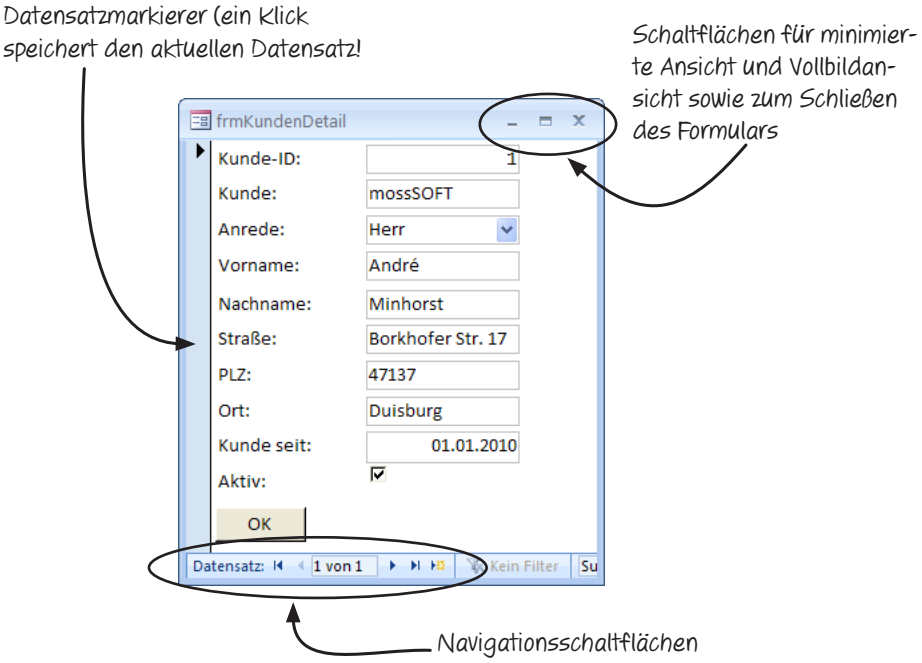
## Funktionen gebundener Formulare

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

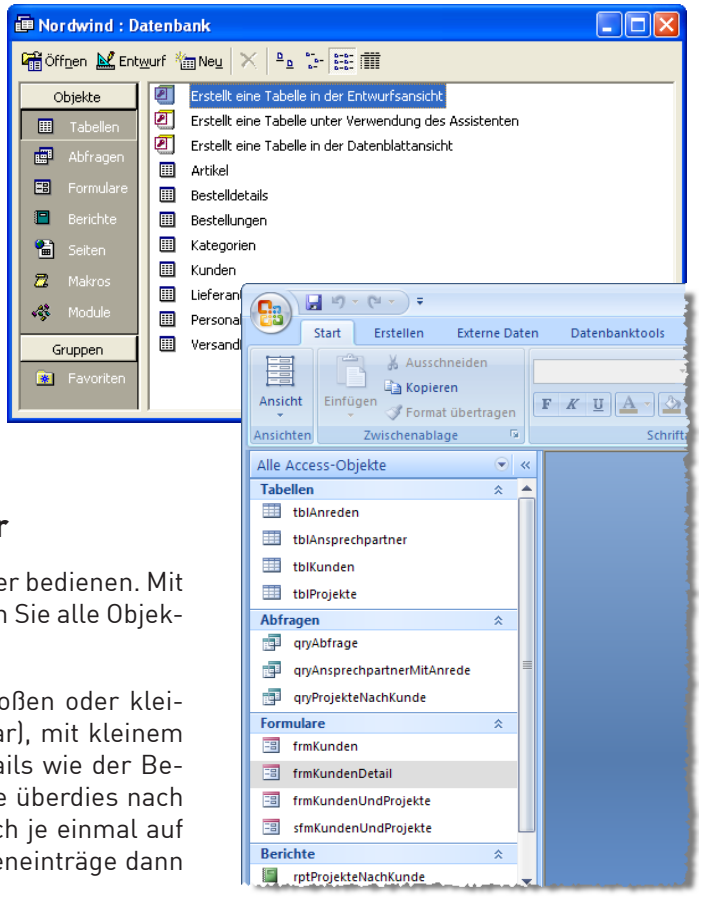
- die Tastenkombination **Strg + S** betätigen,

# 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.

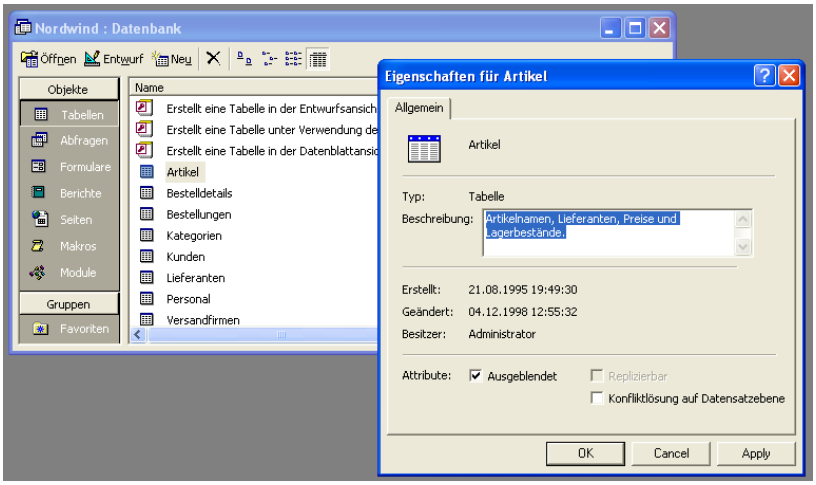
## Das Datenbankfenster unter Access 2003 und älter

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.

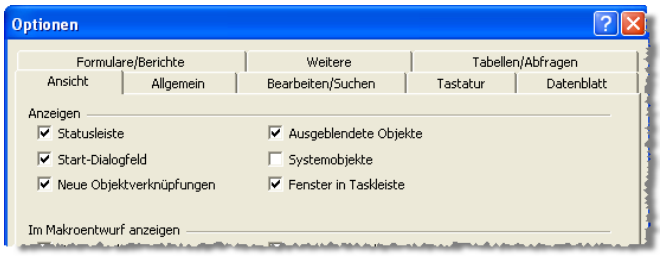
Diese können Sie auf verschiedene Arten darstellen: mit großen oder kleinen Symbolen und Name (alle Objekte sind frei positionierbar), mit kleinem Symbol und Name in Listenform oder in einer Liste mit Details wie der Beschreibung oder dem Erstellungsdatum. Die Liste können Sie überdies nach den verschiedenen 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**

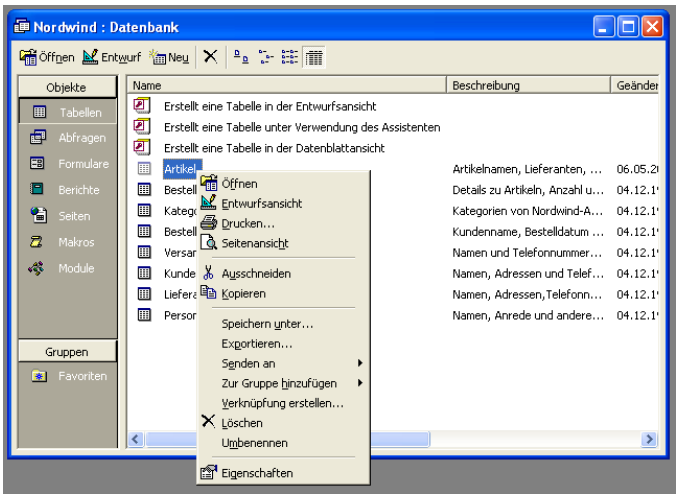
Access speichert einige interne Informationen, zum Beispiel über die enthaltenen Objekte, in Systemtabellen. Diese sind normalerweise nicht sichtbar. Wenn Sie jedoch die Option **Systemobjekte** in obigem Dialog aktivieren, finden Sie im Datenbankfenster auch die Systemtabellen vor. Aktuell können Sie 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.

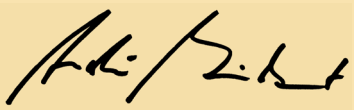
Sie erfahren, wie Sie Projekt- und Kundendaten auf zwei Tabellen aufteilen. Warum? Weil Kunden und Projekte in einer einzigen Tabelle spätestens dann Redundanzen (also doppelt gehaltene Daten) nach sich ziehen, wenn Sie mehr als nur ein Projekt für einen Kunden verwalten möchten.

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**.

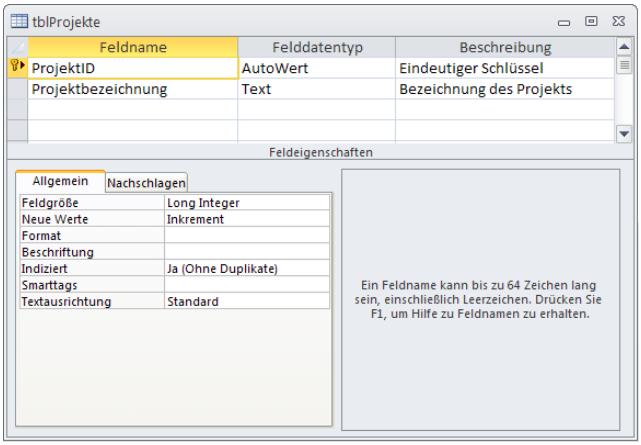
**Projektbezeichnung**: Eine Bezeichnung für das Projekt, zum Beispiel **Access [basics] Ausgabe 3**. Das Feld soll den Datentyp **Textfeld** erhalten →[basics] **Textfeld anlegen**.

### 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 ...

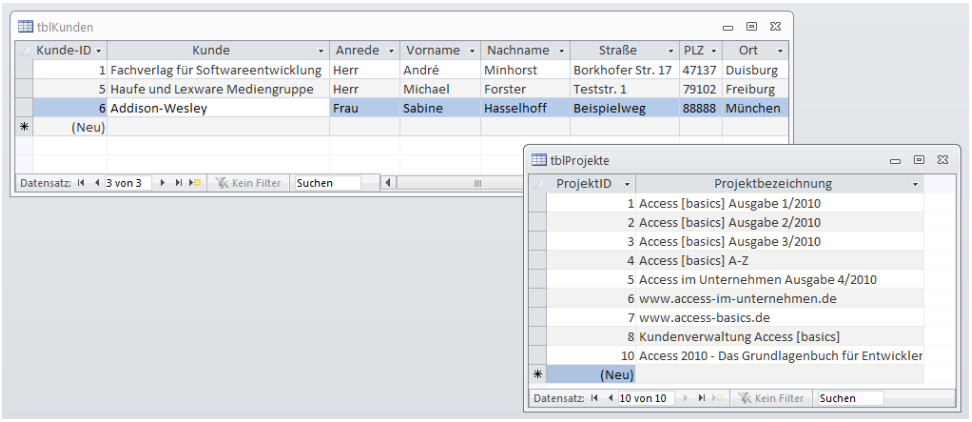
Später, wenn Sie die Projektzeiterfassung erweitern möchten, um beispielsweise auch die Projektplanung zu berücksichtigen, können Sie die Projekttabelle erweitern. Vorerst fehlt uns allerdings nur eine Kleinigkeit, denn: Woher wissen wir eigentlich, für welchen Kunden wir dieses Projekt durchführen?

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

# 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.

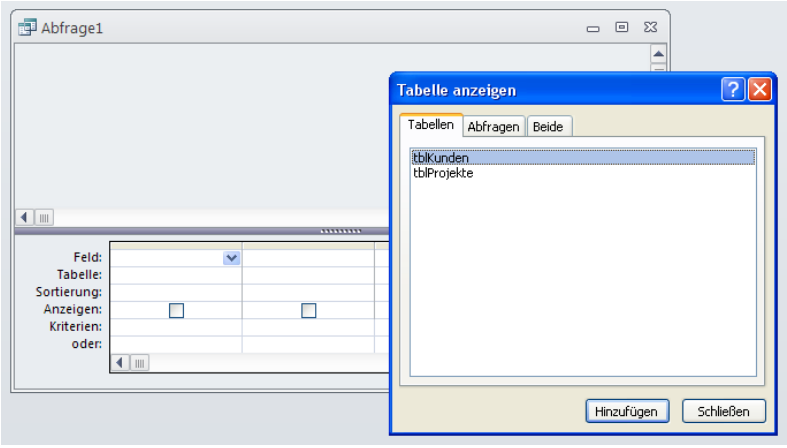
In diesem Artikel beschäftigen wir uns zum Warmwerden mit einfachen Auswahlabfragen. Weiter unten steigern wir dann den Schwierigkeitsgrad ...

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.



# 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.**

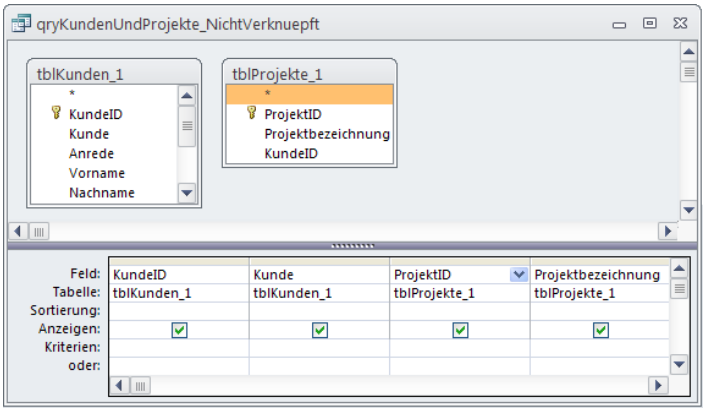
### Was Sie in diesem Artikel lernen

- 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.

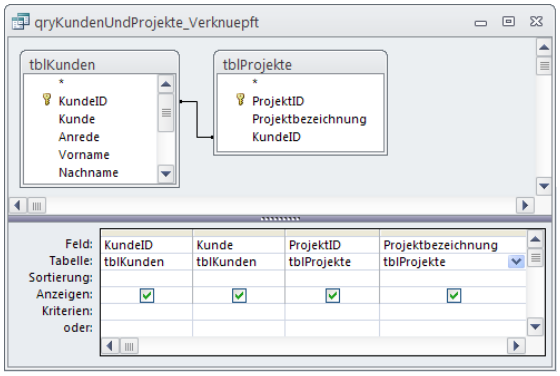



| Kunde-ID | Kunde | ProjektID | Projektbezeichnung |
|---|---|---|---|
| 1 | Fachverlag für Softwareentwicklung | 1 | Access [basics] Ausgabe 1/2010 |
| 5 | Haufe und Lexware Mediengruppe | 1 | Access [basics] Ausgabe 1/2010 |
| 6 | Addison-Wesley | 1 | Access [basics] Ausgabe 1/2010 |
| 7 | Müller 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 |
| 1 | Fachverlag für Softwareentwicklung | 5 | Access im Unternehmen Ausgab |

## 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**.

Bereits die Entwurfsansicht verrät, dass die Dinge hier anders laufen werden: Der Beziehungspfeil zwischen den beiden Tabellen wird nämlich automatisch in den Abfrageentwurf übernommen. Davon abgesehen ziehen wir die gleichen Felder wie in der vorherigen Abfrage in das Entwurfsraster und wechseln anschließend in die Datenblattansicht.




Das Ergebnis sieht wesentlich besser aus: Es zeigt tatsächlich nur diejenigen Kombinationen aus Kunden und Projekten an, die über das Fremdschlüsselfeld **KundeID** der Tabelle **tblProjekte** festgelegt wurden. Die Anzahl der Datensätze entspricht somit der Anzahl der Datensätze der Tabelle **tblProjekte**, denn die Verknüpfung fügt ja jedem Datensatz dieser Tabelle je lediglich einen Datensatz der Tabelle **tblKunden** hinzu.

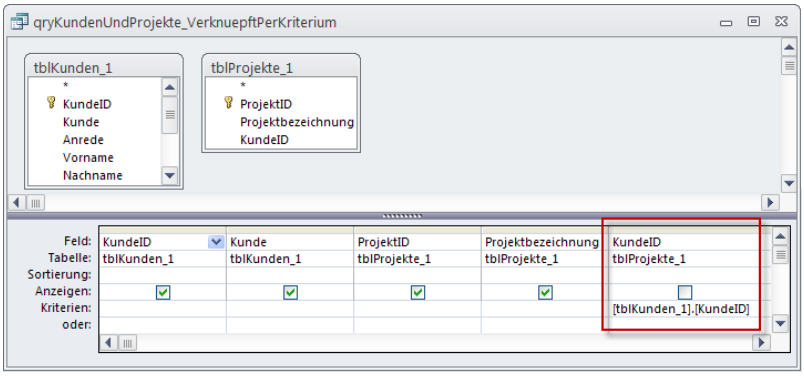
qryKundenUndProjekte_Verknuepft

| Kunde-ID | Kunde | ProjektID | Projektbezeichnung |
|---|---|---|---|
| 1 | Fachverlag für Softwareentwicklung | 1 | Access [basics] Ausgabe 1/2010 |
| 1 | Fachverlag für Softwareentwicklung | 2 | Access [basics] Ausgabe 2/2010 |
| 1 | Fachverlag für Softwareentwicklung | 3 | Access [basics] Ausgabe 3/2010 |
| 1 | Fachverlag für Softwareentwicklung | 4 | Access [basics] A-Z |
| 1 | Fachverlag für Softwareentwicklung | 7 | www.access-basics.de |
| 1 | Fachverlag für Softwareentwicklung | 8 | Kundenverwaltung Access [basics] |
| 5 | Haufe und Lexware Mediengruppe | 5 | Access im Unternehmen Ausgabe 4/2010 |
| 5 | Haufe und Lexware Mediengruppe | 6 | www.access-im-unternehmen.de |
| 6 | Addison-Wesley | 10 | Access 2010 - Das Grundlagenbuch für Entwickler |
| (Neu) | | (Neu) | |

Datensatz: 1 von 9 Kein Filter Suchen

Im Prinzip bewirkt die Übernahme der Beziehung zwischen **tblKunden** und **tblProjekte** also eine Art Kriterium, welches die Anzahl der angezeigten Datensätze des Abfrageergebnisses gegenüber dem ohne Beziehung begrenzt.

Sie glauben nicht, dass sich dies mit einem herkömmlichen Kriterium vergleich lässt? Das tut es aber! Kehren wir doch zu der Abfrage des vorletzten Beispiels zurück, das die beiden unverknüpften Tabellen als Datenherkunft enthielt. Erstellen Sie eine Kopie dieser Abfrage und speichern Sie diese unter dem Namen **qryKundenUndProjekte_VerknuepftPerKriterium**.




Nun ziehen Sie das Fremdschlüsselfeld **KundeID** der Tabelle **tblProjekte** in das Entwurfsraster und erledigen dann zwei Dinge: Erstens tragen Sie als Kriterium den Ausdruck **[tblKunden_1].[KundeID]** ein und zweitens entfernen Sie das Häkchen in der Zeile **Anzeigen** für dieses Feld.

Nach dem Wechsel in die Entwurfsansicht zeigt die Abfrage genau das gleiche Ergebnis an wie die Abfrage mit der Beziehung zwischen den Tabellen **tblKunden** und **tblProjekte**!

Was ist hier geschehen? Im Prinzip nicht viel: Wir haben der Abfrage lediglich ein Kriterium für das Fremdschlüsselfeld **KundeID** der Tabelle **tblProjekte** hinzugefügt, das dafür sorgt, dass die Abfrage nur solche Kombinationen von Datensätzen der Tabellen **tblKunden** und **tblProjekte** liefert, die im Feld **KundeID** den gleichen Wert haben.

# 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.**

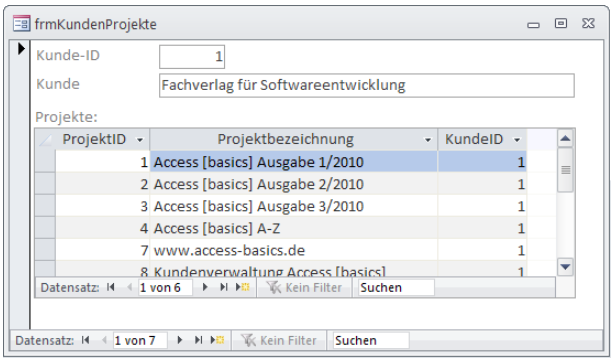
**Was Sie in diesem Artikel lernen**

- 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



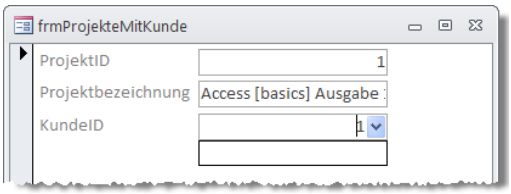

Die einfachere Variante der Darstellung zeigt die Projektdaten plus dem Kunden des Projekts an. Um dies zu realisieren, erstellen Sie einfach wie in →*[basics]* **Gebundenes Formular anlegen** beschrieben ein neues Formular an, das als Datenherkunft die Tabelle **tblProjekte** verwendet und alle Felder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

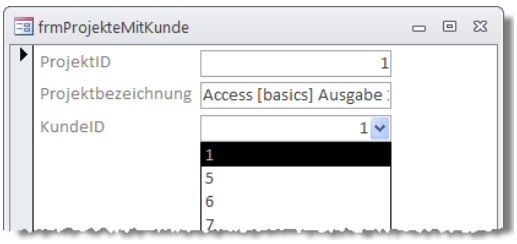
Wenn Sie nun in die Formularansicht wechseln, zeigt das Formular zwar ein Kombinationsfeld an. Wenn Sie dieses aber durch einen Klick auf die Schaltfläche am rechten Rand des Kombinationsfelds aufklappen, finden Sie nur gähnende Leere vor.




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**.

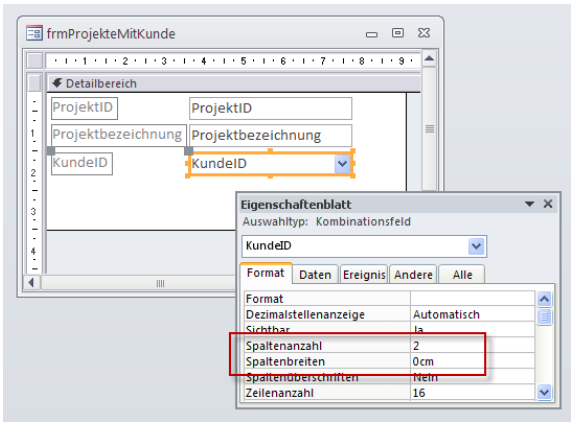
Das neue Zwischenergebnis in der Formularansicht sieht besser aus: Immerhin bietet das Kombinationsfeld nun schon einmal alle Werte des Feldes **KundeID** der Tabelle **tblKunden** an.




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

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.

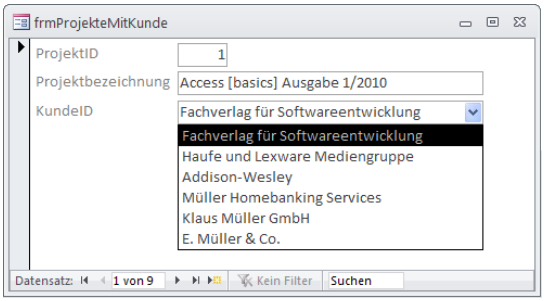
Um dieses nächste Ziel zu erreichen, brauchen Sie nur zwei weitere Eigenschaften zu ändern: Stellen Sie einfach auf der Registerseite Format die Eigenschaft **Spaltenanzahl** auf **2** und **Spaltenbreiten** auf **0cm** ein. Wie meinen? Zwei Spalten, aber nur eine Spaltenbreite angeben – und die auch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




Mal sehen, was diese Änderung bewirkt.

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.

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.

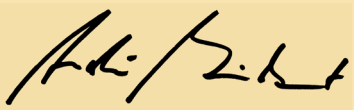
In der letzten Ausgabe haben wir uns um die Kunden und Projekte gekümmert und dabei eine 1:n-Beziehung und verschiedene Möglichkeiten der Darstellung der in den an dieser Beziehung beteiligten Tabellen vorgestellt. In dieser Ausgabe widmen wir uns zunächst einem speziellen Typ der 1:n-Beziehung, nämlich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

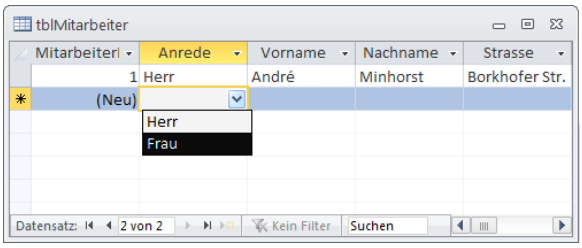
**http://www.access-basics.de/index.php/Heftarchiv.html**.

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!



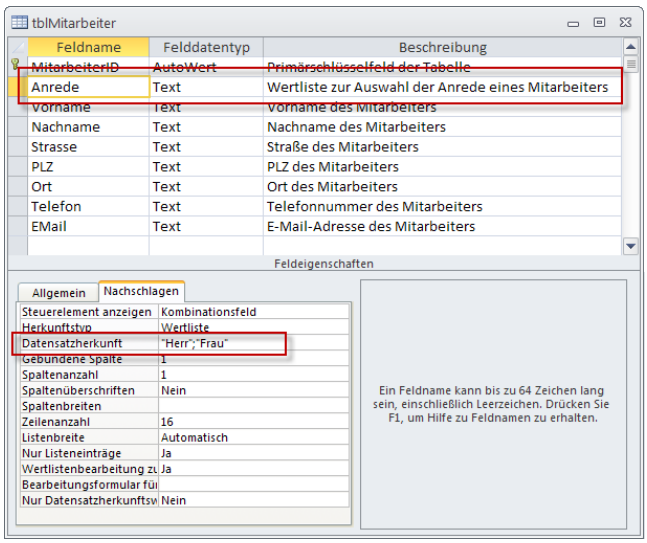

## Wertlisten und Tabellen als Datenquelle von Nachschlagefeldern

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

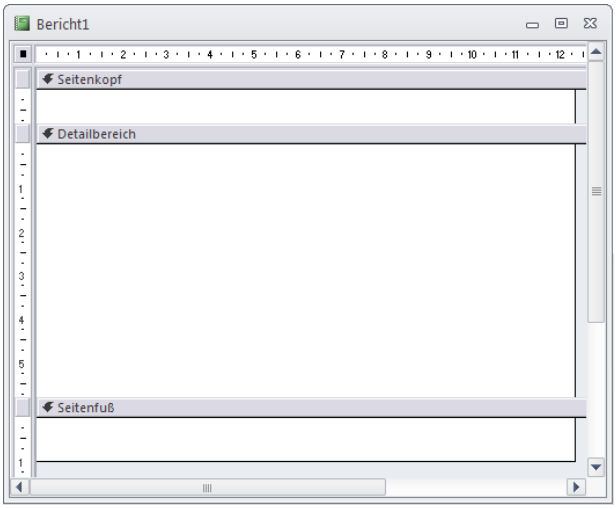
**http://www.access-basics.de/index.php/Heftarchiv.html**.

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**.

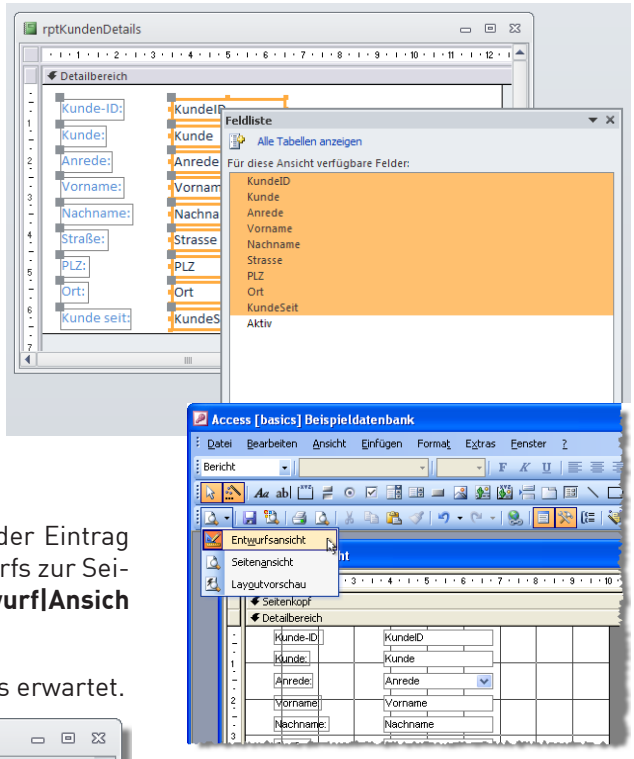
Der Bericht sieht nach dem Anlegen wie in der Abbildung rechts aus. Er enthält noch keine Steuerelemente, aber drei Bereiche: **Seitenkopf**, **Detailbereich** und **Seitenfuß**. Mit diesen wollen wir uns vorerst zufrieden geben – genau genommen mit noch weniger: Blenden Sie einfach mit dem Kontextmenü-Eintrag **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.



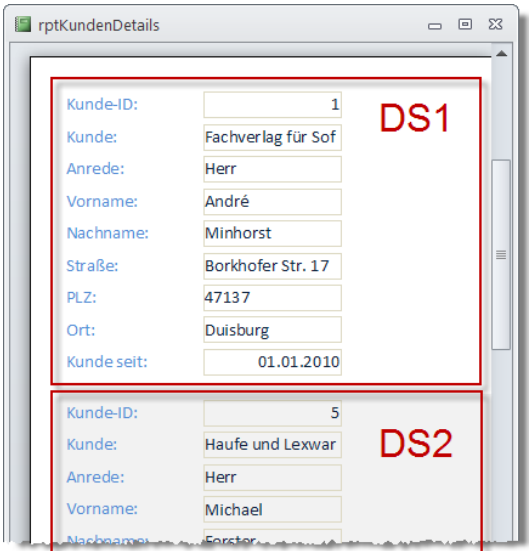

Sie können sich nun gleich einmal das Zwischenergebnis ansehen, indem Sie in die Seitenvorschau des Berichts wechseln. Dazu gibt es verschiedene Möglichkeiten:

**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:

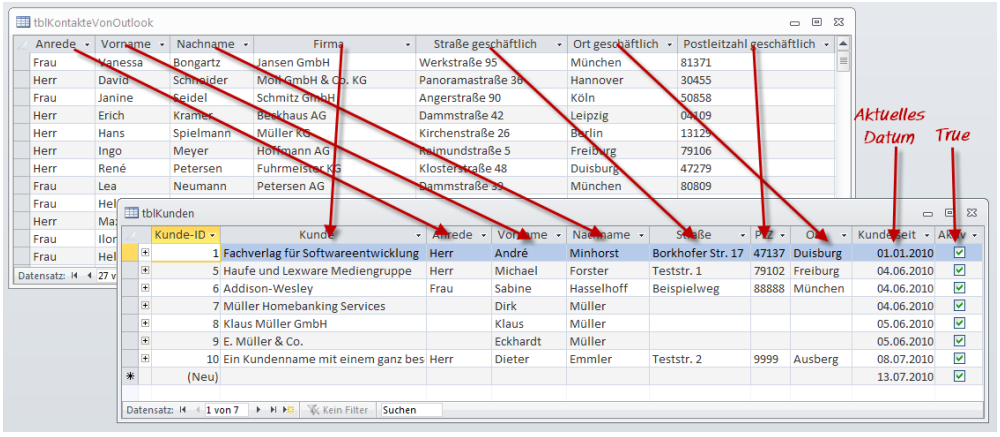
- Die Abfrage liest die anzufügenden Daten aus einer oder mehreren anderen Tabellen oder Abfragen ein und schreibt diese in die Zieltabelle.
- Die Abfrage schreibt nur einen Datensatz in die Zieltabelle, deren Werte Sie manuell vorgeben – also ohne Verwendung einer Quelltabelle.

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.

# 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

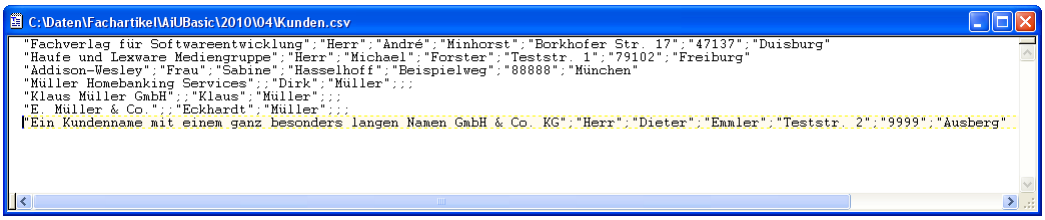
**Beispieldatenbank**

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.

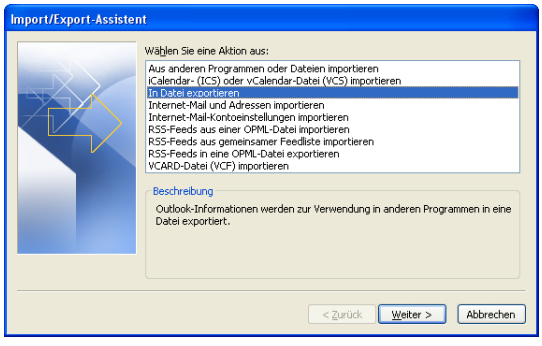



Die Abbildung zeigt ein Beispiel für eine recht einfache und – wie Sie gleich sehen werden – einfach zu importierende CSV-Datei. Die Datei weist eine konsistente Struktur auf: Sie speichert die Daten eines Kunden jeweils in einer Zeile und die einzelnen Informationen sind durch Semikola voneinander getrennt. 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.



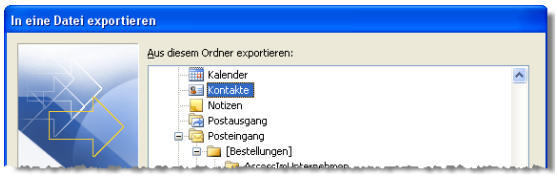

Wählen Sie im Dialog **Import/Export-Assistent** den Eintrag **In Datei exportieren** aus und im nächsten Dialog den Eintrag **Kommagetrennte Werte** oder **Tabulatorgetrennte Werte** aus (außerdem sollten Sie die Windows-Variante verwenden – auf diese Weise verhindern Sie später Probleme mit Sonderzeichen 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.

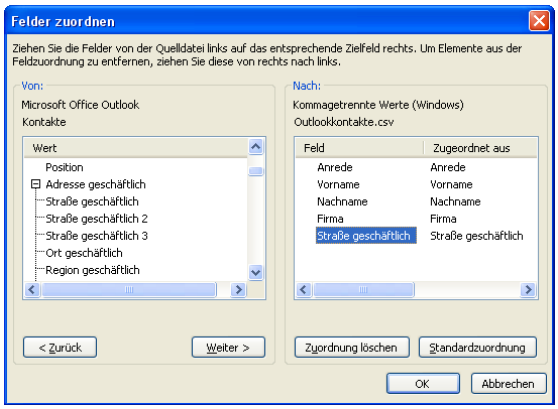
Im deutschsprachigen Raum (und auch anderswo) wird außerdem das Komma als Trennzeichen für Dezimalzahlen verwendet. Die weiter unten vorgestellten Assistenten für das Importieren und Verknüpfen externer Daten meckern übrigens von selbst, wenn das Trennzeichen und das Dezimalzeichen identisch sind. Sorgen 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

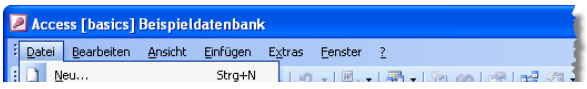
Schauen wir uns nun die verschiedenen Möglichkeiten an, Daten aus einer CSV-Datei zu importieren. Im Wesentlichen gibt es diese Varianten:

- 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:



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

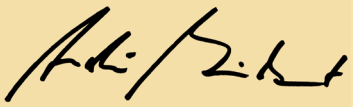


Nach den ersten vier Ausgaben von **Access [basics]**, die großes Interesse auslösten und viele Anregungen brachten, ist es Zeit für ein wenig Optimierung. Sie werden feststellen, dass diese Ausgabe nicht mehr einspaltig mit Screenshots und Kommentaren am rechten Rand, sondern zweispaltig daherkommt - 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.

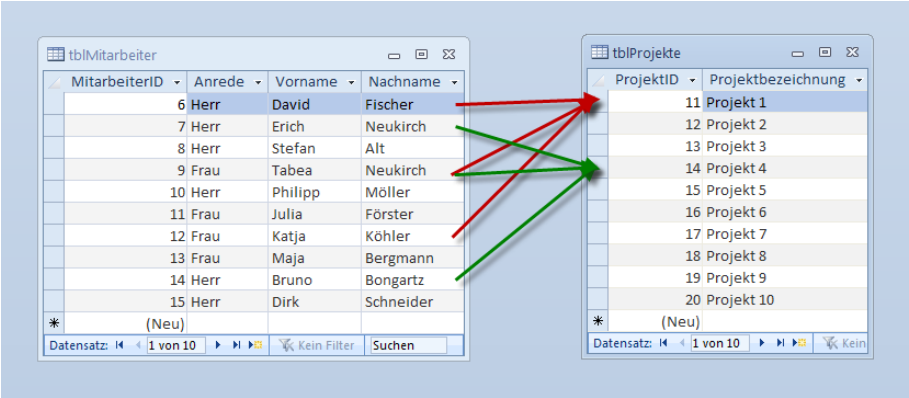



**Bild 1:** Zuordnung von Projekten zu Mitarbeitern und umgekehrt

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.

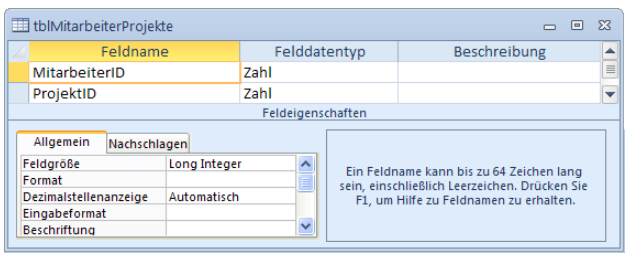



**Bild 2:** Zuordnung von Projekten zu Mitarbeitern und umgekehrt

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**).

# 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.

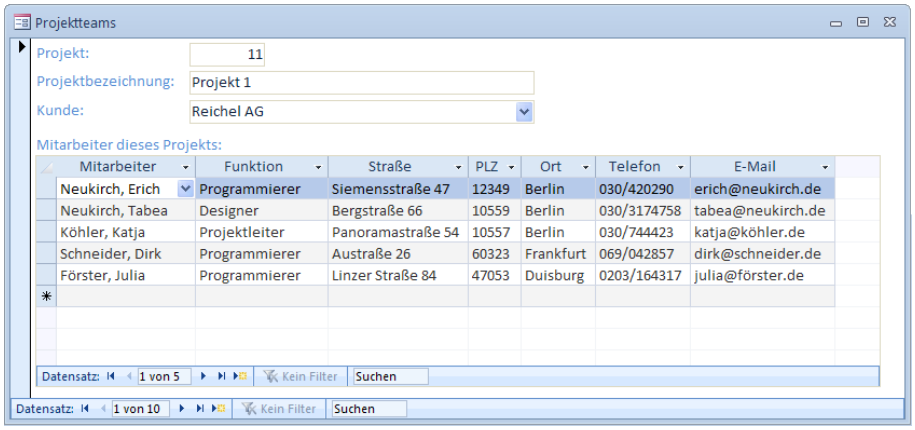



**Bild 1:** So soll das Formular zur Verwaltung von Daten aus zwei per m:n-Beziehung verknüpften Tabellen später aussehen.

## 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

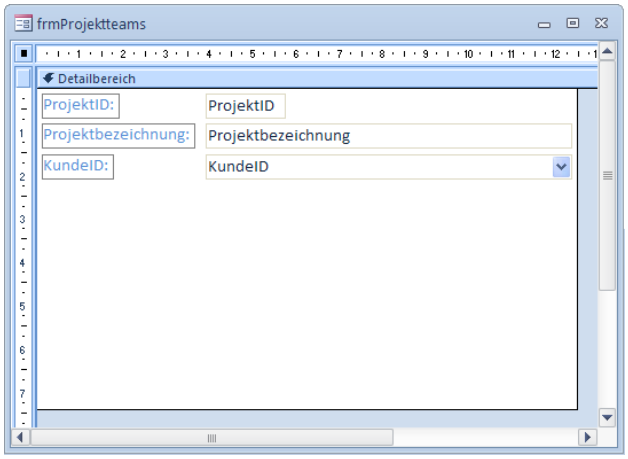



**Bild 2:** Das Hauptformular mit den Feldern der Tabelle **tblProjekte**

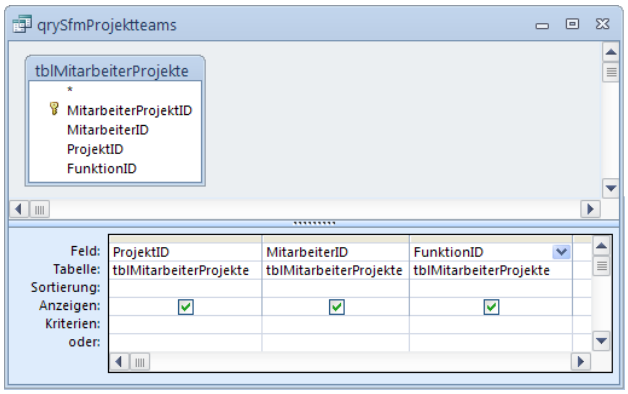



**Bild 3:** Dies ist die Datenherkunft der einfachsten Variante des Unterformulars auf Basis der Tabelle **tblMitarbeiterProjekte**.

**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.

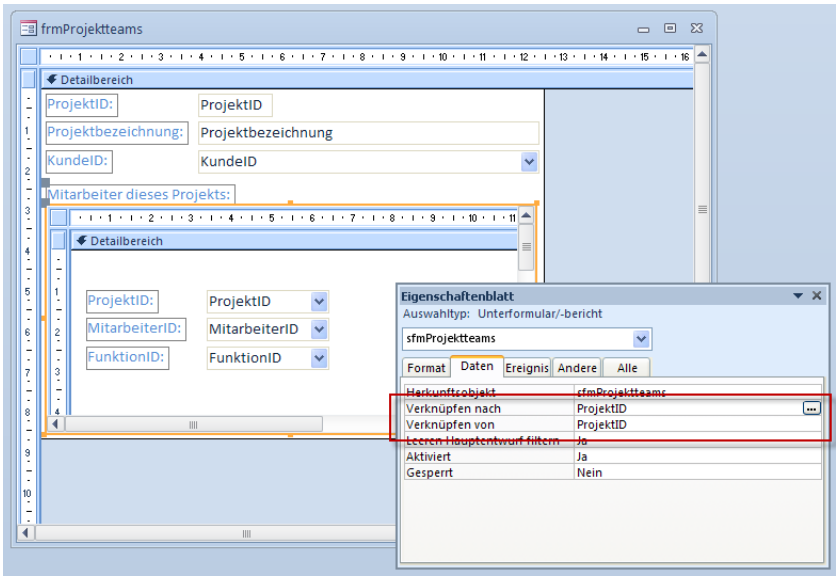



**Bild 4:** Damit das Unterformular die zum Datensatz im Hauptformular passenden Daten anzeigt, müssen Sie diese Eigenschaften entsprechend einstellen.

# 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**.

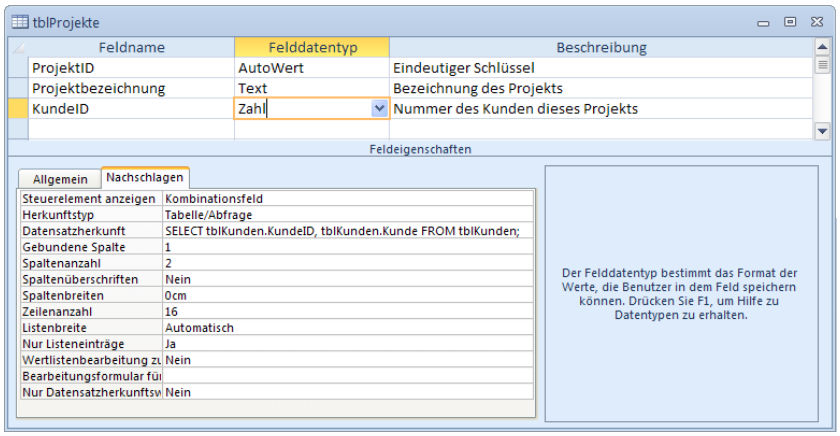



**Bild 1:** Das Feld **KundeID** ist als Nachschlagefeld ausgelegt, was die Eigenschaften der Registerseite **Nachschlagen** der Feldeigenschaften verdeutlichen.

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.

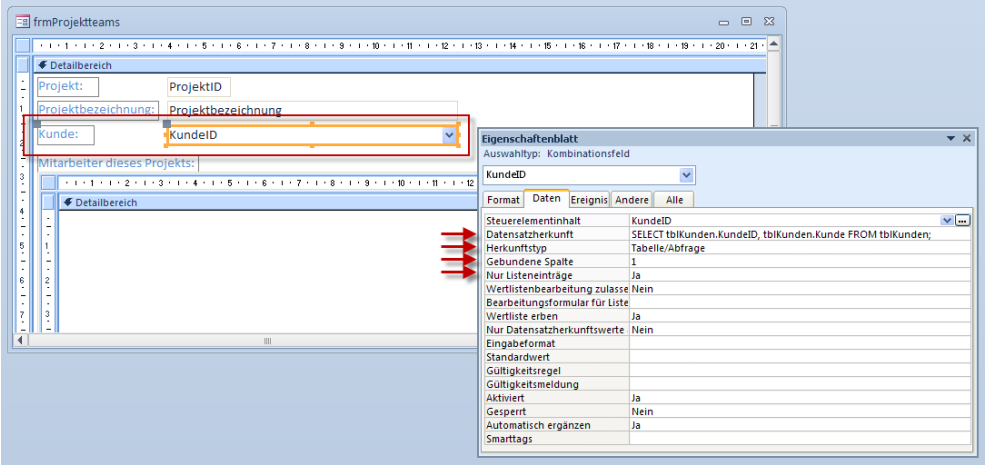



**Bild 2:** Einige der vom Nachschlagefeld übernommenen Eigenschaften finden Sie auf der Registerseite **Daten** des Kombinationsfeldes **KundeID** wieder.

## 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.

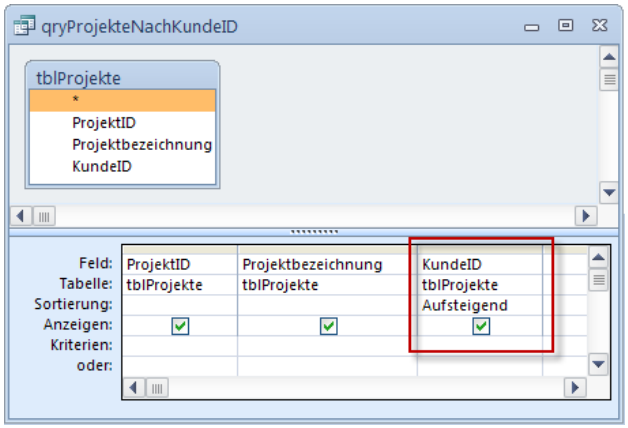



**Bild 3:** Diese Einstellung sollte die Datensätze der Tabelle **tblProjekte** nach dem Kundennamen sortieren.

## 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).

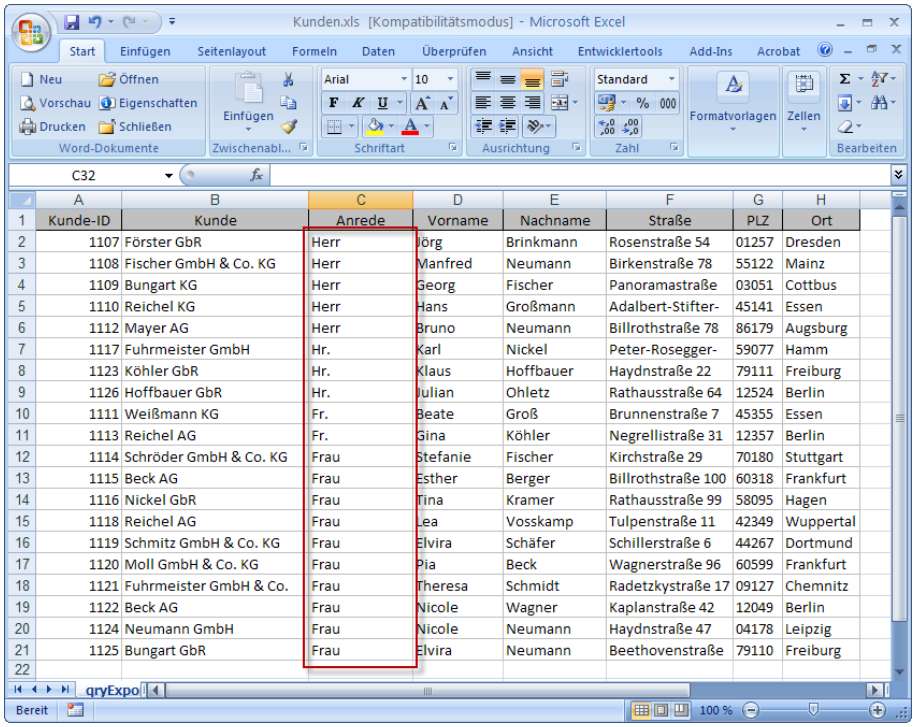
Des Rätsels Lösung ist, dass Access schlicht nach dem eigentlichen Wert des Feldes **KundeID** sortiert. Wenn Sie tatsächlich nach dem Kundennamen sortieren möchten, müssen Sie der Abfrage die verknüpfte Tabelle **tblKunden** hinzufügen und das Feld mit dem eigentlichen Kunden namen als Sortierkriterium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

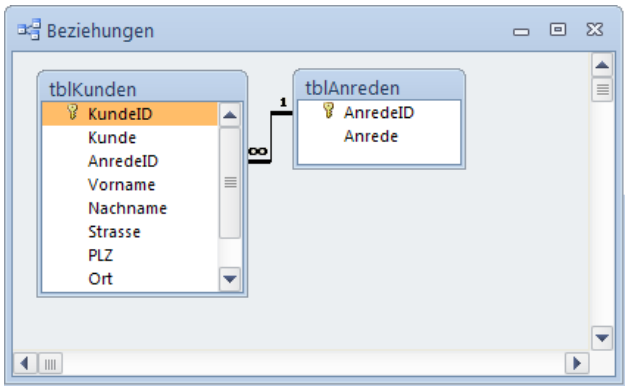



**Bild 2:** Ziel des Kundenimports: Die Tabelle **tblKunden**

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.

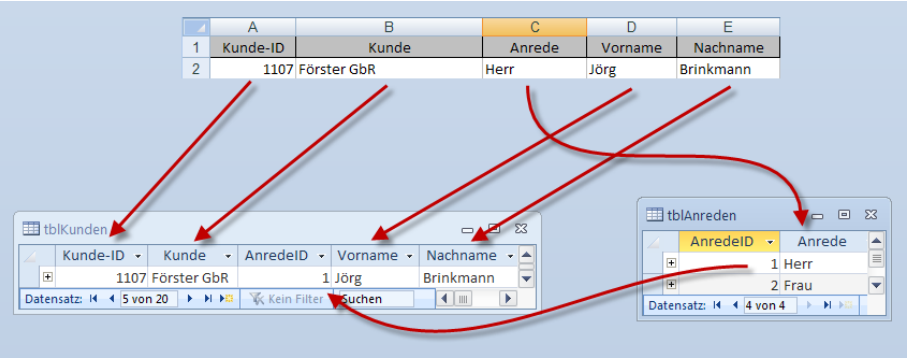



**Bild 3:** Die meisten Felder landen ohne Umweg in der Tabelle **tblKunden**. Nur die Anrede macht einen Umweg über die Tabelle **tblAnreden**.

## 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**).

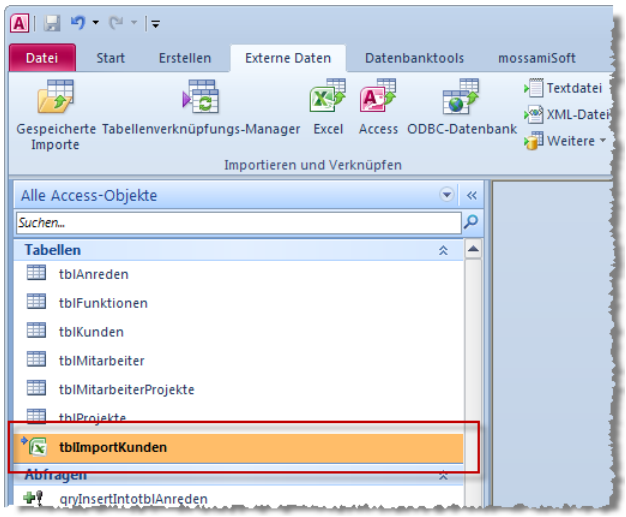



**Bild 4:** Liegt erst einmal eine Verknüpfung auf die Daten vor, können wir diese einfach mit Bordmitteln in die Zieltabellen überführen.

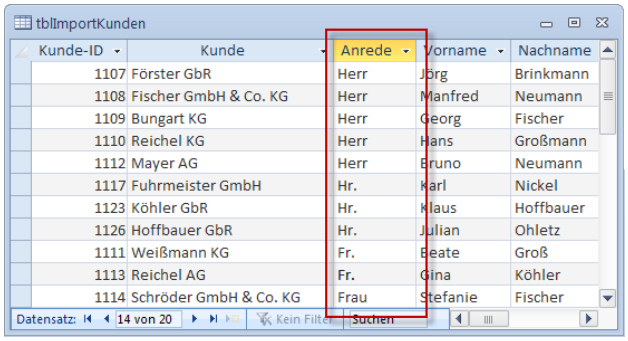



**Bild 5:** Der Inhalt der Verknüpfung sieht nun schon fast gut aus – wenn man vom Feld Anrede absieht ...

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).

## 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.

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

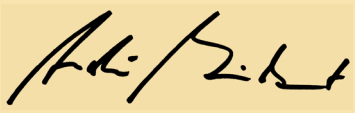
### Was Sie in diesem Artikel lernen

» 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**

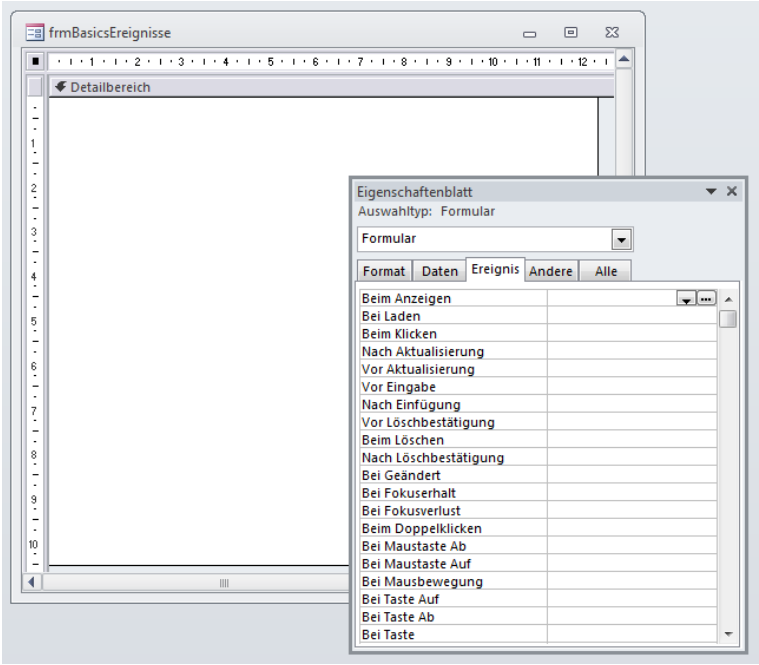



**Bild 1:** Das Eigenschaftsfenster mit den Ereigniseigenschaften

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**

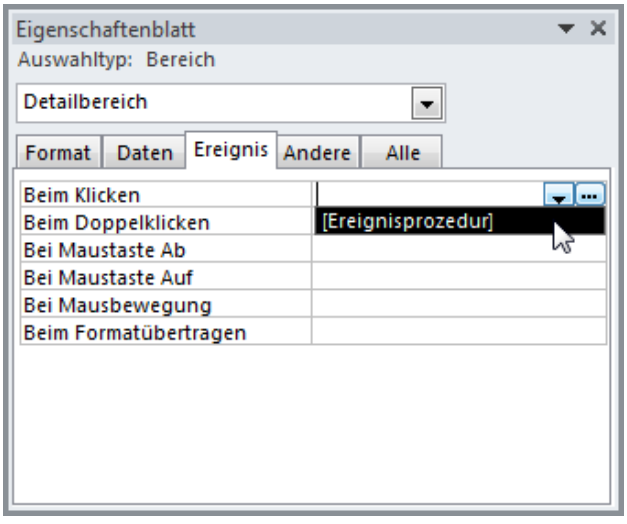



**Bild 2:** Einstellen des Wertes **[Ereignisprozedur]** für eine Ereigniseigenschaft

# 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).

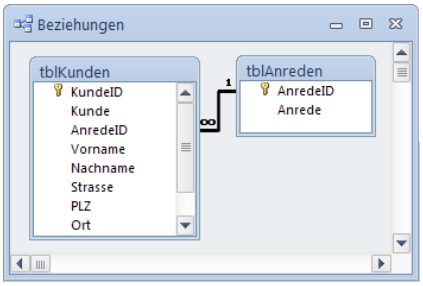



**Bild 1:** Die beiden Tabellen **tblKunden** und **tblAnreden** sind über das Feld **AnredeID** miteinander verknüpft.

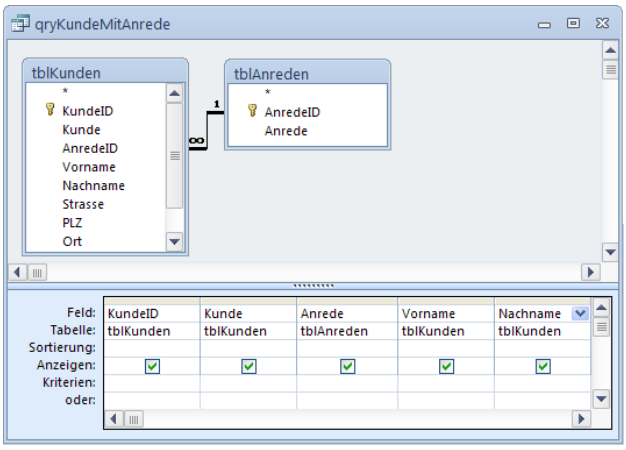



**Bild 2:** Beim Hinzufügen dieser beiden Tabellen wird die Beziehung aus dem Datenbankfenster in die Abfrage übernommen.




**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.

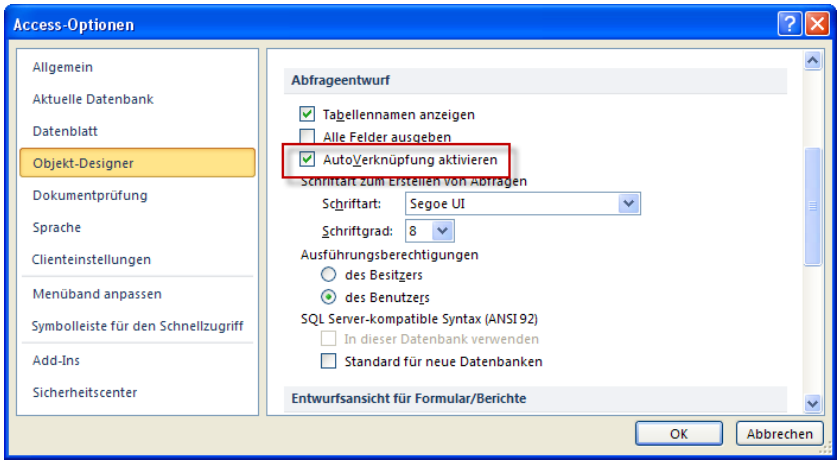



**Bild 5:** Diese Option sorgt mitnichten dafür, dass Beziehungen zwischen zwei Tabellen in den Abfrageentwurf übernommen werden.

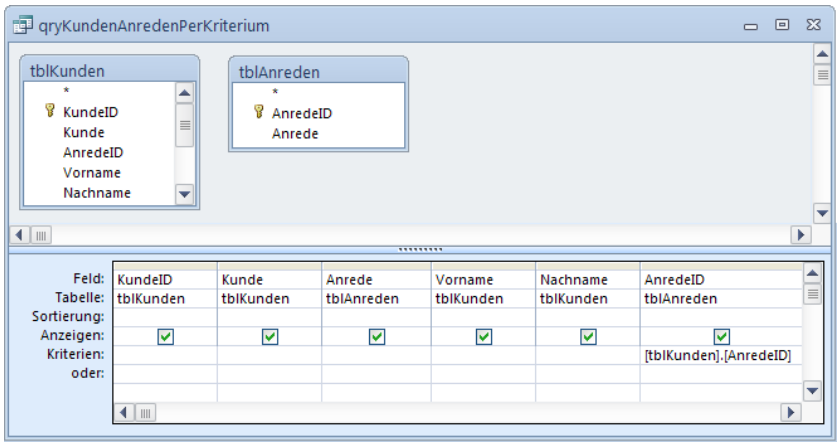



**Bild 6:** Die Funktion des Beziehungspfeils in Abfragen kann auch durch ein Kriterium nachgestellt werden.

# 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
```

# 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.

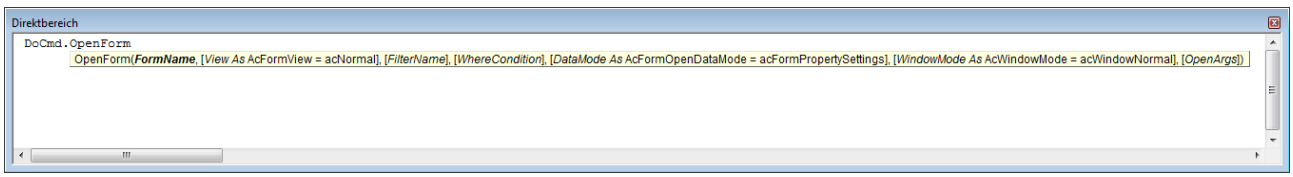



**Bild 1:** Die Parameterliste der DoCmd.OpenForm-Anweisung

# 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

einfach ein Meldungsfenster anzeigen, also nennen Sie diese **cmdMeldung**.

## Schaltflächenbild

Gegebenenfalls möchten Sie der Schaltfläche statt einer Beschriftung ein Bild hinzufügen oder gar Bild und Beschriftung gleichzeitig anzeigen. Beides ist in älteren Access-Versionen nicht ohne Weiteres möglich, aber ab Access 2007 können Schaltflächen auch Bilder und/oder Text anzeigen (wichtige Hinweise dazu finden Sie unter →*[basics]* **Bilder unter Access 2007 und jünger**).

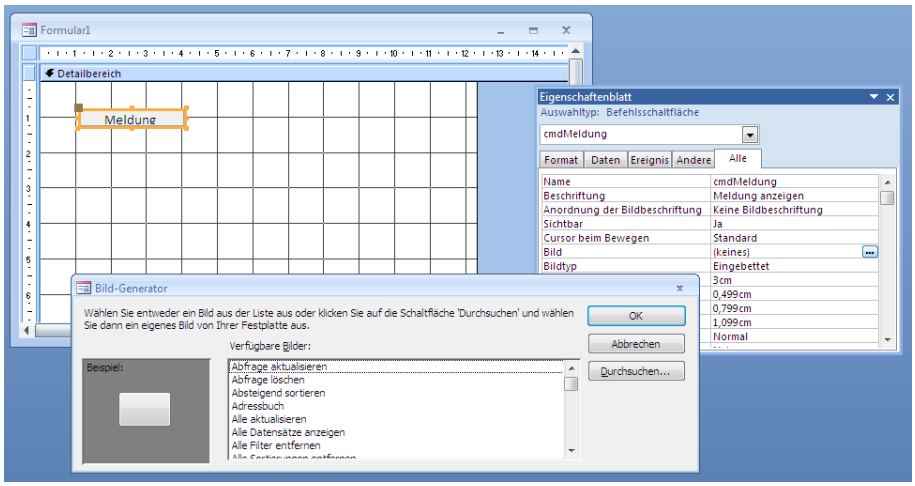



**Bild 1:** Einfügen eines Bildes für eine Schaltfläche

Dazu klicken Sie auf die Eigenschaft **Bild** und auf die dann erscheinende Schaltfläche mit den drei Punkten (...). Dies öffnet den Bild-Generator. Wenn Sie ein eigenes Bild verwenden möchten, sollte es sich um eine **.png**- oder **.gif**-Datei mit transparentem Hintergrund handeln. Eigene Bilder können Sie nach einem Klick auf die Schaltfläche **Durchsuchen...** auswählen.

## Action, bitte!

Nun soll die fertig gestaltete Schaltfläche natürlich auch etwas bewirken. Dazu benötigen Sie etwas VBA-Code. Genau genommen könnten Sie dies auch mit einem Makro erledigen, aber Sie würden damit auf eine Menge Flexibilität verzichten – und das möchten Sie sicher nicht.

Der VBA-Code, den Sie für das Ausführen bestimmter Anweisungen nach dem Anklicken einer Schaltfläche benötigen, landet in einer sogenannten Ereignisprozedur. Der Weg dorthin ist einfach: Als Erstes markieren Sie im Formularentwurf die Schaltfläche, die den Code ausführen soll. Dann aktivieren Sie, sofern es noch nicht sichtbar ist, das Eigenschaftsfenster mit der Taste **F4**. Wechseln Sie zur Registerseite **Ereignis** und klicken Sie auf den Eintrag **Beim Klicken**. Wählen Sie den Wert **[Ereignisprozedur]** aus und klicken Sie dann auf die Schaltfläche mit den drei Punkten (...) – siehe Bild 2. Es erscheint der VBA-Editor, der automatisch ein Klassenmodul angelegt hat, das alle Ereignisprozeduren und weitere Elemente aufnimmt, die Sie in Zusammenhang mit dem jeweiligen Formular erzeugen. Außerdem hat der VBA-Editor gleich den Rahmen der Prozedur erstellt, der wie folgt aussieht:

```
Private Sub cmdMeldung_Click()

End Sub
```

Fügen Sie hier die folgende Zeile ein:

```
MsgBox "Klick!"
```

Nun können Sie die Schaltfläche bereits testen. Wechseln Sie über den entsprechenden Menü-/Ribbon-Eintrag oder mit **Strg + Punkt** in die Formularansicht des Formulars und klicken Sie auf die Schaltfläche. Es erscheint das gewünschte Meldungsfenster.

## Aktivieren und deaktivieren

Außer dem Anklicken gibt es wirklich nicht viel, was Sie mit einer Schaltfläche anstellen können – außer

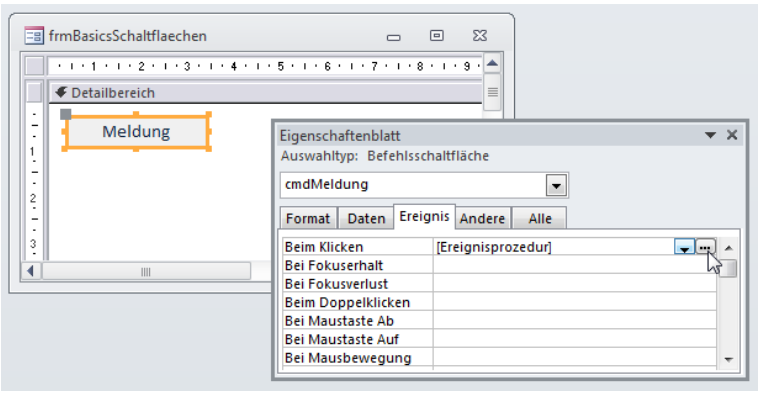



**Bild 2:** Anlegen einer Ereignisprozedur für die Schaltfläche

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

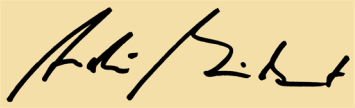
Nach sechs Ausgaben und einem halben Jahr **Access [basics]** ist es Zeit für Veränderungen. Das Magazin soll noch professioneller werden, und dafür haben wir mit Lorenz Hölscher einen weiteren Autor gewonnen. Er hat bereits einige Bücher über Access für Microsoft Press geschrieben und außerdem Artikel 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!“

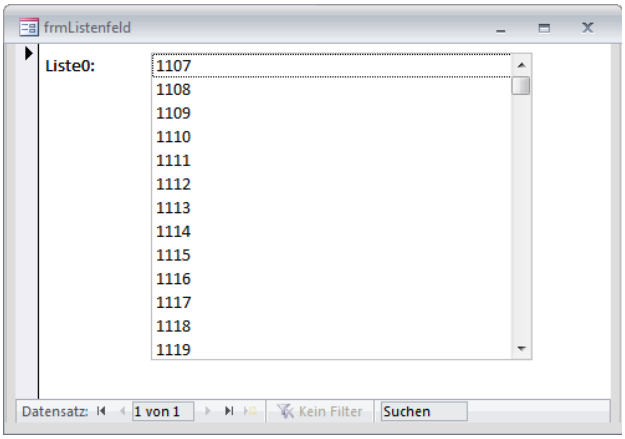



**Bild 2:** Die Liste zeigt die Kunden-IDs an.

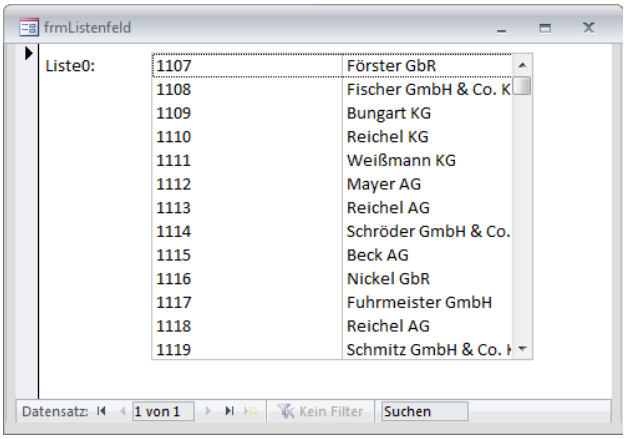



**Bild 1:** Mit der zweiten (Spalte) sieht man besser...

# 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).

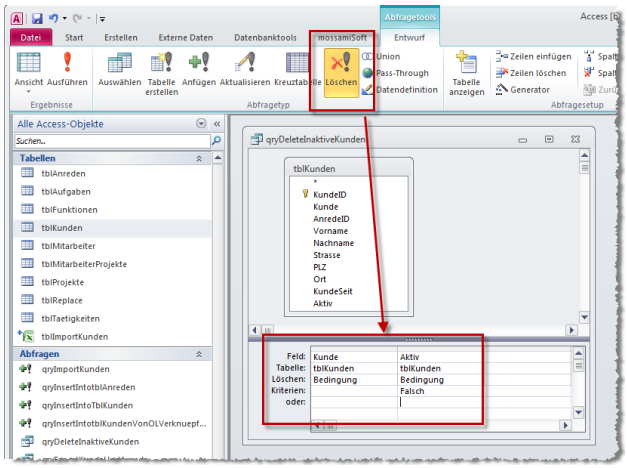



**Bild 1:** Diese Abfrage liefert alle inaktiven Kunden.

### 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

Wenn der Chef der Firma Schwuppdiwupp diesen Vorgang regelmäßig wiederholen möchte, kann er die Sache jedoch noch weiter vereinfachen. Dazu braucht er nur noch eine kleine Änderung an der Abfrage vorzunehmen. Dazu brauchen unter Access 2007 und jünger nur wie in Bild 2 den Ribbon-Eintrag **Entwurf|Abfragetyp|Löschen** zu betätigen. Versionsübergreifend steht der Kontextmenübefehl **Abfragetyp|Löschabfrage** der Abfrage für diesen Zweck bereit (dazu mit der rechten Maustaste auf den grauen Bereich oben im Abfrageentwurf klicken).

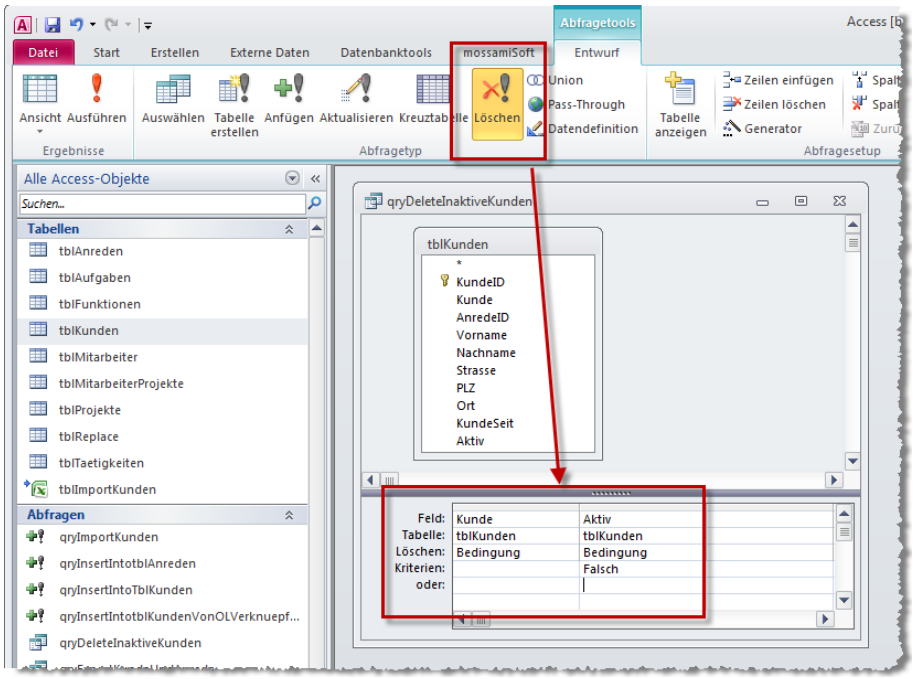



**Bild 2:** Umwandeln einer Auswahlabfrage in eine Löschabfrage

Access ändert den Entwurf nun geringfügig. Die Zeile **Anzeigen** fällt weg und auch eine Sortierung ist für das Löschen von Daten nicht mehr nötig.

Bevor Sie die Daten tatsächlich löschen, können Sie sich die betroffenen Datensätze noch einmal ansehen. Dazu wechseln Sie einfach auf die gewohnte Weise in die Datenblattansicht der Löschabfrage.

Nach der Sichtprüfung führen Sie die Löschabfrage aus, und zwar auf eine der folgenden Methoden:

- Doppelklick auf den Eintrag **qryDeleteInaktiveKunden** im Datenbankfenster beziehungsweise Navigationsbereich
- Öffnen der Abfrage in der Entwurfsansicht und anschließendes Betätigen des Ribbon-Eintrags **Entwurf|Ergebnisse|Ausführen** (Access 2007 und jünger) beziehungsweise den Menüeintrag **Abfrage|Ausführen** (Access 2003 und älter).

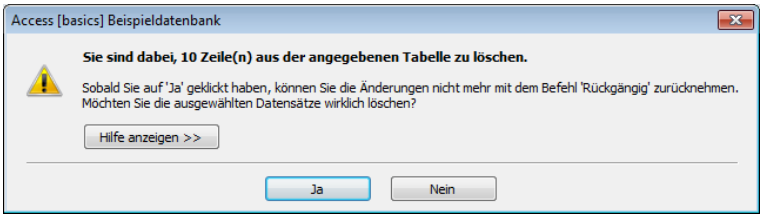



**Bild 3:** Diese Meldung kündigt den anstehenden Löschvorgang an.

Access zeigt nun eine Meldung an, die auf das anstehende Löschen der Datensätze hinweist und letztmalig die Möglichkeit bietet, das Löschen abzubrechen (s. Bild 3). Ein Klick auf **OK**, und die inaktiven Kunden verschwinden im Daten-Nirvana.

## Löschen und die Folgen

Gut gebrüllt, Löwe! Aber leider völlig falsch gedacht! Erstens aus betrieblicher Sicht, denn inaktive Kunden sind keine Störenfriede, sondern potentielle Auftraggeber. Zweitens besteht die Datenbank nicht nur aus der Kundentabelle, sondern aus weiteren Tabellen, von denen einige mit der Kundentabelle verknüpft sind. Und auch mit inaktiven Kunden können beispielsweise Projektdatensätze verknüpft sein, an denen wiederum Aufgaben und Tätigkeiten hängen.

Wer Kunden löscht, sollte dementsprechend auch die Projekte und die damit verknüpften Daten löschen, weil die Datenbank anderenfalls Daten enthält, die nicht mehr eindeutig zugeordnet werden können.

Da wir aber für die Beziehung etwa zwischen den Tabellen tblProjekte und tblKunden referentielle Integrität definiert haben (nachzulesen unter **Tabellen entwerfen - Teil II: Kunden und Projekte verwalten**), können Sie nicht einfach einen Kunden löschen und die mit diesem verknüpften Projektedatensätze in der

# 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

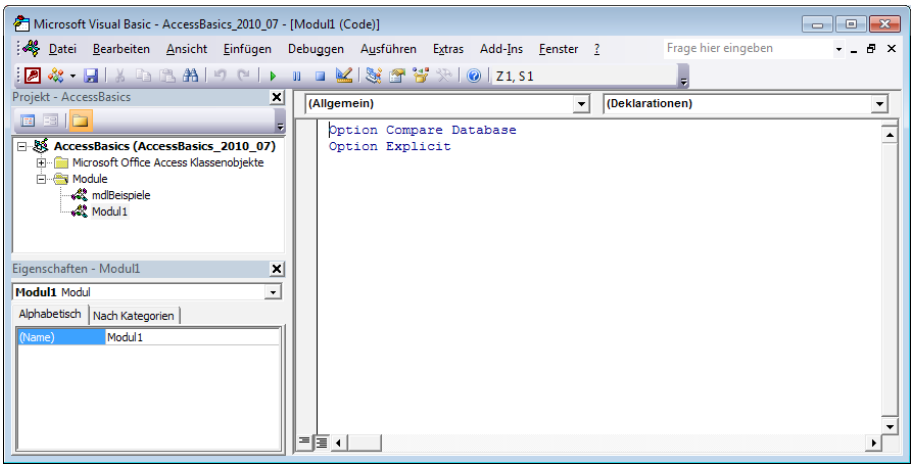



**Bild 1:** Der VBA-Editor enthält ein Fenster für den Projekt-Explorer oben links, die Eigenschaften unten links und als größten Bereich das Code-Fenster rechts.

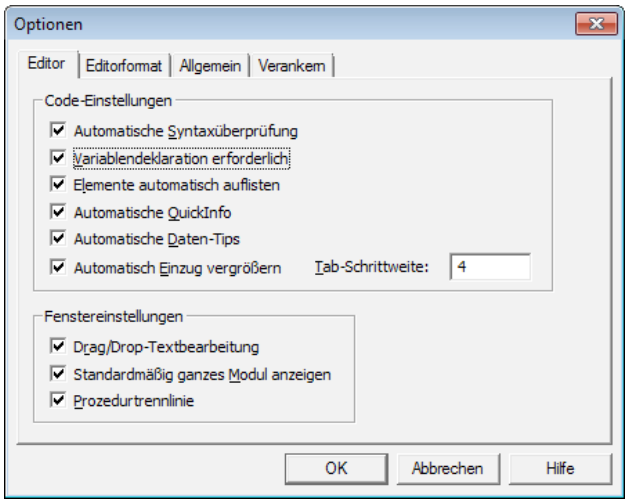



**Bild 2:** Aktivieren Sie unbedingt die Option **Variablendeklaration erforderlich**.

- Der wesentliche Teil ist aber das Code-Fenster rechts, in diesem Bereich stehen die Programme. Es wird beispielsweise durch Doppelklick auf einen Modulnamen im Projekt-Explorer geöffnet.

Das Code-Fenster fehlt bei Ihnen? Vielleicht haben Sie auch noch gar kein Modul eingefügt? Dann sollten wir das schleunigst nachholen! Mit dem Menü Einfügen / Modul legt Access ein solches neues Modul an und öffnet das zugehörige Code-Fenster direkt. In Access-Modulen steht dort schon der Eintrag **Option Compare Database**, was bedeutet, dass ein Vergleich von Zeichenketten in VBA sich an die Spracheinstellung und Sortierung in der Datenbank hält. Das ist jetzt nicht von Interesse, kann aber dort stehen bleiben. Ein anderer Eintrag fehlt dort allerdings, an dem uns viel mehr läge, vor allem zu Beginn der Programmierung: **Option Explicit**. Das würde sicherstellen, dass Sie demnächst Variablen immer vorher ankündigen müssen und nicht (wie BASIC das für Anfänger erlaubt) einfach so nutzen können. Was wie eine Strafverschärfung klingt, erspart Ihnen viele, viele überflüssige Fehler, die Sie sonst machen würden. Bevor es also richtig losgeht, rufen Sie einmalig den Menübefehl **Extras|Optionen** auf und haken wie in Bild 2 auf der Registerkarte **Editor** das fehlende Kontrollkästchen **Variablendeklaration erforderlich** an (warum dies nicht standardmäßig aktiviert ist, weiß übrigens nur Microsoft). Ab jetzt steht in jedem neuen Modul die Zeile **Option Explicit** – aber nur in neuen Modulen. Für dieses schon vorhandene Modul schreiben Sie es bitte oben in die zweite Zeile „zu Fuß" hinein.

In Standardmodulen gibt es – anders als bei den Formularen mit ihren Klassenmodulen – keine vorgegebenen Prozedurnamen wie etwa **cmdMeldung_Click** oder **Form_Unload**. Sie können sich möglichst sprechende Namen weitgehend selber ausdenken, wenn Sie sich an die folgenden Regeln halten:

- Es darf kein VBA-interner Befehl sein, also beispielsweise nicht **Sub**, **If**, **Exit** oder **CurrentDb**.
- Er muss mit einem Buchstaben beginnen und darf nur Buchstaben, Zahlen oder Unterstriche enthalten.
- Die maximale Länge beträgt 255 Zeichen.
- Innerhalb eines Moduls muss der Name eindeutig sein.

Wir würden diese zwingenden Regeln noch um ein paar Empfehlungen erweitern, die Ihnen das Leben leichter machen:

- Nutzen Sie die so genannte Kamel-Schreibweise (nämlich mit Höckern aus Großbuchstaben), also **LiesWert** statt **lies_wert**.
- Vermeiden Sie in Prozedurnamen unbedingt Umlaute und das scharfe S, das gibt immer mal wieder Ärger auf nicht-deutschsprachigen Computern.

# 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.

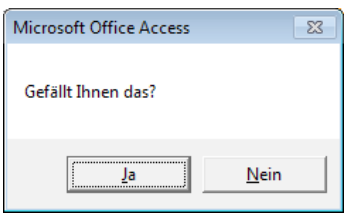



**Bild 1:** So sieht diese Meldung mit zwei Schaltflächen aus.

### 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

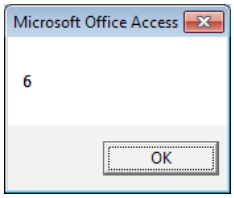



**Bild 2:** Beim Anklicken der **Ja**-Schaltfläche liefert die folgende **MsgBox** den Wert **6**.

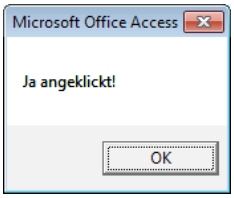



**Bild 3:** Diese Meldung erscheint nur, wenn vorher **Ja** angeklickt wurde.

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.

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.

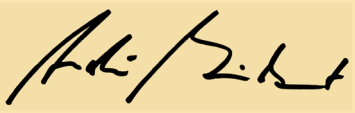
Doch zunächst zu den Abfragen: Zu diesem Thema finden Sie gleich drei Artikel. Im ersten zeigen wir Ihnen, wie Sie die Datensätze der Abfrage nach dem Inhalt von Textfeldern filten können. Es geht allerdings nicht nur um den einfachern Vergleich mit Zeichenketten, denn wir nehmen auch Spezialitäten wie 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**

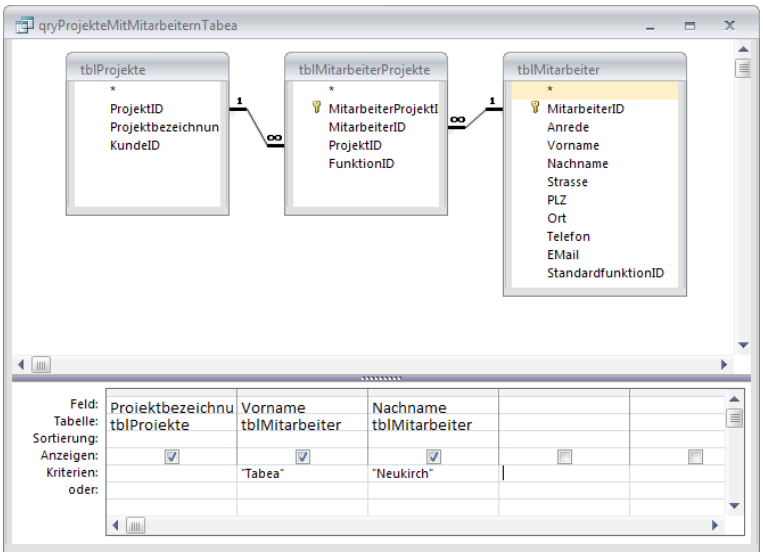



**Bild 1:** Der Entwurf der Abfrage, um die Projekte bestimmter Mitarbeiter zu finden

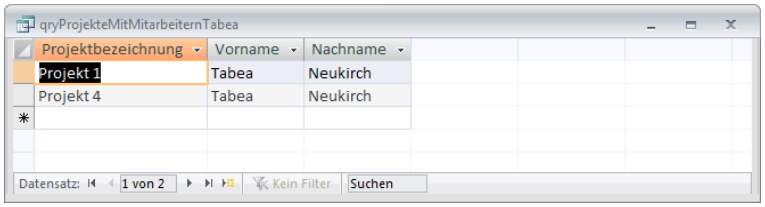



**Bild 2:** Das Abfrage-Ergebnis für Tabea Neukirch

## 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**!

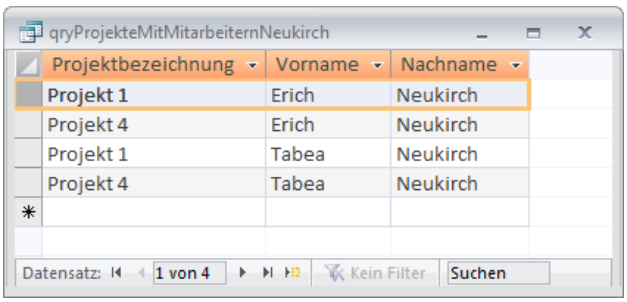



**Bild 3:** Das Abfrage-Ergebnis nur für Neukirch

# 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.

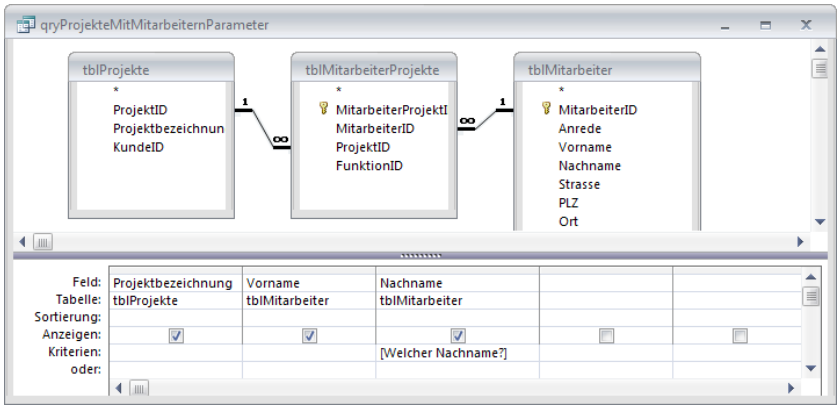



**Bild 1:** So geben Sie einen Parameter an.

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-**

**scher“ Und „Schneider“** stand, fügen Sie nun einen sogenannten Parameter ein. Der Parameter muss in eckigen Klammern stehen und darf nicht den Namen eines der in den zugrunde liegenden Tabellen enthalten. Wenn Sie nun beispielsweise

```
[Welcher Nachname?]
```

als Kriterium eingeben, sind Sie auf der sicheren Seite (s. Bild 1): So heißt garantiert keines der Felder, zumindest, wenn Sie sich sich an die gängigen Konventionen für die Benennung von Feldern halten (siehe →*[basics]* **Konventionen**).

Warum aber formulieren wir den Namen des Parameters als Frage? Nun: Access hat einen eingebauten Mechanismus, der alle Elemente einer Abfrage, die nicht als Feldname oder Schlüsselwort erkannt werden, entdeckt und den Benutzer fragt: „Diesen Wert kenne ich nicht! Sag mir, welchen Wert ich dafür annehmen soll!“

Sobald ein Benutzer diese Abfrage ausführt, erscheint ein kleiner Dialog mit dem Feldnamen darüber und einem Eingabefeld, damit der Benutzer den Inhalt des Kriteriums angeben kann. Geben Sie nun **Fischer** ein und bestätigen den Dialog (s. Bild 2), so sehen Sie ein ganz normales Abfrage-Ergebnis in der Datenblattansicht (s. Bild 3).

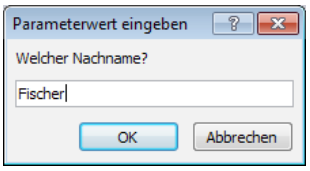



**Bild 2:** Eingabe des Parameters als Kriterium der Abfrage

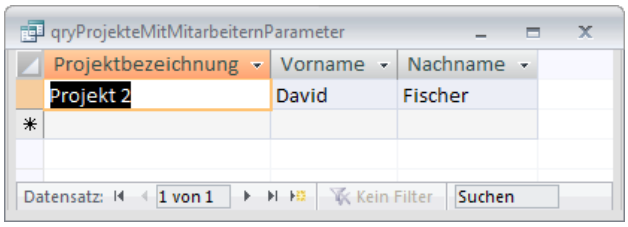



**Bild 3:** Die Parameter-Abfrage hat funktioniert und zeigt die gefilterten Datensätze für Fischer an.

Sobald Sie wieder in den Entwurf der Abfrage zurückwechseln, ist der Parameter-Inhalt vergessen. Sie müssen ihn daher bei jeder Ausführung erneut eingeben. Damit haben wir uns erst einmal ganz viele ähnliche Abfragen erspart.

Eine einzige Parameter-Abfrage ermöglicht die wechselnde Eingabe von Kriterien zur Ausführungszeit. Der Chef muss also nicht mehr in den Abfrage-Entwürfen herumbasteln, sondern braucht diese einfach nur noch zu öffnen.

# 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

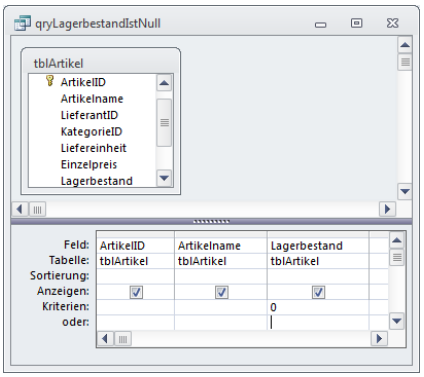



**Bild 1:** Abfrage aller Datensätze, deren Lagerbestand den Wert 0 hat

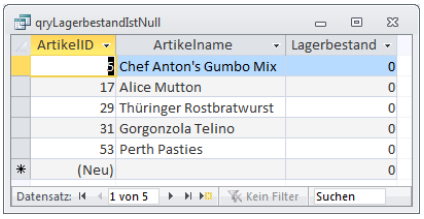



**Bild 2:** Ergebnis der Abfrage aller ausverkauften Datensätze

### 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.

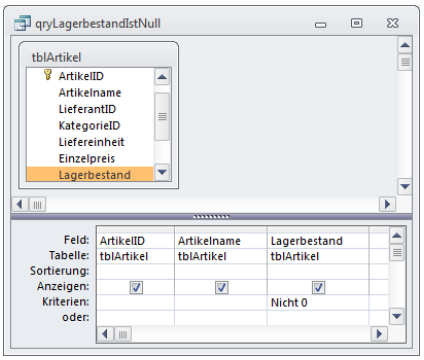



**Bild 3:** Schlüsselwörter wie **NOT** werden im Abfrageentwurf der deutschen Access-Version eingedeutscht.

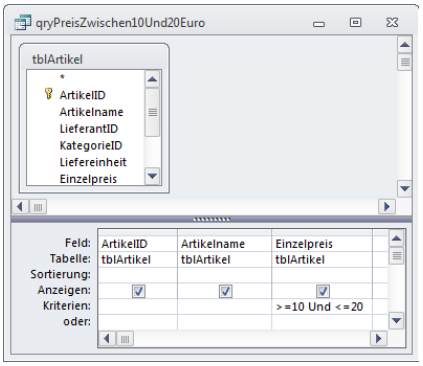



**Bild 4:** Abfragen eines Bereiches von Werten

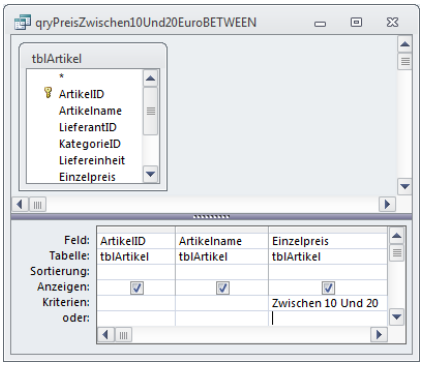



**Bild 5:** Bereichsabfrage, diesmal mit BETWEEN

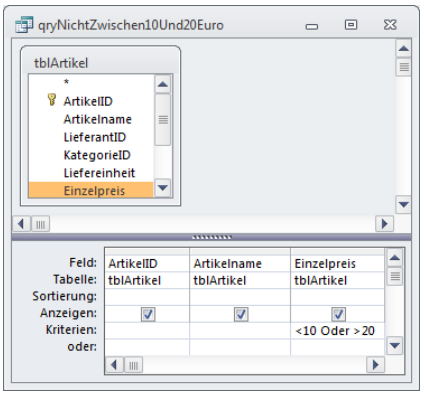



**Bild 6:** Artikel, die weniger als 10,- Euro oder mehr als 20,- Euro kosten

## 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
```

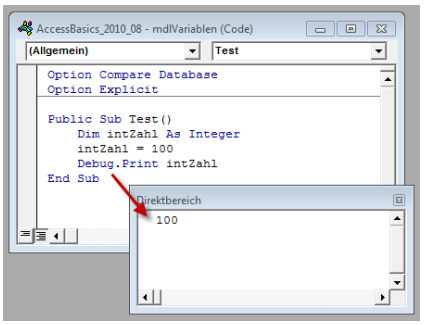



**Bild 1:** Ausgabe des in einer Variablen gespeicherten Wertes im Direktfenster

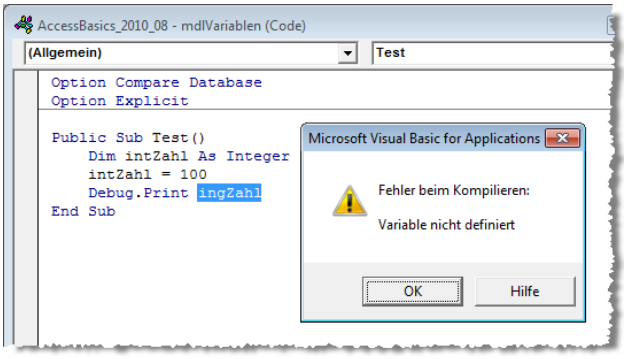



**Bild 2:** Fehlermeldung beim Verwenden einer nicht deklarierten Variablen

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.

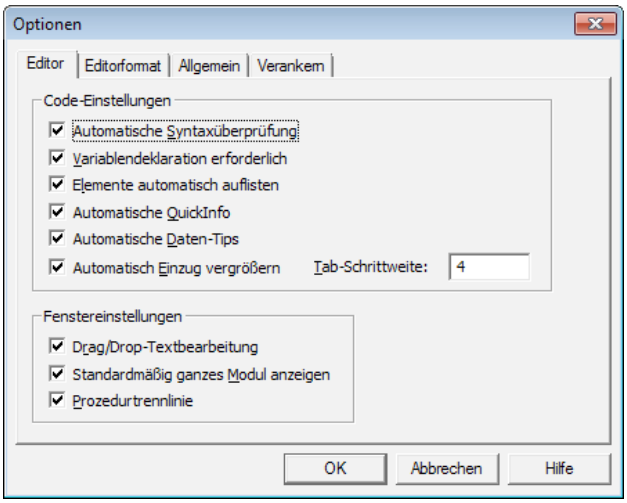



**Bild 3:** Aktivieren des Kontrollkästchens **Variablendeklaration erforderlich**

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?

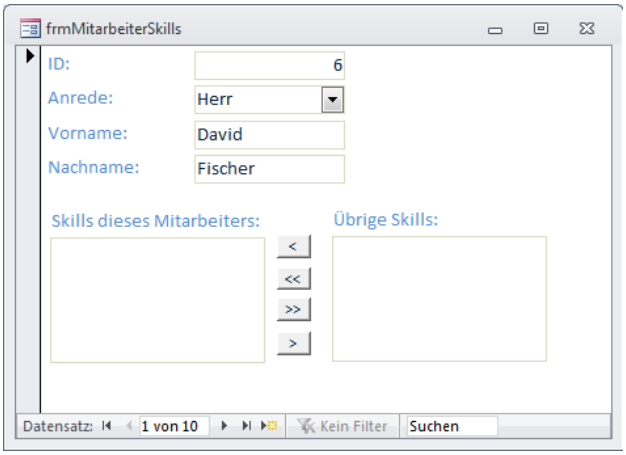



**Bild 1:** Hinzufügen und Entfernen der Skills eines Mitarbeiters

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.

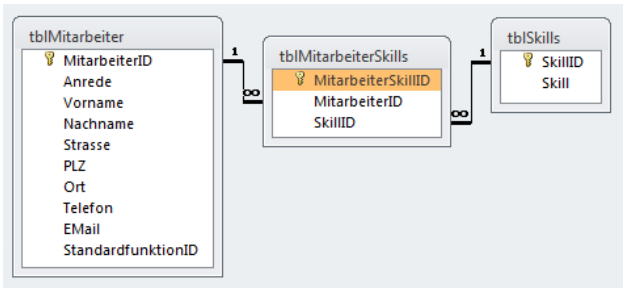



**Bild 2:** Mitarbeiter und ihre Skills im Datenmodell

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-

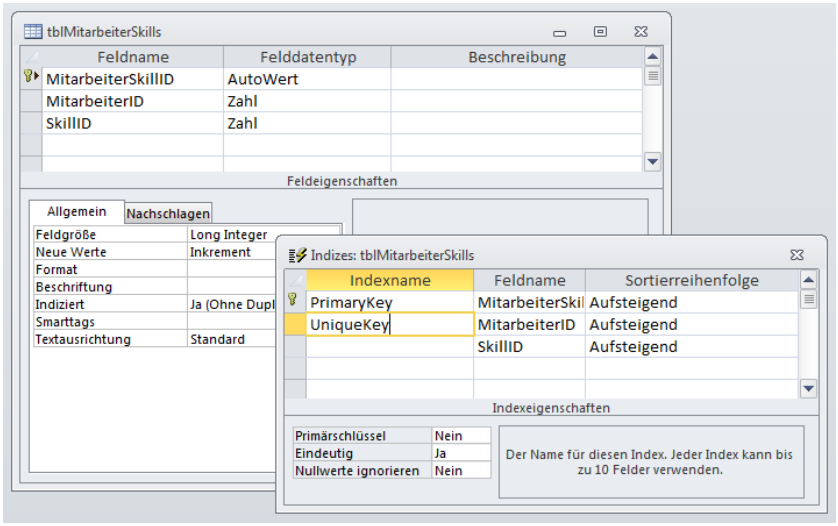



**Bild 3:** Festlegen eines zusammengesetzten eindeutigen Schlüssels für die Tabelle tblMitarbeiterSkills

### 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).

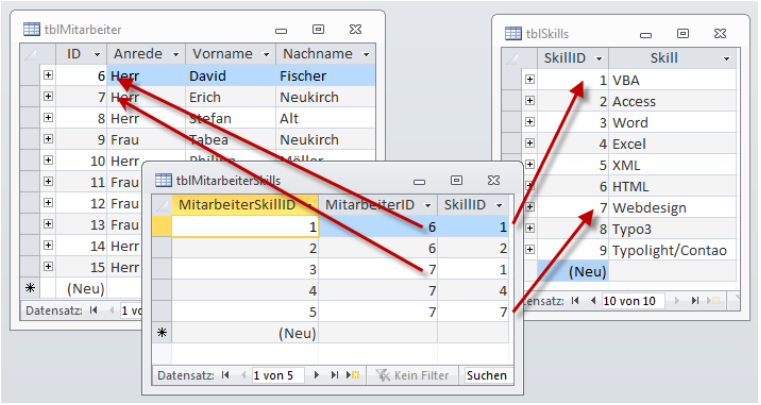



**Bild 4:** Zuordnen von Skills zu Mitarbeitern per Verknüpfungstabelle

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**.

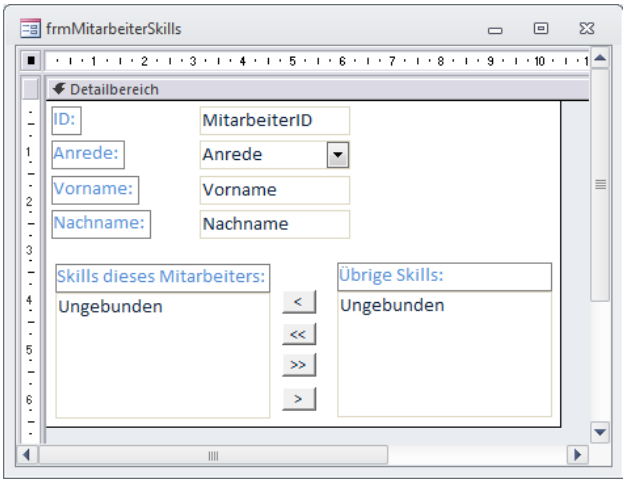



**Bild 5:** Entwurfsansicht des Formulars mit allen Steuerelementen

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

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

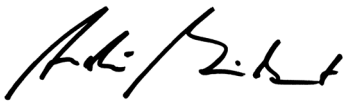
## Access [basics]-News

Mit dieser etwas verspäteten Dezember-Ausgabe gibt es wieder ein paar Neuerungen bei **Access [basics]**. Die erste sehen Sie gleich hier: Die erste Seite des Magazins startet nicht mehr mit dem ersten Artikel, sondern liefert alle Informationen, die Sie vor dem Lesen benötigen. Zum Beispiel den Text, den Sie gerade lesen und der Ihnen mitteilt, was es Neues in der Welt von **Access [basics]** gibt. Außerdem liefern wir Ihnen hier eine erweiterte Inhaltsangabe (siehe rechts) und das Impressum, das bisher viel weiter hinten Platz gefunden hat. Warum diese Umbau-Aktion? Ganz einfach: Weil Sie jeden einzelnen Artikel für sich lesen und ausdrucken können sollen, ohne dass es in jeder Ausgabe einen Artikel gibt, der durch den Inhalt der Startseite verunstaltet wird.

Die zweite Neuerung ist ein überarbeitetes Inhaltsverzeichnis unseres **Access [basics] A-Z**. Unser Leser Bernd Koch (vielen Dank!) wies darauf hin, dass die verschiedenen Informationen zum aktuellen Stand, zum Zeitpunkt des Hinzufügens und der letzten Aktualisierung doch eher verwirren würden. Also haben wir die einzelnen A-Z-Artikel von sämtlichen Daten befreit und stattdessen das Inhaltsverzeichnis um zwei Informationen erweitert: Wann wurde der Beitrag hinzugefügt und wann wurde er zuletzt geändert? Die Aktualisierungen der aktuellen Ausgabe heben wir außerdem farbig hervor.

Und schließlich haben wir der Kopfzeile der einzelnen A-Z-Artikel noch einen Link hinzugefügt, mit dem Sie schnell wieder zum Inhaltsverzeichnis gelangen.

Nun aber viel Spaß beim Lesen!



André Minhorst

## In dieser Ausgabe:

**Tabellen entwerfen – Teil IV: Felder und ihre Eigenschaften** | Lernen Sie die Eigenschaften von Tabellenfeldern kennen. Inklusive Datentypen, Feldgrößen und mehr!

**Abfragen für die Datenauswahl – Teil VI: Daten zusammenführen mit UNION-Abfragen** | Fragen Sie ähnliche Daten aus verschiedenen Tabellen in einem Rutsch ab!

**Steuerelemente – Teil III: Das Textfeld** | Lernen Sie das Textfeld und seine grundlegenden Eigenschaften wie etwa die Datenbindung kennen.

**Steuerelemente – Teil IV: Das Kombinationsfeld** | Wählen Sie gebundene und ungebundene Daten mit dem Kombinationsfeld aus!

**Datensatzauswahl per Kombinationsfeld – Kunden per Kombinationsfeld auswählen und im Formular anzeigen**
Die Lösung zum schnellen Auswählen des gewünschten Datensatzes!

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst | Fachverlag für Softwareentwicklung | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autoren: Lorenz Hölscher, André Minhorst

**ISSN: 2190-8761**

# 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

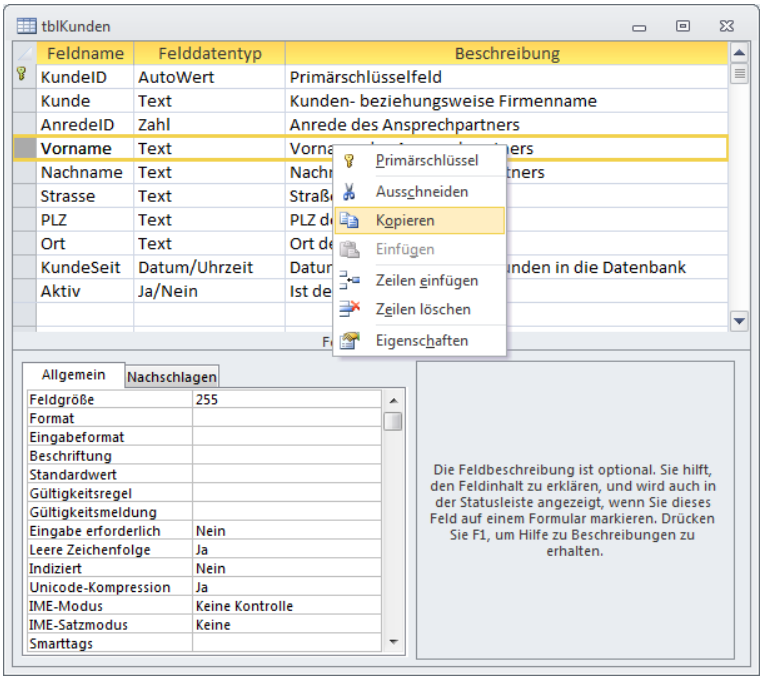



**Bild 1:** Bearbeiten einer Tabelle in der Entwurfsansicht

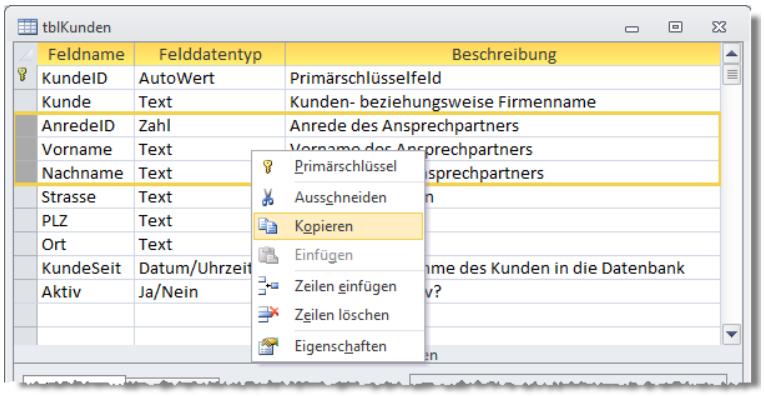



**Bild 2:** Markieren mehrerer zusammenhängender Zeilen

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**

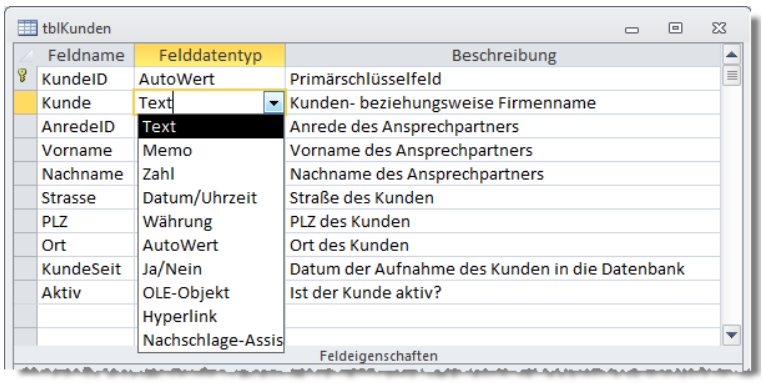



**Bild 3:** Auswahl des Datentyps

# 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.

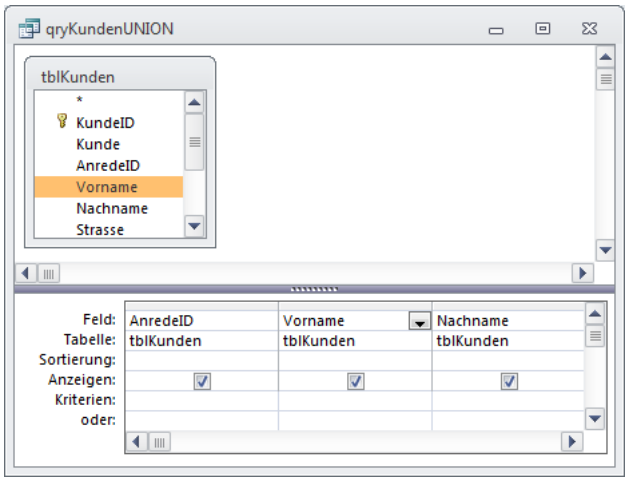



**Bild 1:** Der Kunden-Teil der UNION-Abfrage

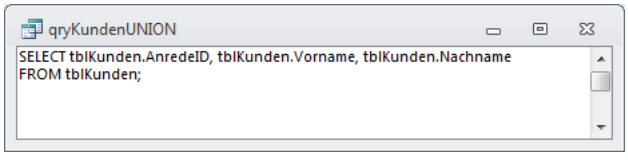



**Bild 2:** Die Abfrage in der SQL-Ansicht

## 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!

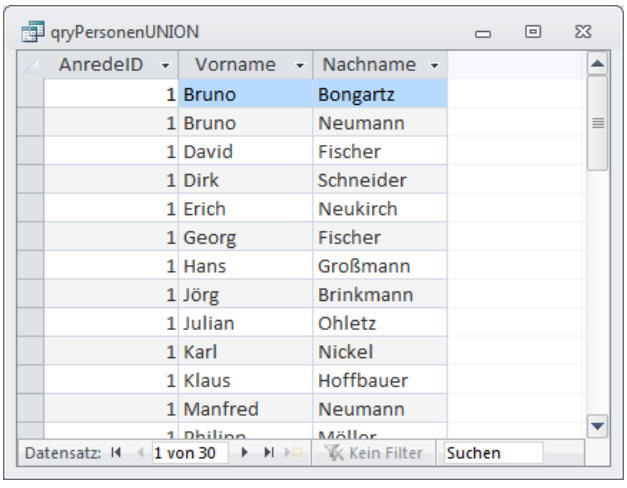



**Bild 3:** Das Ergebnis der UNION-Abfrage mit allen Mitarbeitern und Kunden

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.

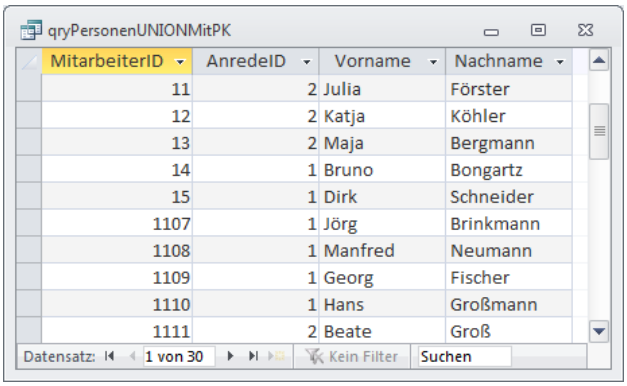
Die wichtigste Grundregel beim Erstellen von **UNION**-Abfragen ist die folgende: Alle **SELECT**-Anweisungen, die durch das **UNION**-Schlüsselwort verkettet werden, müssen genau die gleiche Anzahl Felder enthalten und die gleichen Namen enthalten.

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 |



**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-

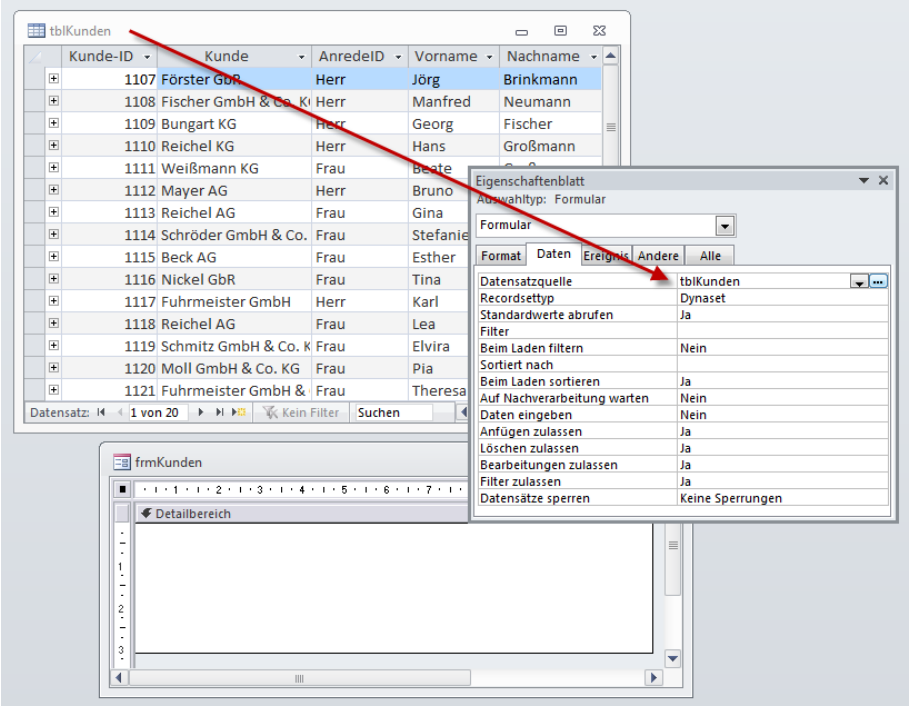



**Bild 1:** Zuweisen der Datenherkunft eines Formulars

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.

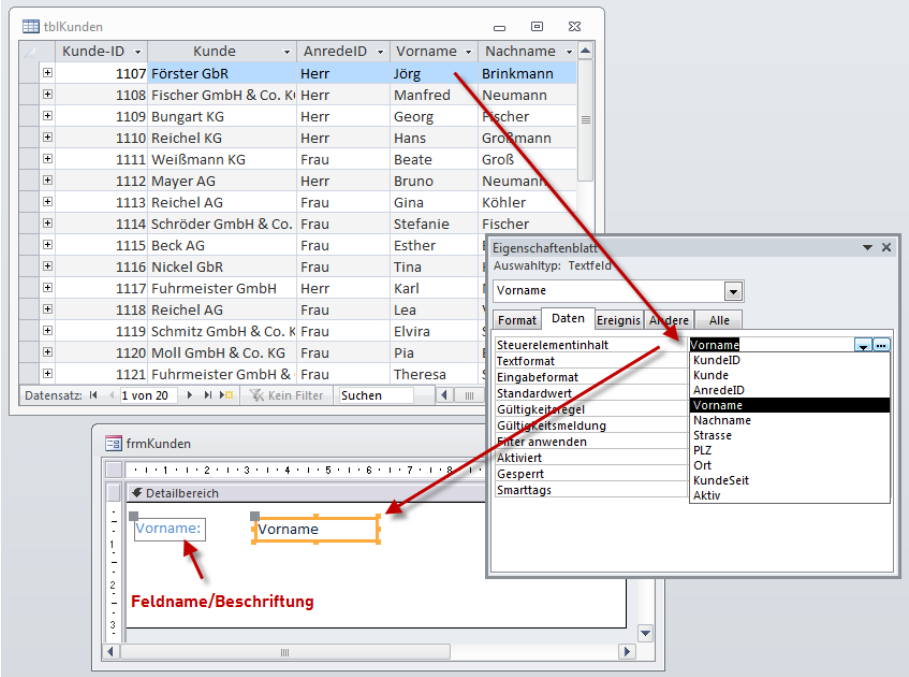



**Bild 2:** Binden eines Textfelds an ein Feld der Datenherkunft

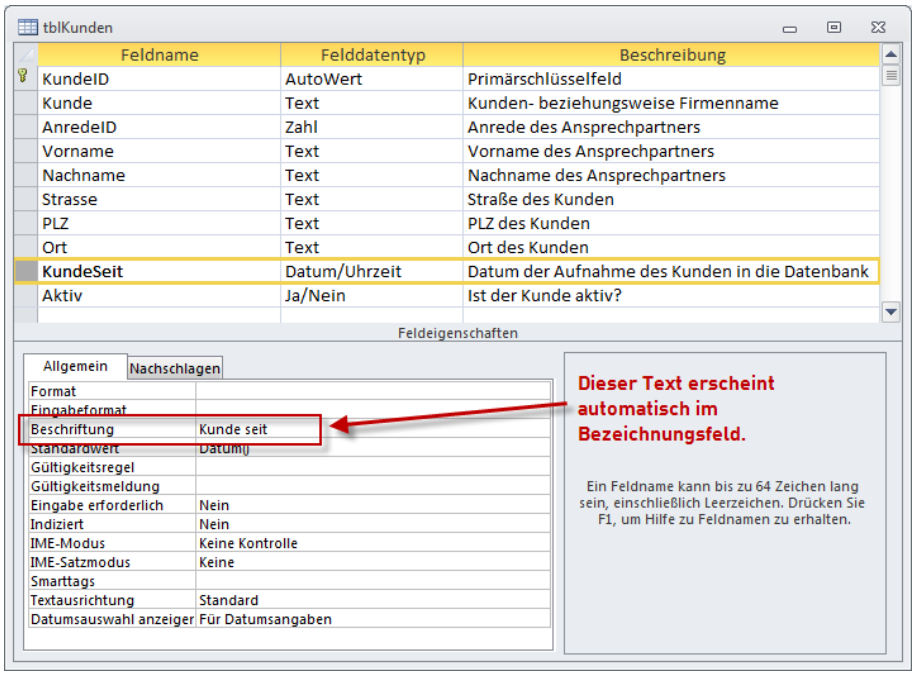



**Bild 3:** Diese Beschriftung wird in das Bezeichnungsfeld eines Textfelds auf Basis dieses Feldes übernommen.

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.

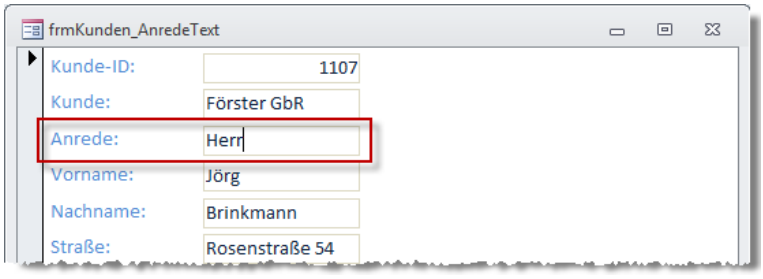



**Bild 2:** In das Formular übernommen erscheint das Feld Anrede zunächst als Textfeld.

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**.

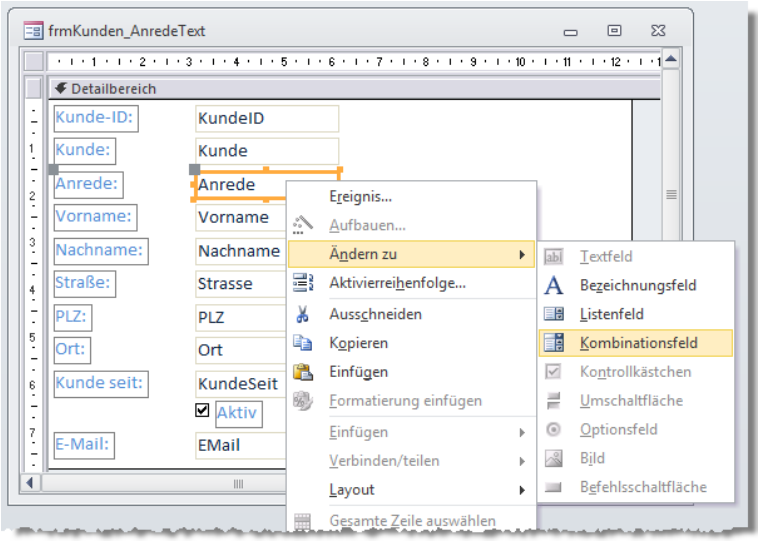



**Bild 3:** Umwandeln eines Textfelds in ein Kombinationsfeld

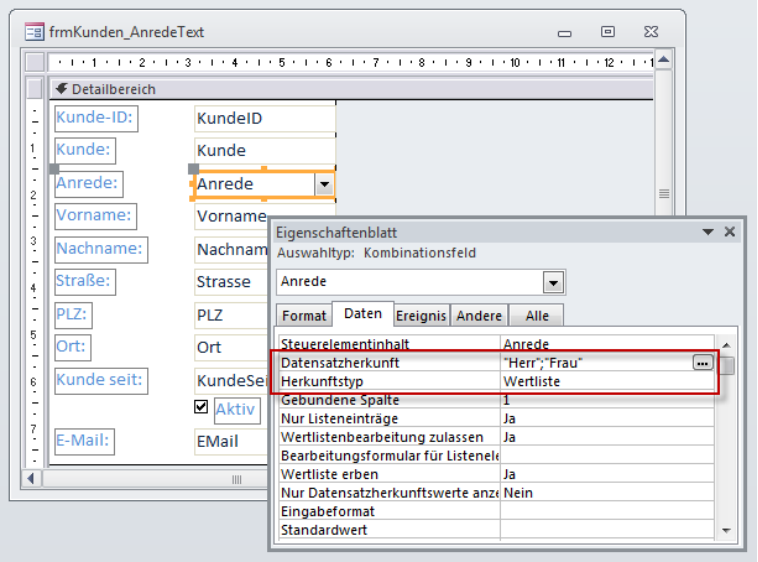



**Bild 4:** Einstellen der anzuzeigenden Einträge

## 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

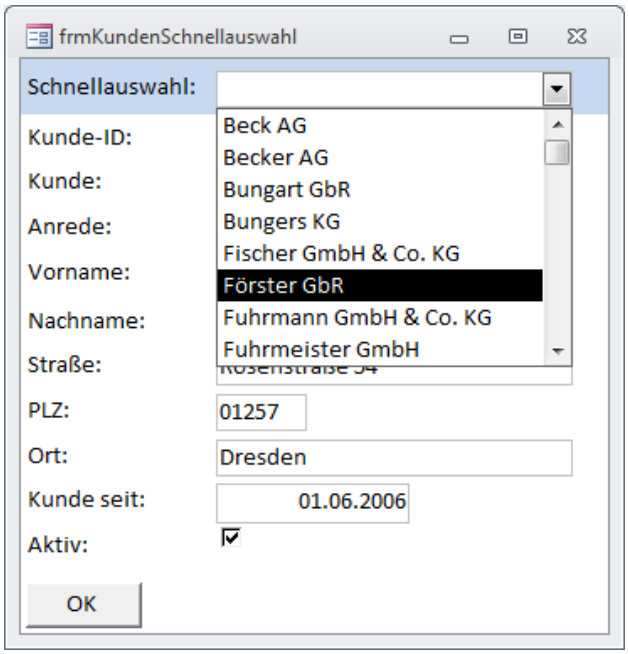



**Bild 1:** Das Suchkombinationsfeld erlaubt das schnelle Ansteuern von Datensätzen.

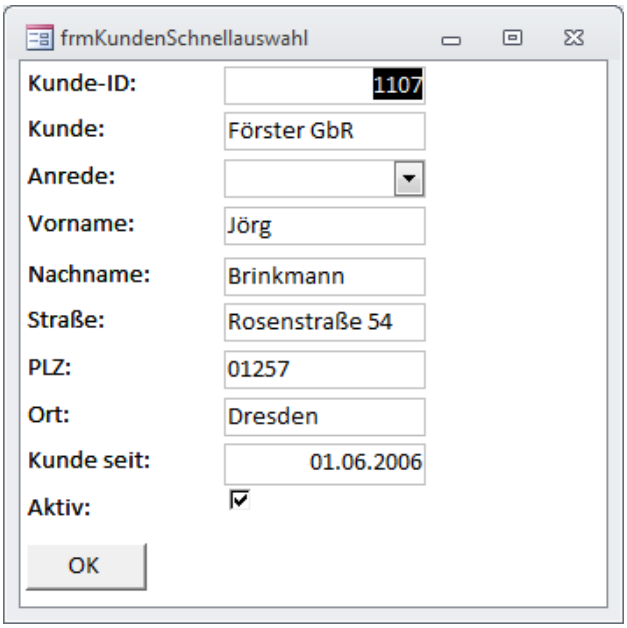



**Bild 2:** Der Ausgangspunkt: Eine Kopie des Formulars **frmKundenDetail** der Beispieldatenank

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.

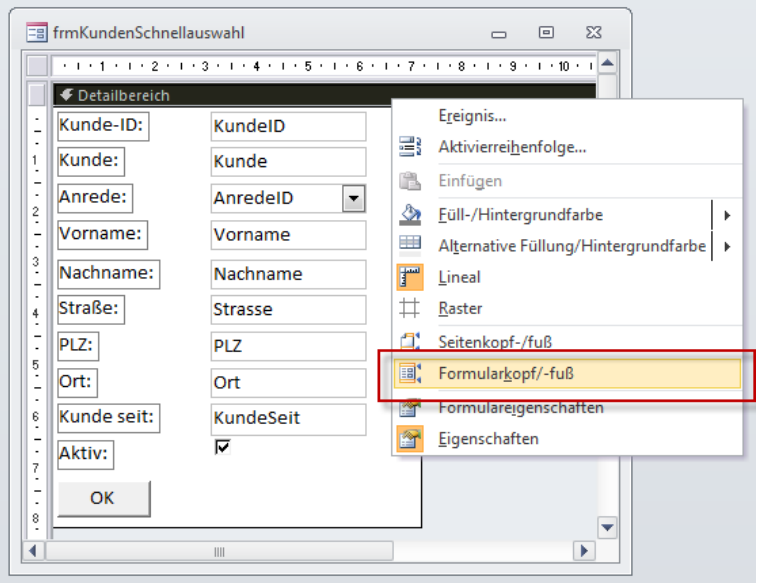



**Bild 3:** Aktivieren von Formularkopf und -fuß

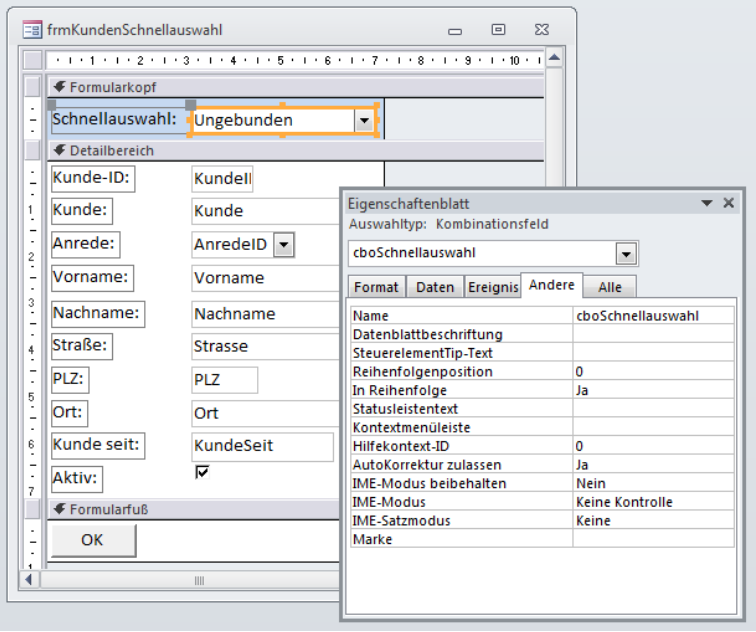



**Bild 4:** Hinzufügen eines Kombinationsfeldes zur Schnellauswahl

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 (...).

Access [basics] 1/2011
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Access [basics]-News

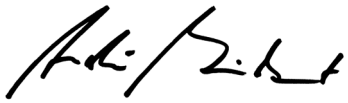
Geschafft! Pünktlich auf die Minute werden Sie um kurz vor 0:00 Uhr, also noch am 31. Januar 2011, die Mail mit dem Link zu dieser neuen Ausgabe von **Access [basics]** im Postfach vorfinden. Da die letzte Ausgabe des Jahres 2010 erst Anfang Januar 2011 erschien, hatte ich ja versprochen, dass die erste Ausgabe des neuen Jahres ebenfalls noch im Januar erscheint. Und das war gar nicht einfach: Immerhin steht im Moment noch das neue Access 2010 - Das Grundlagenbuch für Entwickler auf dem Programm, und auch der eine oder andere Kunde hat Anfang des Jahres Wünsche für seine Softwareprojekte.

In dieser Ausgabe gibt es noch einmal einen Rutsch Grundlagen zu Tabellen und Abfragen, außerdem stelle ich Ihnen mit den Domänenfunktionen eine sehr einfache Möglichkeit vor, auf die Daten innerhalb Ihrer Tabellen zuzugreifen.

Die nächste Ausgabe wird dann richtig spannend: Wir setzen die Arbeiten an der Projektzeiterfassung fort und werden die Formulare zum Eingeben von Aufgaben und Tätigkeiten der einzelnen Projekte erstellen. Mit den Grundlagen der letzten Ausgaben werden wir eine Menge schaffen.

Übrigens: Wenn Sie einmal Themenwünsche haben, dürfen Sie mir diese gern mitteilen, und zwar an die Adresse **info@access-basics.de**. Ich freue mich natürlich auch über Lob und Kritik – also nur los!

Nun aber viel Spaß beim Lesen!



André Minhorst

## In dieser Ausgabe:

**Tabellen entwerfen – Teil V: Eigenschaften von Tabellen** | Lernen Sie die Eigenschaften von Tabellen wie Gültigkeitsregeln, Filter, Sortierungen und Co. kennen.

**Der Ausdrucks-Generator – Ausdrücke schnell zusammenstellen mit einem praktischen Tool** | Nutzen Sie den Ausrucks-Generator, um schnell Ausdrücke für Feldwerte, Kriterien und Eigenschaften zusammenzustellen

**Abfragen für die Datenauswahl – Teil VI: Berechnete Felder in Abfragen und Eigenschaften** | Verwenden Sie Abfragen, um Texte mehrerer Felder zu verknüpfen oder Berechnungsergebnisse anzuzeigen

**Abfragen für die Datenauswahl – Teil VII: Benutzerdefinierte Funktionen in Abfragen** | Wählen Sie gebundene und ungebundene Daten mit dem Kombinationsfeld aus!

**Access-Funktionen – Teil I: Domänenfunktionen** | Fragen Sie schnell mit einem einzigen Befehl Daten aus Tabellen oder Abfragen ab

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst | Fachverlag für Softwareentwicklung | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autoren: André Minhorst

**ISSN: 2190-8761**

# 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.

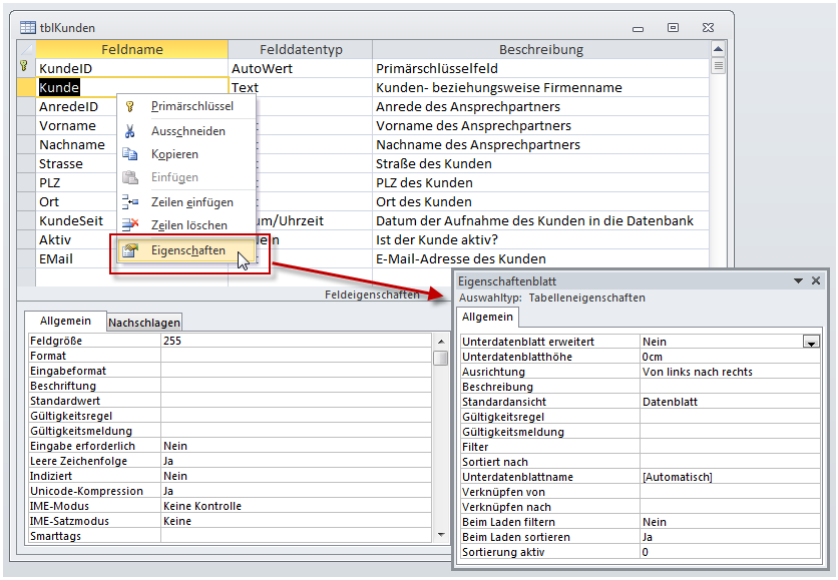



**Bild 1:** Anzeigen des Eigenschaftsfensters einer Access-Tabelle

## 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).

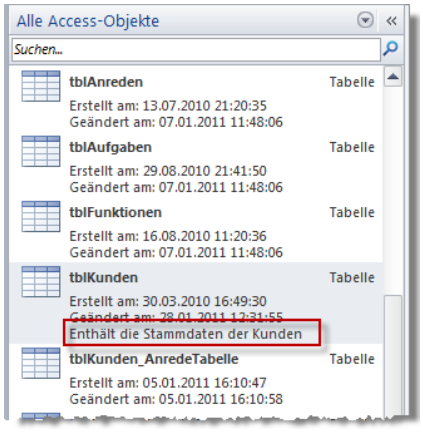



**Bild 2:** Beschreibung einer Tabelle im Navigationsbereich ...

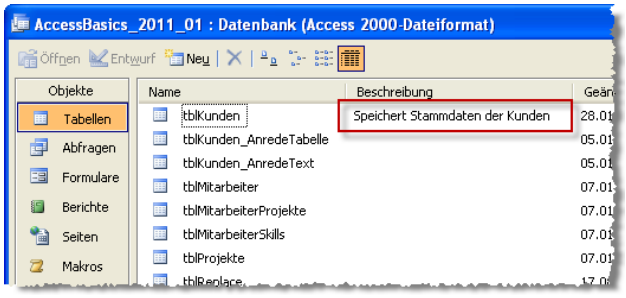



**Bild 3:** ... und im Datenbankfenster.

## 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

des Feldinhalts selbst verwenden. Stattdessen erweitern Sie den zu prüfenden Ausdruck einfach auf meherere Felder. Bild 4 zeigt ein Beispiel: Die Tabelle **tblProjekte** enthält zwei Felder namens **Projektstart** und **Projektende**. Das Enddatum soll logischerweise nicht vor dem Startdatum liegen. Wenn Sie nicht vorbeugen, wird es dem Benutzer in der Regel dennoch gelingen, ungültige Werte einzugeben. Also tragen Sie für die **Gültigkeitsregel** der Tabelle den folgenden Ausdruck ein:

```
[Projektstart]<[Projektende]
```

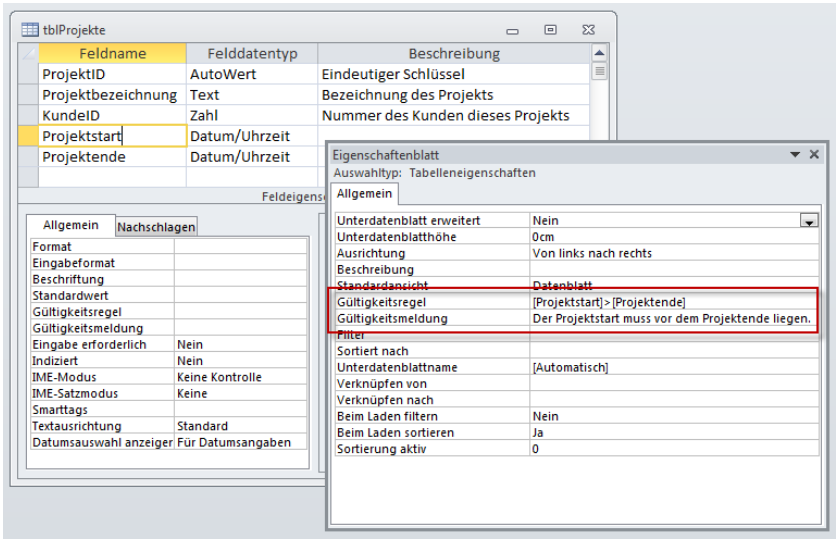



**Bild 4:** Beispiel für den Einsatz von Gültigkeitsregel und Gültigkeitsmeldung

Beachten Sie, dass die Gültigkeitsregel einen Ausdruck enthalten muss, der die gültigen Daten enthält! Die Eigenschaft **Gültigkeitsmeldung** erhält diesen Wert:

```
Der Projektstart muss vor dem Projektende liegen.
```

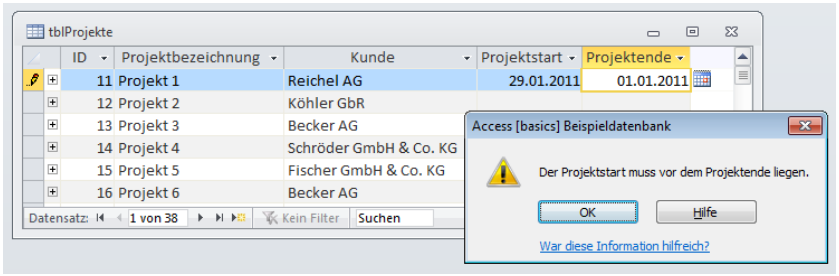



**Bild 5:** Anzeige der Gültigkeitsmeldung beim Verletzten einer Gültigkeitsregel

Wenn Sie nun eine ungültige Kombination von Werten eintragen, zeigt Access die Gültigkeitsmeldung an – allerdings erst, wenn Sie versuchen, den Datensatz zu speichern (siehe Bild 5). Im Gegensatz zu den beiden gleichnamigen Eigenschaften von Tabellenfeldern gibt es also zwei wesentliche Unterschiede:

- Auf Tabellenebene können Sie alle Felder der Tabelle in der Gültigkeitsregel referenzieren. Auf Feldebene gelingt dies nur für das jeweilige Feld.
- Auf Tabellenebene erfolgt die Gültigkeitsprüfung vor dem Speichern des Datensatzes, auf Feldebene vor dem Aktualisieren des Feldinhalts.

Man kann hier auch kompliziertere Ausdrücke zusammensetzen. Dabei ist einem der Ausdrucks-Generator behilflich. Diesen beschreiben wir in einem weiteren Artikel namens **Der Ausdrucks-Generator**.

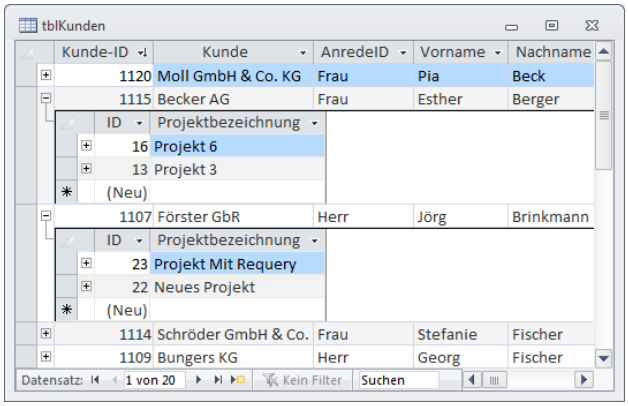



**Bild 6:** Tabelle mit Unterdatenblättern

## Filter und Sortiert nach (2000)

Mit der **Filter**-Eigenschaft stellen Sie einen Filter ein, zum Beispiel **KundeID = 1111**. Diesen aktivieren Sie in der Datenblattansicht unter Access 2003 und älter, indem Sie den Kontextmenüeintrag **Filter/Sortierung anwenden** der Titelzeile des Datenblatts auswählen (**Filter/Sortierung entfernen** zeigt wieder alle Daten an).

In neueren Access-Versionen, also Access 2007 und jünger, steht dieser Kontextmenübefehl nicht mehr zur Verfügung. Sie können den Filter stattdessen über einen entsprechenden Ribbon-Eintrag aktivieren.

# 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.

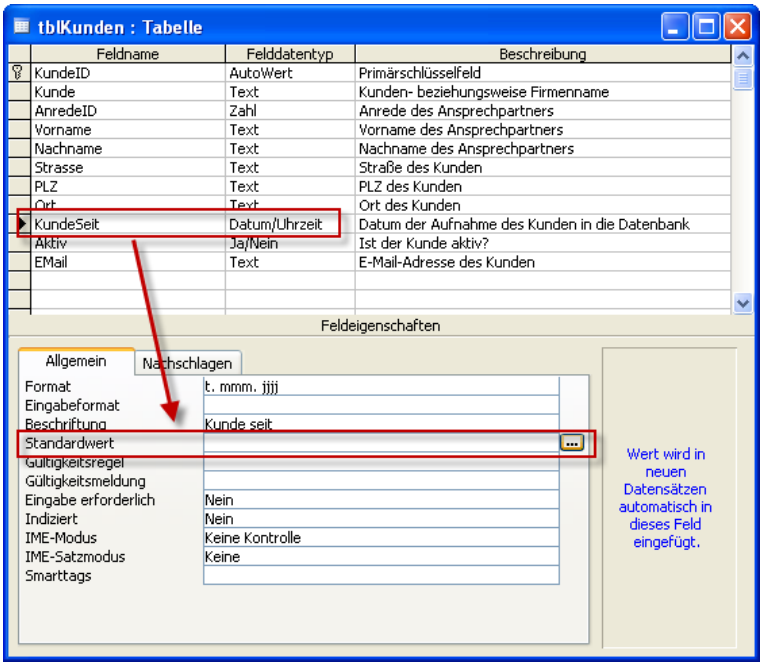



**Bild 1:** Eigenschaften eines Datumsfeldes

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).

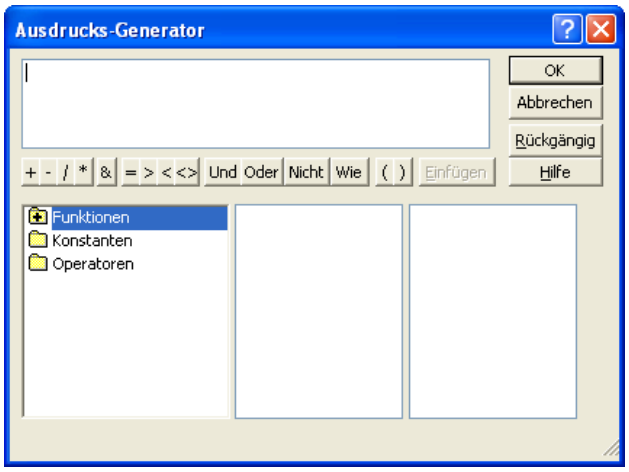



**Bild 2:** Der Ausdrucks-Generator unter Access 2007 und jünger

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).

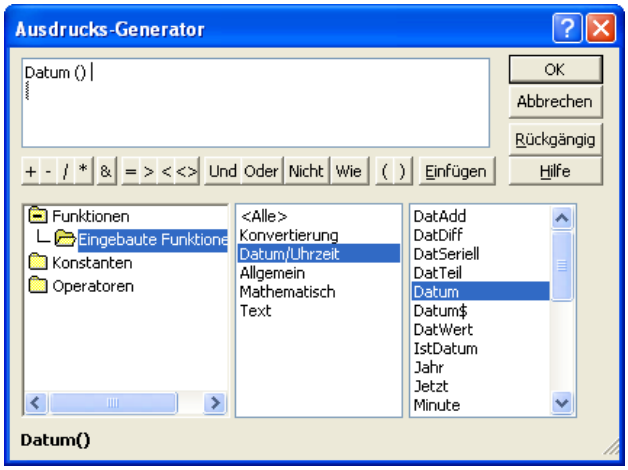



**Bild 3:** Die Funktion zum Einfügen des aktuellen Datums ist schnell gefunden.

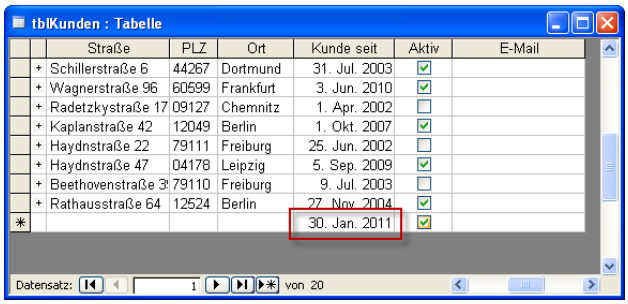



**Bild 4:** Automatisches Datum für neue Datensätze

## 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).

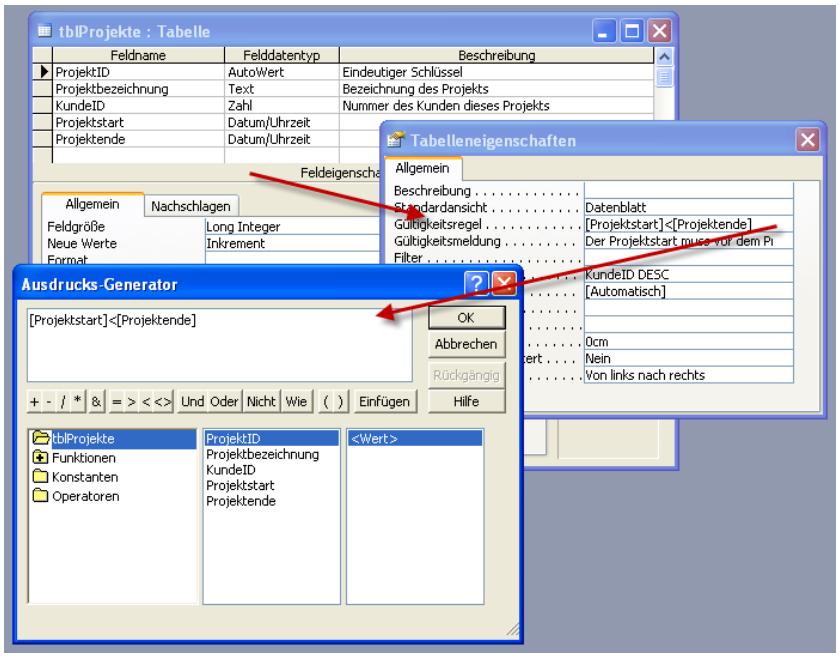



**Bild 5:** Erstellen einer Gültigkeitsregel mit dem Ausdrucks-Generator

# 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.

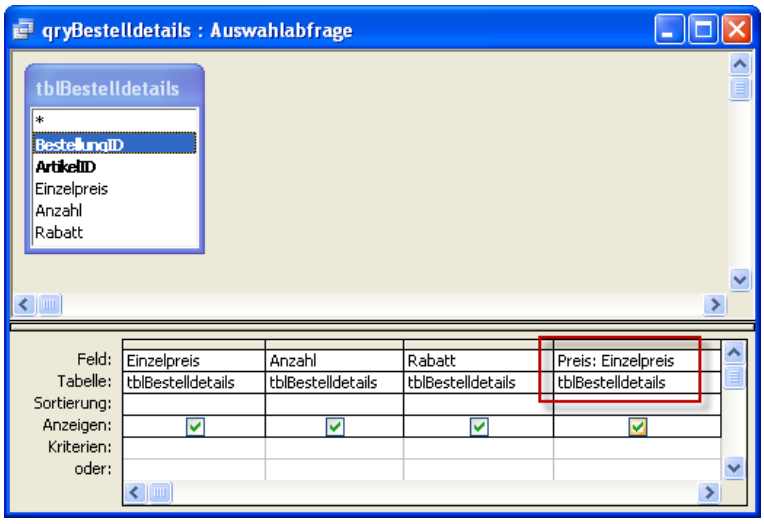



**Bild 1:** Ein berechnetes Feld, das einfach nur umbenannt wurde und den Wert eines anderen Feldes anzeigen soll

## 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).

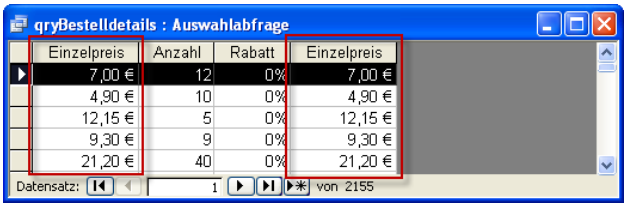



**Bild 2:** Das berechnete Feld und das Originalfeld sehen genau gleich aus.

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.

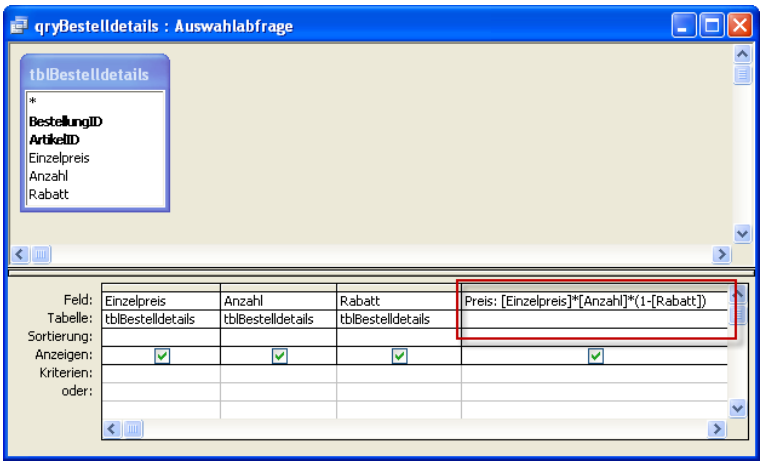



**Bild 3:** Besteht das berechnete Feld aus mehr als nur einem Feld der Datenherkunft der Abfrage, verschwindet die Anzeige der Herkunftstabelle.

## 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.

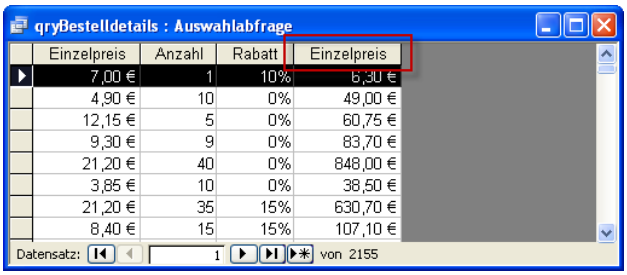



**Bild 4:** Außerdem zeigt der Spaltenkopf nun die Bezeichnung des berechneten Feldes an.

## 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.

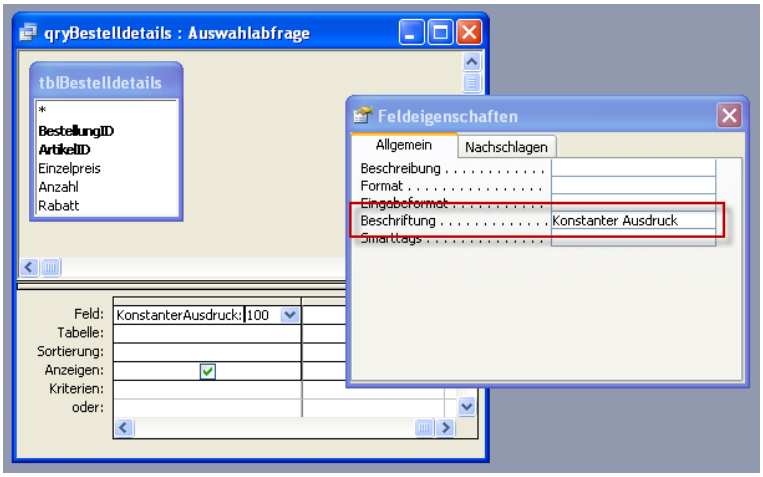



**Bild 5:** Anpassen des Spaltenkopfes eines berechneten Feldes

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.

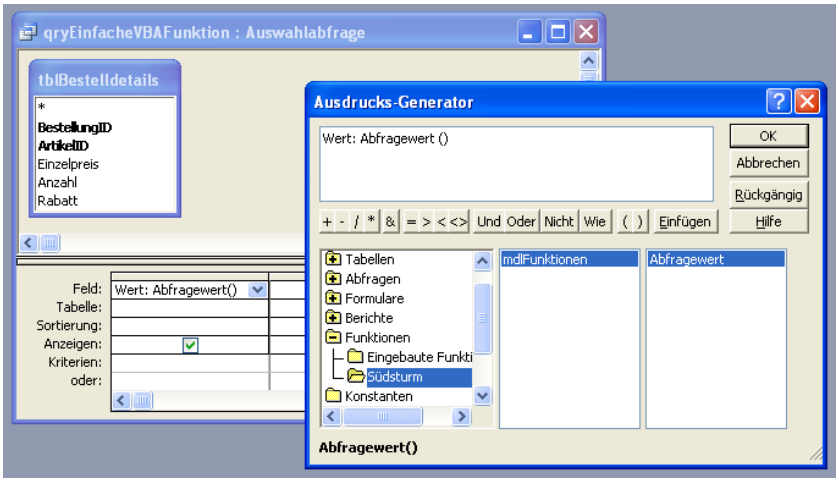



**Bild 1:** Hinzufügen eines berechneten Felds auf Basis einer benutzerdefinierten VBA-Funktion

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**.

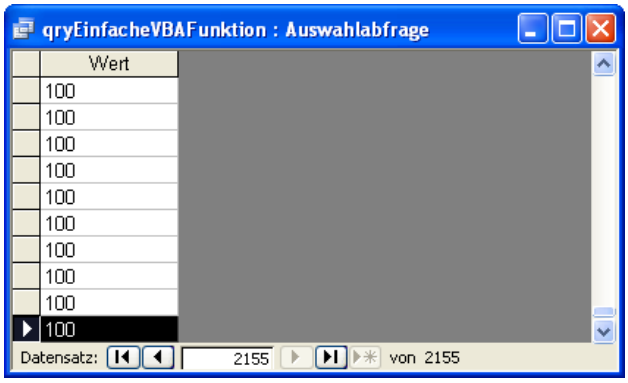



**Bild 2:** Das Ergebnis der Abfrage mit dem berechneten Feld

Wie Bild 2 zeigt, funktioniert die Abfrage mit der VBA-Funktion. Aber warum wird der Wert 100 so oft angezeigt und nicht nur einmal? Ganz einfach: Die Abfrage gibt immer soviele Datensätze aus, wie es die enthaltenen Tabellen vorgeben – egal, ob sie Felder der enthaltenen Tabellen anzeigt oder nicht.

Bevor wir uns dem nächsten Beispiel zuwenden, schauen wir uns etwas Interessantes an: Wenn die Abfrage einige hundert Datensätze enthält, wird die Abfrage dann eigentlich für jeden Datensatz einmal aufgerufen?

Um das herauszufinden, nutzen wir eine wichtige Funktion des VBA-Editors: den Haltepunkt. Öffnen Sie den VBA-Editor mit der soeben angelegten Funktion und klicken Sie einfach auf den grauen Rand links von der einzigen Anweisung der Funktion.

Wenn Sie nun die Abfrage **qryEinfacheVBAFunktion** in der Datenblattansicht öffnen, wechselt Access zum VBA-Editor und hinterlegt die betroffene VBA-Zeile gelb und hält die Ausführung des Codes an dieser Stelle an (siehe Bild 3).

Wenn Sie nun auf F5 klicken, sollte die Funktion bis zum Erreichen des nächsten Haltepunkts fortgesetzt werden. Die Funktion wird beendet, aber der Haltepunkt kein weiteres Mal angesprungen – das heißt, dass die Funktion nur ein einziges Mal aufgerufen wird! Anscheinend merkt Access, dass die Funktion einen statischen Wert zurückliefert.

## Berechnungen mit Parametern

Im zweiten Beispiel wollen wir einer Funktion die Werte der drei Felder **Einzelpreis**, **Anzahl** und **Rabatt** übergeben und daraus den Preis der Bestellposition ermitteln lassen.

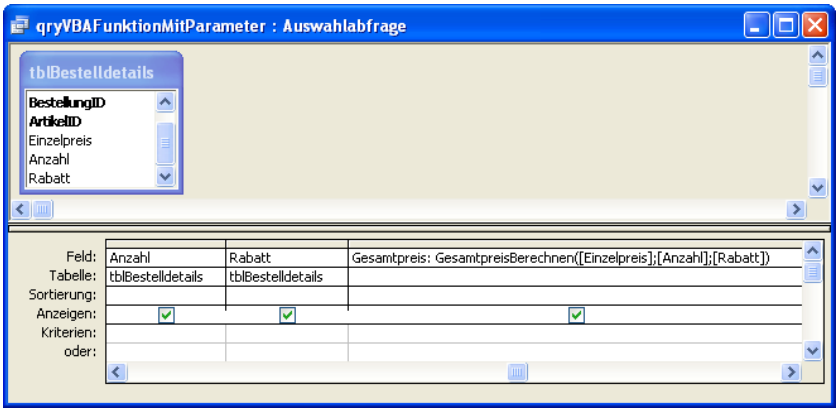



**Bild 4:** Entwurf der Abfrage mit dem Aufruf der VBA-Funktion mit Parametern

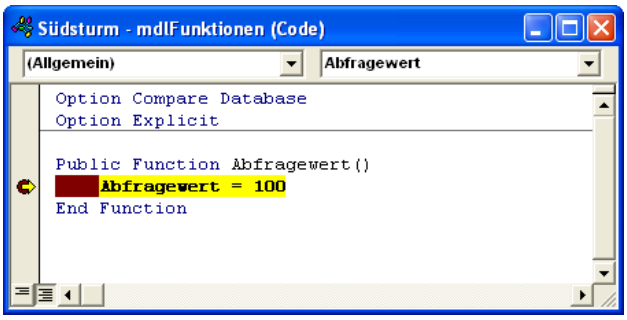



**Bild 3:** Mit einer Haltemarke stellen wir fest, ob und wann die VBA-Funktion aufgerufen wird.

Dazu erstellen Sie zunächst die folgende VBA-Funktion:

```
Public Function GesamtpreisBerechnen( _
        curEinzelpreis As Currency, _
        intAnzahl As Integer, _
        curRabatt As Currency) As Currency
    GesamtpreisBerechnen = curEinzelpreis * _
        intAnzahl * (1 - curRabatt)
End Function
```

Die Funktion erwartet beim Aufruf Werte für die Parameter **curEinzelpreis**, **intAnzahl** und **curRabatt**. Diese übergeben Sie beim Aufruf aus der Abfrage heraus durch Verweise auf die entsprechenden Felder des jeweiligen Datensatzes (siehe Bild 4):

```
Gesamtpreis: GesamtpreisBerechnen([Einzelpreis];
[Anzahl];[Rabatt])
```

Beim Anzeigen der Abfrage in der Datenblattansicht liest Access die Werte für den aktuellen Datensatz aus, ruft die VBA-Funktion mit diesen Parametern auf und gibt schließlich die ermittelten Werte aus.

Hierbei ist zu beachten, dass die VBA-Funktion tatsächlich für jeden Datensatz aufgerufen wird. Aber nicht auf einen Schlag! Die VBA-Funktion wird zunächst nur für die aktuell sichtbaren Datensätze ausgeführt. Dies können Sie ebenfalls durch Setzen eines Haltepunktes feststellen. Einfacher ist es jedoch, folgende Zeile in die Funktion einzufügen:

```
curEinzelpreis * intAnzahl * (1 -
curRabatt)
```

Der VBA-Editor zeigt dann die Parameterwerte der bis dato behandelten Datensätze im Direktfenster an.

# 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.

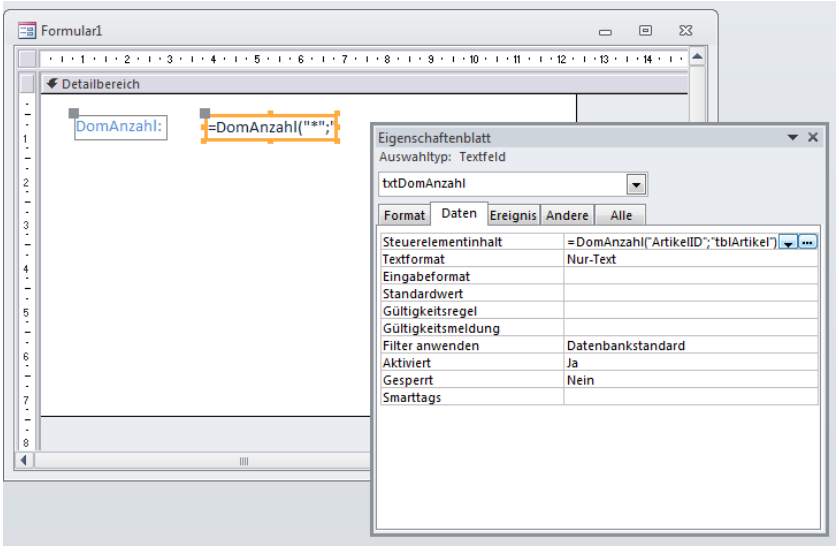



**Bild 1:** Textfeld mit einer Domänenfunktion als Steuerelementinhalt

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.

Access [basics] 2/2011
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Access [basics]-News

Schön, dass Sie uns mit Ihren Fragen und Themenvorschlägen unterstützen! Für diese Ausgabe haben Sie sich beispielsweise etwas zum Thema Adressen gewünscht.

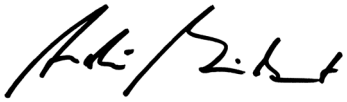
Im konkreten Fall lautete die Frage, wie man Adressblöcke etwa für ein Anschreiben zusammenstellt. Das Problem hierbei sind unvollständige Datensätze – hier fehlt ein Vorname, dort die Straße, und im nächsten Fall soll noch ein Adresszusatz eingearbeitet werden.

Wie dies funktioniert, erfahren Sie in dieser Ausgabe. Außerdem kommen wir auf die in den News der vorherigen Ausgabe angekündigten Themen zurück: Wir werfen einen Blick auf die Möglichkeiten von Kombinationsfeldern und nutzen diese zum Erweitern unserer Beispieldatenbank, der Projektzeiterfassung.

Sie erfahren hier, wie Sie Aufgaben und Tätigkeiten zu einem Projekt anlegen. Wozu brauchen Sie hier Kombinationsfelder? Nun, zum Beispiel zum Auswählen des Projekts, für das Sie eine Aufgabe anlegen wollen. Oder in doppelter Ausführung zum Auswählen des Projekts und der Aufgabe, für die Sie eine Tätigkeit erstellen – ein willkommender Anlass, die Technik der abhängigen Kombinationsfelder zu erläutern!

Wenn Sie weitere Themenwünsche haben oder eine technische Frage zu Access haben, wenden Sie sich einfach mit einer E-Mail an die Adresse **info@access-basics.de** an mich. Ich freue mich schon auf Ihre Anregungen.

Viel Erfolg mit der neuen Ausgabe!



André Minhorst

## In dieser Ausgabe:

**Abfragen für die Datenauswahl – Teil VIII: Abfrageeigenschaften**

**Adressblöcke in Berichten – Anschreiben mit Empfänger-Adressen ausstatten**

**Access-Funktionen – Teil II: Bedingungen: Wenn(), Schalter() und Wahl()**

**Formulare für die Dateneingabe – Teil VI: Abhängige Kombinationsfelder**

**Formulare für die Dateneingabe – Teil VII: Standardwerte statisch und dynamisch einstellen**

**Projektzeiterfassung – Aufgaben und Tätigkeiten verwalten**

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst | Fachverlag für Softwareentwicklung | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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).

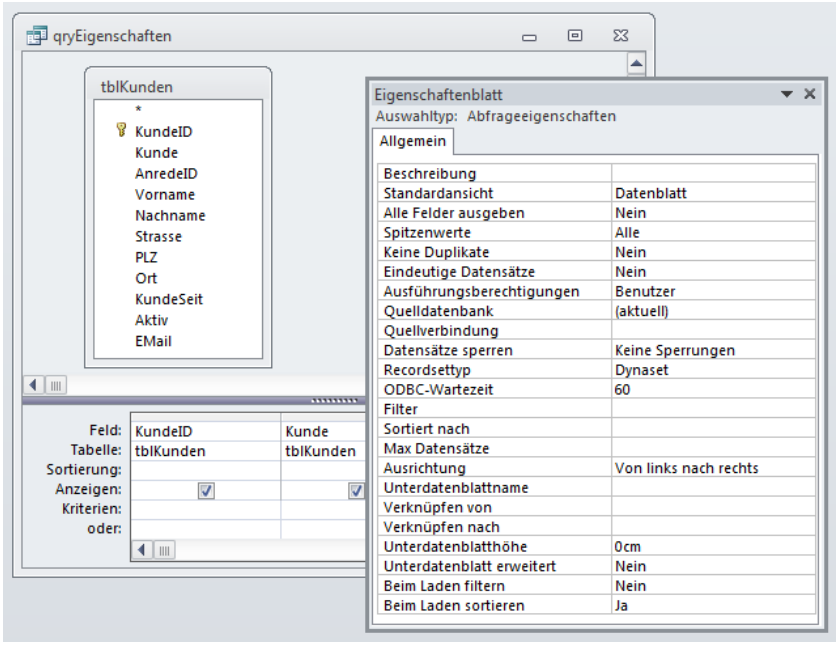



**Bild 1:** Das Eigenschaftsfenster einer Abfrage

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**.

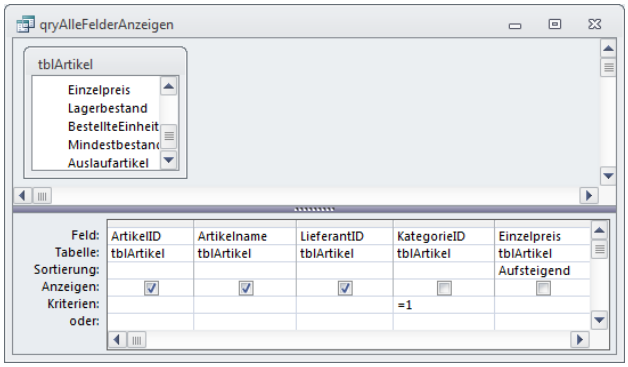



**Bild 2:** Eine Abfrage mit Feldern, die nicht im Ergebnis erscheinen sollen

## 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.

Dies ist auch in der Abfrage aus Bild 2 der Fall. Die Eigenschaft **Alle Felder ausgeben** können Sie nun verwenden, wenn Sie die ausgeblendeten Felder zu Kontrollzwecken doch einmal im Abfrageergebnis sehen möchten. Access zeigt dann alle Felder an, die überhaupt im Entwurfsraster enthalten sind.

Sie brauchen dann nicht erst mehrere Häkchen zu setzen oder zu entfernen, sondern können dies mit einer einzigen Eigenschaft erledigen.

## Spitzenwerte

Neben den üblichen Kriterien für Abfragen gibt es auch noch ein feldübergreifendes Kriterium: Die Eigenschaft **Spitzenwerte** legt fest, wieviele Datensätze beziehungsweise wieviel Prozent der Datensätze des eigentlichen Abfrageergebnisses angezeigt werden sollen (siehe Bild 3).

Sie können hier einen der voreingestellten Werte wie **5** oder **25%** auswählen, aber auch benutzerdefinierte Werte einstellen. Wenn Sie nur den ersten Datensatz anzeigen möchten, tragen Sie den Wert **1** ein, soll es die erste Hälfte sein, verwenden Sie den Wert **50%**.

Diese Eigenschaft wirkt sich auf die aktuell festgelegte Sortierung aus. Im Beispiel sind die Artikel nach dem Einzelpreis sortiert. Liegt keine Sortierung vor, wird die Primärschlüsselspalte als Sortierkriterium verwendet. Sollte die Spalte, die als Sortierspalte dient, Nullwerte enthalten, und die Sortierung ist aufsteigend, enthält das Abfrageergebnis auch die Datensätze mit dem Nullwert im betroffenen Feld. Die Datensätze mit Nullwert werden jedoch zusätzlich zu den ersten x Datensätzen angezeigt.

Wenn Sie also **Spitzenwerte** auf **10** einstellen und die Tabelle enthält einen Datensatz mit einem Nullwert in der Sortierspalte, liefert die Abfrage elf Datensätze zurück. Dieses Verhalten tritt bei der prozentualen Anzeige nicht auf: Hier werden die Datensätze mit Nullwerten im Sortierfeld mitgezählt.

Sollte der letzte Datensatz des Ergebnisses einen mehrfach vorkommenden Wert in der Sortierspalte aufweisen, werden diese Datensätze zusätzlich zur Ergebnismenge hinzugefügt. Wenn Sie als einen Artikel zu zehn Euro, einen zu acht Euro und drei zu fünf Euro haben und die ersten drei Datensätze ausgeben lassen, erhalten Sie trotzdem fünf Datensätze.

## Keine Duplikate

Stellen Sie die Eigenschaft **Keine Duplikate** auf **Ja** ein, liefert das Abfrageergebnis nur Datensätze, deren Inhalte in den durch die Abfrage ausgegebenen Feldern gleich sind.

In Abfragen, die das Primärschlüsselfeld der Haupttabelle enthalten, macht diese Einstellung also keinen Sinn – durch den eindeutigen Schlüssel ist sowieso jeder Datensatz eindeutig.

Wenn Sie aber beispielsweise herausfinden wollen, in welchen Orten sich Ihre Kunden befinden, können

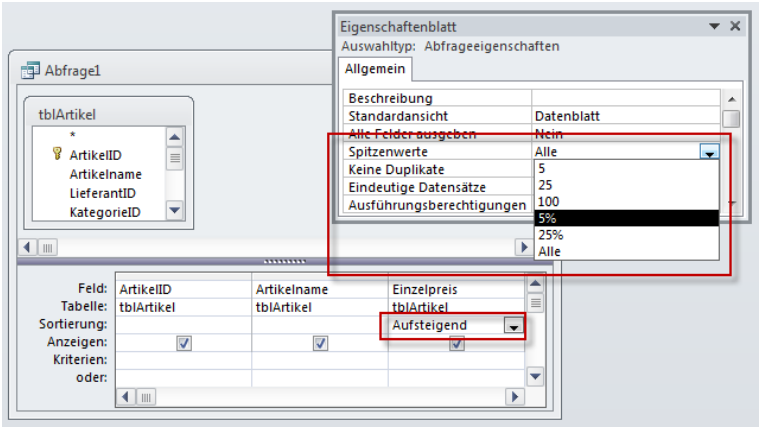



**Bild 3:** Festlegen der Anzahl/Prozentzahl der anzuzeigenden Datensätze

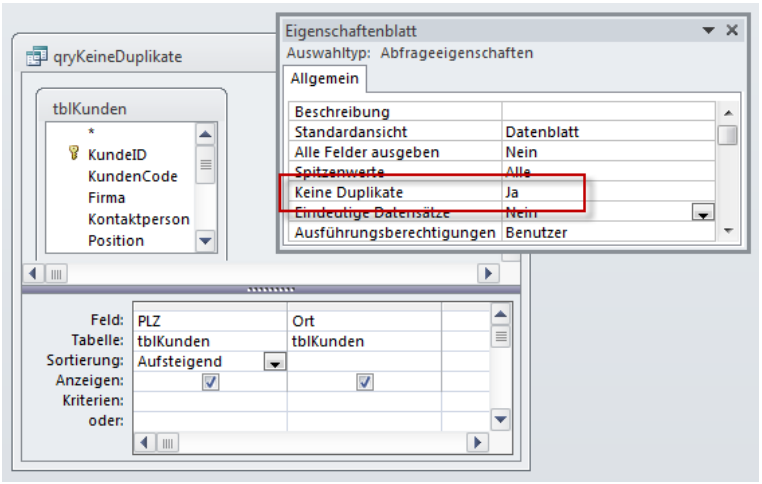



**Bild 4:** Ermitteln aller eindeutigen Orte der Kundentabelle

# 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.

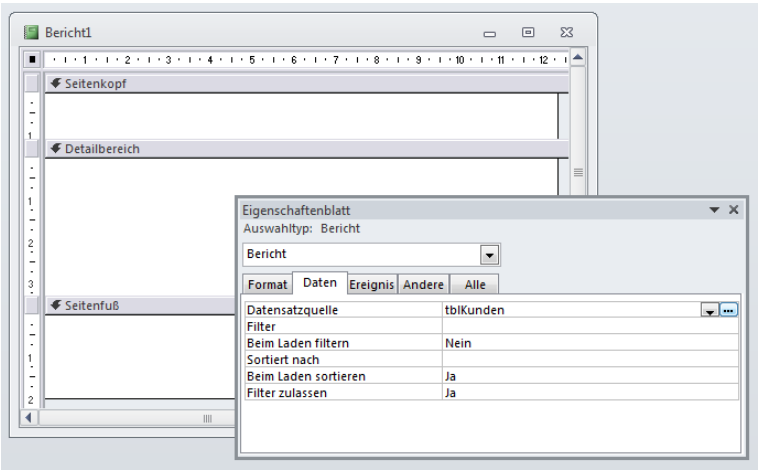



**Bild 1:** Zuweisen der Datenherkunft zum Bericht

## 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.

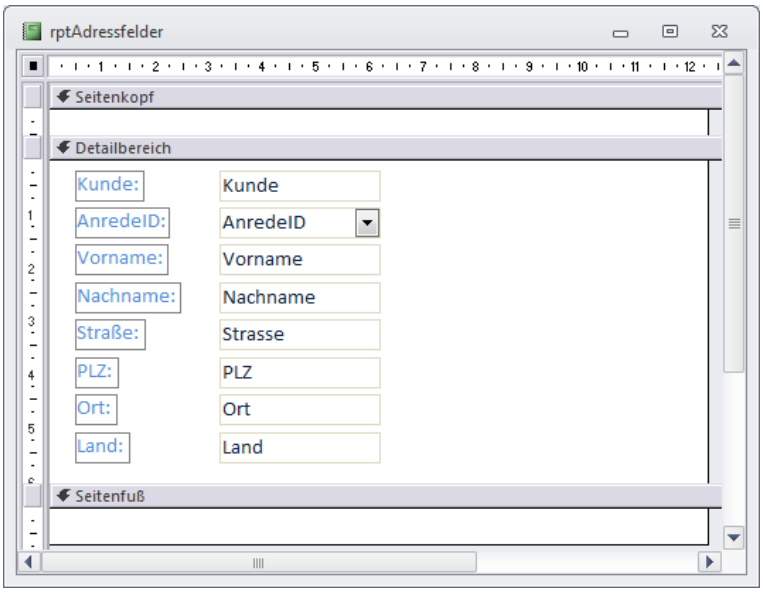



**Bild 2:** Adressfeld, ganz grob

## 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).

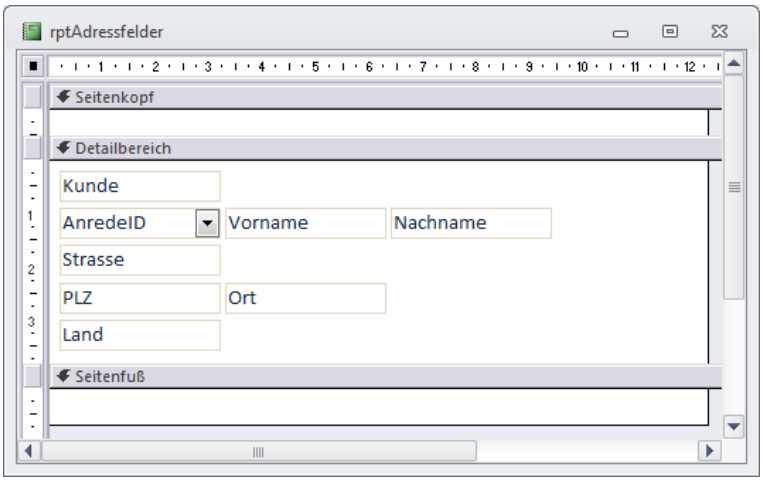



**Bild 3:** Adressfeld, ganz grob

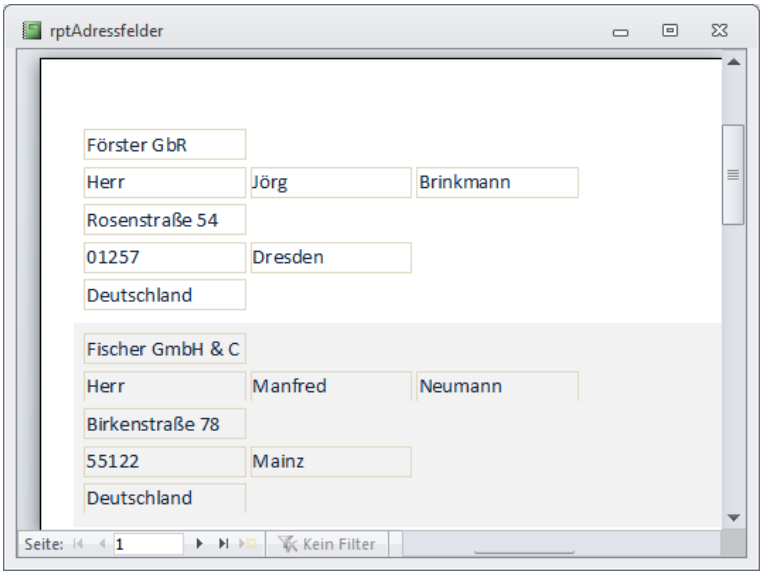



**Bild 4:** Ein erster Wechseln in die Seitenansicht zeigt dieses Bild.

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.

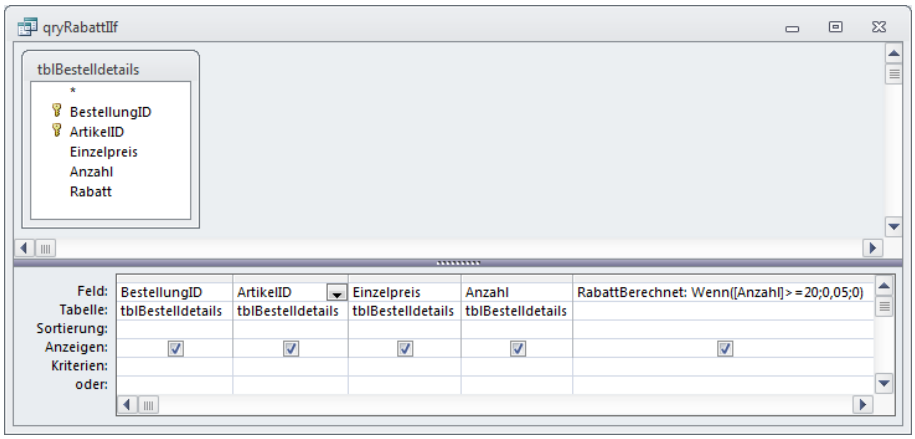



**Bild 1:** Eine Wenn-Funktion in einer Abfrage

- **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))
```

Die Abfrage arbeitet erst die äußere **Wenn**-Bedingung ab. Ist die Anzahl größer gleich 20, wird als Rabatt 0,05 festgelegt. Falls nicht, kommt die innere **Wenn**-Bedingung zum Einsatz. Wobei dies nicht ganz korrekt ist: Die innere Bedingung wird auf jeden Fall ebenfalls berechnet – unabhängig davon, ob sie überhaupt relevant ist.

## Die Schalter-Bedingung

Sie werden jedoch schnell feststellen, dass das Verschachteln mehrere **Wenn**-Bedingungen schnell unübersichtlich wird. In diesem Fall können Sie die **Schalter**-Bedingung (unter VBA: **Switch**) einsetzen. Die **Schalter**-Bedingung erwartet ein oder mehrere Ausdruck-Wert-Paare. Sie gibt den ersten Wert zurück, dessen Ausdruck den Wert **True** liefert. Erweitern wir also das Beispiel so, dass der Rabatt bei einer Anzahl von 50 oder mehr bei 10%, bei 20 oder mehr bei 5% und bei 10 oder mehr bei 2% liegt. Dann sieht die Funktion in der Abfrage von oben etwa so aus:

```
RabattSchalter: Schalter([Anzahl]>=50;0,1;
[Anzahl]>=20;0,05;[Anzahl]>=10;0,02)
```

Der Ausdruck hat nur einen kleinen Schönheitsfehler: Wenn die Anzahl kleiner als 10 ist, liefert die Funktion den Wert **Null**, hinterlässt also ein leeres Feld. Also greifen wir hier auf einen der folgenden Artikel dieser Reihe vor und setzen die **Nz**-Funktion ein. Diese prüft, ob der enthaltene Wert **Null** ist und ersetzt diesen gegebenenfalls durch den im zweiten Parameter der **Nz**-Funktion angegebenen Wert. Im folgenden Beispiel soll das Feld den Wert **0** anzeigen, wenn keiner der Rabatte zutrifft:

```
RabattSchalter: Nz(Schalter([Anzahl]>=50;0,1;
[Anzahl]>=20;0,05;[Anzahl]>=10;0,02);0)
```

Auch hier werden jedoch alle Bedingungen geprüft.

## Auswahl per Index

Die dritte Funktion im Bunde ist die **Wahl**-Bedingung (unter VBA: **Choose**). Sie hat keine Ausdruck-Wert-Paare, sondern fragt nur im ersten Parameter einen variablen Ausdruck ab. Dieser sollte Werte wie **1**, **2**, **3** und so weiter annehmen. Die folgenden Parameter enthalten genau diesem Index entsprechende Werte. Enthält der erste Parameter beispielsweise den Wert **1**, wird der erste der folgenden Parameterwerte zurückgegeben. Dies ist zum Beispiel praktisch, wenn Sie Texte zu den Zahlenwerten der Optionen einer Optionsgruppe ermitteln möchten. Im Beispielformular aus Bild 3 finden Sie beispielsweise eine Optionsgruppe namens **ogrSchulnoten**. Die einzelnen Optionen dieser Optionsgruppe haben die Werte **1** bis **6**. Das Textfeld rechts heißt **txtSchulnote** und besitzt als Steuerelementinhalt den folgenden Ausdruck:

```
=Wahl([ogrSchulnoten];"Sehr Gut";"Gut";"Befriedigend"
;"Ausreichend";"Mangelhaft";"Ungenügend")
```

Dies bedeutet, dass die Wahl-Funktion jeweils den Wert von **ogrSchulnoten** ermittelt und im Textfeld **txtSchulnote** die entsprechende Bezeichnung anzeigt. Hat **ogrSchulnoten** beispielsweise den Wert **2**, wählt die **Wahl**-Funktion aus den folgenden Parametern den Eintrag **"Gut"** aus. Das Ergebnis wird direkt im Textfeld **txtSchulnote** angezeigt.

## Mögliche Probleme

Wie bereits erwähnt, werten alle drei Funktionen alle enthaltenen Ausdrücke aus. Wenn Sie dort also beispielsweise eine VBA-Funktion unterbringen, wird diese für jeden Parameter, der diese enthält, ausgeführt. Dies können Sie schnell etwa mit der VBA-Variante der Funktion **Wahl** ausprobieren. Geben Sie im Direktfenster des VBA-Editors (zu öffnen mit **Strg + G**) einmal den folgenden Ausdruck ein:

```
Choose(2,InputBox("Geben Sie Wert 1 ein"),
InputBox("Geben Sie Wert 2 ein"))
```

Obwohl der erste Parameter den Wert **2** enthält und dementsprechend nur der im dritten Parameter enthaltenen Ausdruck ausgewertet werden sollte, geschieht das für alle beide – es erscheinen gleich zwei Eingabefenster.

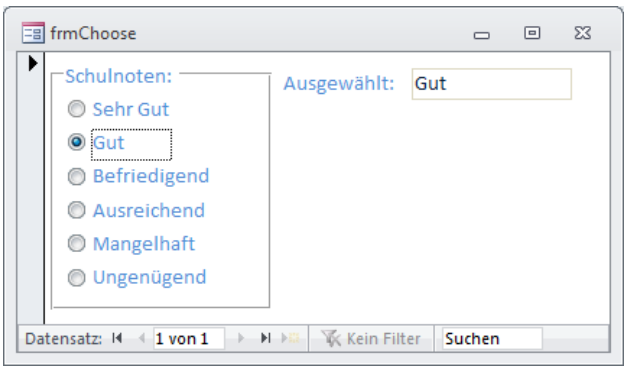



**Bild 3:** Anzeige des ausgewählten Textes mithilfe der **Wahl**-Bedingung

# 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.

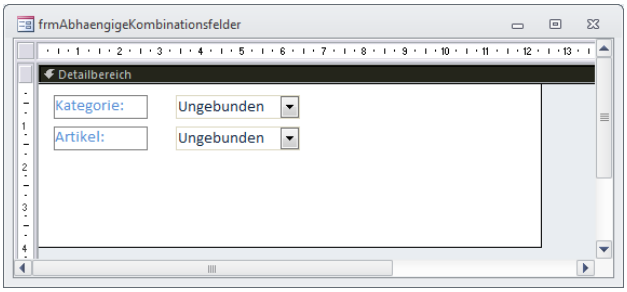



**Bild 1:** Formular mit den beiden Kombinationsfeldern im Rohbau

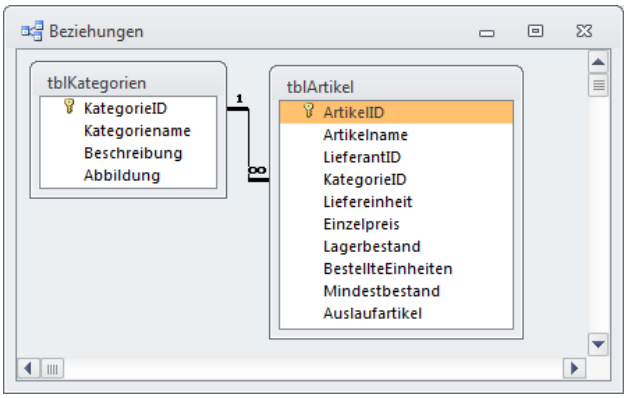



**Bild 2:** Beziehung zwischen Kategorien und Artikeln

### 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.

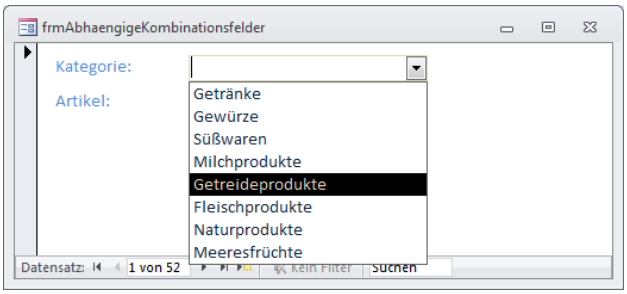



**Bild 3:** Auswahl der Kategorie per Kombinationsfeld

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.

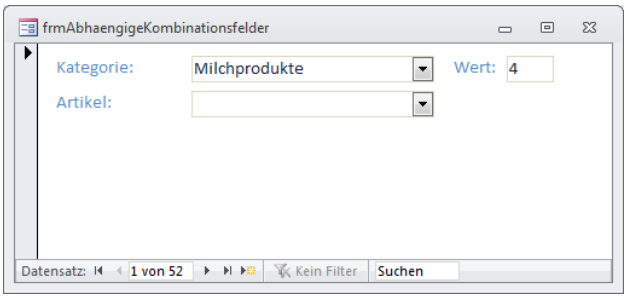



**Bild 4:** Anzeige des eigentlichen Wertes des Kombinationsfeldes

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:

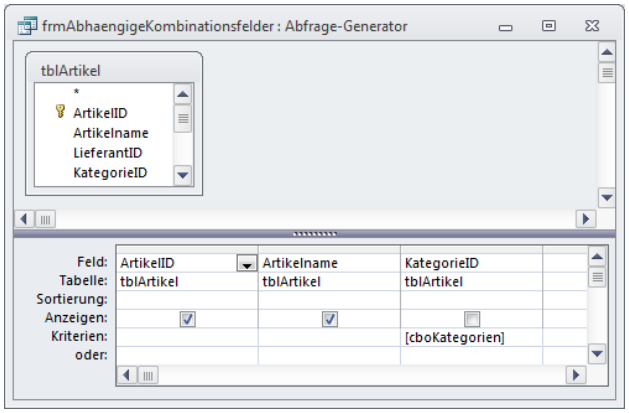



**Bild 5:** Datensatzherkunft des zweiten Kombinationsfeldes

```
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.

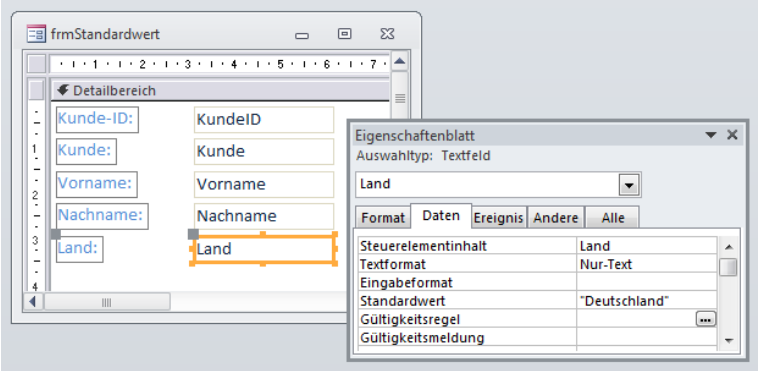



**Bild 1:** Einstellen des Standardwerts in einem Formular

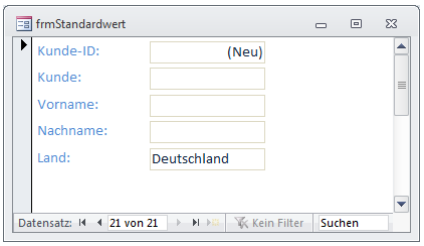



**Bild 2:** Felder mit Standardwert werden vorbelegt.

## 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):

```
Private Sub Form_AfterUpdate()
    Me!Land.DefaultValue = Me!Land.Value
End Sub
```

Das Eintragen eines Wertes wie **Deutschland** in den aktuellen Datensatz und der anschließende Wechseln zu einem neuen Datensatz liefert nicht das gewünschte Ergebnis (siehe Bild 3). Warum nicht? Nun: Die Eigenschaft **DefaultValue** erwartet eine Zeichenkette als Wert. Die einzige Anweisung unserer Prozedur interpretiert den übergebenen Wert, beispielsweise Deutschland, nicht als String, sondern als Variablen-/Funktionsnamen. Dementsprechend sucht das Feld im nächsten Schritt nach einer Variablen oder Funktion namens Deutschland, die aber nicht da ist.

Damit die Eigenschaft **Standardwert** den neuen Wert tatsächlich als Zeichenkette erkennt, müssen Sie diesen in Anführungszeichen einfassen. Sie müssen dem Ausdruck also ein führendes und ein folgendes Anführungszeichen hinzufügen. Da ein Anführungszeichen aber selbst eine spezielle Aufgabe hat, nämlich den Beginn und das Ende von Zeichenketten zu markieren, müssen Sie es einer Sonderbehandlung unterziehen. In diesem Fall verdoppeln Sie es einfach. Ein einzelnes Anführungszeichen als Zeichenkette wird also durch den Ausdruck **""""** repräsentiert – ein öffnendes, ein verdoppeltes und ein schließendes Anführungszeichen. Die Anweisung ändert sich dadurch wie folgt:

```
Me!Land.DefaultValue = """" & Me!Land.Value & """"
```

Wenn Zeichenketten in Textfeldern schon eine Sonderbehandlung benötigen, wie sieht es dann mit den anderen Felddatentypen aus? Um es kurz zu machen: Es gibt weitere Probleme und Sonderbehandlungen. Dies liegt darin begründet, dass die **DefaultValue**-Eigenschaft Zeichenketten als Wert aufnimmt. Wenn die eigentlichen Werte der Textfelder aber einen anderen Datentyp als Text besitzen, gibt es Probleme dabei, diesen Wert erst für den Verbleib in der **DefaultValue**-Eigenschaft in eine Zeichenkette zu konvertieren und diese anschließend wieder mit dem entsprechenden Datentyp weiterzuverwenden.

Eine Dezimalzahl beispielsweise, die auf deutschen Systemen in der Regel mit dem Komma als Dezimaltrennzeichen angezeigt, aber für die interne Weiterverarbeitung den Punkt als Trennzeichen erwartet, führt zu folgendem Problem: Der gespeicherte Wert ist beispielsweise **1.5**. Angezeigt wird aber **1,5**.

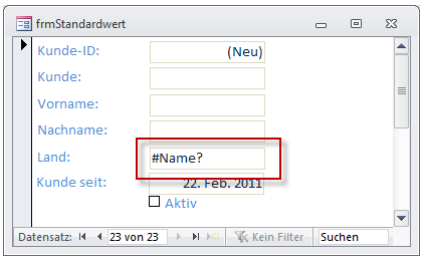



**Bild 3:** Felder mit Standardwert werden vorbelegt.

Wenn Sie diesen Wert der Eigenschaft **DefaultValue** zuweisen, speichert diese auch **1,5** als Zeichenkette. Ein neuer Datensatz kann diesen Wert jedoch wegen dem Komma nicht verwerten und zeigt stattdessen den Ausdruck **#Name?** an. Damit aus der Dezimalzahl mit Komma eine Dezimalzahl mit Punkt als Dezimaltrennzeichen wird, müssen Sie diese mit der **CDbl**-Funktion und dann mit der **Str**-Funktion behandeln, also etwa so:

```
Me!Zahlenfeld.DefaultValue =
Str(CDbl(Me!Zahlenfeld.Value))
```

Gleiches gilt beim Datum, das ja intern als Double-Wert gespeichert wird (mehr dazu später). Wenn Access dann einen Ausdruck wie **1.1.2011 10:00:30** im **Standardwert** speichert und diesen später in einem Feld, das eigentlich eine **Double**-Zahl erwartet, anzeigen soll, erscheint ebenfalls der Ausdruck **#Name?**. Hier wenden Sie die gleiche Behandlung wie bei Zahlenfeldern an.

Bei den übrigen Felddatentypen wie Ganzzahlen oder Ja/Nein können Sie der Eigenschaft **DefaultValue** jedoch einfach den Wert des Feldes zuweisen.

## Warum nicht einfach einen Wert statt des Standardwerts einstellen?

Sie können jederzeit den Wert eines Feldes auf einen gültigen Wert einstellen. Dies funktioniert im Gegensatz zum Standardwert auch für Datensätze, die bereits angezeigt werden. Der wichtige Unterschied ist, dass das Ändern des Feldwertes auch gleich den Datensatz ändert – diese ist also nicht mehr unberührt.

Wenn der Benutzer es sich also anders überlegt und doch keinen neuen Datensatz mehr anlegen möchte, muss er die Änderungen am Datensatz zunächst etwa mit der Escape-Taste verwerfen. Wenn Sie nur den Standardwert ändern, wirkt sich dies nicht auf den Zustand des Datensatzes aus – dieser bleibt unberührt. Das betroffene Feld zeigt zwar einen Wert an, aber dieser steckt lediglich in der Eigenschaft **Standardwert** und nicht im Feld selbst.

# 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.

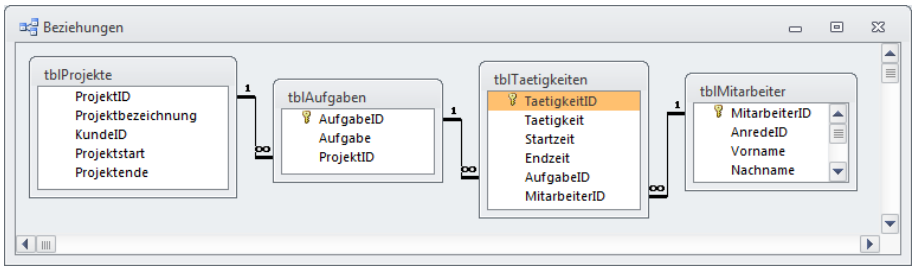



**Bild 1:** Hierarchie bestehend aus Projekten, Aufgaben, Tätigkeiten und Mitarbeitern

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.

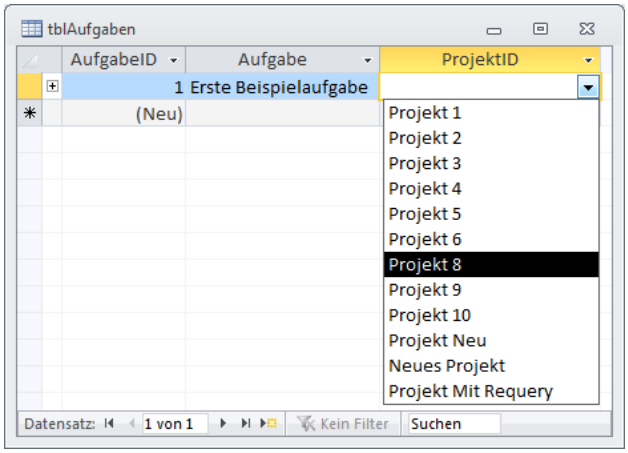



**Bild 2:** Hierarchie bestehend aus Projekten, Aufgaben, Tätigkeiten und Mitarbeitern

## 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.

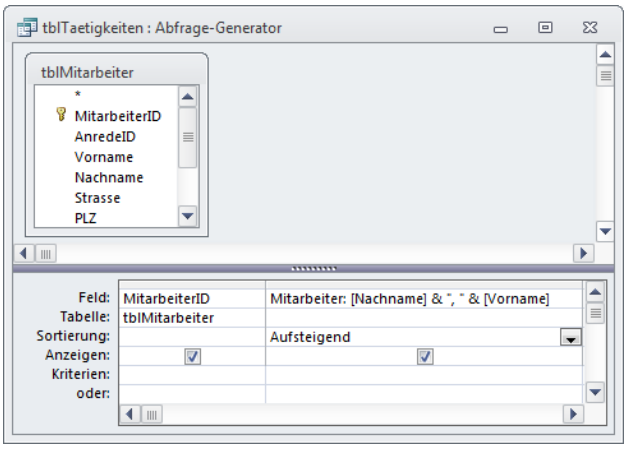



**Bild 3:** Datensatzherkunft des Nachschlagefeldes **MitarbeiterID** der Tabelle **tblTaetigkeiten**

## 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**.

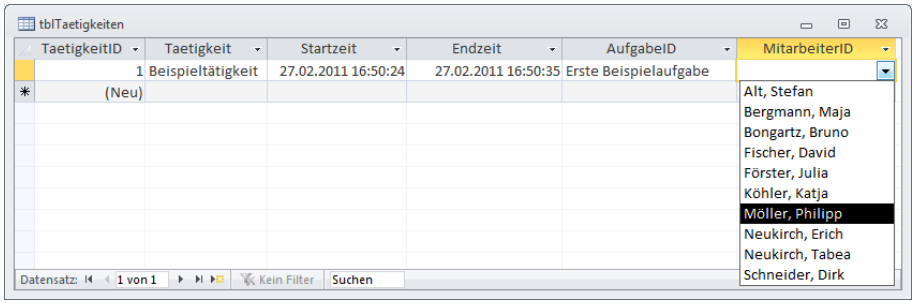



**Bild 4:** Komforable Auswahl des Mitarbeiters, der eine Tätigkeit durchführt

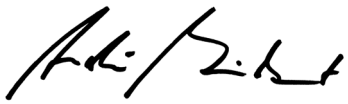
Access [basics] 3/2011
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Access [basics]-News

Schön! In dieser Ausgabe arbeiten wir wieder eine Menge grundlegender Themen ab. Sie lernen zum Beispiel mit den Indizes ein Werkzeug kennen, mit dem Sie die Sortierung von und die Suche nach Daten in Tabellen optimieren können (**Tabellen entwerfen - Teil VI: Indizierung von Feldern**). Mit Indizes können Sie auch festlegen, dass ein Feld nur eindeutige Werte speichern darf. Um sicherzustellen, dass Sie einem Feld einen solchen Index zuweisen können, müssen Sie zunächst die Duplikate rauswerfen – mehr dazu erfahren Sie unter **Abfragen für die Datenauswahl - Duplikate per Abfrage finden**. Sie lernen, wie Sie Daten tabellarisch in Berichten aufarbeiten (**Reporting mit Berichten - Einfache Tabelle in der tabellarischen Ansicht darstellen**). Außerdem fügen wir eine Ebene hinzu und betrachten die Handhabung verknüpfter Daten wie etwa von Kunden und ihren Projekten in einem Bericht (**Reporting in Berichten - Darstellung von 1:n-Beziehungen per Gruppierung**). Schließlich werfen wir einen sehr genauen Blick auf Daten, die im Datum/Zeit-Format gespeichert sind (**Datum und Zeit, Teil I - Umgang mit Datums- und Zeitangaben in Tabellen, Formularen, Berichten und VBA**).

Warum das "schön" ist? Weil **Access [basics]** sich nun mehr und mehr fortgeschritteneren und anspruchsvolleren Themen widmen kann! Und das ist ja auch Ihr Ziel: Die Grundlagen genau kennenzulernen und dieses Wissen später für echte Problemlösungen einzusetzen.

In diesem Sinne – viel Spaß beim Lesen der neuen Ausgabe!



André Minhorst

## In dieser Ausgabe:

**Tabellen entwerfen - Teil VI: Indizierung von Feldern** | Was sind Indizes und wie setze ich diese in meiner Datenbank ein?

**Abfragen für die Datenauswahl - Duplikate per Abfrage finden** | Doppelte Daten führen mitunter zu Inkonsistenzen. Raus damit!

**Reporting mit Berichten - Einfache Tabelle in der tabellarischen Ansicht darstellen** | Tabellarische Daten formschön in Berichten abbilden

**Reporting in Berichten - Darstellung von 1:n-Beziehungen per Gruppierung** | Verknüpfte Daten in Berichten darstellen – hier am Beispiel von Kunden und Projekten

**Datum und Zeit, Teil I - Umgang mit Datums- und Zeitangaben in Tabellen, Formularen, Berichten und VBA** | Alles, was Sie über den Datum/Zeit-Datentyp wissen müssen

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst | Fachverlag für Softwareentwicklung | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autoren: André Minhorst

**ISSN: 2190-8761**

# 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

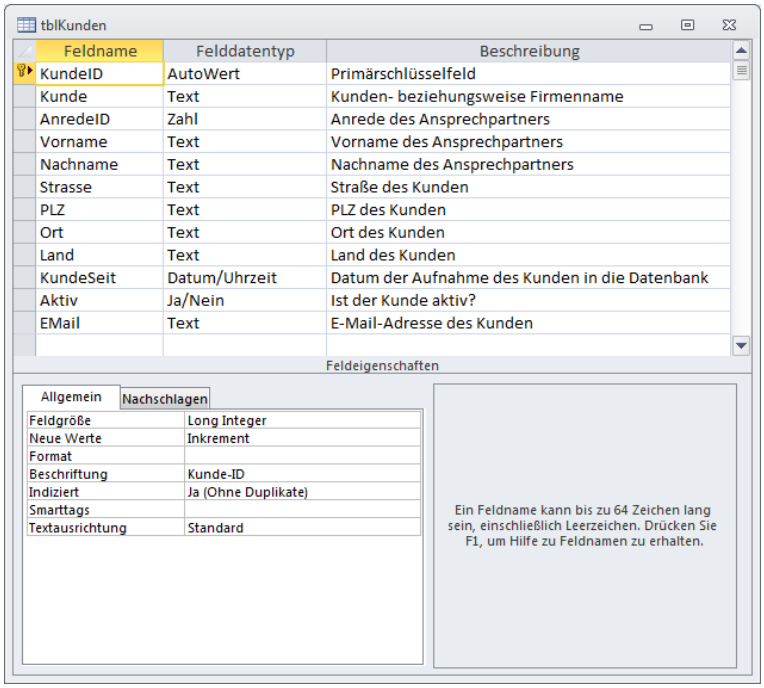



**Bild 1:** Die Tabelle **tblKunden** in der Entwurfsansicht

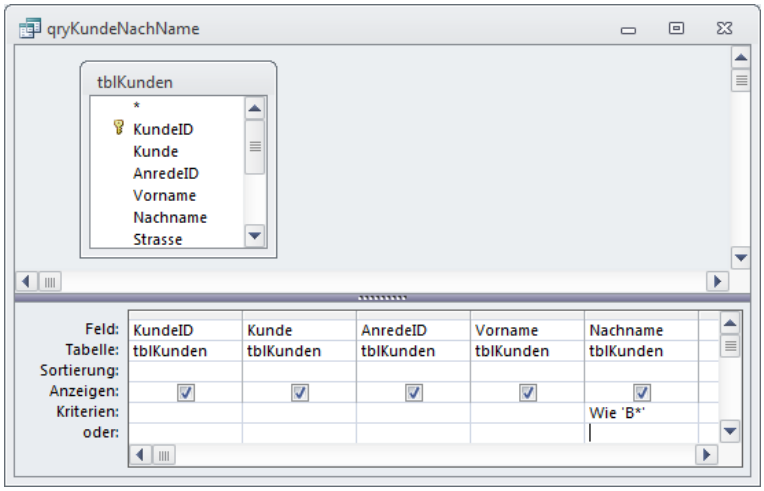



**Bild 2:** Abfrage nach dem Nachnamen in der Entwurfsansicht ...

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.

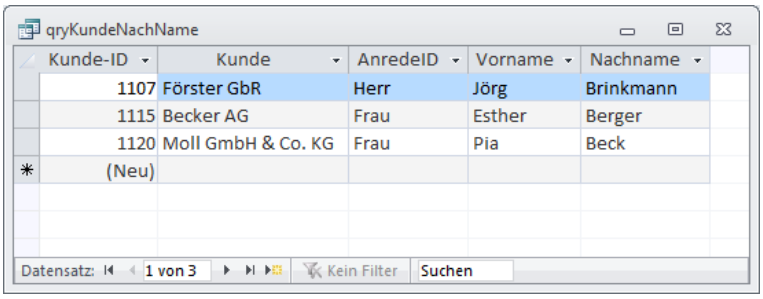



**Bild 3:** ... und das Ergebnis der Abfrage.

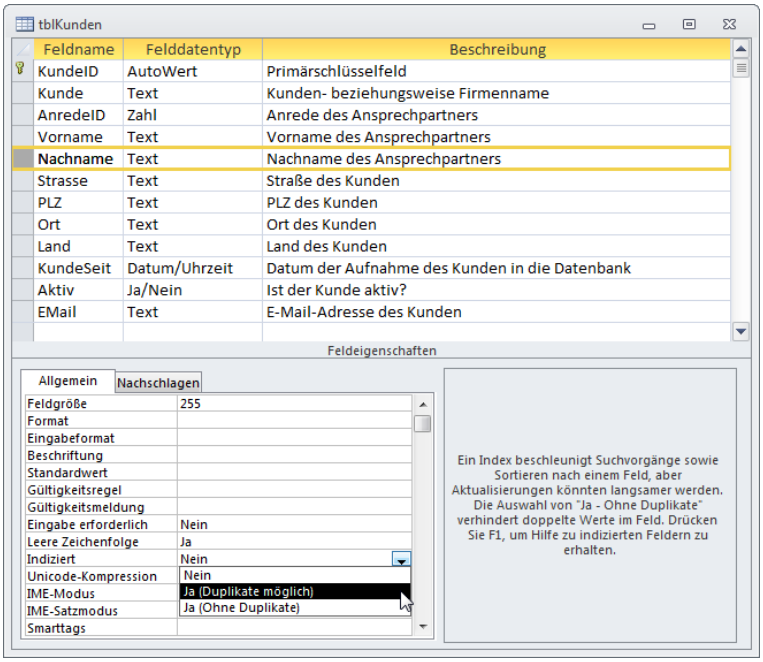



**Bild 4:** Anlegen eines Indexes für das Feld **Nachname**

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.

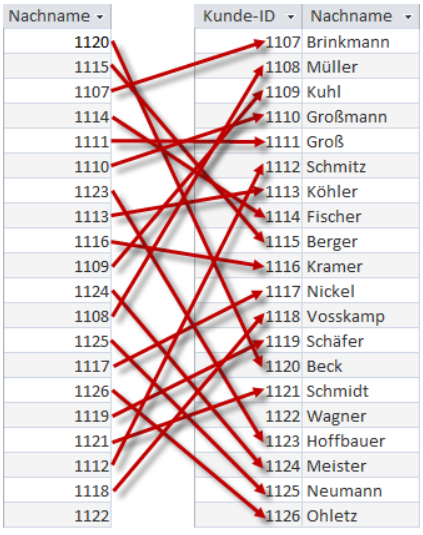



**Bild 5:** Schematisches Aussehen einer Indextabelle

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).

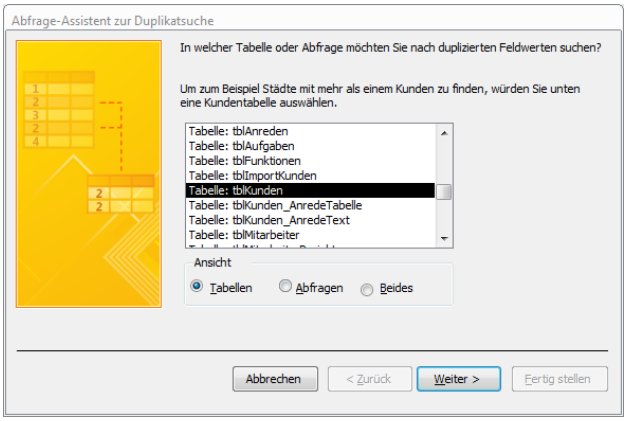



**Bild 1:** Zu untersuchende Tabelle oder Abfrage auswählen

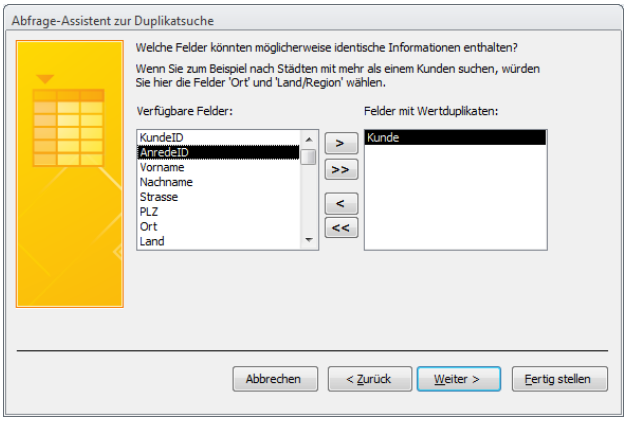



**Bild 2:** Auswählen der zu untersuchenden Felder

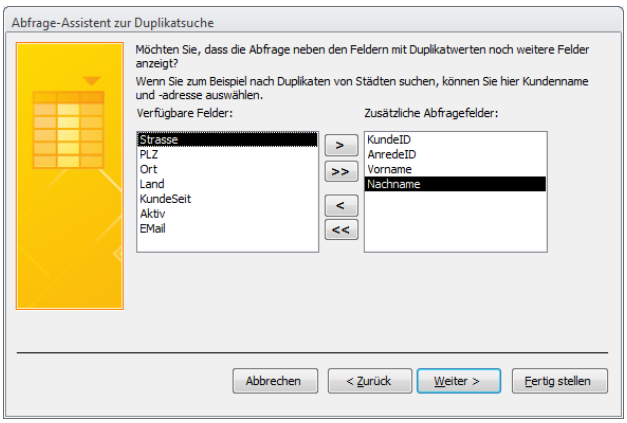



**Bild 3:** Weitere Felder für die Anzeige auswählen

Interessant ist der Aufbau der Abfrage, die Access nach Ihren Angaben im Assistenten erstellt hat (siehe Bild 6). Die Abfrage enthält zunächst alle Felder, die überhaupt für den Duplikatsvergleich und die Anzeige ausgewählt wurden, wobei das beziehungsweise die Vergleichsfelder ganz links erscheinen.

Soweit gibt es keinen Unterschied zu den Auswahlabfragen, die Sie bisher in den Artikeln dieser Reihe kennen gelernt haben – wenn das Kriterium für das Feld **Kunde** nicht wäre. Dieses lautet nämlich folgendermaßen:

```
In (SELECT [Kunde] FROM [tblKunden] As Tmp
GROUP BY [Kunde] HAVING Count(*)>1 )
```

Aber auch die in diesem Kriterium verwendete IN-Klausel haben Sie schon kennengelernt, nämlich im Artikel **Abfragen für die Datenauswahl - Teil V: Filtern nach Zahlen**.

Lediglich die Gruppierung mit dem **GROUP BY**-Schlüsselwort und die anschließend verwendete **HAVING**-Klausel haben wir noch nicht verwendet. Dieser **SELECT**-Ausdruck gruppiert die Datensätze der Tabelle **tblKunden** zunächst nach dem Feld **Kunde**. Anschließend legt es eine Bedingung fest, nämlich **Count(*)>1**. Dies bedeutet: Liefere alle Werte von Kunde zurück, für die es mehr als ein Vorkommen gibt.

Eigentlich wäre dies schon die Antwort auf die Frage nach den doppelt vorkommenden Kunden, aber da wir noch einige weitere Informationen wie **KundeID** et cetera sehen wollen, liefert die Hauptabfrage

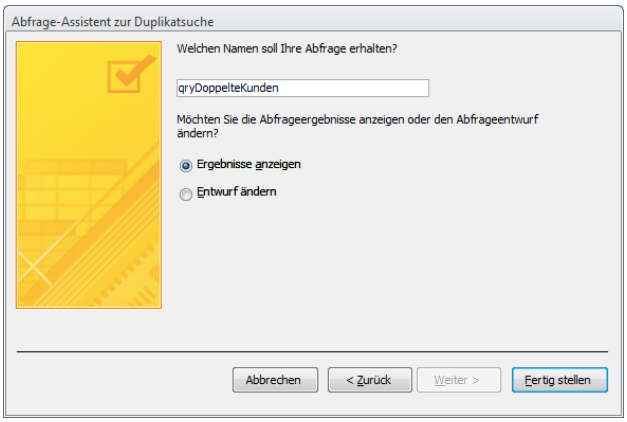



**Bild 4:** Festlegen eines Speichernamens für die Duplikatsabfrage

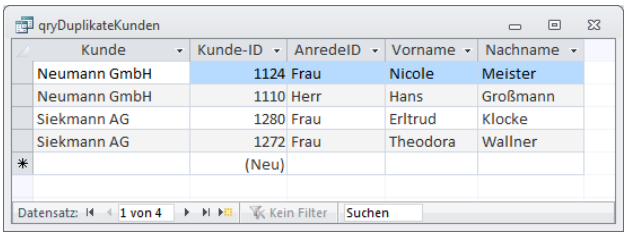



**Bild 5:** Gefundene Duplikate in der Tabelle tblKunden

genau die Datensätze der Tabelle, deren Feld **Kunde** dem Ergebnis der Unterabfrage entspricht.

Mit diesem Abfrageergebnis können Sie nun schon arbeiten und beispielsweise tatsächlich doppelte Kunden entfernen. Zumindest theoretisch: In der Praxis müssen Sie natürlich zuvor prüfen, ob nicht etwa schon Projekte für beide Kopien des Kunden angelegt wurden. Diese müssen dann natürlich vor dem Löschen des einen Exemplars auf das andere übertragen werden.

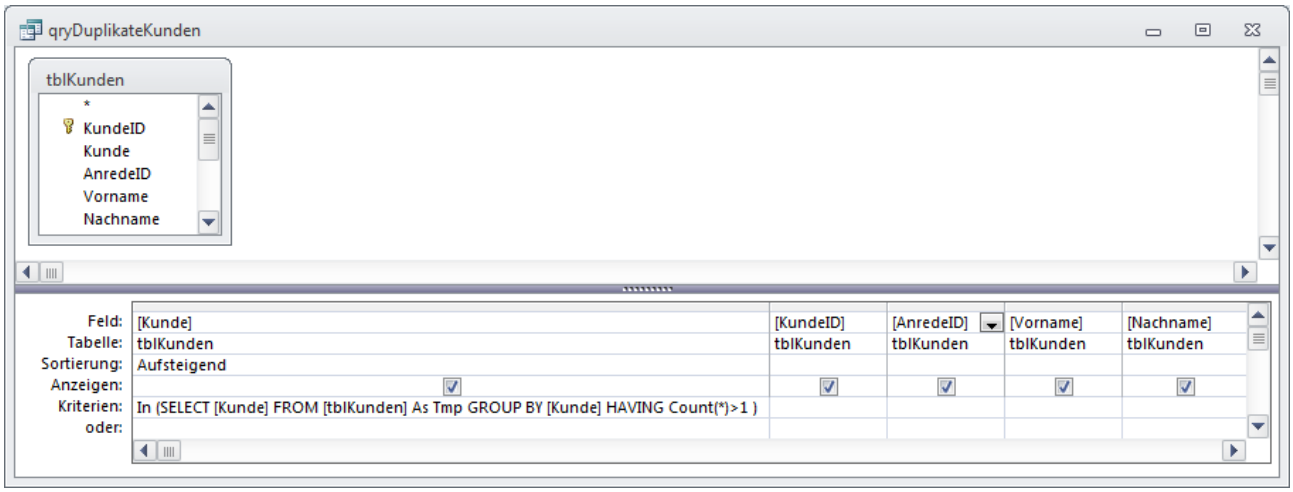



**Bild 6:** Die Abfrage zum Ermitteln aller mehrfach vorkommenden Kundennamen in der Entwurfsansicht

# 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-**

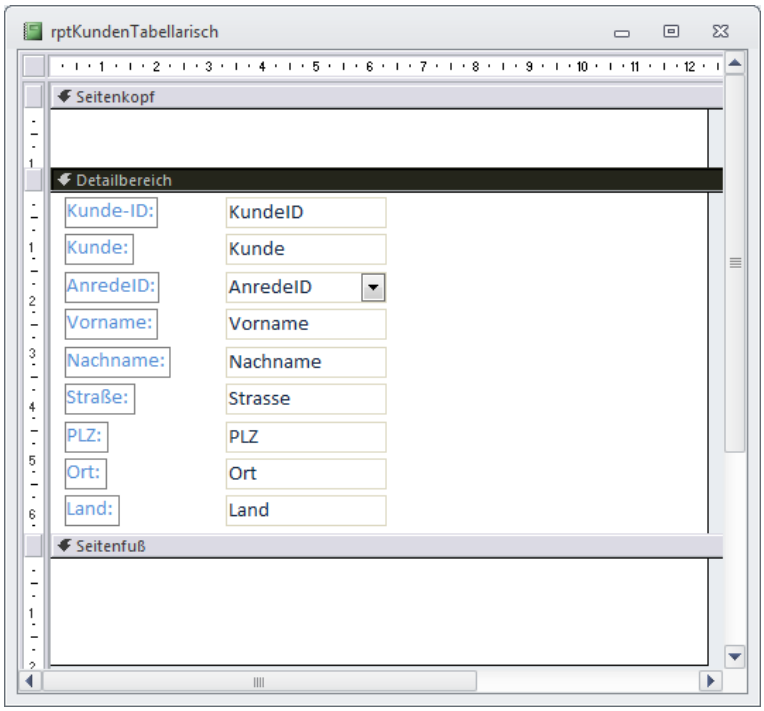



**Bild 1:** Die gewünschten Felder sind schonmal im Bericht gelandet.

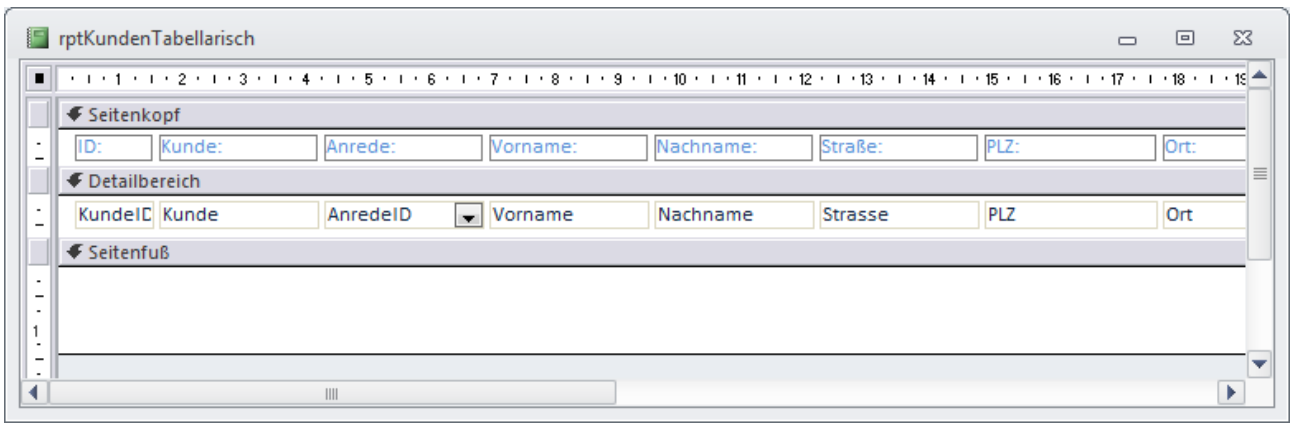



**Bild 2:** Die Bezeichnungsfelder landen im Seitenkopf, der Rest im Detailbereich.

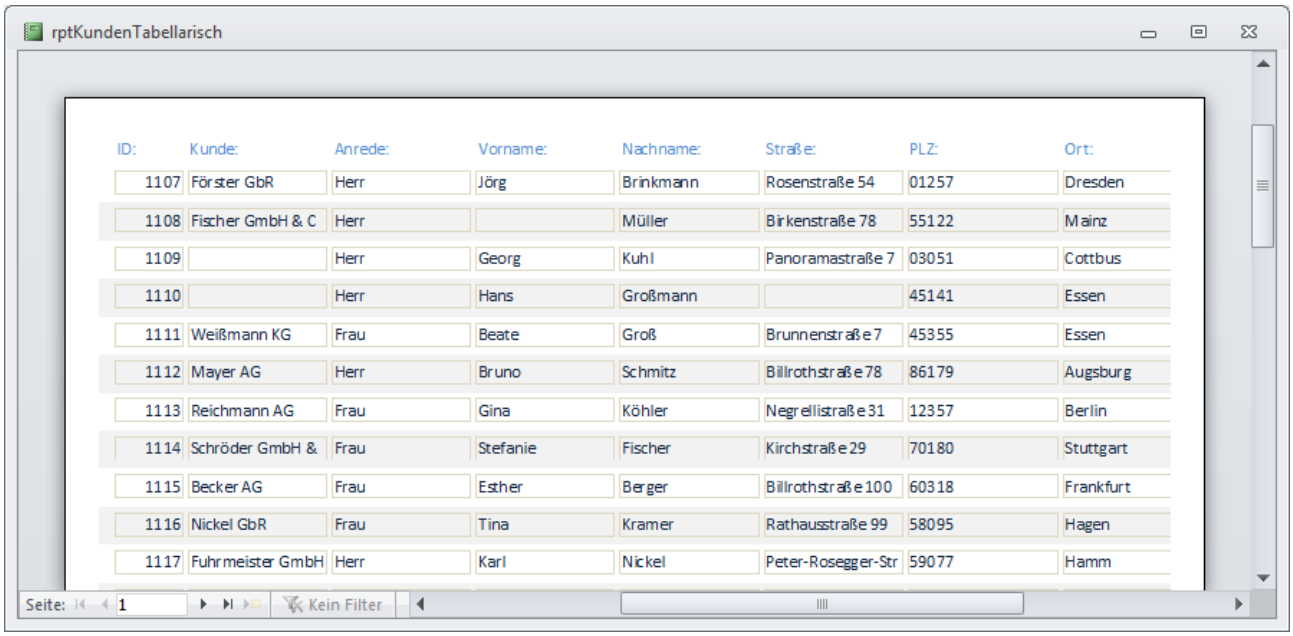



**Bild 3:** Der erste Entwurf in der Berichtsvorschau

**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.

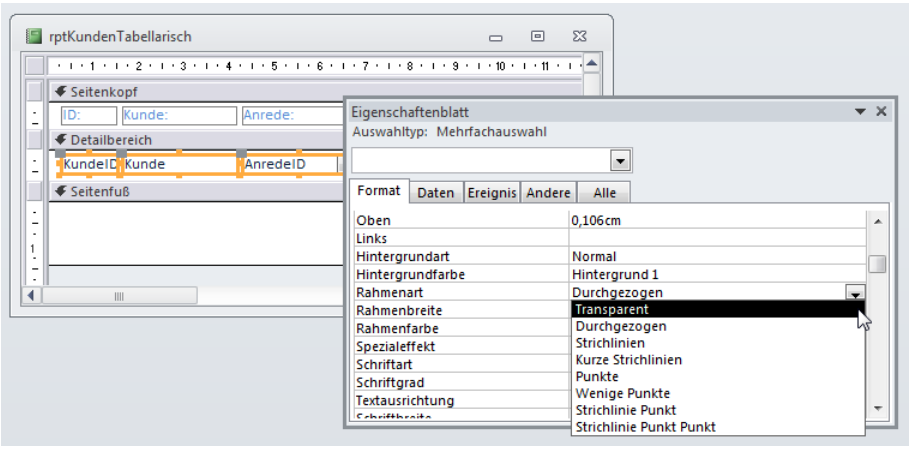



**Bild 4:** Entfernen der Textfeld-Rahmen

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).

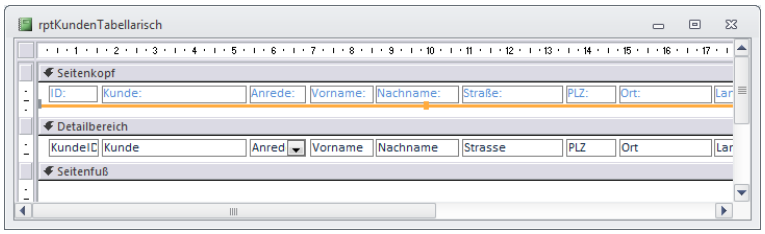



**Bild 5:** Positionierung der Trennlinie unter den Spaltenüberschriften

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.

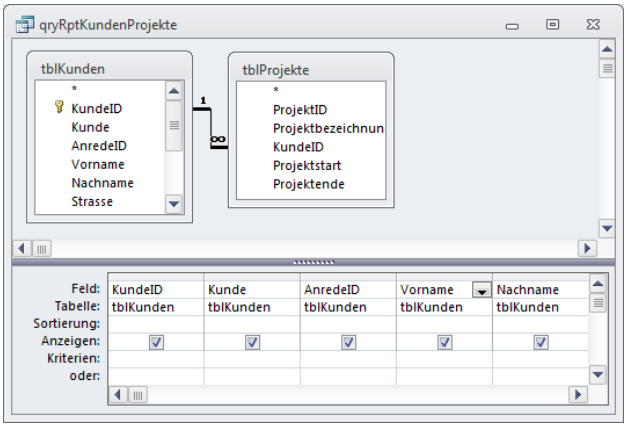



**Bild 1:** Datenherkunft des Berichts zur Anzeige von Kunden und Projekten

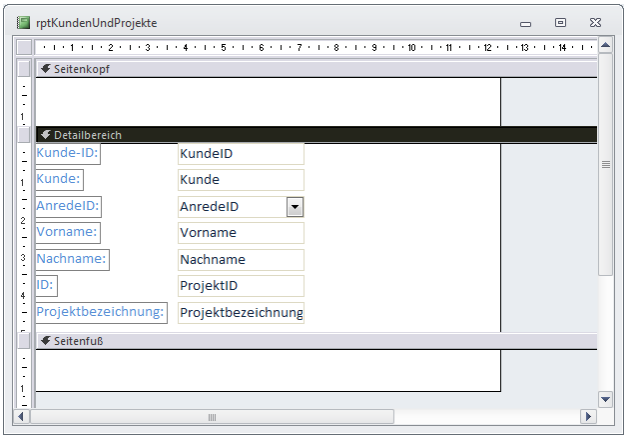



**Bild 2:** Erster Entwurf des Berichts

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).

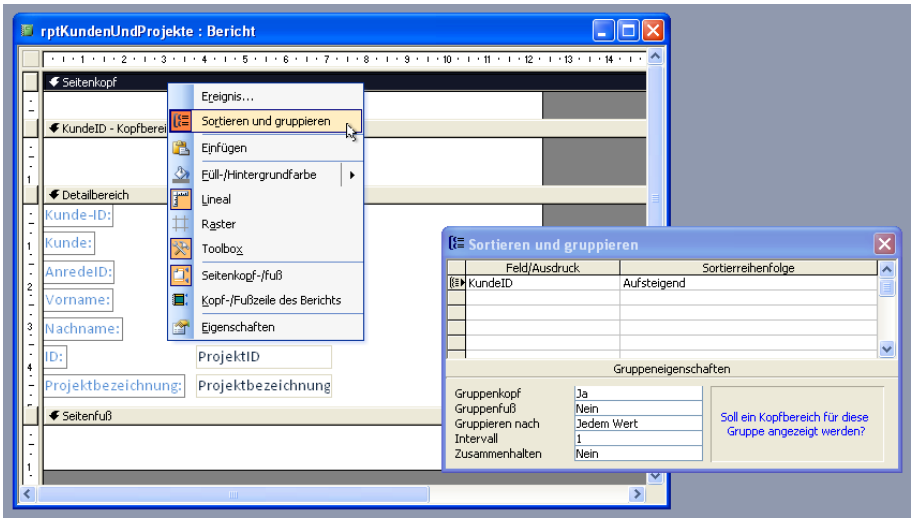



**Bild 4:** Einrichten einer Gruppierung unter Access 2003 und älter

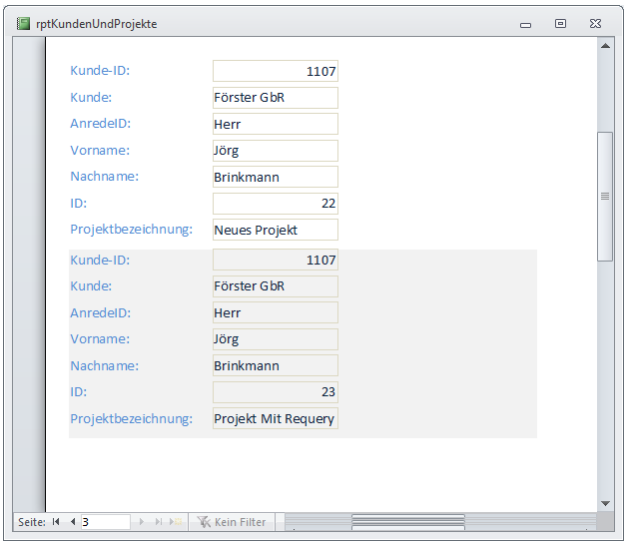



**Bild 3:** Im ersten Ansatz werden die Kundendaten für jedes Projekt des Kunden einmal ausgegeben.

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).

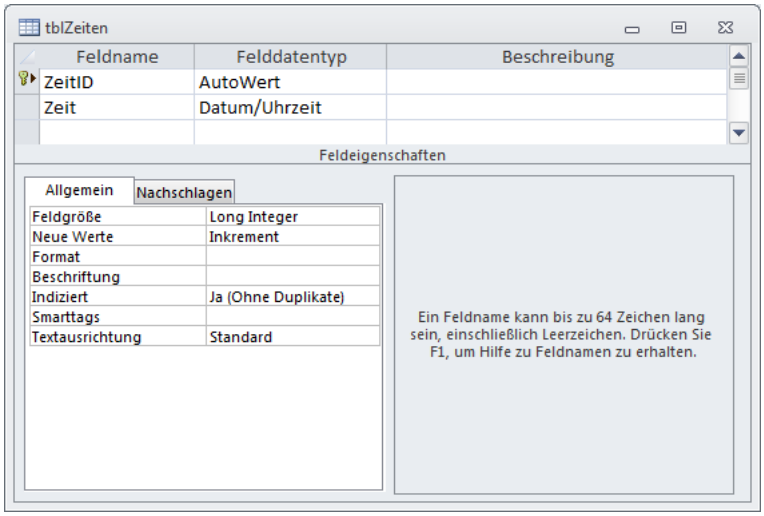



**Bild 1:** Die Tabelle **tblZeiten** in der Entwurfsansicht

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

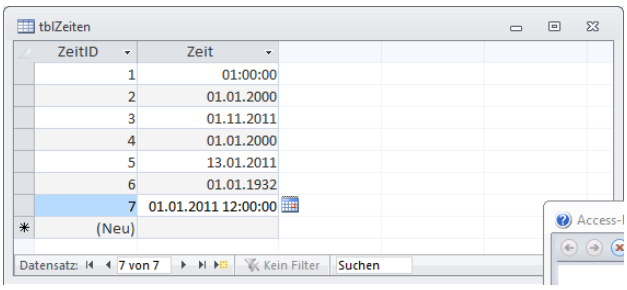



**Bild 2:** Die Zeit-Tabelle mit einigen Beispieldatensätzen

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:

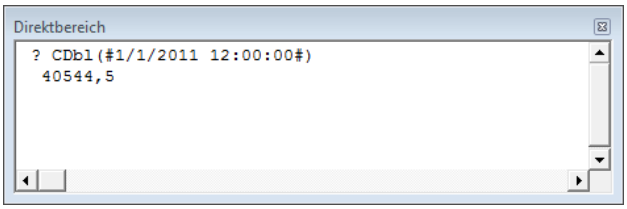



**Bild 3:** Umwandeln eines Datums mit der **CDbl**-Funktion

```
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**.

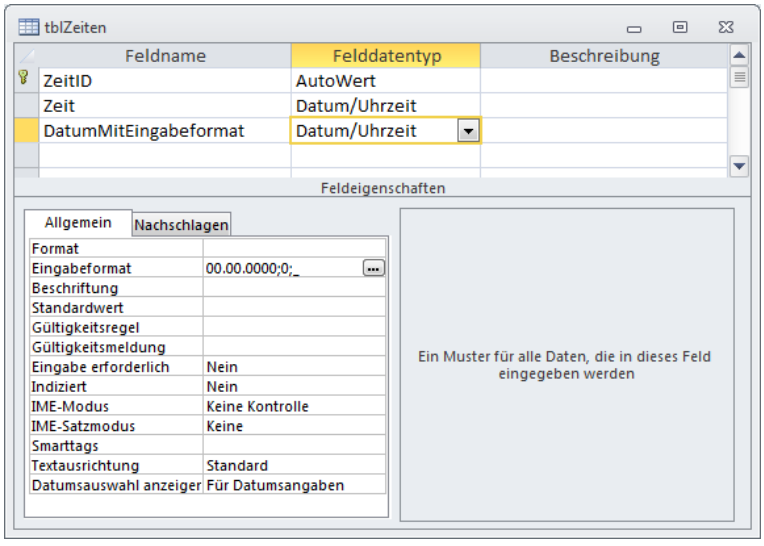



**Bild 4:** Eingabe eines Eingabeformats

Access [basics] 4/2011
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

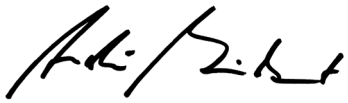
## Access [basics]-News

Geschafft! Diese Ausgabe ist die 13. Ausgabe und somit hat **Access [basics]** das erste Jahr erfolgreich hinter sich gebracht. Und Ihr Zuspruch gibt uns Recht: Die deutschsprachigen Access-Benutzer, Poweruser und Entwickler brauchen ein Magazin, dass ihnen die benötigten Informationen auf den Punkt nahebringt. Mit dieser Jubiläumsausgabe erhalten Sie als Bonus ein kostenloses Kapitel meines neuen Buch **Access 2010 – Das Grundlagenbuch für Entwickler**, dass Anfang Juni in den Buchläden verfügbar sein wird.

Das Kapitel heißt **Tabellen und Datenmodellierung** und ist natürlich auch für die Leser unter Ihnen, die noch nicht mit Access 2010 arbeiten, interessant. Dort geht es nämlich nicht nur um die grundlegenden Informationen zum Erstellen von Tabellen: Das Kapitel stellt auch eine Reihe von Datenmodellen vor, die Sie vielleicht als Anregung für den Entwurf eigener Anwendungen nutzen können.

In dieser Ausgabe dreht es sich gleich in zwei Artikeln um das Thema Datum und Zeit. Zunächst erfahren Sie, welche Funktionen Access für den Umgang mit Datum- und Zeitangaben bietet (**Access-Funktionen – Teil III: Datum und Zeit**). Anschließend zeigt der Beitrag **Abfragen für die Datenauswahl – Teil X: Datum und Zeit**, wie Sie in Abfragen auf Datum- und Zeit zugreifen. Außerdem lernen Sie die verschiedenen Arten von Schleifen kennen, mit denen Sie unter VBA Anweisungen wiederholt durchführen lassen können (**VBA-Grundlagen – Schleifen in VBA**) und fügen Ihren Formularen Funktionen zum Validieren hinzu (**Formulare für die Dateneingabe – Validieren von Benutzereingaben**).

Viel Spaß beim Lesen der neuen Ausgabe!



André Minhorst

## In dieser Ausgabe:

**Formulare für die Dateneingabe – Validieren von Benutzereingaben** | Erweitern Sie Ihre Formulare um Funktionen zum Prüfen der Eingaben in Textfelder.

**Access-Funktionen – Teil III: Datum und Zeit** | Lernen Sie die Funktionen für den Zugriff auf das aktuelle Datum und die Zeit und für den Umgang mit diesen Informationen kennen.

**Abfragen für die Datenauswahl – Teil X: Datum und Zeit** | Greifen Sie auf verschiedene Zeit-Informationen zu und filtern Sie Datensätze nach Datum und Uhrzeit.

**VBA-Grundlagen – Schleifen in VBA** | Erfahren Sie, wie Sie in VBA bestimmte Anweisungen nach verschiedenen Kriterien wiederholt ausführen können.

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst | Fachverlag für Softwareentwicklung | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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.

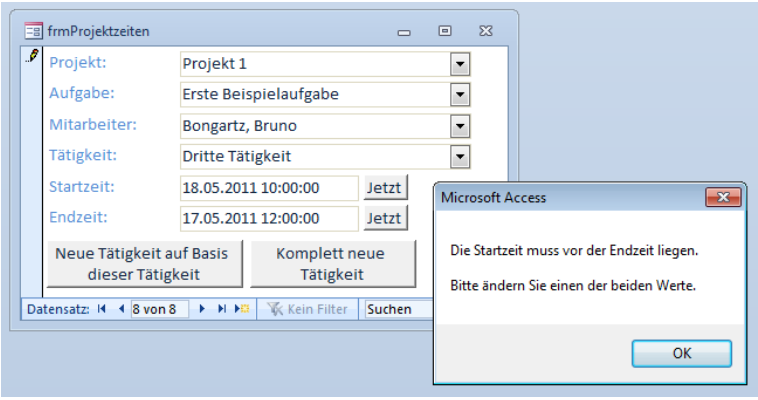



**Bild 1:** Beispiel für eine Validierungsmeldung

## 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

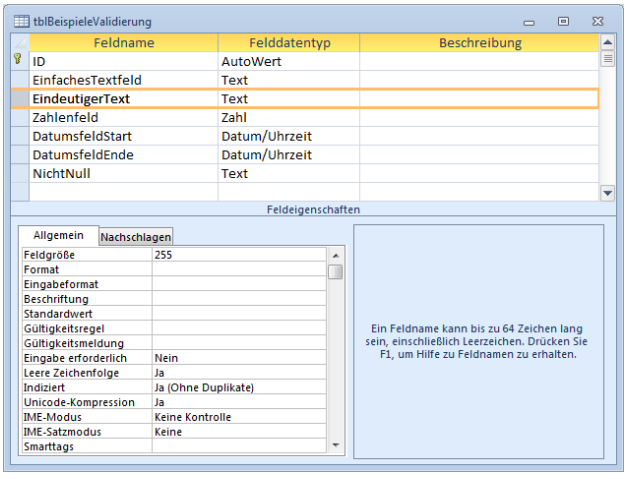



**Bild 2:** Das Feld **EindeutigerText** der Beispieltabelle hat einen eindeutigen Index.

# 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).

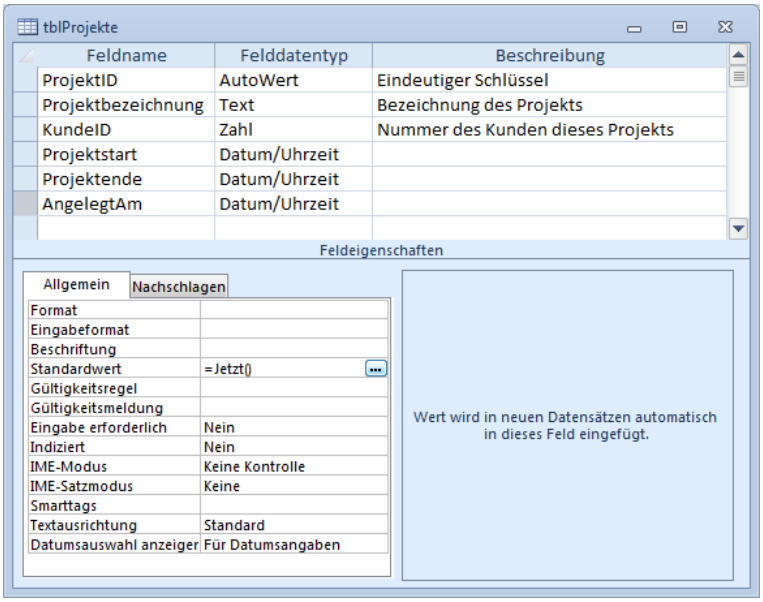



**Bild 1:** Anlegen einer Datumsfunktion als Standardwert eines Tabellenfeldes

## 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:

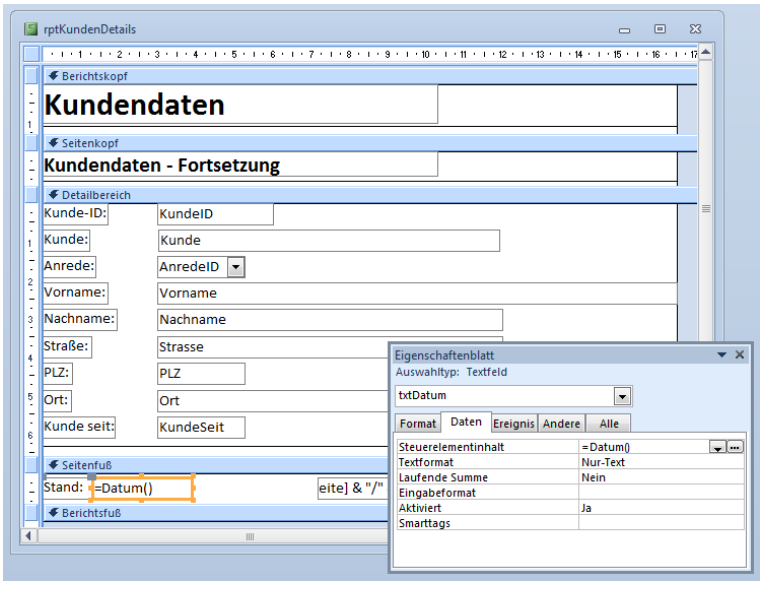



**Bild 2:** Verwenden einer Datumsfunktion im Seitenfuß eines Berichts

```
? 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

# 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.

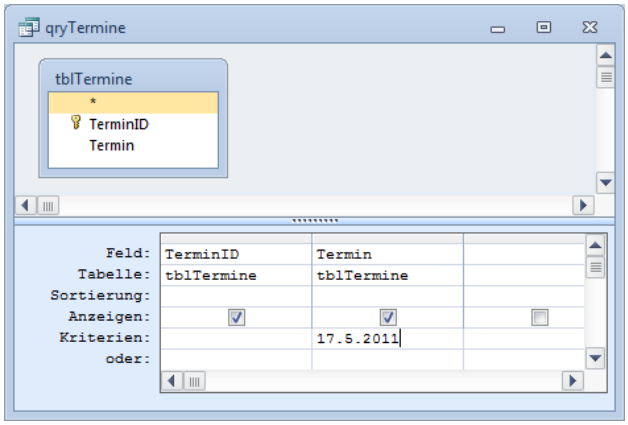
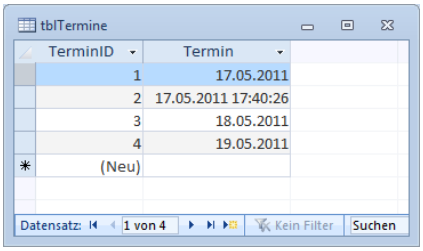



**Bild 1:** Eingabe eines einfachen Datums als Kriterium

## 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



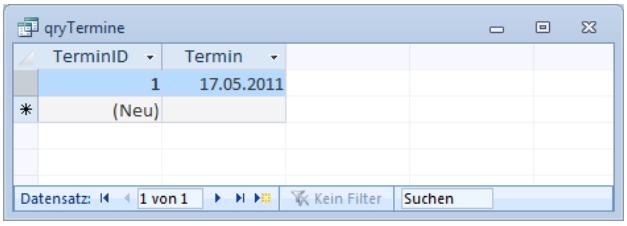




**Bild 3:** Sind das alle Termine am 17.5.2011?

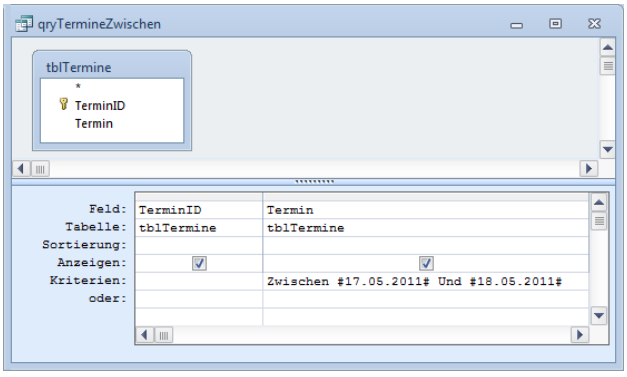



**Bild 4:** Abfrage mit einem Zwischen-Kriterium

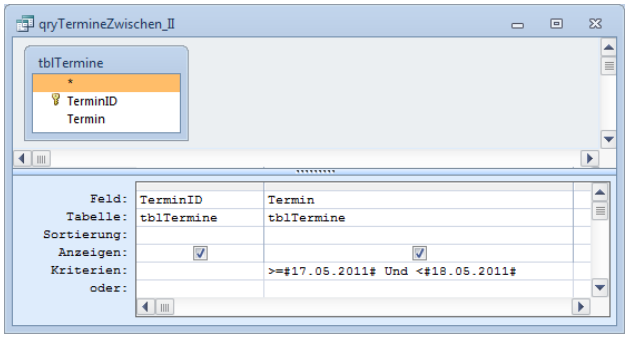



**Bild 5:** Eingrenzen des Datums mit anderen Mitteln

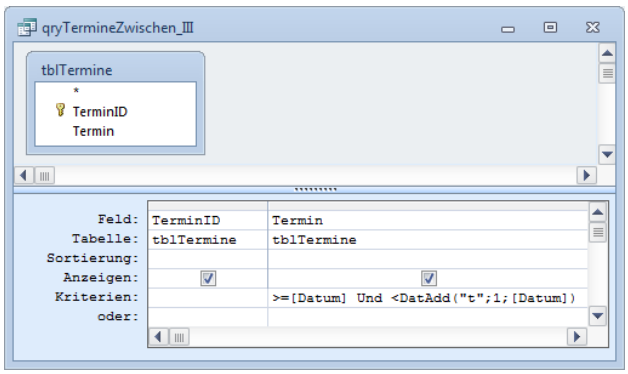



**Bild 6:** Termine eines Tages mit Parameter

mit einem Zahlenwert von **40681** genau hinein. Das **Zwischen**-Schlüsselwort ist also nur auf den ersten Blick praktisch. Einfacher ist es hier, mit den herkömmlichen Operatoren wie Größer (>), Kleiner (<), Größer/Gleich (>=), Kleiner/Gleich (<=) oder Gleich (=) zu arbeiten. Dies sieht für unseren Fall wie in Bild 5 aus.

## Alle Datumsangaben zu einem Tag ermitteln

Wenn Sie diese Abfrage mit einem Parameter zur Eingabe eines Datums ausstatten möchten und die Abfrage alle Datensätze zurückliefern soll, die zu dem mit dem Parameter übergebenen Tag gehören, verwenden Sie diesen Ausdruck als Kriterium (siehe Bild 6):

```
>=[Datum] Und <DatAdd("t";1;[Datum])
```

Die **DatAdd**-Funktion addiert einen Tag zum Parameter **[Datum]** hinzu. Das Kriterium liefert also alle Datensätze, bei denen das Datum dem mit dem Parameter **[Datum]** übergebenen Wert entspricht – gleich, ob das Feld **Termin** eine Uhrzeit enthält oder nicht.

## Alle Geburtsdaten eines Jahres ermitteln

Schauen wir uns nun die Tabelle **tblKunden** an, der wir das Feld **Geburtsdatum** hinzugefügt haben. Die Abfrage **qryAlleGeburtstage1980** liefert mit dem Kriterium alle Datensätze, deren Feld **Geburtstag** einen Wert aus dem Jahr **1980** enthält (siehe Bild 7).

Wenn wir diese Abfrage flexibler gestalten und einen Parameter zur Eingabe des gesuchten Geburtsjahres hinzufügen möchten, wird es schwierig: Sie können hier nicht einfach den Text **1980** gegen einen Parameter wie **[Geburtsjahr]** austauschen. Dazu ist schon ein etwas komplizierterer Ausdruck nötig:

```
>=ZDate("01.01." & [Geburtsjahr]) Und
<DatAdd("jjjj";1;ZDate("1.1." & [Geburtsjahr]))
```

Hier werden zunächst Zeichenketten aus dem Datumsteil **01.01.** und dem per Parameter erfragten Jahr zusammengestellt und dann mit **ZDate** in Datumswerte umgewandelt. Beim zweiten Wert wird außerdem noch mit **DatAdd** ein Jahr hinzugefügt.

Dies geht allerdings noch einfacher, wenn wir nur für das Kriterium ein weiteres Feld zur Abfrage hinzufügen, das nur das Jahr des Geburtsdatums liefert. Diesem können Sie dann direkt den Parameter **[Geburtsjahr]** als Kriterium zuweisen (siehe Bild 8).

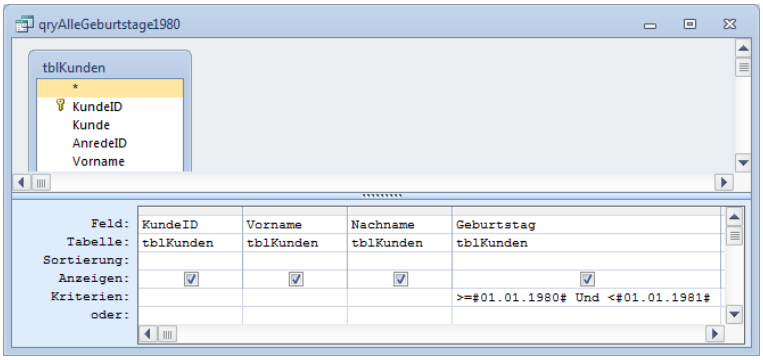



**Bild 7:** Ermittlung aller Geburtstage des Jahres 1980

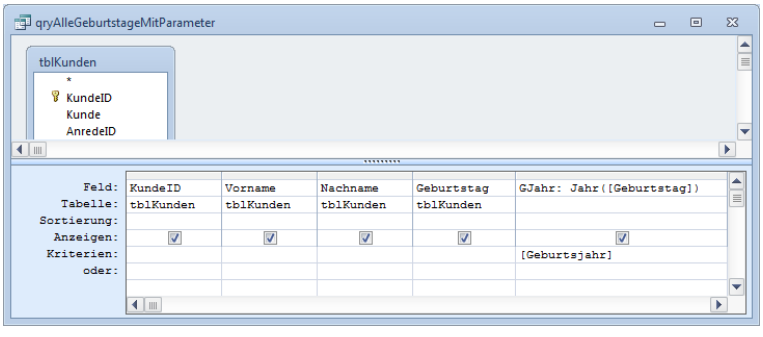



**Bild 8:** Geburtstage mit einfachem Kriterium ermitteln

# 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.

## 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
```

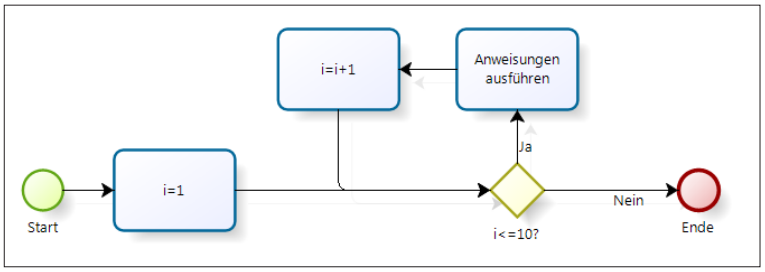



**Bild 11:** Flussdiagramm für eine **For...Next**-Schleife

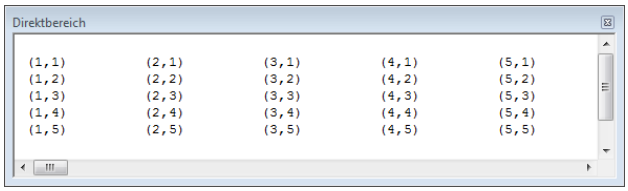



**Bild 1:** Ausgabe zweier verschachtelter **For...Next**-Schleifen

## 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
```

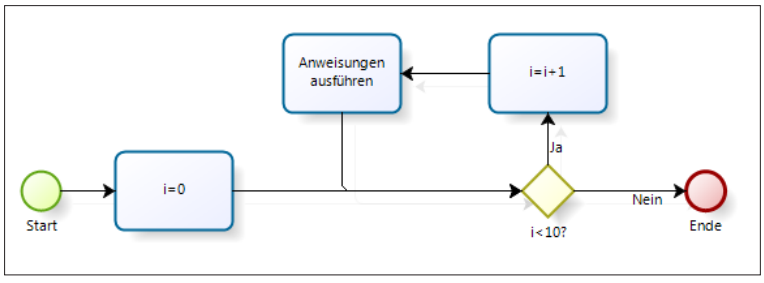



**Bild 2:** Flussdiagramm für eine **Do While...Loop**-Schleife

Access [basics] 5/2011
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Access [basics]-News

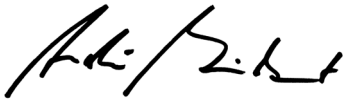
Mit der neuen Ausgabe von **Access [basics]** stellen wir die Beispieldateien von einer großen Datenbank auf eine Datenbank je Artikel um. Die Beispieldatenbank ist doch recht unübersichtlich geworden, eine einzelne Datei zu jedem Artikel scheint mir praktischer zu sein. Ich werde in der nächsten Zeit auch die bestehenden Datenbanken aufteilen, damit Sie die Beispiele zu den Artikeln schneller finden können. Im Artikeltext finden Sie gleich zu Beginn jeweils einen Hinweis auf den Dateinamen.

Außerdem wird ab dieser Ausgabe die Webseite aufgewertet: Sie können nun zunächst die neuen Artikel und später auch die bereits vorhandenen Artikel online im HTML-Format lesen. Dort finden Sie auch Links zur jeweiligen Beispieldatenbank. Start für die Online-Artikel ist der 24. Juni 2011.

Und schließlich habe ich einige kleine Änderungen am Design der Heftseiten vorgenommen: Der Titel der Artikel, der meist die Rubrik markierte (etwa **Tabellen entwerfen**) ist in die Kopfzeile gerutscht, der eigentliche Titel, der sonst im Untertitel steckte, wird nun als einziger Titel angezeigt. Auf den Seiten nach der ersten Seite eines jeden Artikels stehen Rubrik und Titel oben in der Kopfzeile. Verwirrend? Schauen Sie sich einfach an, wie es aussieht.

Und noch eine Kleinigkeit in eigener Sache: Mein neues Buch ist vor kurzem erschienen! Wem es mit **Access [basics]** nicht schnell genug geht, den lade ich herzlich ein, sich einmal auf der Webseite zum Buch umzusehen: **www.acciu.de/aeb2010**

Und nun: Viel Spaß beim Lesen der neuen Ausgabe!



André Minhorst

## In dieser Ausgabe:

**Werte zu Kombinationsfeldern hinzufügen** | Erfahren Sie, wie Sie neue Werte für Lookup-Tabellen direkt im Kombinationsfeld eingeben.

**Aktionsabfragen per VBA ausführen** | Lernen Sie die Anweisung zum Ausführen von Lösch-, Aktualisierungs- und weiteren Abfragetypen kennen.

**Daten per Kombinationsfeld auswählen und löschen** | Wählen Sie einen zu löschenden Datensatz aus und löschen Sie diesen per Mausklick.

**Meldungsfenster anzeigen und auswerten** | Verwenden Sie die MsgBox-Anweisung zum Anzeigen von Meldungen und zum Erfragen einfacher Informationen.

**Daten in der Endlosansicht von Formularen anzeigen** | Setzen Sie Endlosformular ein, um mehrere Datensätze gleichzeitig anzuzeigen und zu bearbeiten.

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst | Fachverlag für Softwareentwicklung | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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.

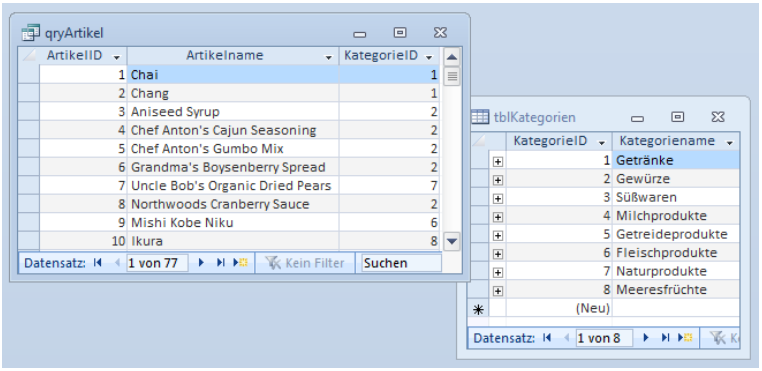



**Bild 1:** Artikel und Kategorien werden über das Fremdschlüsselfeld **KategorieID** verknüpft

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).

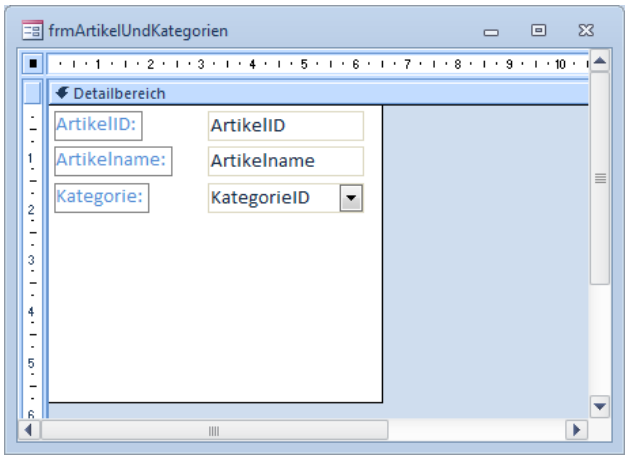



**Bild 2:** Das Beispielformular in der Entwurfsansicht

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

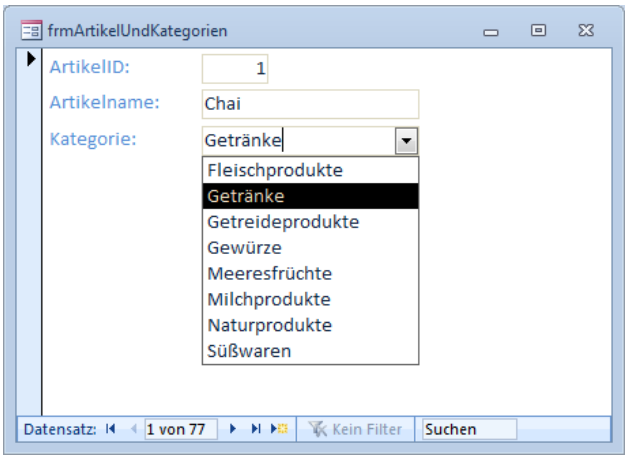



**Bild 3:** Kombinationsfeldeinträge auswählen

**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

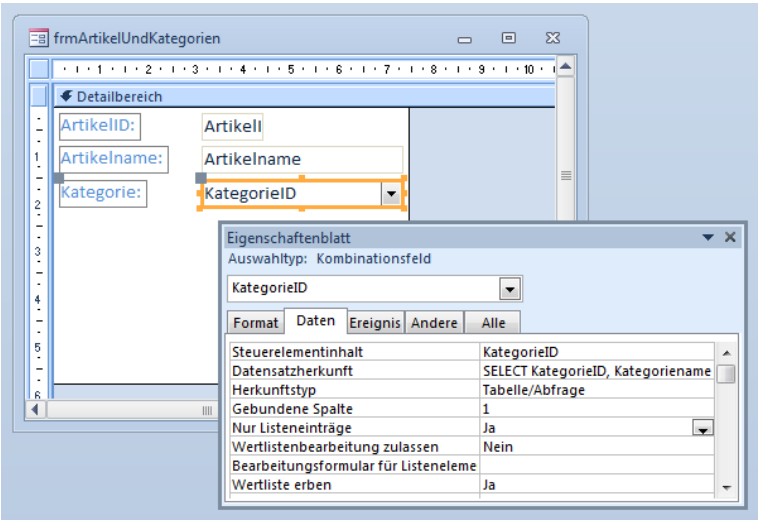



**Bild 4:** Die Eigenschaft **Nur Listeneinträge**

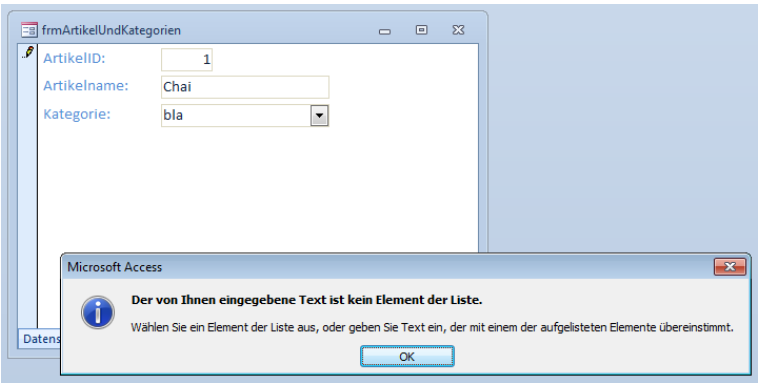



**Bild 5:** Diese Fehlermeldung erscheint, wenn die Eigenschaft **Nur Listeneinträge** den Wert **Ja** enthält.

# 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:

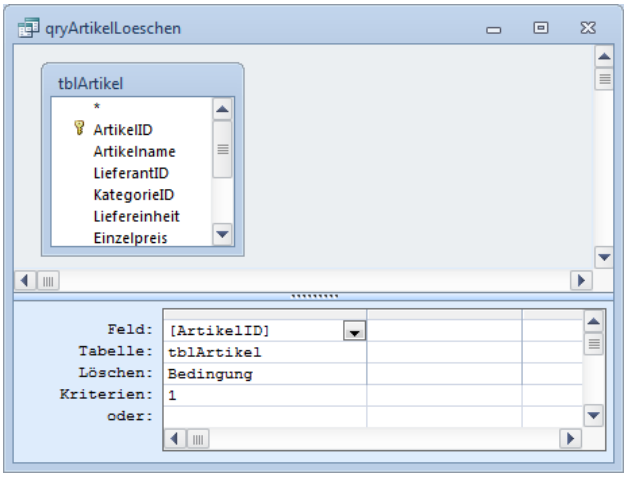



**Bild 1:** Entwurfsansicht einer Abfrage zum Löschen des Artikels mit dem Wert **1** im Feld **ArtikelID**.

```
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.

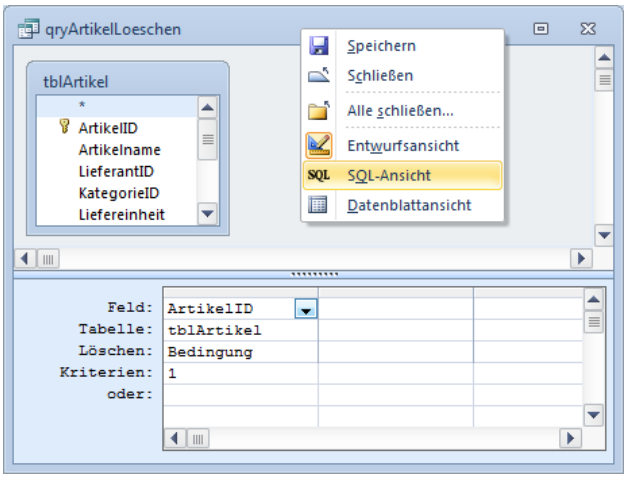



**Bild 2:** SQL-Ansicht einer Abfrage aktivieren

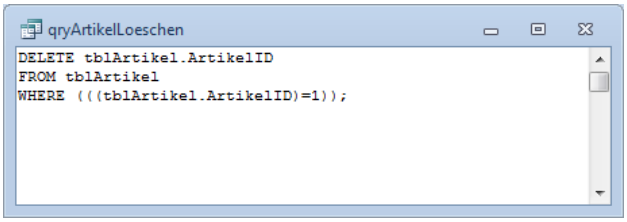



**Bild 3:** SQL-Ansicht einer Abfrage

# 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

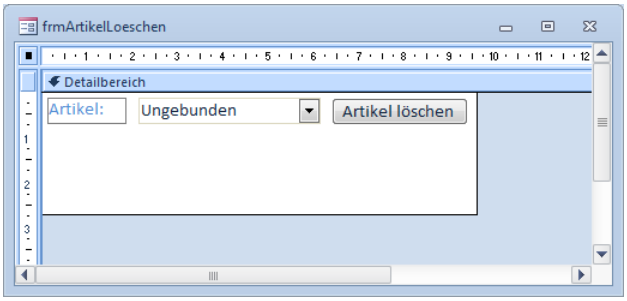



**Bild 1:** Entwurfsansicht des Formulars zum Löschen von Artikeln

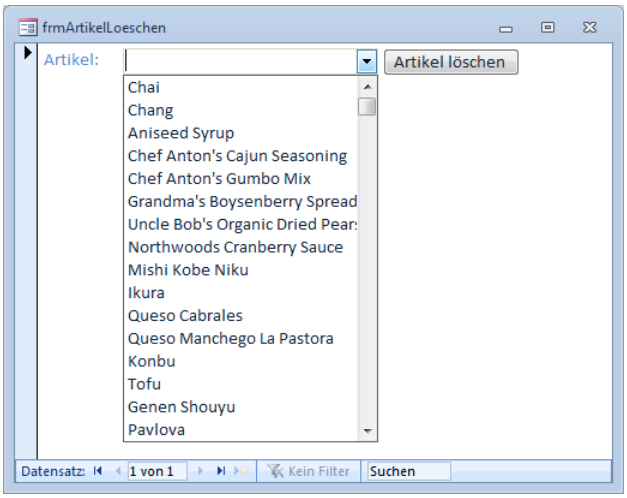



**Bild 2:** Auswahl eines Artikels per Kombinationsfeld

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
```

Diese Prozedur füllen wir nun mit einigen Anweisungen, die in **Aktionsabfragen per VBA ausführen** genauer erläutert werden:

```
Private Sub cmdArtikelLoeschen_Click()
    Dim db As DAO.Database
    Dim lngArtikelID As Long
    Dim strSQL As String
    Set db = CurrentDb
    lngArtikelID = Me!cboArtikelID
    strSQL = "DELETE FROM tblArtikel WHERE ↵
                    ArtikelID = " & lngArtikelID
    db.Execute strSQL
    Set db = Nothing
End Sub
```

Wenn wir davon ausgehen, dass der Benutzer einen Eintrag des Kombinationsfeldes ausgewählt hat, wird **lngArtikelID** mit dem Wert des Feldes **ArtikelID** des betreffenden Artikels gefüllt, also beispielsweise mit dem Wert **10**. Der SQL-Ausdruck sieht dann etwa so aus:

```
DELETE FROM tblArtikel WHERE ArtikelID = 10
```

Die folgende **Execute**-Anweisung führt diese Aktionsabfrage aus und löscht den Datensatz, soweit dieser vorhanden ist. Das sollte jedoch der Fall sein, da er soeben noch durch den Benutzer ausgewählt werden konnte.

Fertig – das Auswählen des zu löschenden Datensatzes und das anschließende Löschen funktionieren!

## Finetuning

Allerdings dürfen Sie niemals davon ausgehen, dass der Benutzer ein solches Formular nach Ihren Vorstellungen verwendet. Sie wissen, dass Sie einen Artikel auswählen müssen und diesen dann durch einen Klick auf die Schaltfläche löschen können. Benutzer tun jedoch gelegentlich unerwartete Dinge.

Zum Beispiel probieren sie Schaltflächen aus, ohne die vorher zu erledigenden Schritte durchzuführen. In diesem Fall löst das den Fehler aus Bild 3 aus.

Was ist hier passiert? Wenn Sie auf die Schaltfläche Debuggen klicken, erfahren Sie mehr. Es erscheint dann nämlich der VBA-Editor und die fehlerhafte Zeile wird farbig hinterlegt (siehe Bild 4). In diesem Fall handelt es sich um die folgende Zeile:

```
lngArtikelID = Me!cboArtikelID
```

Da der Benutzer keinen Eintrag aus dem Kombinationsfeld **cboArtikelID** ausgewählt hat, enthält dieses den Wert **Null**. Wir kommen in einem späteren Artikel auf die Funktion dieses Wertes zurück. Aktuell reicht die Information, dass **Null** kein Zahlenwert ist und deshalb nicht der **Long**-Variablen **lngArtikelID** zugeordnet werden kann. Genau das löst den hier aufgetretenen Fehler auf.

Also bauen Sie eine Prüfung ein, die sicherstellt, dass der Benutzer mit dem Kombinationsfeld einen Wert auswählt und erst dann auf die Schaltfläche **cmdArtikelLoeschen** klickt. Die folgenden Zeilen fügen Sie direkt vor der Zeile **Set db = CurrentDb** ein:

```
If IsNull(Me!cboArtikelID) Then
    MsgBox "Wählen Sie erst den zu löschenden ↵
                                    Artikel aus."
    Exit Sub
End If
```

Wenn der Benutzer noch keinen Datensatz ausgewählt hat, erscheint nun eine entsprechende Meldung und die Prozedur wird beendet.

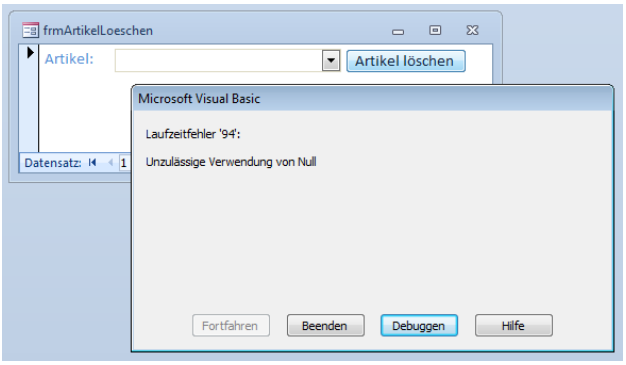



**Bild 3:** Wurde kein Datensatz ausgewählt, führt ein Klick auf die Schaltfläche zu diesem Fehler.

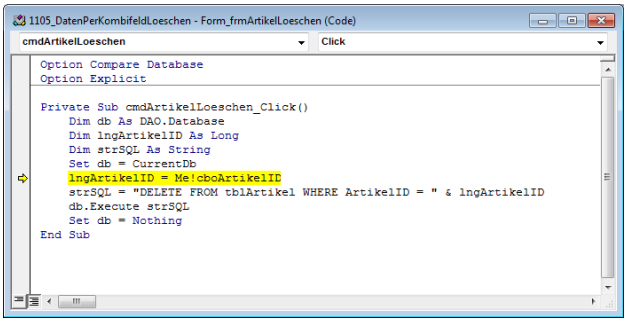



**Bild 4:** Nach einem Klick auf die **Debuggen**-Schaltfläche zeigt der VBA-Editor die fehlerhafte Zeile an.

# 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.

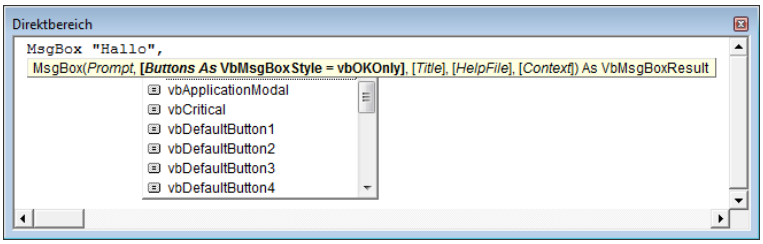



**Bild 2:** Der zweite Parameter offenbart mehr Optionen als sein Name besagt.

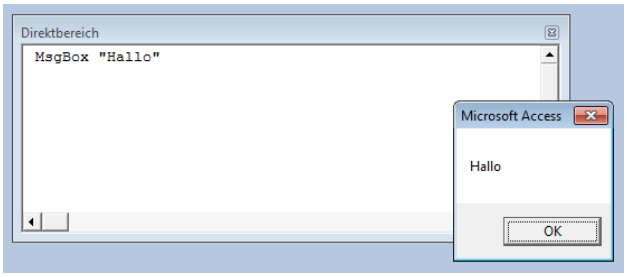



**Bild 1:** Die einfachste Variante eines Meldungsfensters

- **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.

- **vbOKOnly**: Zeigt nur die Schaltfläche **OK** an.
- **vbRetryCancel**: Zeigt die Schaltflächen **Wiederholen** und **Abbrechen** an.
- **vbYesNo**: Zeigt die Schaltflächen **Ja** und **Nein** an.
- **vbYesNoCancel**: Zeigt die Schaltflächen **Ja**, **Nein** und **Abbrechen** an.

Eine Übersicht der verschiedenen Kombinationen zeigt Bild 3. Mit einem der oben genannten Werte stellen Sie also die Schaltflächenkombination ein – wie Sie diese auswerten, beschreiben wir weiter unten.

Aber was fangen Sie mit den übrigen Werten für den Parameter **buttons** an? Ganz einfach: Mit diesem einen Parameter legen Sie gleich mehrere Eigenschaften fest – nicht nur die anzuzeigende Schaltflächenkombination!

Dazu tragen Sie einfach zwei oder mehr Werte durch den Operator **Or** voneinander getrennt für den Parameter **buttons** ein.

Als Standardwert verwendet die **MsgBox**-Anweisung übrigens den Wert **vbOKOnly**.

## Icon des Meldungsfensters

Dazu sehen wir uns als Nächstes vier weitere Werte an, die für verschiedene Icons im Meldungsfenster stehen:

- **vbCritical**: Warnhinweis
- **vbExclamation**: Ausrufezeichen auf gelbem Hintergrund
- **vbInformation**: Informations-Symbol
- **vbQuestion**: Fragezeichen-Symbol

Das Aussehen der vier Icons entnehmen Sie Bild 4.

Wenn Sie eine Schaltflächenkombination und ein bestimmtes Icon kombinieren möchten, verwenden Sie eine Anweisung wie die folgende:

```
MsgBox "OK und Abbrechen mit Fragezeichen-Icon",
vbOKCancel Or vbQuestion
```

Standardmäßig zeigt die **MsgBox**-Anweisung gar kein Icon an, Sie müssen also explizit einen der hier angegebenen Werte zur Eigenschaft **buttons** hinzufügen.

## Standardschaltfläche festlegen

Damit ist der Fundus der Werte für den Parameter **buttons** immer noch nicht erschöpft. Mit den folgenden vier Werten legen Sie fest, welche Schaltfläche als Standardschaltfläche verwendet wird:

- **vbDefaultButton1**
- **vbDefaultButton2**
- **vbDefaultButton3**
- **vbDefaultButton4**

Die Standardschaltfläche ist diejenige, die beim Anzeigen des Meldungsfensters aktiviert ist und etwa durch das Betätigen der Leertaste oder der Eingabetaste ausgelöst werden kann. Mit der **Tabulator**-Taste oder den Cursor-Tasten kann der Benutzer bei

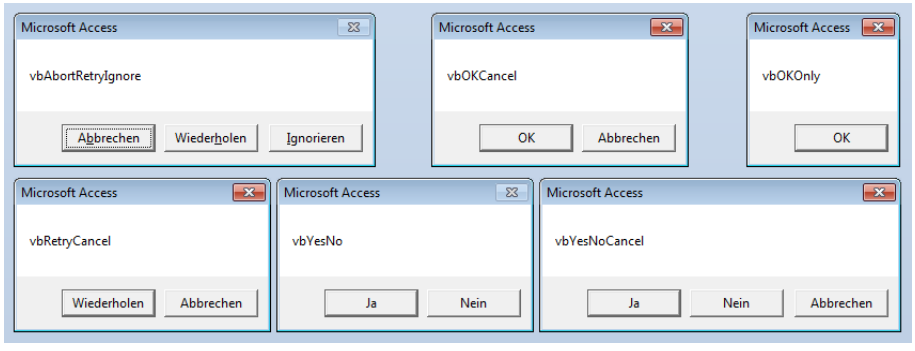



**Bild 3:** Alle möglichen Schaltflächen-Kombinationen der **MsgBox**-Anweisung

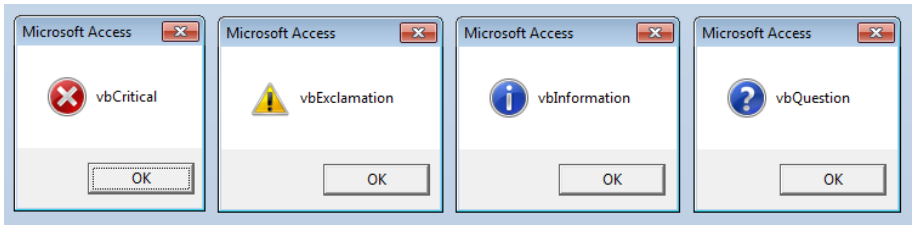



**Bild 4:** Die verschiedenen Icons im Meldungsfenster

# 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).

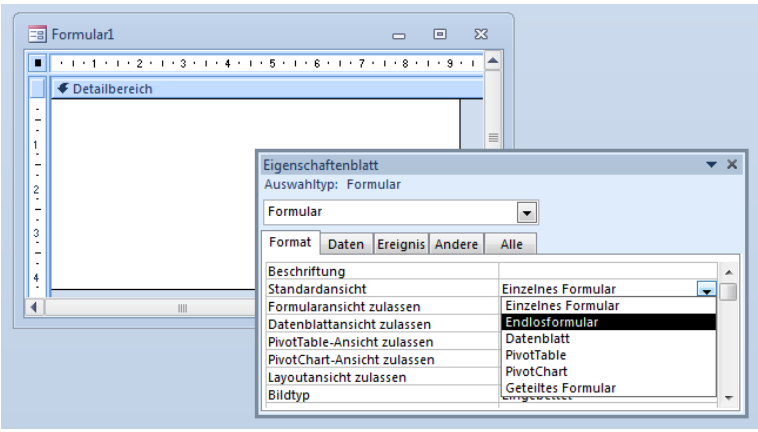



**Bild 1:** Die verschiedenen Icons im Meldungsfenster

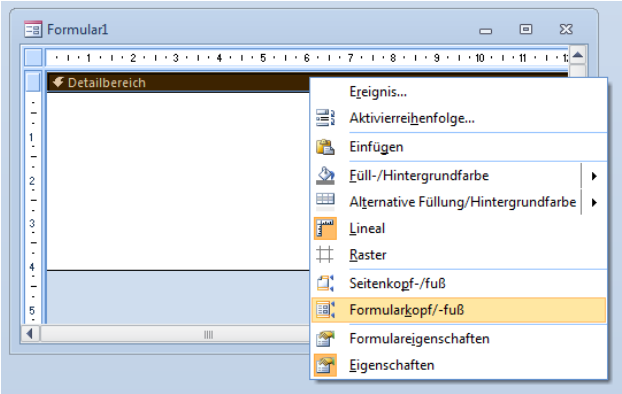



**Bild 2:** Einblenden von Formularkopf und -fuß

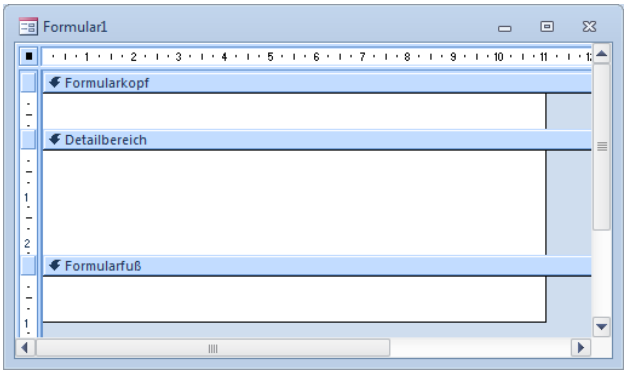



**Bild 3:** Formular mit Formularkopf und -fuß

- 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.

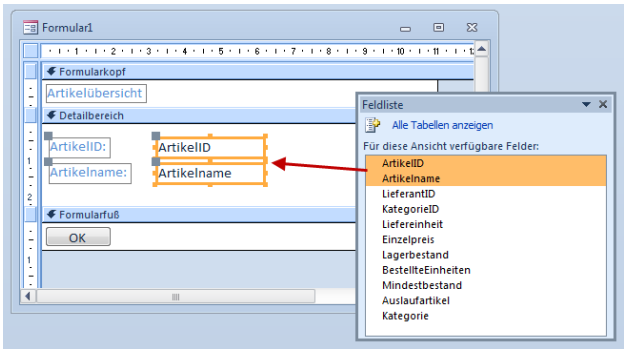



**Bild 4:** Hinzufügen der Felder zum Detailbereich der Entwurfsansicht

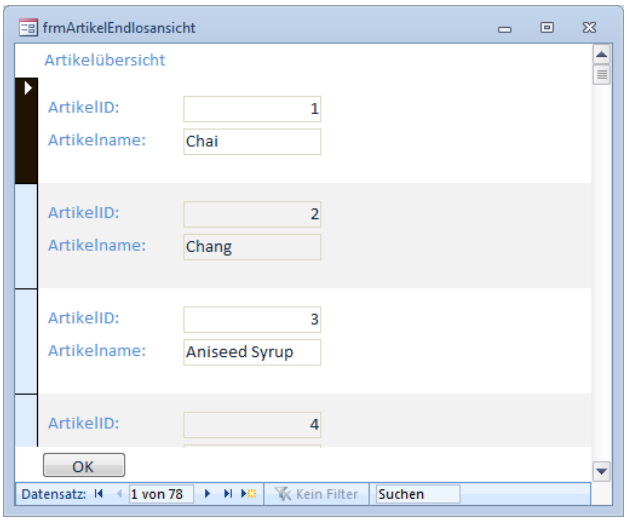



**Bild 5:** Ein Formular mit mehreren Datensätzen in der Endlosansicht

Access [basics] 6/2011
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Access [basics]-News

In dieser Ausgabe dreht sich alles um zwei Themen. Das erste ist der Export von Daten im Textformat, das zweite das Aufteilen von Datenbanken.

Weil Artikel sich am leichtesten schreiben und sich auch toll nachvollziehen lassen, wenn ihnen ein Beispiel aus der Praxis zugrunde liegt, gingen die ersten beiden Artikel dieser Ausgabe mir ganz leicht von der Hand. Es geht um den Export von Daten in Text-Dateien, und zwar im sogenannten CSV-Format. Dieses Format zeichnet sich dadurch aus, dass die Datensätze durch Zeilenumbrüche und die einzelnen Felder durch ein Zeichen wie etwa ein Komma oder ein Semikolon voneinander getrennt werden.

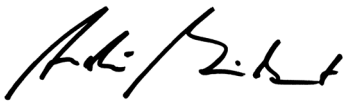
Genau dieses Format erwartet die Schnittstelle eines bekannten Versanddienstleisters. Und da bei mir gerade zum Zeitpunkt der Erstellung dieser Ausgabe eine größere Versandaktion anstand, schrieb ich auch gleich einen Artikel über das Thema. Wenn Sie erfahren möchten, wie Sie Ihre Adressdaten nach bestimmten Vorgaben exportieren, lesen Sie unter **CSV-Export für Adressdaten** weiter. Passend zu diesem Thema erfahren Sie unter dem Titel **Im- und Export-Spezifikationen definieren und verwenden**, wie Sie die Einstellungen solcher Exporte leicht speichern und dann mit einer einzigen Codezeile aufrufen können.

Der zweite Schwerpunkt dieser Ausgabe liegt im Aufteilen von Datenbanken. Gründe dafür gibt es genug: Entweder man möchte die Mehrbenutzerfähigkeiten von Access nutzen oder vielleicht auch nur die Wartbarkeit einer Datenbankanwendung verbessern. Beides gelingt, wenn Sie die Anwendung in Frontend und Backend aufteilen und vom Frontend aus über geeignete Verknüpfungen auf die im Backend gespeicherten Tabellen zugreifen.

Die Grundlagen zum Aufteilen und Verknüpfen liefert der Artikel **Datenbanken aufteilen**. Die Aufteilung einer Datenbank soll es in den meisten Fällen ermöglichen, mit mehreren Benutzern auf die Daten zuzugreifen. Was dabei zu beachten ist, erfahren Sie im Artikel **Datenbanken im Mehrbenutzerbetrieb**.

Und schließlich wird auch eine Backend-Datenbank gelegentlich verschoben. Access verfolgt den Standort von Backend-Datenbanken nicht automatisch, also müssen Sie selbst Hand anlegen. Alles Wichtige über das Neuverknüpfen aufgeteilter Datenbanken liefert der Artikel **Tabellenverknüpfungen pflegen**.

Und nun: Viel Spaß beim Lesen der neuen Ausgabe!



André Minhorst

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst | Fachverlag für Softwareentwicklung | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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.

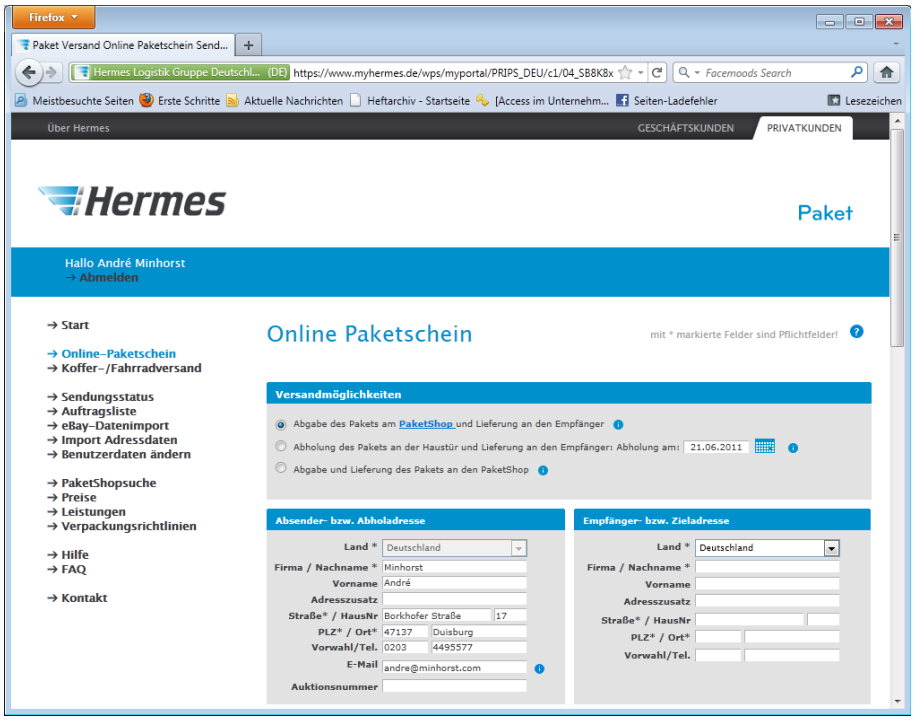



**Bild 1:** Beispiel für einen Online-Paketschein

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:

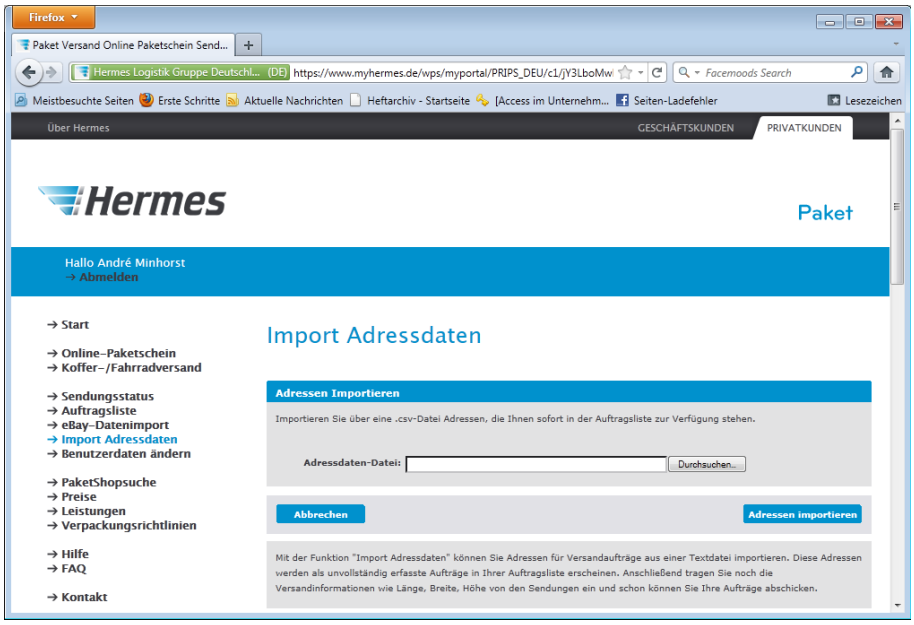



**Bild 2:** Schnittstelle zum Hochladen von CSV-Dateien mit Adressenlisten

- 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

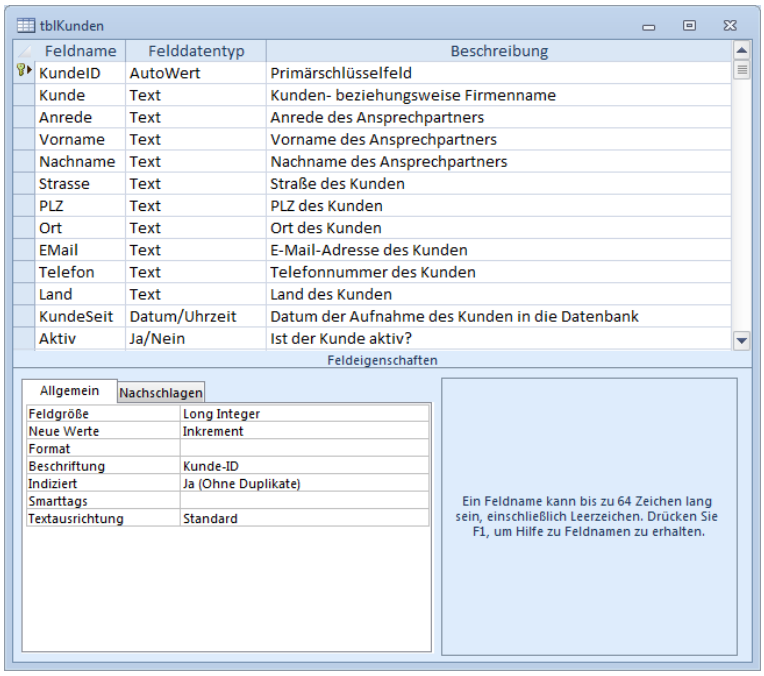



**Bild 3:** Schnittstelle zum Hochladen von CSV-Dateien mit Adressenlisten

# 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).

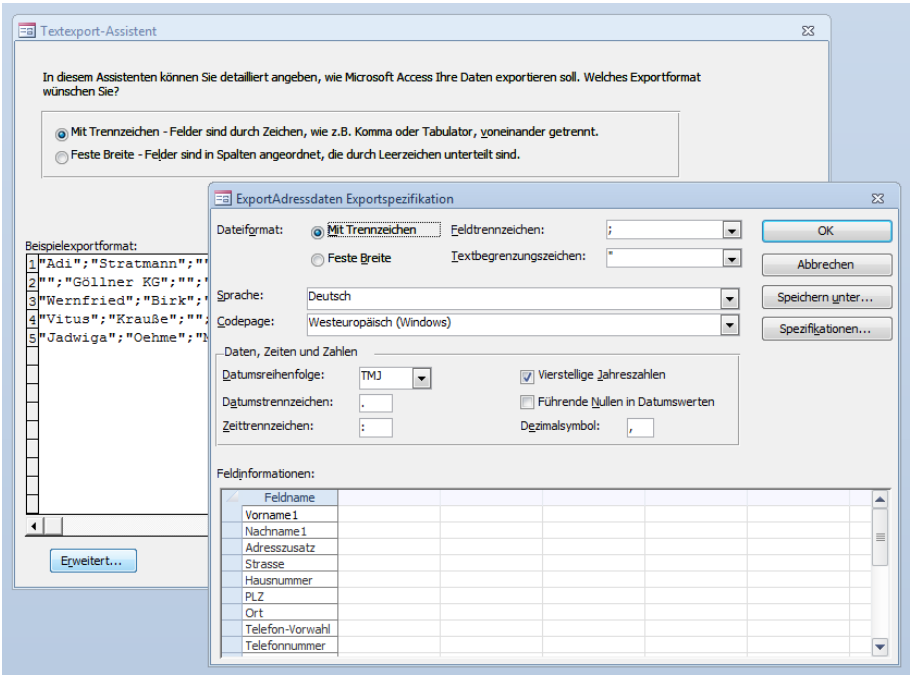



**Bild 1:** Der Dialog mit der Zusammenfassung der Einstellungen

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.

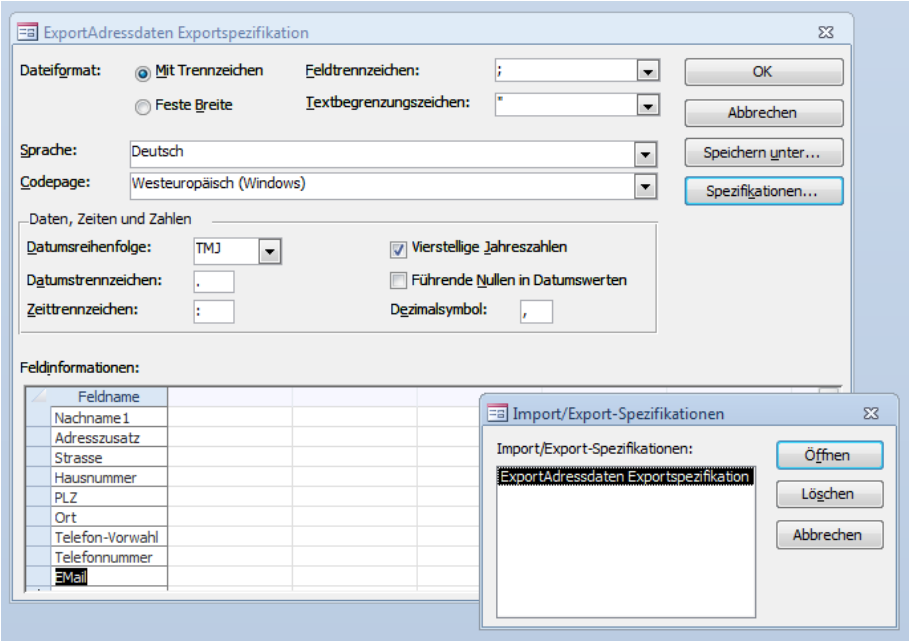



**Bild 2:** Öffnen oder Löschen bestehender Spezifikationen

## Import/Export per VBA starten

Am meisten nützen Spezifikationen denjenigen, die regelmäßig die gleichen Import/Export-Operationen durchführen. Wenn Sie einmal einen Import oder Export als Spezifikation gespeichert haben, können Sie die Operation mit einer einzigen VBA-Zeile aufrufen.

Dazu verwenden Sie die **TransferSpreadsheet** oder die **TransferText**-Anweisung des **DoCmd**-Objekts. Lassen Sie sich von Ausdrücken wie **DoCmd**-Objekt an dieser Stelle nicht durcheinanderbringen und nehmen Sie die nachfolgend dokumentierten Befehle vorerst einfach hin.

Die **TransferText**-Anweisung dient generell dem Import-, dem Export und dem Verknüpfen von Textdateien. Die **TransferSpreadsheet**-Anweisung ist prinzipiell eine Spezialisierung von **TransferText**, denn es bezieht sich auf Excel-Tabellen und enthält noch einige auf Excel ausgelegte Besonderheiten.

Wir schauen uns an dieser Stelle jedoch nur die **TransferText**-Anweisung an. Sie können diese zu Testzwecken im Direktfenster des VBA-Editors ausprobieren.

Das Direktfenster öffnen Sie von Access aus am schnellsten mit der Tastenkombination **Strg + G**.

Wenn Sie dort **DoCmd** gefolgt von einem Punkt eingeben, bietet der VBA-Editor gleich alle möglichen Anweisungen dieses Objekts an. Das Eintippen der ersten Buchstaben der gewünschten Anweisung führt dann schnell zum Eintrag **TransferText** (siehe Bild 3).

Wenn Sie diese Anweisung eingegeben haben und noch ein Leerzeichen hinterher schicken, erscheint gleich die Parameterliste der **TransferText**-Anweisung (siehe Bild 4).

Die Anweisung verwendet die folgenden Parameter (es gibt noch zwei weitere Parameter, die aber in diesem Fall nicht zum Einsatz kommen):

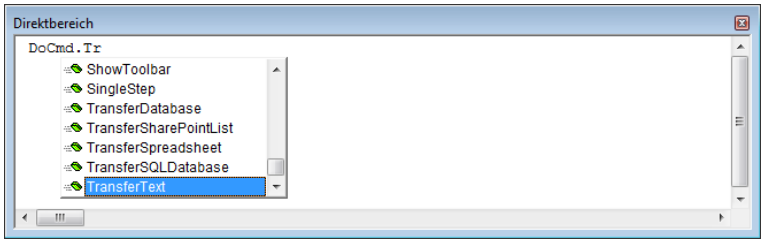



**Bild 3:** Der VBA-Editor unterstützt Sie bei der Eingabe von **DoCmd**-Anweisungen.

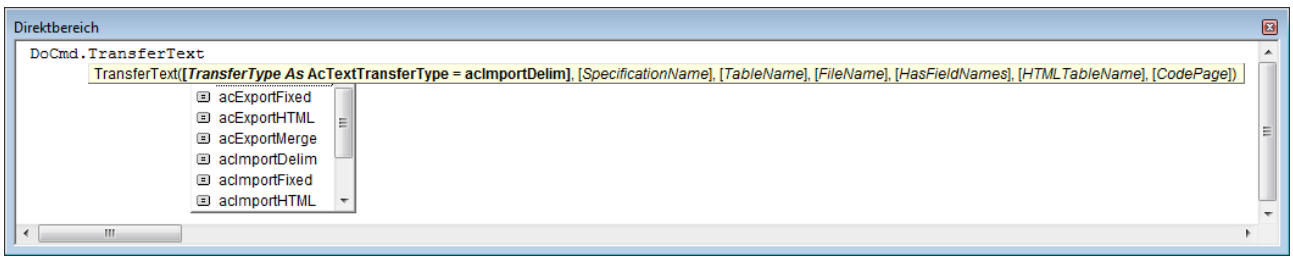



**Bild 4:** Auch beim Festlegen der Parameter hilft der VBA-Editor mit IntelliSense weiter.

# 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).

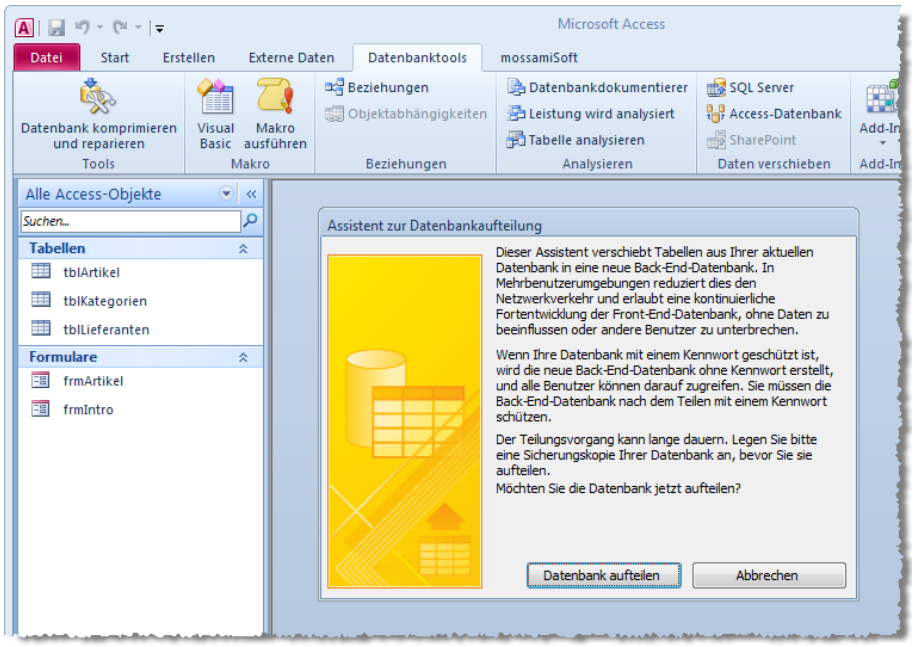



**Bild 1:** Start des Assistenten zur Datenbankaufteilung

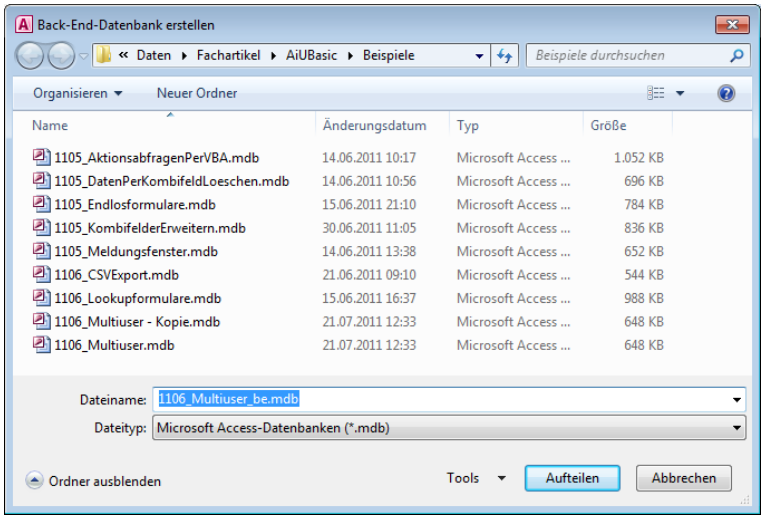



**Bild 2:** Festlegen des Namens der neuen Backend-Datenbank

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:

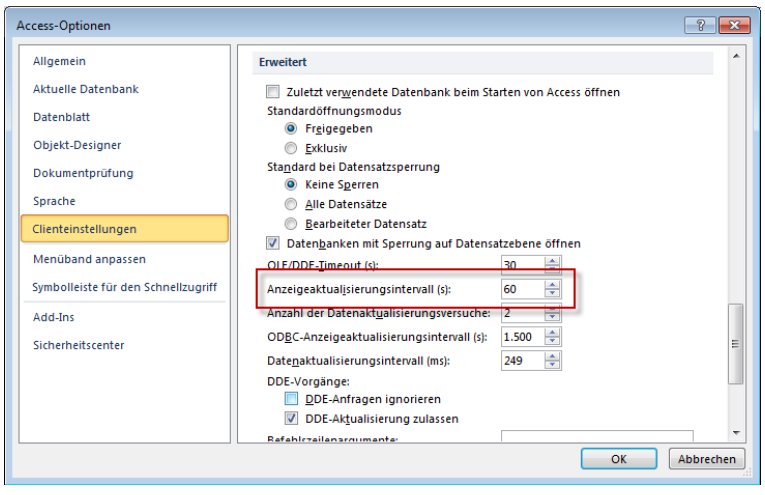



**Bild 1:** Einstellen des Aktualisierungsintervalls

**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.

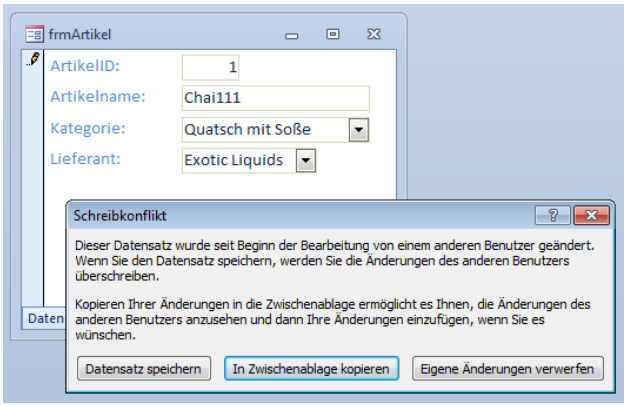



**Bild 2:** Meldung eines Schreibkonflikts

Diese Meldung verlangt vom Benutzer eine Entscheidung: Soll er seine Version des Datensatzes speichern und somit gegebenenfalls die Änderungen des anderen Benutzers überschreiben? Oder soll er den aktuellen Stand des durch ihn geänderten Datensatzes in die Zwischenablage kopieren, um zu prüfen, wie die Änderungen des anderen Benutzers aussehen? Oder verwirft er seine eigenen Änderungen zugunsten der Änderungen des anderen Benutzers?

Je weniger Datensätze es gibt und je mehr Benutzer diese bearbeiten, desto höher ist das Risiko eines Schreibkonflikts.

Im Idealfall bearbeitet jeder Mitarbeiter seinen eigenen Kreis von Daten, beispielsweise Bestellungen für bestimmte Kunden. Dann ist das Auftreten eines Schreibkonfliktes sehr unwahrscheinlich.

Das Zulassen von Änderungen von Datensätzen, die sich bereits in einem geänderten Zustand befinden, nennt sich übrigens **Optimistisches Sperren**. Das Gegenteil dazu erläutern die folgenden Abschnitte.

## Pessimistisches Sperren

Sie können Schreibkonflikte jedoch auch auf andere Weise unterbinden. Dazu verwenden Sie eine weitere Option namens **Standard bei Datensatzsperrung**, die sich im Optionen-Dialog der verschiedenen Access-Versionen in unmittelbarer Nähe der soeben vorgestellten Option zum Einstellen der Aktualisierungsgeschwindigkeit befindet (siehe Bild 3).

Die Standardeinstellung lautet **Keine Sperren** und wirkt sich durch nachträgliche Schreibkonflikte bei gleichzeitiger Änderung des gleichen Datensatzes aus. Wenn Sie hier die Einstellung **Alle Datensätze** wählen, wird die komplette Tabelle beim Bearbeiten eines Datensatzes gesperrt, bei Bearbeiteter Datensatz nur der aktuelle Datensatz.

Aber Achtung: Die Optionen wirken sich nur auf das Erstellen neuer Formular aus! Wenn Sie also ein bestehendes Formular verwenden und die Option **Standard bei Datensatzsperrung** wählen, ändert sich die Sperrungsstrategie für Datensätze in bestehenden Formularen nicht.

Sie müssen dann zunächst das betroffene Formular in der Entwurfsansicht öffnen und die Option **Daten|Datensätze sperren** wie in Bild 4 einstellen.

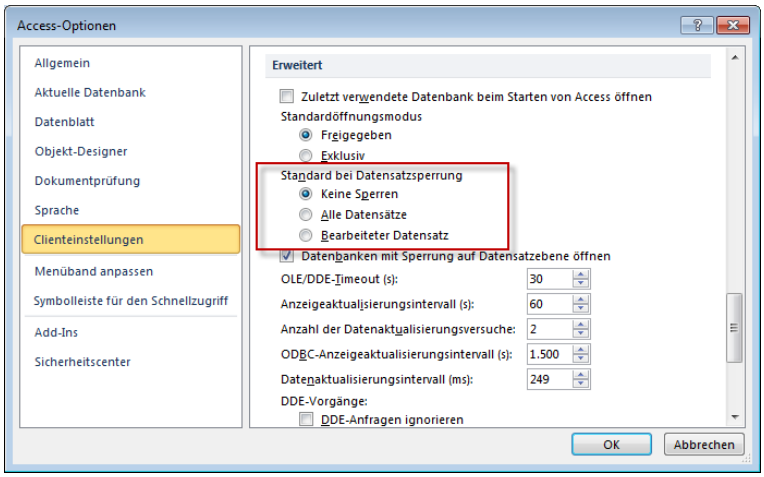



**Bild 3:** Arten der Sperrung von Datensätzen

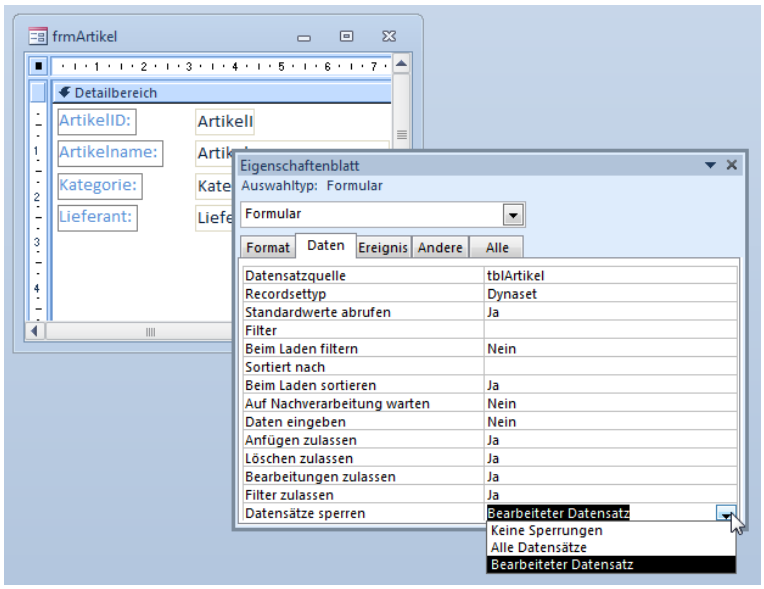



**Bild 4:** Sperrungsoptionen im Formularentwurf

# 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**.

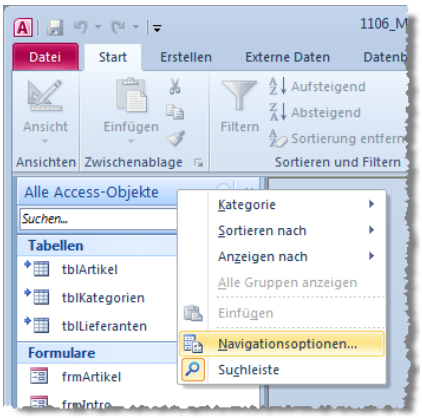



**Bild 1:** Aufrufen der Navigationsoptionen unter Access 2007 und jünger

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.

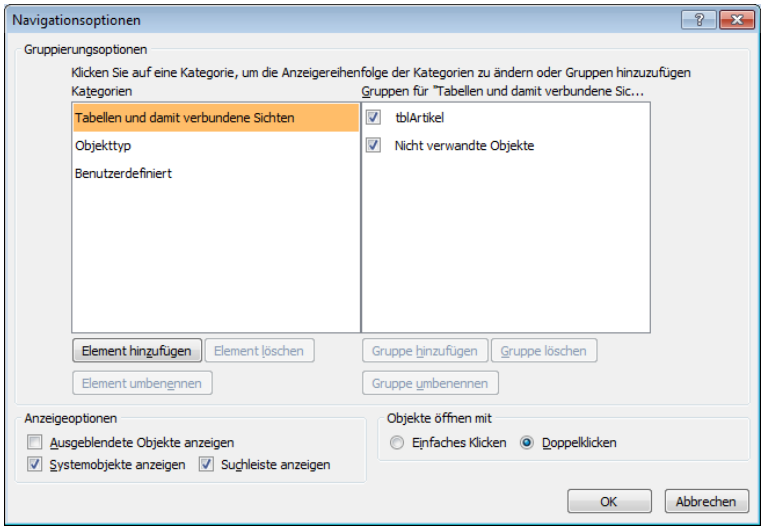



**Bild 2:** Aktivieren der Anzeige der Systemobjekte unter Access 2007 und jünger

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:

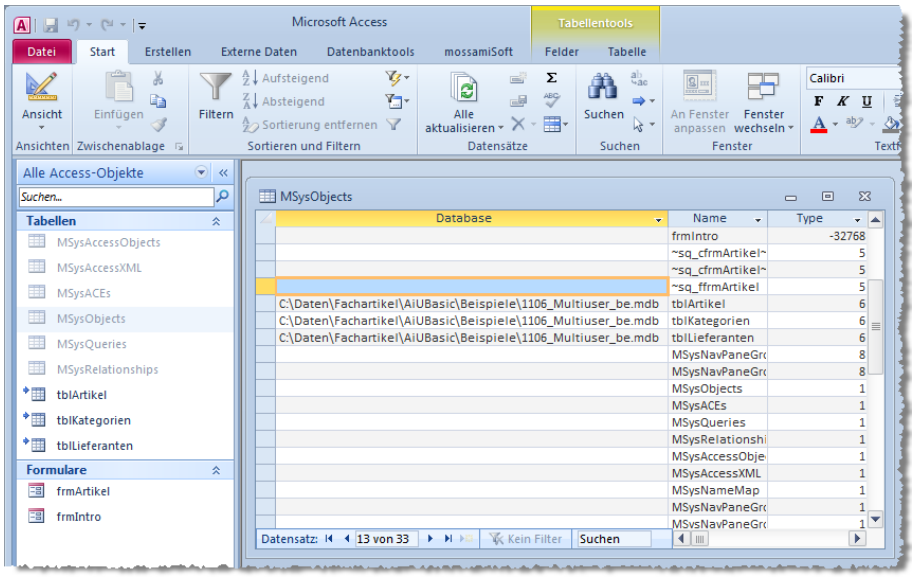



**Bild 3:** Die Tabelle **MSysObjects** zeigt den Pfad der verknüpften Tabellen an.

Access [basics] 7/2011
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Editorial

Kennen Sie das auch? Es ist noch ein überschaubarer Zeitraum bis zum Urlaub – sagen wir zehn Tage. Die sind natürlich noch komplett mit Arbeit zugepackt.

Kofferpacken? Machen wir zwei Stunden vor der Abreise, wie immer. Dann geschieht das Unerwartete: Die Deadline für ein Projekt wird vom Kunden um einen Monat nach hinten geschoben. Genug Zeit, das Projekt nach dem Urlaub zu vollenden!

Also nur noch die kleine Aufgabe erledigen, die neben dem großen Projekt noch auf dem Plan bis zum Urlaub stand – zwei, vielleicht drei Tage Arbeit. Leicht zu erledigen in den verbleibenden zehn Tagen! Doch erstmal erholen wir uns schonmal vorab zwei Tage und gehen dann gestärkt in das kleine Restprojekt.

Oh, da kommt noch ein kleines, dringendes Projekt eines anderen Kunden dazwischen – kein Thema, die zwei Tage sind locker drin.

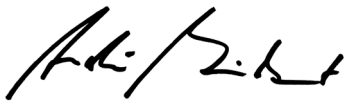
Es werden nicht zwei, sondern vier Tage, weil man schon so schön in Urlaubsstimmung ist und zwischendurch noch zwei Ausflüge mit den Kindern eingeschoben werden.

Uuups – es bleiben nur noch vier Tage, um den Rest des dringend abzugebenden Projekts zu erledigen – und das alles in der hektischen Betriebsamkeit des Kofferpackens (nicht meine Koffer, die vom Rest der Familie ...)!

Dreieinhalb Tage später, 23:30 Uhr, am nächsten Morgen geht der Flieger. Nur noch schnell das Editorial schreiben, das PDF erstellen, die Beispieldatenbanken zusammenstellen, alles hochladen und noch die Rundmail an die Abonnenten rausschicken. Fast geschafft!

Ungefähr so haben sich die vergangenen zehn Tage abgespielt – und es hätte so schön sein können. Aber es scheint in der Natur der meisten Menschen zu liegen (und besonders von Access-Programmierern), dass Projekte um so mehr Zeit fressen, je länger man dafür Zeit hat. Nach dem Urlaub werde ich dieses Problem mal analysieren und versuchen, ein Projekt, dessen Abgabetermin noch weit weg ist, zügig fertigzustellen – das Ergebnis dieses Versuchs stelle ich dann hier vor. Wenn es klappt.

Bis dahin haben Sie hoffentlich viel Spaß mit der vorliegenden Ausgabe von **Access [basics]**. In diesem Sinne, bis demnächst – in alter Frische!



André Minhorst

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst | Fachverlag für Softwareentwicklung | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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.

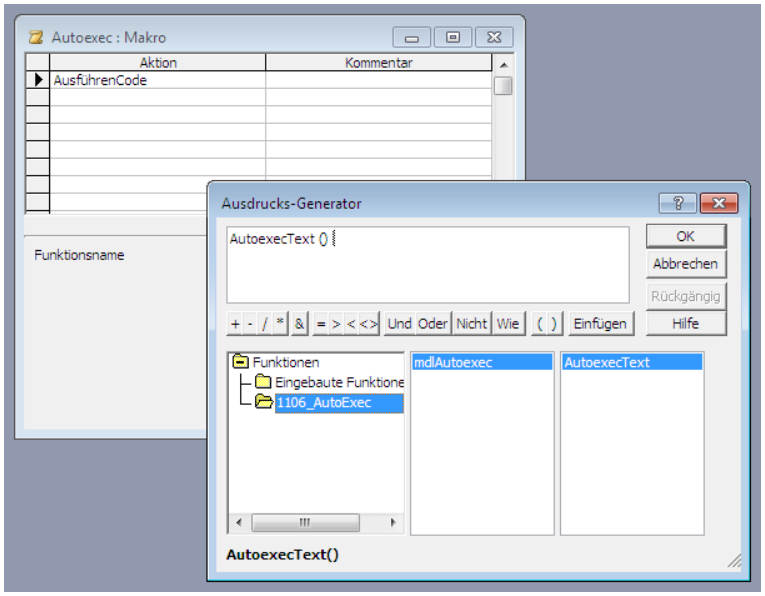



**Bild 1:** Anlegen des **Autoexec**-Makros unter Access 2007 und älter

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.

## 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

der aktuellen Datenbank, in diesem Fall **1106_Autoexec**. Es erscheint eine Liste der Standardmodule und für das markierte Standardmodul eine weitere Liste der enthaltenen öffentlichen Function-Routinen (siehe Bild 1).

Wählen Sie die gewünschte Funktion aus und schließen Sie den Ausdrucks-Generator wieder. Speichern Sie dann das Makro unter dem Namen **Autoexec**.

## Autoexec-Makro unter Access 2010 anlegen

Der Makro-Editor von Access 2010 sieht ein wenig anders aus als der älterer Access-Versionen. Eine neue Makroaktion fügen Sie hier aber ebenfalls über die Auswahlliste hinzu – wählen Sie auch hier den Eintrag **AusführenCode** aus.

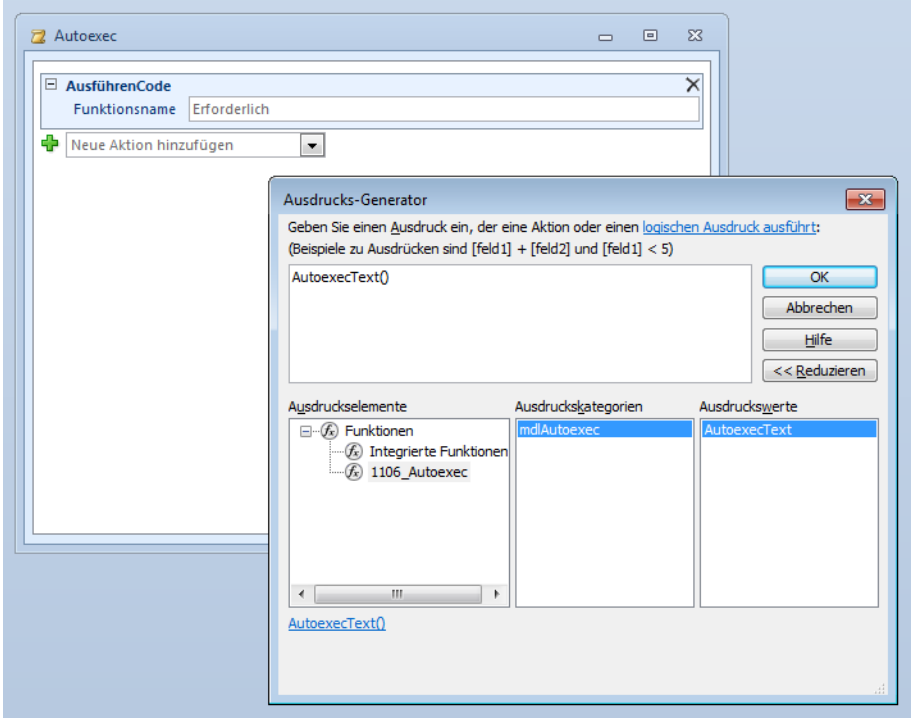



**Bild 2:** Anlegen des **Autoexec**-Makros unter Access 2010

Ein Nachteil ist, dass es hier keine Schaltfläche zum Öffnen des Ausdrucks-Generators gibt. Zum Glück funktioniert dieser trotzdem: Sie müssen einfach nur die Einfügemarke im Parameterfeld **Funktionsname** positionieren und dann die Tastenkombination **Strg + F2** betätigen.

Im Ausdrucks-Generator stellen Sie dann wie in Bild 2 den gewünschten Ausdruck zusammen.

Anschließend speichern Sie das Makro unter dem Namen **Autoexec**.

## Autoexec-Makro testen

Sie können das Makro nun testen, indem Sie doppelt auf den entsprechenden Eintrag im Datenbankfenster beziehungsweise im Navigationsbereich klicken. Es erscheint das Meldungsfenster, dass durch die **MsgBox**-Funktion der durch das Makro aufgerufenen VBA-Funktion **AutoexecText** erzeugt wird.

Die eigentliche Aufgabe dieses Makros ist es jedoch, die angegebene Funktion beim Öffnen der Datenbankdatei aufzurufen. Also schließen Sie die Datenbankdatei und öffnen diese erneut. Wenn Sie alles richtig gemacht haben, sollte das Meldungsfenster auch in diesem Fall erscheinen.

## Weitere Möglichkeiten

Sie können mit dem Autoexec-Makro nicht nur eine VBA-Funktion aufrufen, sondern noch viele weitere Dinge erledigen. Einen Überblick der Möglichkeiten erhalten Sie, wenn Sie sich die Liste der Makroaktionen im Makro-Editor ansehen.

Dort gibt es beispielsweise auch einen Befehl namens **ÖffneFormular**. Damit können Sie gleich beim Öffnen der Datenbank ein Formular öffnen.

Dies ist jedoch die einzige Aktion, die Sie auch auf andere Weise automatisieren können. Dazu geben Sie einfach den Namen des zu öffnenden Formulars für die Eigenschaft **Formular/Seite anzeigen** in den Start-Optionen von Access 2003 und älter (Menü-Eintrag **Extras|Start**) beziehungsweise in den Access-Optionen von Access 2007 und jünger unter **Aktuelle Datenbank|Formular anzeigen** an.

# 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

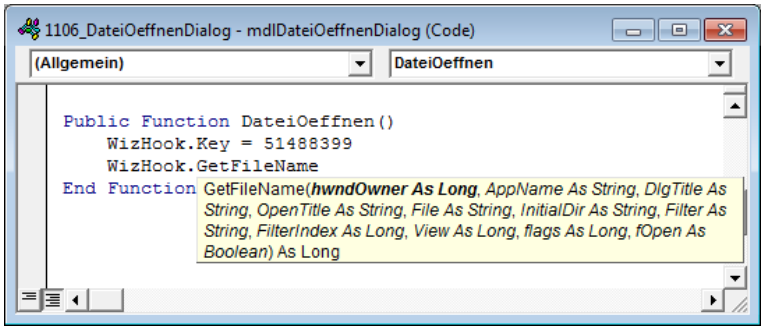



**Bild 1:** Parameter der Funktion **GetFileName**

- **Flags**: Zahlenwerte, die verschiedene weitere Optionen aktivieren

- **fOpen**: Gibt an, ob der aktuell ausgewählte Ordner beim Klick auf die Öffnen-Schaltfläche geöffnet wird oder nicht

## Einfache Dateiauswahl

Eigentlich könnte man diesen Befehl mit einer einzigen Zeile aufrufen, aber irgendwie müssen Sie ja die ausgewählte(n) Datei(en) auslesen, und diese landen im Parameter **File**. Also müssen Sie zumindest hierfür eine Variable deklarieren und als Parameter übergeben. Anschließend können Sie den Inhalt dieser Variablen auslesen. Schauen wir uns das für einen einfachen Fall an:

```
Public Function DateiAuswaehlen() As String
    Dim strFile As String
    WizHook.Key = 51488399
    Call WizHook.GetFileName(7
                        Application.hWndAccessApp, 7
                "Microsoft Access", "Datei öffnen", 7
            "Öffnen", strFile, CurrentProject.Path, 7
              "Access-Datenbanken (*.mdb,*.accdb)", 7
                                    0, 0, 0, False)
    Debug.Print strFile
End Function
```

Diese Routine legt zunächst die Variable **strFile** zum Speichern der ausgewählten Datei an. Dann aktiviert sie die **Wizhook**-Klasse durch Zuweisen der **Key**-Eigenschaft. Schließlich folgt der eigentliche Aufruf des Dateidialogs, der den Dialog aus Bild 2 hervorruft.

Die ersten beiden Parameter können Sie immer wie hier angegeben verwenden. Der dritte gibt den anzuzeigenden Titel des Dialogs an, der vierte die Bezeichnung der Schaltfläche zum Abschließen der Auswahl.

Der fünfte Parameter erwartet eine Variable, die beim Beenden der Auswahl mit Pfad und Dateiname der selektierten Datei gefüllt wird und nachher ausgewertet werden kann. Der sechste Parameter übergibt das Verzeichnis, in dem die Auswahl beginnen soll – hier das mit **CurrentProject.Path** ermittelte Verzeichnis der aktuellen Datenbankdatei.

Danach folgt die Angabe der Filter, wobei wir hier nur das Anzeigen von Access-Datenbankdateien erlauben. Die folgende **0** gibt an, dass der erste Eintrag der Filterliste beim Öffnen des Dialogs markiert sein soll. Die folgenden beiden Parameter, die ebenfalls mit **0** gefüllt wurden, betrachten wir später. Der letzte Parameter ist für die reine Dateiauswahl irrelevant.

## Einfache Ordnerauswahl

Auch wenn die Funktion **GetFileName** heißt, können Sie damit dennoch auch Ordner auswählen. Dazu ändern Sie die folgenden Parameter:

- **DlgTitle**: Wird auf **Ordner auswählen** eingestellt.

- **OpenTitle**: Wird auf **Auswählen** eingestellt.

- **Flags**: Wird auf **32** eingestellt.

- **fOpen**: Wird auf **True** eingestellt.

- **Filter**: Kann eine leere Zeichenfolge enthalten, da der Parameter nicht berücksichtigt wird.

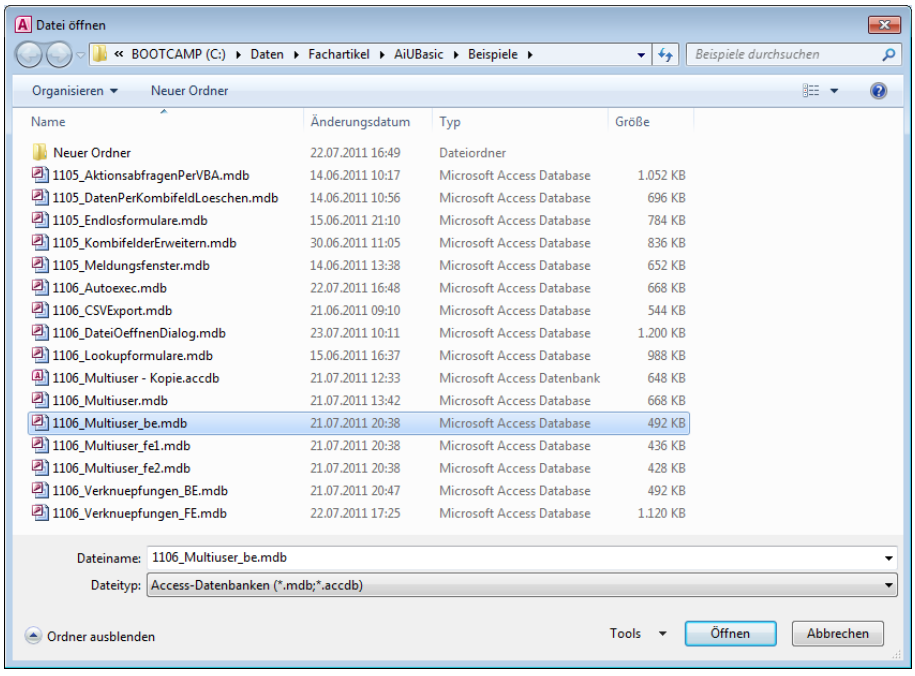



**Bild 2:** Beispiel für einen Dateidialog

# 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.

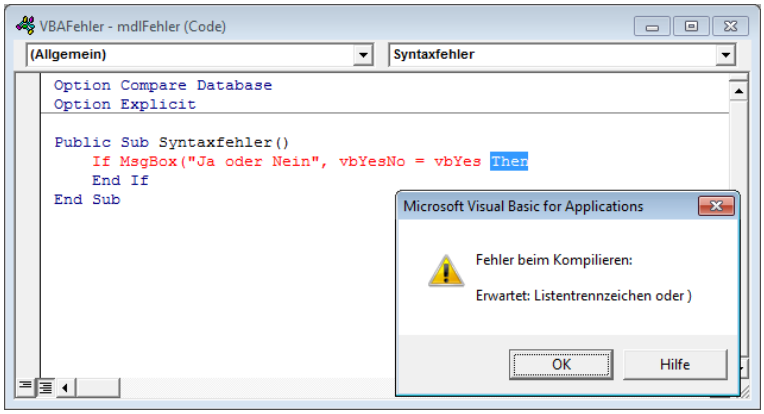



**Bild 1:** Syntaxfehler durch eine fehlende Klammer

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:

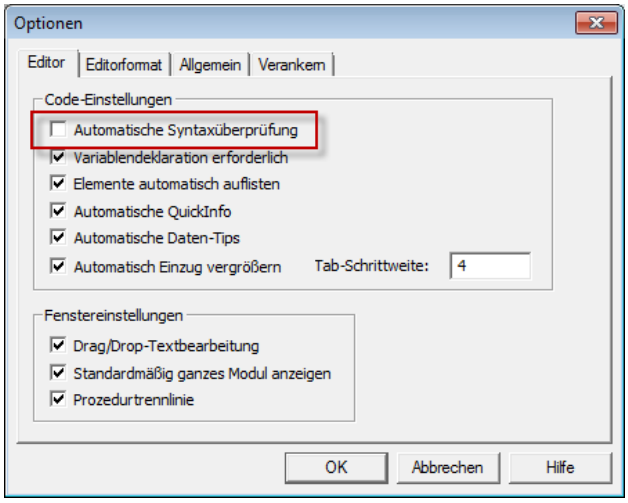



**Bild 2:** Deaktivieren der Anzeige von Syntaxfehlern

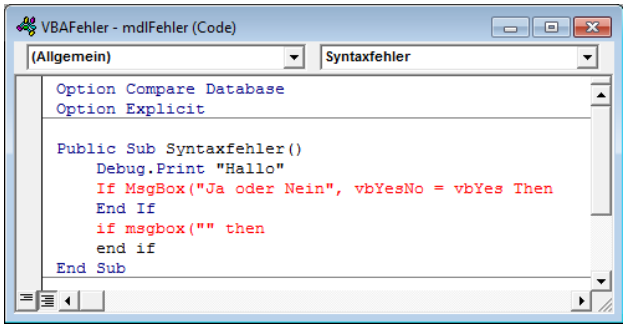



**Bild 3:** Ohne automatische Syntaxüberprüfung können auch mehrere fehlerhafte Zeilen farbig hinterlegt werden.

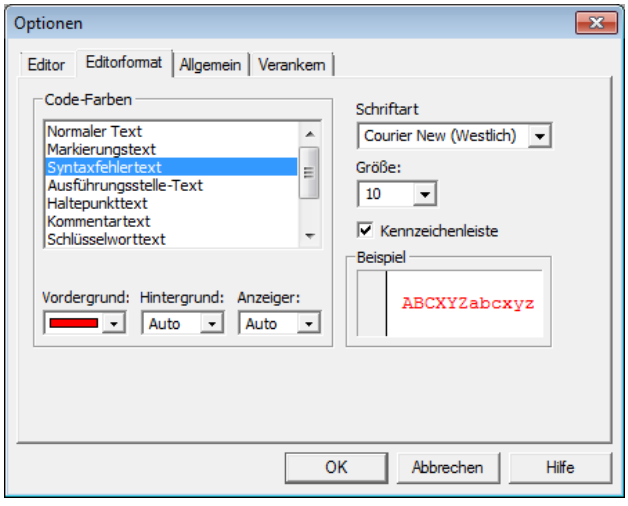



**Bild 4:** Einstellen der Textfarbe etwa für Zeilen mit Syntaxfehlern

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:

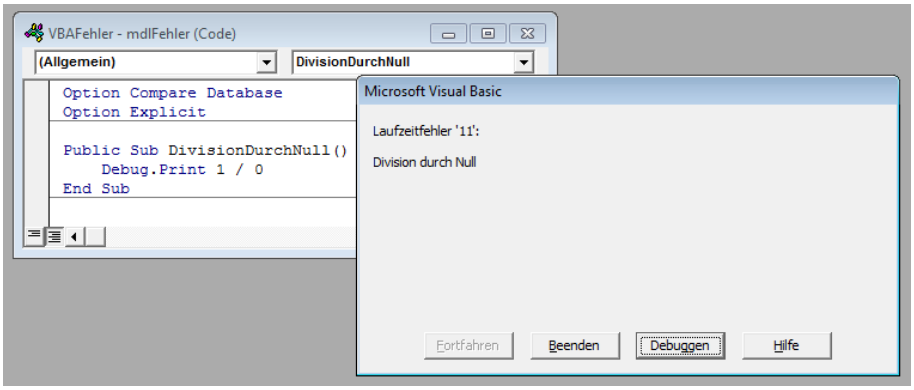



**Bild 1:** Fehlermeldung eines unbehandelten Fehlers

```
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
```

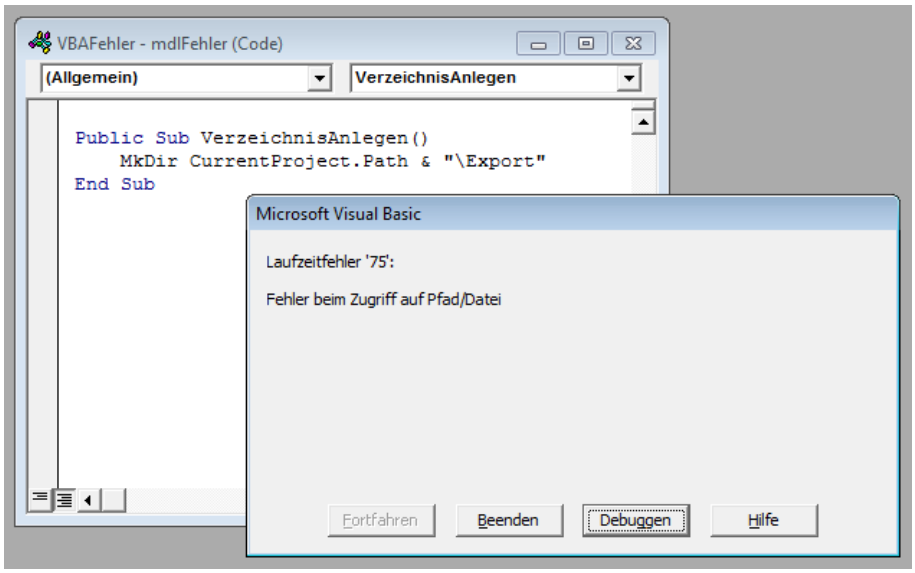



**Bild 2:** Ein bereits vorhandenes Verzeichnis lässt sich nicht nochmals anlegen.

Access [basics] 8/2011
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Editorial

In dieser Ausgabe schauen wir uns mal wieder ein sehr grundlegendes Thema an – die Bearbeitung von Daten in der Datenblattansicht. Durch Datensätze und Felder navigieren, Daten anlegen, bearbeiten und löschen, kopieren, ausschneiden und einfügen und noch ein wenig sortieren. Das Thema ist Ihnen zu billig? Das dachte ich auch. Aber man kann sich doch mit dem einen oder anderen Kniff selbst die Arbeit in der Datenblattansicht von Tabellen, Abfragen oder Formularen noch erleichtern. Lesen Sie mehr unter **Recordsets in der Datenblattansicht** ab Seite 2!

Danach geht es gleich weiter mit den Erleichterungen, nämlich mit den Formular- und Berichtsvorlagen. Wenn Sie neue Formulare anlegen, stellen Sie möglicherweise immer wieder Eigenschaften wie die Schriftgröße von Steuerelementen um. So etwas ist nicht nötig: Mit Formularvorlagen definieren Sie, mit welchen Attributen ein frisches Formular samt Steuerelementen ausgestattet werden soll.

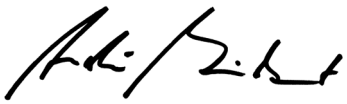
Das ist für alte Hasen natürlich auch ein alter Hut, aber für die gibt es auch noch was Neues: Wir zeigen Ihnen nämlich, wie Sie mehrere Formularvorlagen in der gleichen Datenbank definieren und diese per Mausklick zur aktuellen Formularvorlage machen. Ab Seite 6 erfahren Sie unter **Formular- und Berichtsvorlagen**, wie das funktioniert.

Wer Tabellen miteinander verknüpft, zeigt die Daten früher oder später auch in Haupt- und Unterformular an. Dabei kann es passieren, dass der Benutzer Daten im Unterformular anlegt, ohne dass das Hauptformular einen Datensatz enthält – der Datensatz des Unterformulars landet dann im Nirwana. Wie Sie dies verhindern, zeigt unser Artikel **Datensätze in Unterformularen anlegen** ab Seite 10.

Und schließlich setzen wir mit zwei weiteren Artikeln das Thema **Fehlerbehandlung** fort, das wir in der vorherigen Ausgabe gestartet haben. Unter **Fehlerbehandlung in der Praxis** ab Seite 13 erfahren Sie, wie Sie Fehler im täglichen Access-Leben behandeln können.

Eine besondere Art, Fehler zu behandeln, ist bei solchen Fehlern nötig, die durch die Verletzung von Restriktionen auf Tabellen- oder Feldebene auftreten. Dies gelingt gezielt nur in Formularen, und zwar mithilfe einer speziellen Ereignisprozedur. Alles, was Sie dazu wissen müssen, liefert der Artikel **Fehlerbehandlung in Formularen** ab Seite 18.

Und nun: Viel Spaß beim Lesen!



Ihr André Minhorst

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst | Fachverlag für Softwareentwicklung | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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.

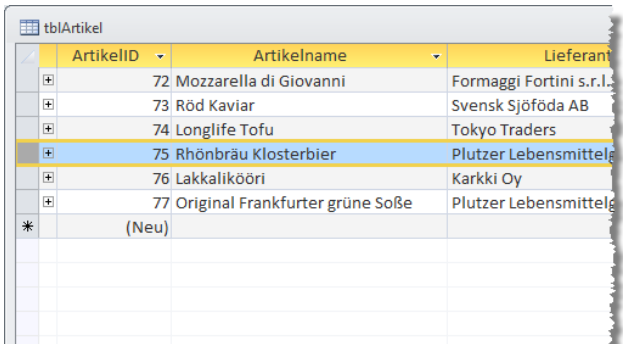



**Bild 1:** Ein Recordset in der Datenblattansicht einer Tabelle

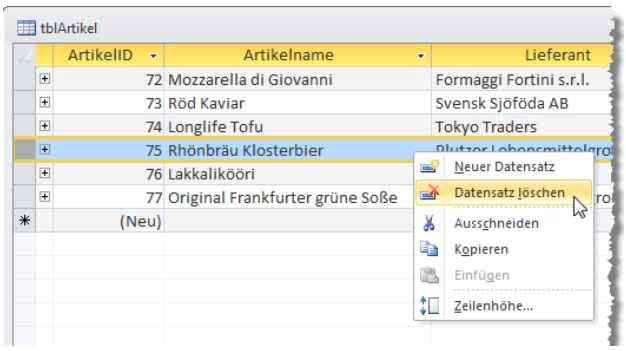



**Bild 2:** Löschen eines Records

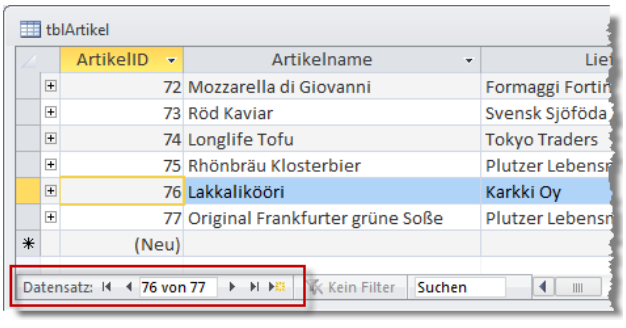



**Bild 3:** Navigationsleiste mit Datensatznummer

## 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.

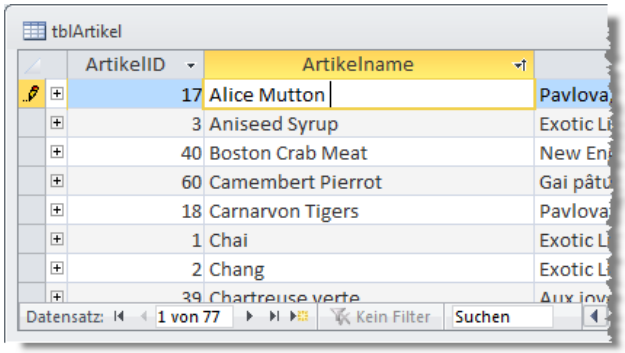



**Bild 4:** Bearbeiten eines Datensatzes

## 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.

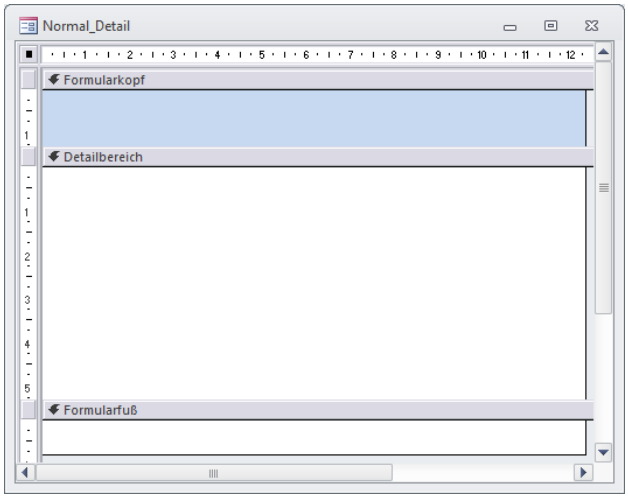



**Bild 1:** Einstellungen für eine Formularvorlage

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:

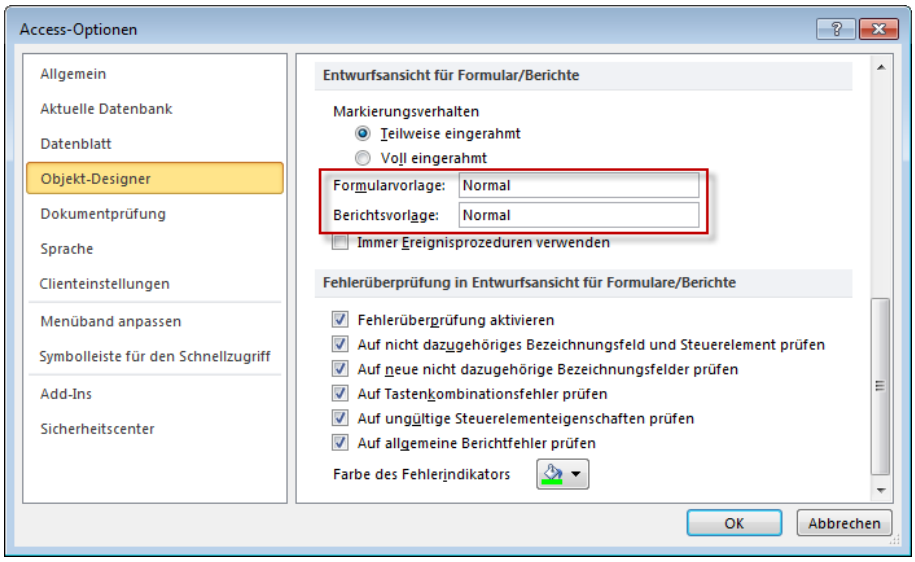



**Bild 2:** Einstellen der Namen des Standardformulars beziehungsweise -berichts

- 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

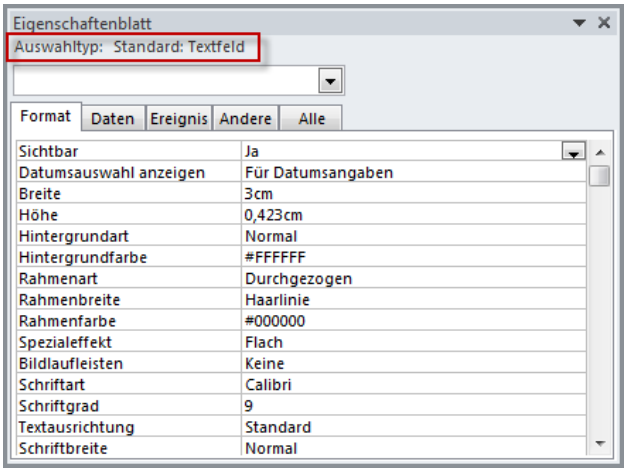



**Bild 3:** Einstellen von Standardeigenschaften für ein Steuerelement

# 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.

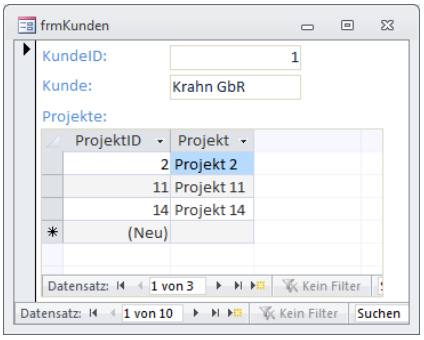



**Bild 1:** Normalerweise zeigt das Unterformular die verknüpften Datensätze an.

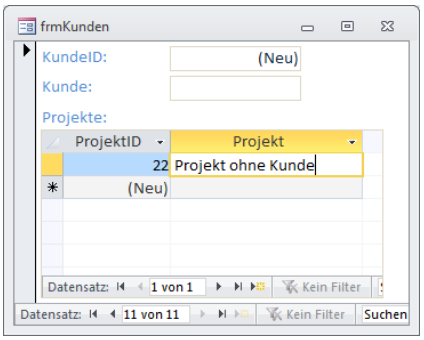



**Bild 2:** Was aber, wenn man ein Projekt eingibt, ohne dass das Hauptformular einen Kunden anzeigt?

Gerade Letzteres führt dazu, dass neue Datensätze im Unterformular, hier also Projekte, gleich dem im Hauptformular angezeigten Kunden zugeordnet werden können. Wenn das Hauptformular jedoch keinen Datensatz anzeigt, lautet der Wert des Primärschlüsselfeldes **NULL**. Wenn Sie nun einen neuen Datensatz im Unterformular anlegen, stellt Access den Wert des für die Eigenschaft **Verknüpfen von** angegebenen Feldes automatisch ebenfalls auf diesen Wert ein.

Dies führt dann dazu, dass die Tabelle **tblProjekte** Datensätze enthält, deren Feld **KundeID** keinen Wert enthält, der Wert also **NULL** ist (siehe Bild 3). Wenn der Benutzer dann im Hauptformular zu einem anderen Kunden-Datensatz und wieder zurück zum neuen, leeren Datensatz wechselt, könnte das Unterformular theoretisch die noch nicht mit einem Datensatz der Tabelle **tblKunden** verknüpften Projektdatensätze anzeigen – dies gelingt jedoch nicht und würde auch keinen Sinn machen.

tblProjekte

| ProjektID | Projekt | KundeID |
|---|---|---|
| 14 | Projekt 14 | Krahn GbR |
| 15 | Projekt 15 | Bolte GmbH |
| 16 | Projekt 16 | Krahl GbR |
| 17 | Projekt 17 | Peukert AG |
| 18 | Projekt 18 | Ziemann KG |
| 19 | Projekt 19 | Ziemann KG |
| 20 | Projekt 20 | Peukert AG |
| 22 | Projekt ohne Kunde | |
| 23 | Projekt ohne Kunde II | |
| (Neu) | | |

Datensatz: 1 von 22 | Kein Filter | Suchen

**Bild 3:** Projekte, die ohne Kunde angelegt wurden, enthalten im Feld **KundeID** den Wert **NULL**.

## Unverknüpfte Datensätze verhindern I

Wenn Sie dies verhindern möchten, haben Sie eine einfache Möglichkeit: Sie brauchen nur den Wert der Eigenschaft **Eingabe erforderlich** des Fremdschlüsselfeldes **KundeID** auf den Wert **Ja** einzustellen (siehe Bild 4 – die Beispielobjekte zu dieser Lösung heißen **tblKunden**, **tblProjekte**, **frmKunden** und **sfmProjekte**).

tblProjekte

| Feldname | Felddatentyp | Beschreibung |
|---|---|---|
| ProjektID | AutoWert | |
| Projekt | Text | |
| KundeID | Zahl | |

Feldeigenschaften

Allgemein | Nachschlagen

| | |
|---|---|
| Feldgröße | Long Integer |
| Format | |
| Dezimalstellenanzeige | Automatisch |
| Eingabeformat | |
| Beschriftung | |
| Standardwert | |
| Gültigkeitsregel | |
| Gültigkeitsmeldung | |
| Eingabe erforderlich | Ja |
| Indiziert | Ja (Duplikate möglich) |
| Smarttags | |
| Textausrichtung | Standard |

Ein Feldname kann bis zu 64 Zeichen lang sein, einschließlich Leerzeichen. Drücken Sie F1, um Hilfe zu Feldnamen zu erhalten.

**Bild 4:** Durch Einstellung dieser Eigenschaft verhindern Sie, dass Datensätze ohne Fremdschlüsselwert angelegt werden.

Danach tritt allerdings eine Fehlermeldung auf, sobald Sie einen Datensatz im Unterformular anlegen möchten, ohne dass das Hauptformular seinerseits bereits einen Datensatz anzeigt. Dazu reicht es, auch nur ein Zeichen in das Feld **Projekt** einzutippen (siehe Bild 5).

frmKunden

KundeID: (Neu)

Kunde:

Projekte:

| ProjektID | Projekt |
|---|---|
| 27 | |

Datensatz:

Datensatz:

Microsoft Access

Sie müssen einen Wert in das Feld 'tblProjekte.KundeID' eingeben.

OK | Hilfe

War diese Information hilfreich?

**Bild 5:** Diese Fehlermeldung erscheint, wenn Sie dennoch einen Datensatz ohne Fremdschlüsselfeldwert anlegen.

Mit dieser Meldung kann der Benutzer natürlich nicht viel anfangen. Genaugenommen ist das Feld **KundeID** der Tabelle **tblProjekte**, also das Fremdschlüsselfeld, ja noch nicht einmal im Unterformular enthalten.

Also verwenden wir die Möglichkeit, dem Formular eine Fehlerbehandlung zuzuweisen. Grundlegende Informationen hierzu finden Sie unter **Fehlerbehandlung in Formularen**. Im vorliegenden Fall soll das Formular den Fehler abfangen, der durch das Anlegen eines Datensatzes ohne vorherigens Füllen des Fremdschlüsselwertes im Unterformular ausgelöst wird. Dazu schließen Sie beide

# 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.

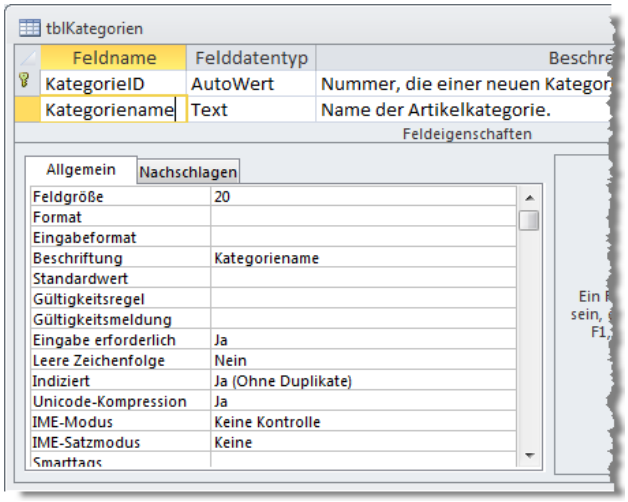



**Bild 1:** Das Hinzufügen doppelter Werte in dieses Feld löst einen Fehler aus.

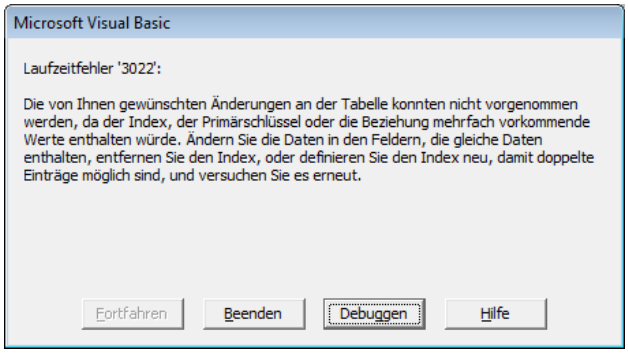



**Bild 2:** Fehlermeldung beim Anlegen doppelter Werte im eindeutigen Feld

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,

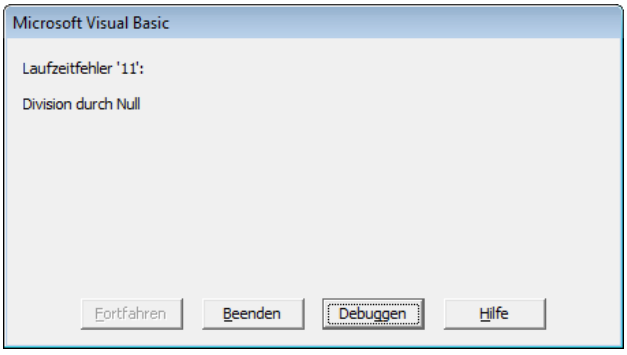



**Bild 3:** Eingebaute Fehlermeldung

# 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

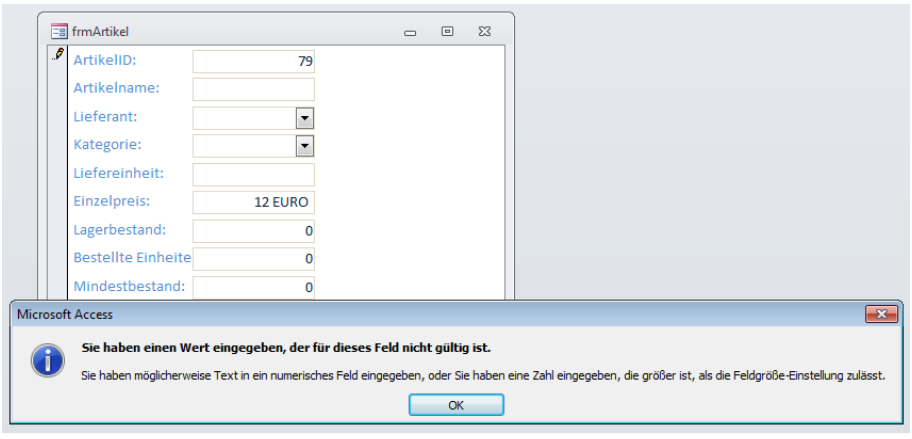



**Bild 1:** Dieser Meldung erscheint beim Eingeben von Text in numerische Felder.

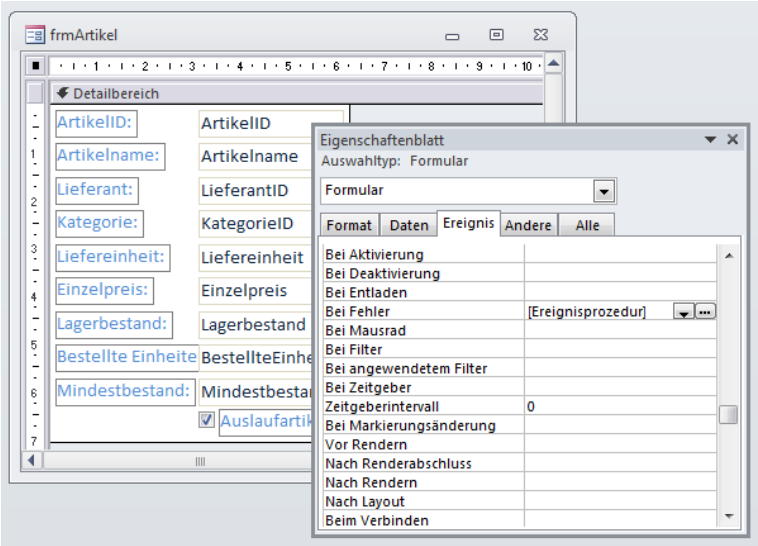



**Bild 2:** Anlegen der Ereignisprozedur, die bei Fehlern im Formular ausgelöst wird

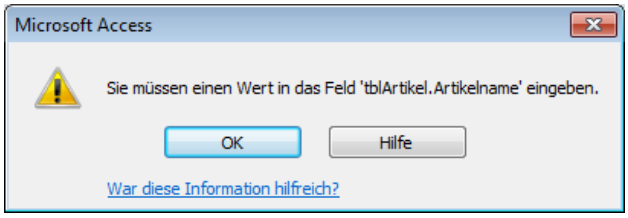



**Bild 3:** Fehlermeldung beim Speichern eines Datensatzes mit leeren Pflichtfeldern

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
```

Access [basics] 9/2011
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Editorial

Wollen Sie Daten wie Postleitzahlen, Orte, Länder und so weiter immer wieder komplett eintippen müssen? Nein! Sollen solche Daten deshalb in einer Lookup-Tabelle landen, damit Sie diese per Kombinationsfeld auswählen können? Nein! Also zeigen wir Ihnen im ersten Artikel der neuen Ausgabe, wie Sie ganz einfach auf die bereits einmal eingegebenen Werte eines Feldes zugreifen, indem Sie die Datensatzherkunft eines Kombinationsfeldes mit den entsprechenden Daten füllen. Sie geben dann nur noch die ersten paar Zeichen ein und betätigen die Tabulator-Taste, sobald die Autovervollständigen-Funktion den gesuchten Eintrag anzeigt! Wie dies genau funktioniert, zeigt unser Beitrag **Kombinationsfeld als Eingabehilfe für Textfelder** ab Seite 2.

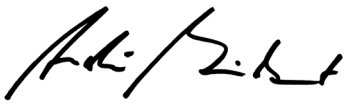
In einem weiteren Artikel stellen wir Ihnen die Optionsgruppe als Eingabesteuerelement für gebundene und ungebundene Daten vor. Damit erlauben Sie dem Benutzer, schnell per Mausklick eine von mehreren Auswahlmöglichkeiten zu selektieren, statt dafür erst ein Kombinationsfeld zu öffnen oder den Wert sogar von Hand eintippen zu müssen. Alles zu den beiden Steuerelementen Optionsgruppe und Optionsfeld lesen Sie im Artikel **Optionsgruppen und Optionsfelder** ab Seite 5.

Wenn Sie noch nicht so viele Berichte erstellt haben, sollten Sie unbedingt einen Blick in den Artikel **Berichtsbereiche** ab Seite 11 werfen. Dort lernen Sie die einzelnen Bereiche von Berichten und deren Funktion kennen.

Auch im Artikel **Sortieren in Berichten** ab Seite 14 dreht es sich um Berichte. Hier erfahren Sie, dass in Tabellen und Abfragen festgelegte Sortierungen nur sporadisch in Berichten funktionieren. Dort gibt es nämlich ein eigenes Werkzeug zum Festlegen der Sortierung.

Und schließlich liefern wir im Artikel **Tipps und Tricks** ab Seite 17 eine Reihe interessanter Informationen. Dort erfahren Sie etwa, wie Sie das Standardverzeichnis einer Datenbank einstellen, Objekte wie Tabellen, Abfragen, Formulare und Berichte im Datenbankfenster beziehungsweise Navigationsbereich ausblenden und somit vor den Blicken der Benutzer schützen und wie Sie diese temporär wieder sichtbar machen. Außerdem zeigen wir Ihnen, wie Sie Datum und Uhrzeit schnell per Tastenkombination einfügen und wie Sie den Wert eines Feldes aus dem vorherigen Datensatz auch im aktuellen Datensatz einfügen.

Und nun: Viel Spaß beim Lesen!



Ihr André Minhorst

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst | Fachverlag für Softwareentwicklung | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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.

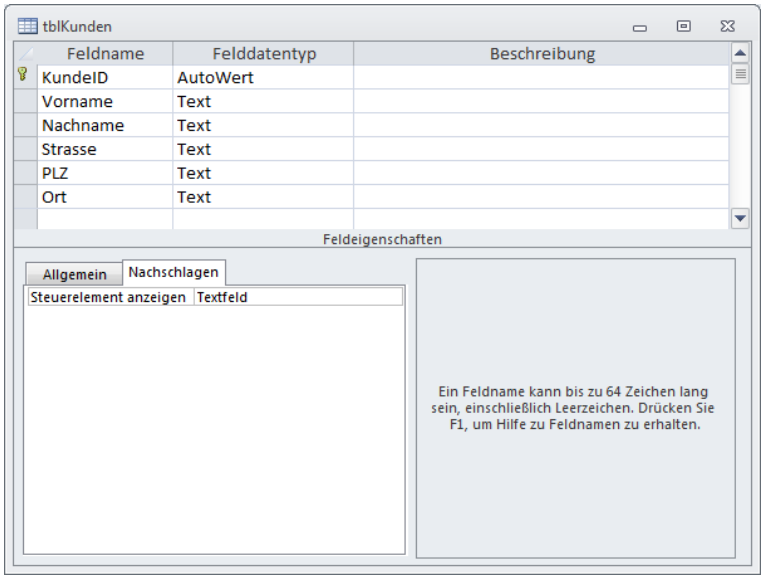



**Bild 1:** Beispieltabelle **tblKunden**

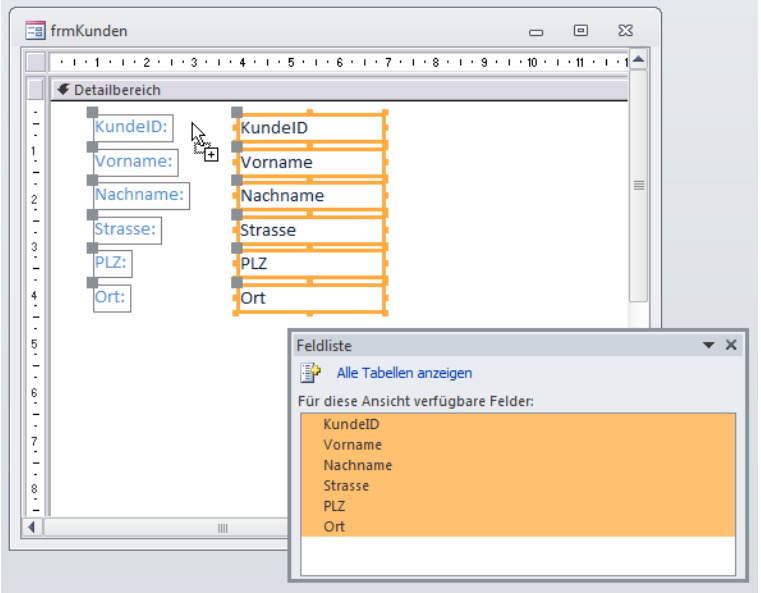



**Bild 2:** Hinzufügen der Felder der Tabelle **tblKunden** zum Formular **frmKunden**

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.

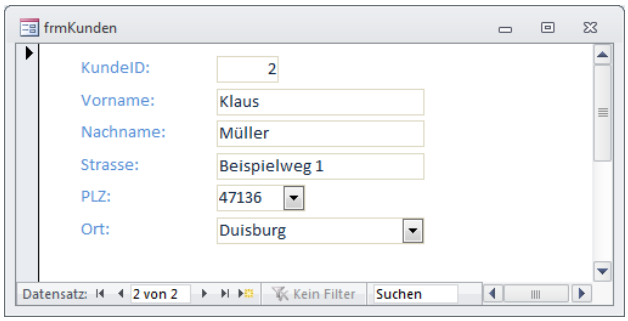



**Bild 3:** Eintragen von Daten in Textfelder, die in Kombinationsfelder umgewandelt wurden

Nach dem Wechsel in die Formularansicht tut sich nicht viel: Sie können ganz gewöhnlich Daten in die Kombinationsfelder eintragen (siehe Bild 3). Das Aufklappen der Kombinationsfelder bietet keine weiteren Auswahlmöglichkeiten an.

Das ändern wir jetzt, und zwar zunächst für die Postleitzahl. Wechseln Sie zurück in die Entwurfsansicht des Formulars und klicken Sie auf das Steuerelement **cboPLZ**. Klicken Sie in die Eigenschaft **Datensatzherkunft** und dann auf die nun erscheinende Schaltfläche mit den drei Punkten (...). Sie öffnen damit die Abfrage-Entwurfsansicht und können nun eine Abfrage als Datensatzherkunft für das Kombinationsfeld einstellen.

Wählen Sie im Dialog **Tabelle anzeigen** den Eintrag **tblKunden** aus und schließen Sie diesen Dialog. Ziehen Sie das Feld **PLZ** aus der Tabelle **tblKunden** in das Entwurfsraster des Abfrageentwurfs und stellen Sie die **Sortierung** auf **Aufsteigend** ein. Wenn Sie nun in die Datenblattansicht wechseln, zeigt diese bereits alle Postleitzahlen der Tabelle an – allerdings auch doppelte. Damit dies nicht geschieht, stellen Sie die Eigenschaft **Keine Duplikate** der Tabelle auf **Ja** ein (siehe Bild 4). Sollte das Eigenschaftsfenster gerade nur Feldeigenschaften anzeigen, ist aktuell ein Feld markiert – vermutlich, weil Sie dieses gerade bearbeitet haben. Klicken Sie dann einfach in den grau hinterlegten Bereich oben im Abfrageentwurf, um die Abfrage-Eigenschaften einzublenden.

Schließen Sie den Abfrageentwurf und kehren Sie zum Formular zurück. Wechseln Sie zur Formularansicht und geben Sie für einen neuen Kunden die Postleitzahl ein, die testweise genauso beginnt wie eine bereits vorhandene Postleitzahl.

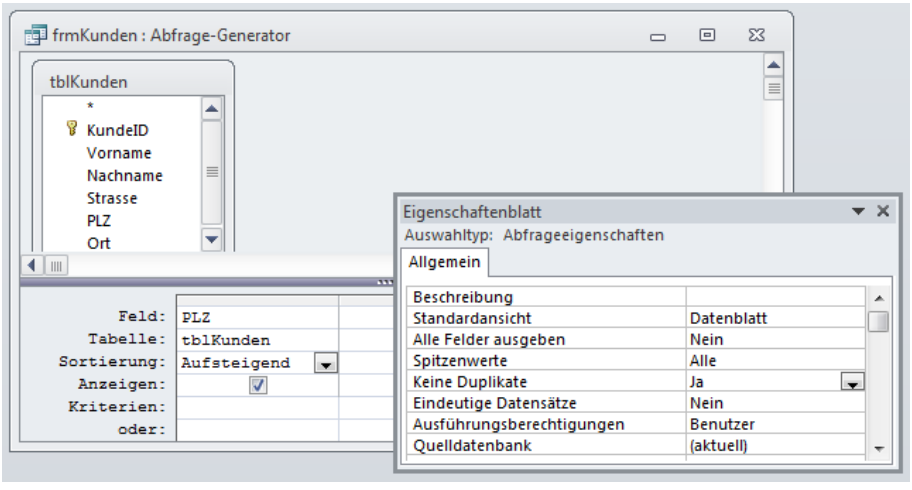



**Bild 4:** Einstellen der Datensatzherkunft des Kombinationsfeldes

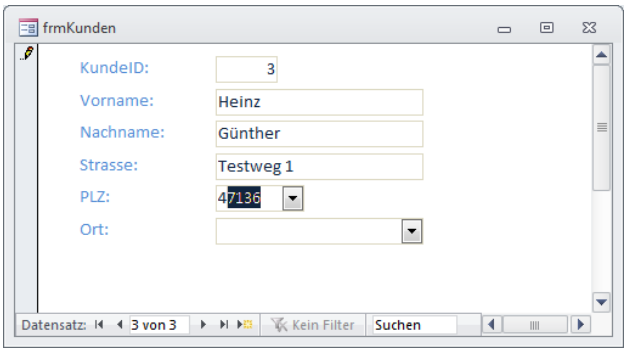



**Bild 5:** Kombinationsfeld mit automatischer Ergänzung

Und das Ergebnis überzeugt: Das Kombinationsfeld zeigt gleich den ersten Treffer der bisher eingegebenen PLZs an, das mit den bereits eingegebenen Zeichen übereinstimmt (siehe Bild 5). Der Benutzer könnte nun direkt zum nächsten Feld springen und den aktuellen Wert so übernehmen. Sie haben aber noch mehr Möglichkeiten: Sie können zum Beispiel das Kombinationsfeld aufklappen und einen der bisher vergebenen Werte auswählen. Für Tastatur-Fans: Ein Kombinationsfeld öffnen Sie mit **F4**.

## Automatische Ergänzung aktiv?

Das die automatische Ergänzung so funktioniert (wenn sie funktioniert), liegt an der Einstellung der Eigenschaft **Automatische Ergänzung** des Kombinationsfeldes. Wenn bei Ihnen nicht automatisch der nächste verfügbare Eintrag angezeigt wird, ist diese Eigenschaft möglicherweise auf den Wert **Nein** eingestellt. Ändern Sie den Wert in diesem Fall einfach auf **Ja**.

# 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.

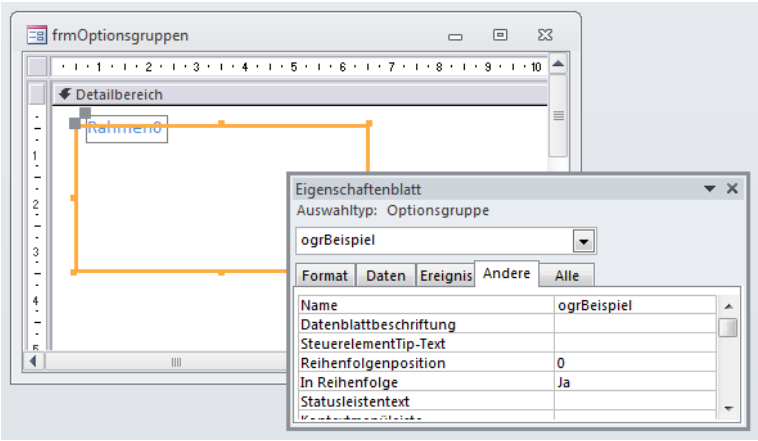



**Bild 1:** Einstellen des Namens einer frisch angelegten Optionsgruppe

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

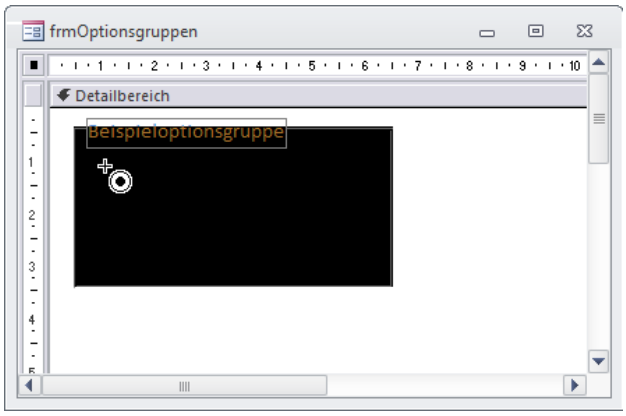



**Bild 2:** Hinzufügen eines Optionsfeldes zu einer Optionsgruppe

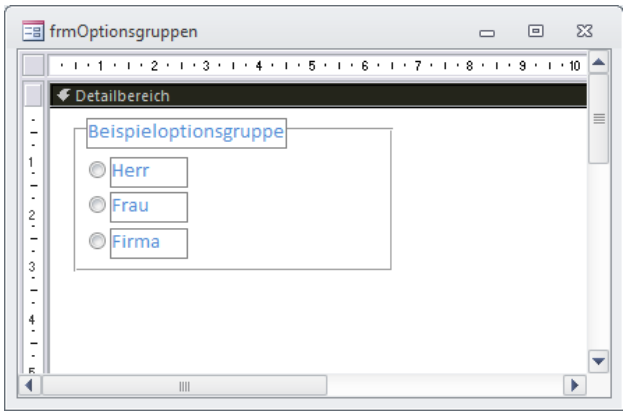



**Bild 3:** Hinzufügen weiterer Optionsfelder

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.

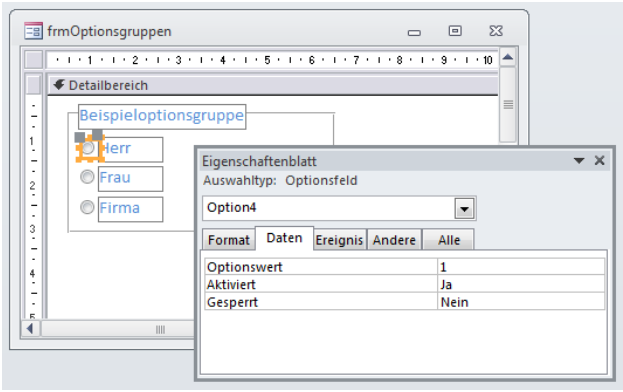



**Bild 4:** Wert eines der Optionsfelder

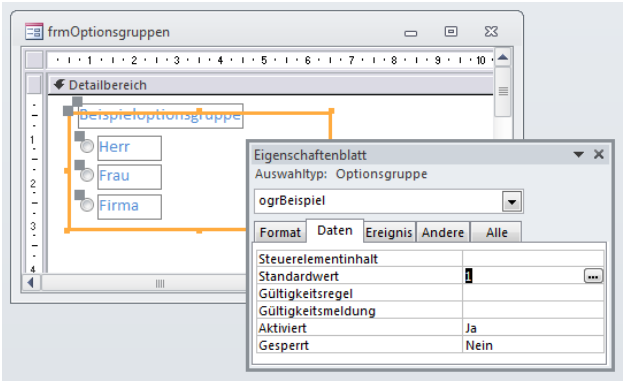



**Bild 5:** Standardwert der Optionsgruppe festlegen

# 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-

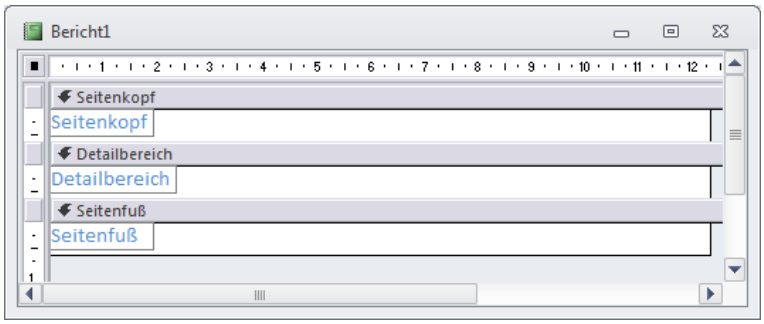



**Bild 1:** Standardmäßig angezeigte Berichtsbereiche

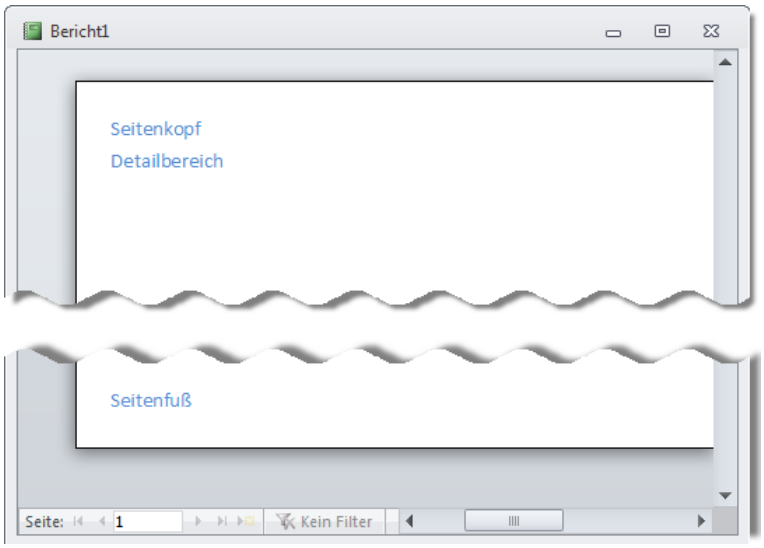



**Bild 2:** Berichtsbereiche in der Seitenansicht

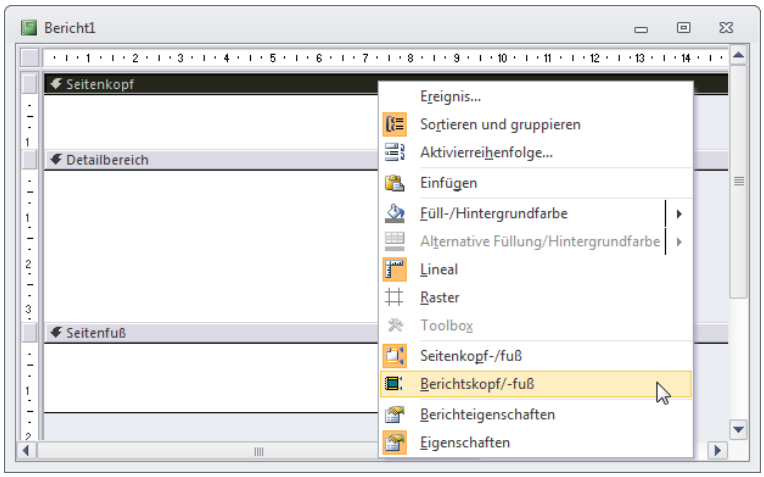



**Bild 3:** Ein- und Ausblenden von Berichtsbereichen

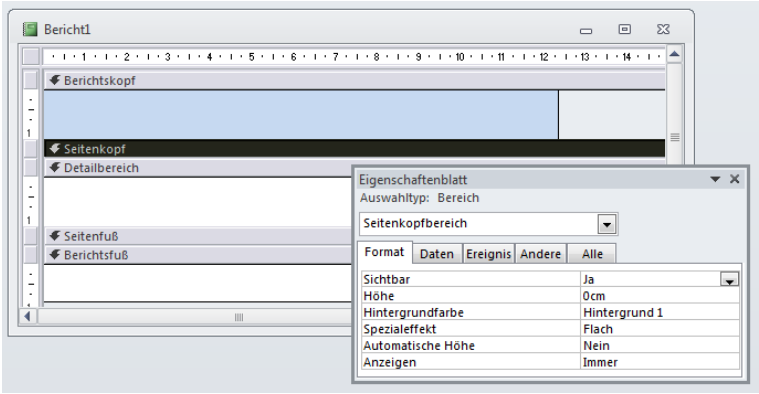



**Bild 4:** Bericht mit ausgeblendeten Berichtsbereichen

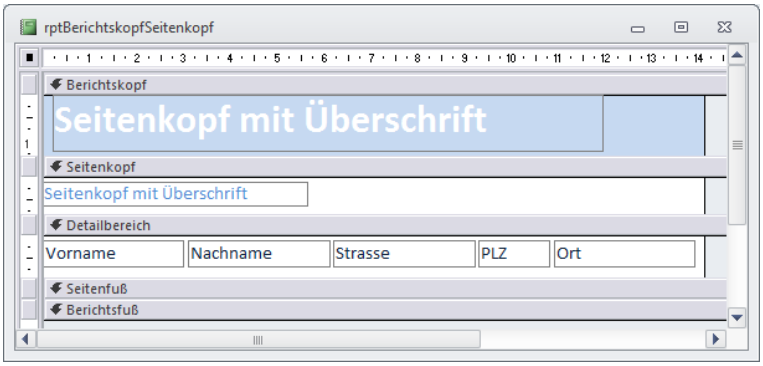



**Bild 5:** Bericht mit Berichtskopf und Seitenkopf




**Bild 6:** Berichts- und Seitenkopf erscheinen beide auf der ersten Seite.




**Bild 7:** Seitenkopf ab Seite 2? Kein Problem mit dieser Eigenschaft!

flussen. Die Kontextmenü-Variante funktioniert jedoch versionsübergreifend.

Wenn Sie nur den Seitenkopf oder -fuß beziehungsweise den Berichtskopf oder -fuß benötigen, aktivieren Sie jeweils beide Bereiche und stellen die Eigenschaft Höhe des unerwünschten Bereichs im Eigenschaftsfenster auf **0** ein. Alternativ verschieben Sie einfach den darunter liegenden Bereich soweit nach oben, bis der nicht erwünschte Bereich nicht mehr sichtbar ist (wenn der Bereich sich ganz unten befindet, ziehen Sie den unteren Rand einfach mit der Maus nach oben, bis die Höhe **0** ist). Bild 4 zeigt einen Bericht mit Berichts- und Seitenbereichen, von denen jeweils einer durch Einstellen der Eigenschaft **Höhe** auf den Wert **0cm** ausgeblendet wird.

## Überschriften in Berichts- und Seitenkopf

Wenn Sie einen Bericht mit Berichtskopf und Seitenkopf ausstatten, hat das meist folgenden Grund: Sie möchten auf der ersten Seite eine große Überschrift abbilden, die auf den folgenden Seiten wiederkehrt – allerdings möglichst nicht mehr allzu groß. Dann legen Sie wie in einen entsprechenden Berichtskopfbereich und einen Seitenkopfbereich an (siehe Bild 5).

Ein Wechsel in die Seitenansicht sorgt allerdings für Ernüchterung: Das der Seitenkopf bereits auf der ersten Seite angezeigt wird, war so nicht gedacht (siehe Bild 6). Aber woher soll Access auch wissen, dass der Seitenkopf erst ab der zweiten Seite erscheinen soll? Ganz einfach: Sie teilen es mit einer entsprechenden Eigenschaft mit.

Dazu wechseln Sie erneut in die Entwurfsansicht des Berichts. Die Eigenschaft **Seitenkopf** ist nicht etwa eine Eigenschaft des entsprechenden Bereichsbereichs, sondern des Berichts selbst. Deshalb markieren Sie zunächst den Bericht und finden dann unter dem Registerreiter **Format** die gewünschte Eigenschaft (siehe Bild 7).

# 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

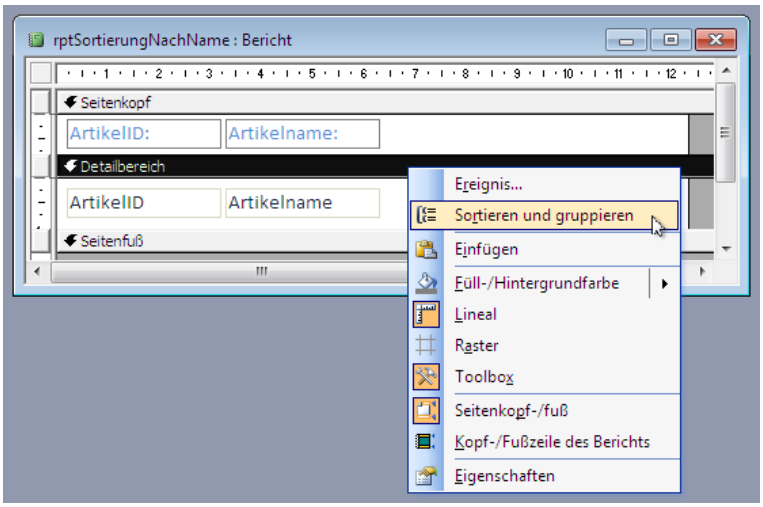



**Bild 1:** Aufruf des Dialogs zum Angeben einer Sortierung für die Daten des Berichts

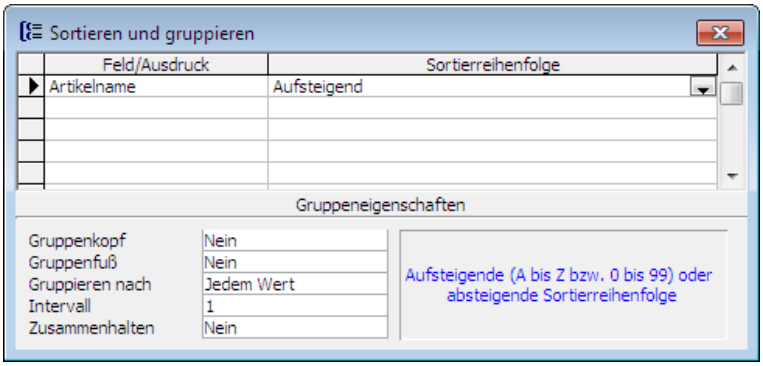



**Bild 2:** Angeben der Sortierreihenfolge eines Berichts

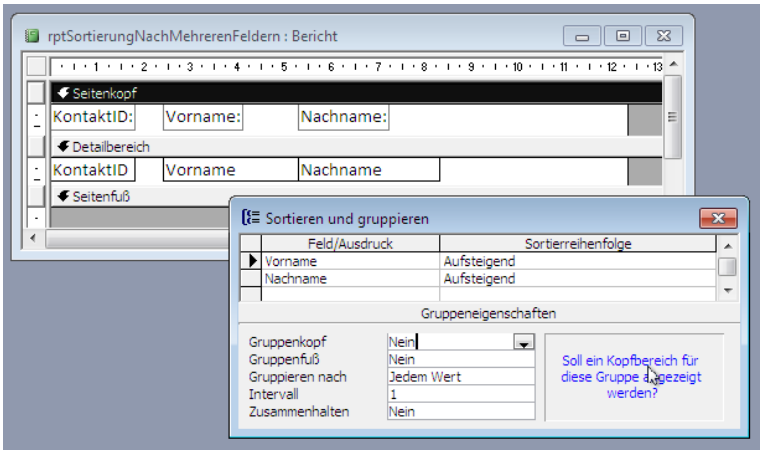



**Bild 3:** Bericht mit zweifacher Sortierung

können. Wenn Sie beispielsweise eine Artikelliste nach dem Inhalt des Feldes **Artikelname** sortieren möchten, wählen Sie unter **Feld/Ausdruck** den Eintrag **Artikelname** aus und behalten rechts unter Sortierreihenfolge den Wert **Aufsteigend** bei.

Wenn Sie wie in Bild 3 gleich nach mehreren Felder sortieren möchten, berücksichtigen Sie, dass die Sortierung von oben nach unten abgearbeitet wird. Das bedeutet in diesem Beispiel, dass erst nach dem Vornamen und dann nach dem Nachnamen sortiert wird (siehe Bild 4).

## Sortieren nach Nachschlagefeldern

Access macht es dem Entwickler an vielen Stellen sehr einfach. Sie können beispielsweise Datensätze der Tabelle **tblArtikel** mitsamt Kategorie- und Lieferantennamen in einem Bericht anzeigen, ohne dass die Tabellen **tblKategorien** und **tblLieferanten** explizit zur Datenherkunft hinzugefügt werden. Die anzuzeigenden Werte liegen lediglich in Form der Datensatzherkunft der in der Tabelle **tblArtikel** definierten Nachschlagefelder vor. Wenn Sie diese Felder zum Berichtsentwurf hinzufügen, werden diese zwar noch als Kombinationsfeld angezeigt (Access übernimmt hier die im Tabellenentwurf festgelegten Einstellungen, aus einem Nachschlagefeld wird also ein Kombinationsfeld). Beim Wechsel in die Seitenansicht und auch beim Drucken ersetzt Access die Kombinationsfelder jedoch durch einfache Textfelder und zeigt schlicht die Texte an, die auch die Kombinationsfelder angezeigt hätten.

Was aber geschieht, wenn Sie etwa nach dem Feld **LieferantID** der Tabelle **tblArtikel** sortieren (siehe Bild 5)? Der Bericht zeigt zwar die Lieferantennamen statt der ID an, aber die Sortierung berücksichtigt die im Fremdschlüsselfeld gespeicherten Werte (siehe Bild 6).

Wenn Sie in einem Bericht also tatsächlich nach Feldinhalten sortieren wollen, die in Kombinationsfeldern angezeigt werden, müssen Sie die entsprechenden Felder der verknüpften Tabellen mit in die Datenherkunft aufnehmen und diese im Dialog **Sortieren und gruppieren** angeben. Die Abfrage mit den Feldern **Lieferant** und **Kategorie** sähe dann etwa wie in Bild 7 aus.

Nachdem Sie diese Abfrage als Datenherkunft des Berichts eingestellt haben, können Sie im Dialog **Sortieren und gruppieren** das Feld **Firma** als Sortierkriterium auswählen (siehe Bild 8). Danach gelingt auch die Sortierung nach Feldern wie **Firma** oder **Kategoriename** (siehe Bericht **rptArtikelNachLieferant**).

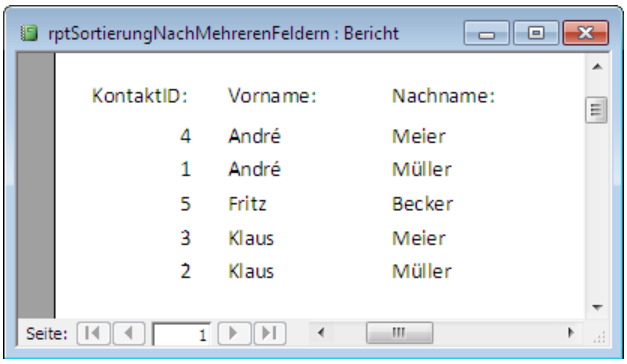



**Bild 4:** Sortierung nach Vorname und Nachname

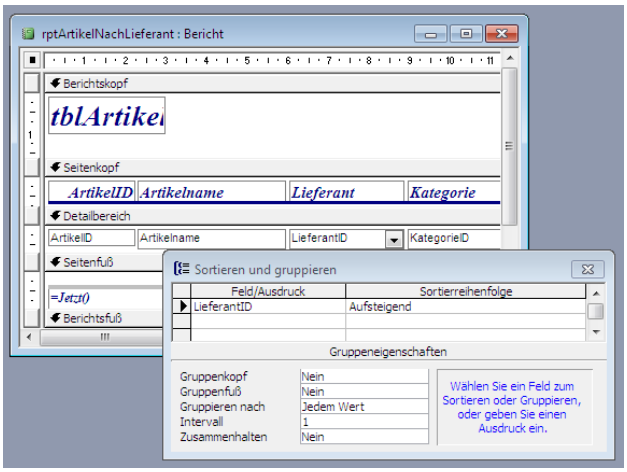



**Bild 5:** Einrichten der Sortierung nach einem Fremdschlüssel-/Nachschlagefeld

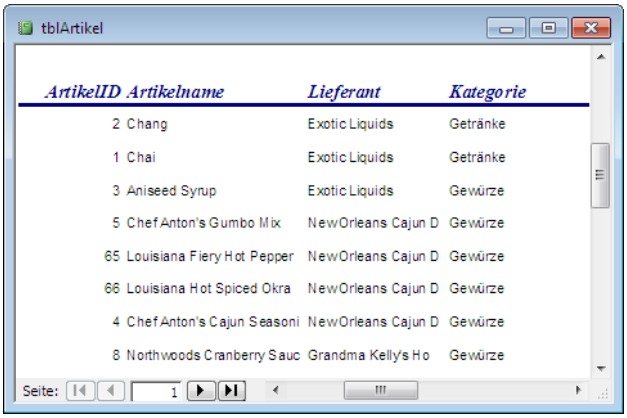



**Bild 6:** Das Sortieren nach Werten aus Nachschlagefeldern funktioniert nicht wie erwartet.

## Dynamisch sortieren

Gegebenenfalls möchten Sie dem Benutzer die Möglichkeit bieten, selbst die Sortierreihenfolge im Be-

# Tipps und Tricks

**In dieser Ausgabe zeigen wir Ihnen, wie Sie das Standardverzeichnis beim Öffnen und Erstellen von Datenbanken anpassen, wie Sie Datenbankobjekte ausblenden, damit Otto Normalverbraucher diese nicht mehr im Datenbankfenster beziehungsweise im Navigationsbereich vorfindet, wie Sie schnell die aktuellen Werte für Zeit und Datum in Textfelder eintragen, wie Sie den Wert aus dem gleichen Feld des vorherigen Datensatzes übernehmen und vieles mehr!**

## Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1109_TippsUndTricks.mdb**.

## Standardverzeichnis einstellen

Wenn Sie unter Access eine neue Datenbank anlegen oder eine Datenbank öffnen möchten, zeigt Access immer den entsprechenden Dateidialog an. Standardmäßig wird dort je nach Access-Version ein Verzeichnis wie **Eigene Dokumente** oder ähnlich angezeigt.

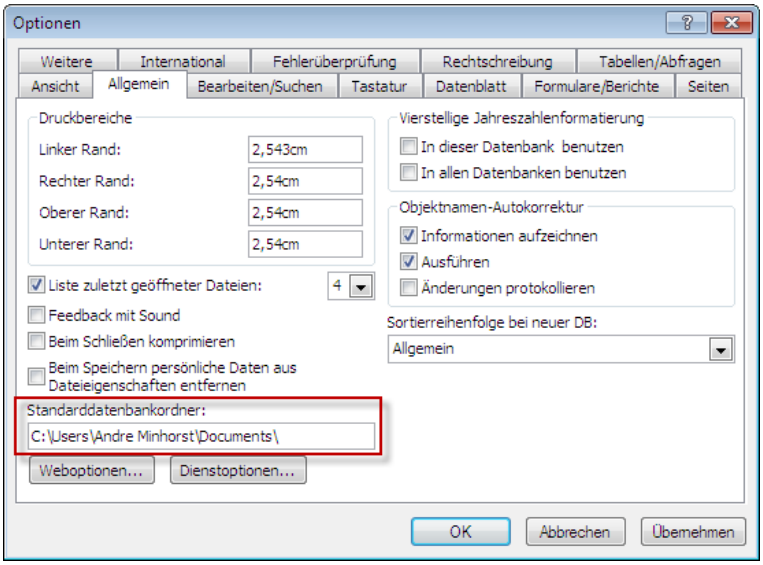



**Bild 1:** Einstellen des Standardverzeichnisses von Access

Normalerweise verwenden Sie aber ein anderes Verzeichnis, um Ihre Access-Datenbanken zu speichern – oder zumindest ein gemeinsames übergeordnetes Verzeichnis, von dem aus Sie mit ein paar Mausklicks zum Zielverzeichnis gelangen.

Da ist es doch praktisch, dass Sie das Verzeichnis, das beim Öffnen oder Anlegen von Datenbankdateien angezeigt wird, selbst festlegen können. Dazu gehen Sie wie folgt vor:

- Unter Access 2003 und älter wählen Sie den Menüeintrag **Extras|Optionen** (dieser Eintrag ist nur bei geöffneter Datenbank aktiviert) und wechseln im nun erscheinenden Dialog zur Registerseite **Allemein**. Dort finden Sie die Option **Standarddatenbankordner**, mit der Sie das gewünschte Verzeichnis festlegen können (siehe Bild 1).

- Unter Access 2007 klicken Sie auf den Office-Button und dann auf die Schaltfläche **Access-Optionen**. Die Option **Standarddatenbankordner** befindet sich im Bereich **Häufig verwendet**.

- Unter Access 2010 wähle Sie den Ribbon-Eintrag **Datei|Optionen** aus, wechseln zum Bereich **Allgemein** und stellen das Standardverzeichnis unter **Datenbanken erstellen|Standarddatenbankordner** ein.

## Objekte ausblenden

Normalerweise sollte eine professionelle Anwendung eine eigene Benutzeroberfläche bestehend aus einer Menü- oder Ribbon-Steuerung, Formularen und Berichten mitbringen. Das Datenbankfenster (bis Access 2003) beziehungsweise der Navigationsbereich (ab Access 2007) sollten für den Anwender Ihrer Datenbanken tabu sein.

Wenn die Anwendung jedoch nicht ganz so professionellen Ansprüchen genügen muss, kann man den Benutzer die benötigten Access-Objekte natürlich über Datenbankfenster/Navigationsbereich öffnen lassen.

Dann sollten Sie aber zumindest nur die für die Benutzung der Datenbank wichtigen Objekte anzeigen. Die anderen blenden Sie einfach aus, und zwar so:

- Unter Access 2003 klicken Sie im Datenbankfenster mit der rechten Maustaste auf das fragliche Objekt und wählen den Kontextmenüeintrag **Eigenschaften** aus (siehe Bild 2). Im nun erschein-

den Eigenschaftsfenster für dieses Objekt aktivieren Sie die Option **Ausgeblendet** (siehe Bild 3).

- Unter Access 2007 und Access 2010 gehen Sie genauso vor – mit dem Unterschied, dass Sie nicht auf das Objekt im Datenbankfenster, sondern im Navigationsbereich klicken und dass der Eintrag im Kontextmenü je nach Objekt **Tabelleneigenschaften**, **Objekteigenschaften** et cetera heißt.

Nach dem Schließen des jeweiligen Eigenschaftsfensters blendet Access den Eintrag des betroffenen Elements aus.

## Ausgeblendete Objekte anzeigen

Was dem Benutzer die Arbeit mit Ihrer Anwendung erleichtert, ist für den Entwickler hinderlich: Wie soll man nun auf die Schnelle auf die ausgeblendeten Objekte zugreifen? Auch das ist kein Problem, denn Access hält eine Option bereit, mit der Sie auch ausgeblendete Objekte im Datenbankfenster beziehungsweise im Navigationsbereich anzeigen können.

- Unter Access 2003 und älter betätigen Sie dazu zunächst den Menüeintrag **Extras|Optionen** und wechseln im Optionen-Dialog zur Registerseite **Ansicht**. Dort finden Sie die Option **Ausblendete Objekte**, die Sie aktivieren müssen (siehe Bild 4). Wenn Sie den Optionen-Dialog nun schließen, zeigt Access die ausgeblendeten Objekte mit abgeblendetem Symbol an (siehe Bild 5).

- Unter Access 2007 und jünger finden Sie diese Option, wenn Sie mit der rechten Maustaste auf die Titelzeile des Navigationsbereichs klicken und den Eintrag **Navigationsoptionen** auswählen. Der nun erscheinenden Dialog **Navigationsoptionen** bietet ebenfalls die Option **Ausgeblendete Objekte** an. Hier werden die wieder sichtbar gemachten ausgeblendeten Objekte zusätzlich in grauer Schrift angezeigt.

## Aktuelles Datum und aktuelle Zeit blitzschnell

Datum und Zeit muss man immer wieder mal in Textfelder eingeben, manchmal sogar beides. Dummerweise muss man beim Datum meist erstmal überlegen (okay, manch einer hat auch immer das aktuelle Datum parat), und die Uhrzeit kann man unten rechts in der Taskleiste ablesen. Aber diese Informationen

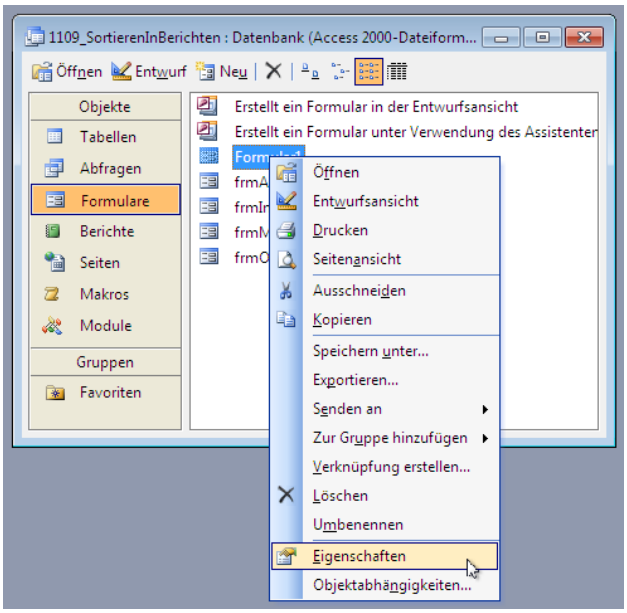



**Bild 2:** Anzeigen der Eigenschaften eines Datenbank-Objekts

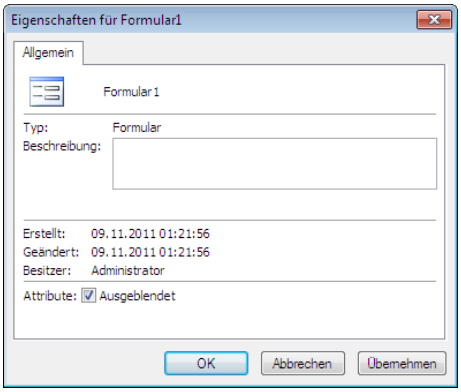



**Bild 3:** Ausblenden eines Objekts

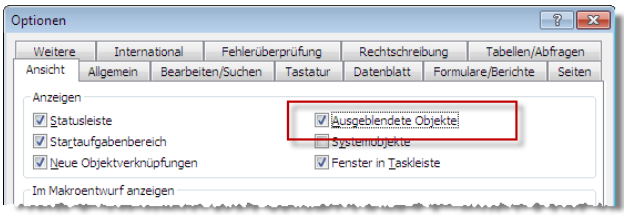



**Bild 4:** Einblenden ausgeblendeter Objekte

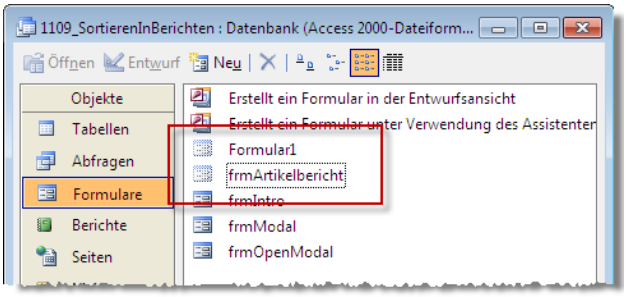



**Bild 5:** Ausgeblendete Objekte wieder eingeblendet

Access [basics] 10/2011
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Editorial

Diese Ausgabe von **Access [basics]** bietet gleich sechs Artikel. Der erste zeigt, was Sie alles mit der Datenblattansicht anstellen können – und das, ohne diese per VBA zu programmieren. Richtig tolle Möglichkeiten zum Filtern der angezeigten Daten bieten dabei erst Access 2007 und Access 2010. Aber lesen Sie doch selbst nach – im Artikel **Datenblattansicht im Griff** ab Seite 2.

Wie oft kommt es vor, dass Sie gern den aktuell im Formular angezeigten Datensatz kopieren möchten, um einen neuen Datensatz auf Basis eines bestehenden Datensatzes zu erstellen? Vermutlich oft genug, um dafür eine eigene Schaltfläche einzurichten, mit der Sie per Mausklick einen Datensatz duplizieren können. Details dazu lesen Sie im Beitrag **Aktuellen Datensatz kopieren** ab Seite 6.

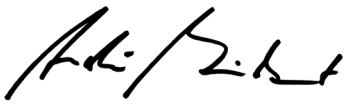
Der vierte Beitrag dreht sich wiederum um Formulare. Je länger Sie mit Access entwickeln und je professioneller die Anwendungen werden, desto mehr werden Sie mit VBA arbeiten. Da kann es nicht schaden, wenn Sie wissen, wie Sie per VBA auf Formulare, Unterformular und die enthaltenen Steuerelemente zugreifen können. Das ist zwar eigentlich ganz einfach, wenn man das System einmal verinnerlicht hat, aber aller Anfang ist schwer – deshalb haben wir einen kompletten Artikel nur zu diesem Thema geschrieben. Lesen Sie also ab Seite 8 sehr aufmerksam, wenn es heißt: **Formulare und Steuerelemente referenzieren**.

Das Öffnen von Berichten ist doch eigentlich ganz einfach? Nicht, wenn Sie dies per VBA erledigen und dabei beispielsweise einen Filter oder Öffnungsargumente übergeben wollen oder den Bericht in verschiedenen Ansichten präsentieren möchten. Ab Seite 12 lernen Sie im Artikel **Berichte öffnen** alles, was Sie beim Öffnen eines Berichts mit VBA beachten müssen.

Kann man eigentlich per VBA auf Informationen wie den Datenbanknamen et cetera zugreifen? Dazu bietet Access eine Bibliothek namens DAO an: Sie enthält eine ganze Reihe von Anweisungen, mit denen Sie die gesuchten Informationen erhalten. Unter **DAO: Datenbank im Griff mit dem Database-Objekt** erfahren Sie ab Seite 17, welche Möglichkeiten diese Bibliothek zum Verwalten der aktuellen Datenbank bietet.

Und schließlich lernen Sie im Artikel **For Each-Schleifen** noch die einzige bislang nicht vorgestellte Schleifenart kennen (ab Seite 20).

Und nun: Viel Spaß beim Lesen!



Ihr André Minhorst

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst | Fachverlag für Softwareentwicklung | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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.

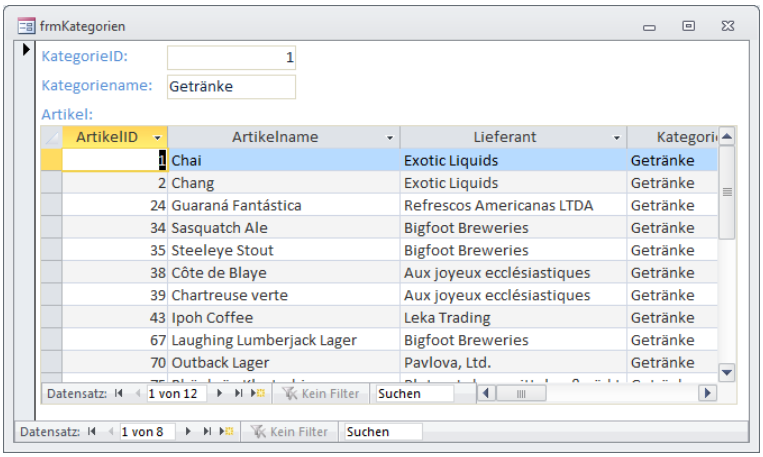



**Bild 1:** Unterformular in der Datenblattansicht

## 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

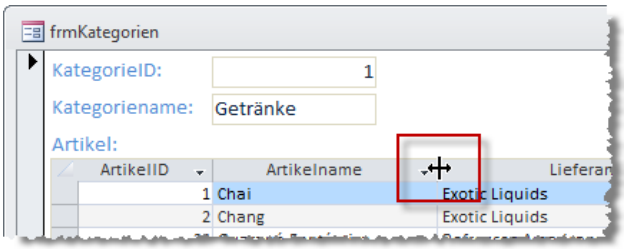



**Bild 2:** Spaltenbreite einstellen

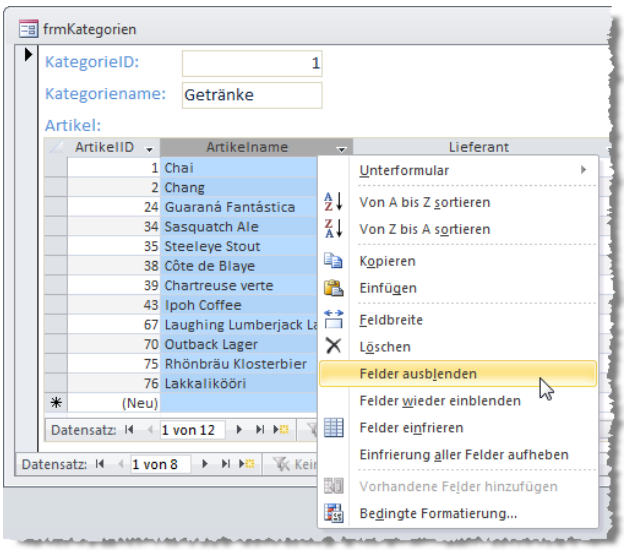



**Bild 3:** Spalten ausblenden

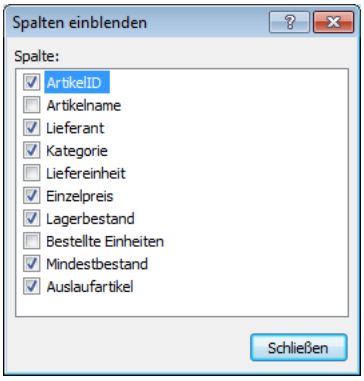



**Bild 4:** Spalten ein- und ausblenden

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

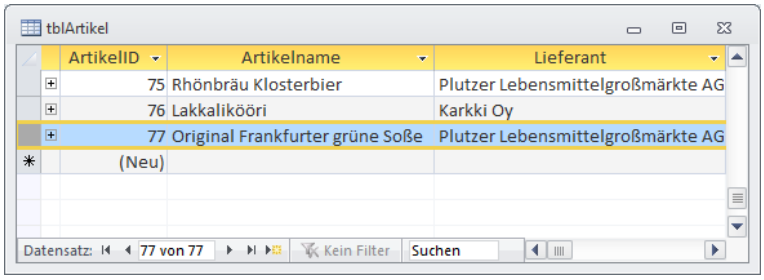



**Bild 1:** Markieren eines kompletten Datensatzes

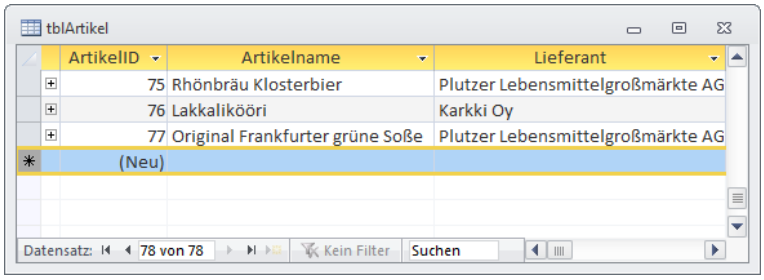



**Bild 2:** Markieren des Zieldatensatzes, in diesem Fall eines neuen Datensatzes

- 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.

Für das folgende Beispiel verwenden Sie ein einfaches Formular. Weisen Sie dem Formular in der Entwurfsansicht die Tabelle **tblArtikel** als Datenherkunft zu und ziehen Sie alle Felder dieser Tabelle in die Entwurfsansicht des Formulars.

Aktivieren Sie den Formularkopf und -fuß und fügen Sie im Formularkopf eine Schaltfläche ein, die mit dem Text **Datensatz duplizieren** beschriftet ist und den Namen **cmdDatensatzDuplizieren** erhält. Das Formular finden Sie in der Beispieldatenbank unter dem Namen **frmDatensatzDuplizieren_I** (siehe Bild 3).

Legen Sie nun eine Prozedur an, die durch das Ereignis **Beim Klicken** der Schaltfläche **cmdDatensatzDuplizieren** an. Dazu tragen Sie für die Eigenschaft den Wert **[Ereignisprozedur]** ein und klicken dann auf die Schaltfläche mit den drei Punkten.

Die nun erscheinende Ereignisprozedur ergänzen Sie wie folgt:

```
Private Sub cmdDatensatzDuplizieren_Click()
    DoCmd.RunCommand acCmdSelectRecord
    DoCmd.RunCommand acCmdCopy
    DoCmd.RunCommand acCmdPasteAppend
End Sub
```

Die erste Anweisung markiert den kompletten aktuelle Datensatz. Die zweite kopiert diesen in die Zwischenablage und die dritte fügt den Datensatz als neuen Datensatz ein.

Und als ob das nicht schon einfach genug wäre, springt der Datensatzzeiger nach dem Ausführen dieser Prozedur auch noch zum frisch erstellten Duplikat. Hier brauchen Sie nun nur noch die gewünschten Änderungen durchzufügen.

## Kopieren von Datensätzen ohne Autowert

Wenn die Tabelle mit den zu kopierenden Datensätzen keinen Autowert als Primärschlüsselfeld verwendet, müssen Sie den Wert des Primärschlüsselfeldes vor dem Speichern manuell auf einen noch nicht vorhandenen Wert einstellen, da Access sonst eine entsprechende Fehlermeldung ausgibt.

In der Beispieldatenbank haben wir hierzu eine Kopie der Tabelle **tblArtikel** namens **tblArtikelOhneAutowert** angelegt, deren Primärschlüsselfeld wir in einen einfachen Zahlenwert geändert haben.

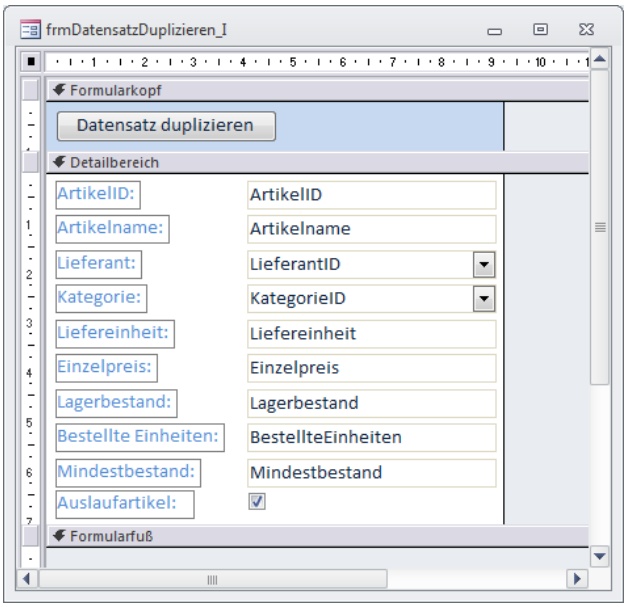



**Bild 3:** Mit diesem Formular können Sie den aktuellen Datensatz per Mausklick duplizieren.

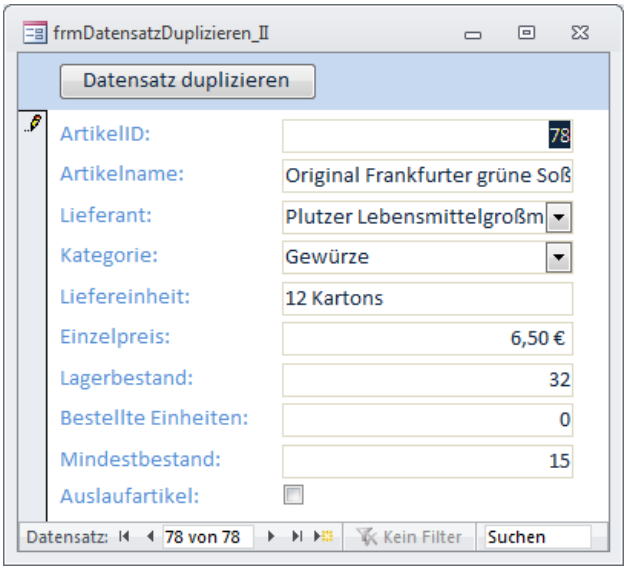



**Bild 4:** Frisch kopierter Datensatz mit automatisch geändertem Primärschlüsselwert

Wenn Sie dies auch noch durch die Prozedur erledigen lassen möchten, fügen Sie eine Zeile wie die folgende hinzu:

```
Me!ArtikelID = DMax("ArtikelID", _
    "tblArtikelOhneAutowert") + 1
```

Dieses Beispiel finden Sie im Formular **frmDatensatzDuplizieren_II** (siehe Bild 4)

# 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.

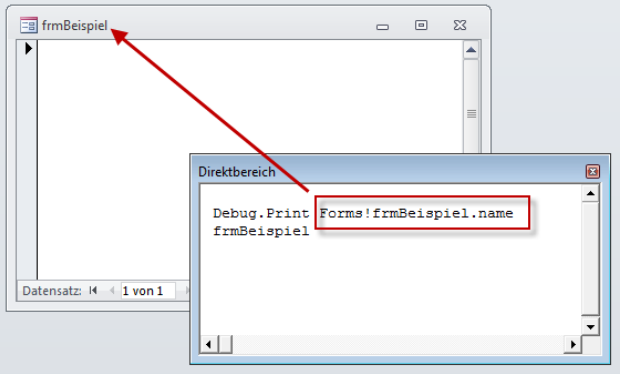



**Bild 1:** Referenzieren eines Formulars

## 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.

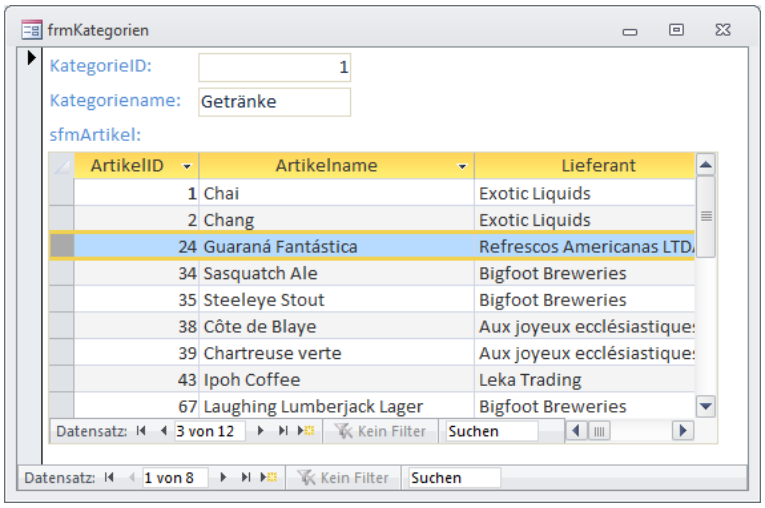



**Bild 2:** Referenzieren eines Unterformulars

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.

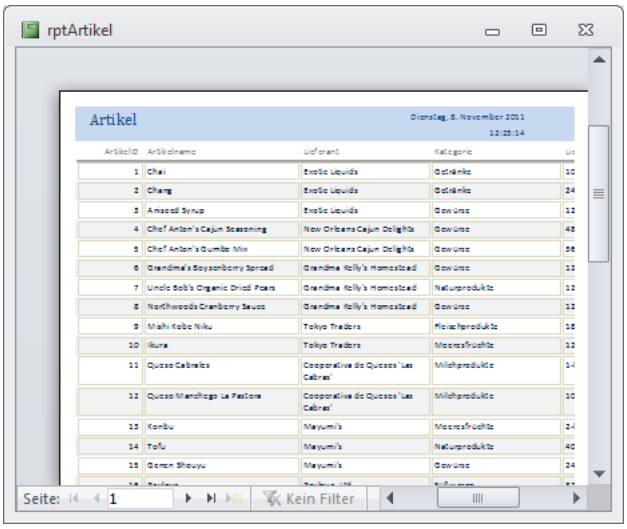



**Bild 1:** Ein in der Seitenvorschau geöffneter Bericht

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:

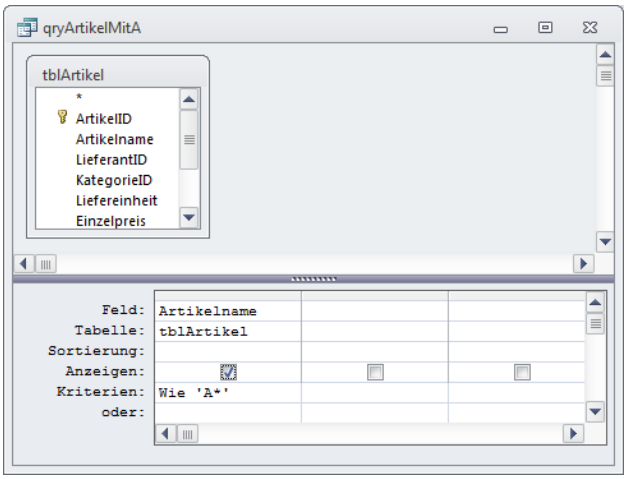



**Bild 2:** Diese Abfrage dient als Filter eines Berichts und wird mit dem Parameter **Filtername** übergeben.

```
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-**

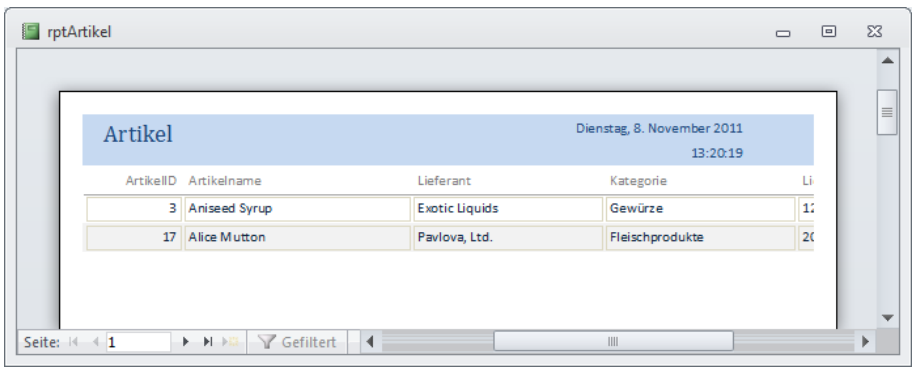



**Bild 3:** Dieser Bericht wurde mithilfe einer für das Kriterium **Filter** angegebenen Abfrage gefiltert.

# 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**.

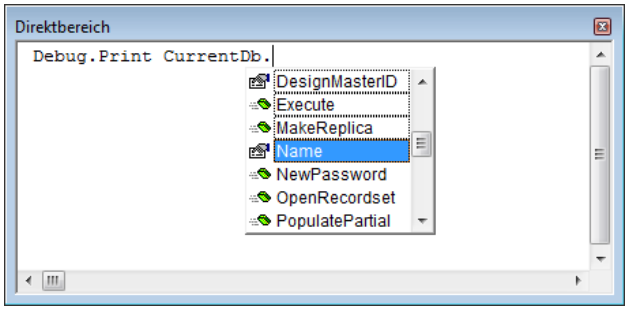



**Bild 1:** Eigenschaften und Methoden des mit **CurrentDb** gelieferten **Database**-Objekts

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
```

Wo jetzt die **Debug.Print**-Anweisung steht, können Sie beliebige weitere Zugriffe auf den in der Variablen **db** gespeicherten Verweis auf die aktuelle Datenbank absetzen.

## Alle Tabellen auslesen

Mit dem **Database**-Objekt können Sie alle in einer Access-Datenbank enthaltenen Tabellen auslesen. Die Tabellen macht das **Database**-Objekt über eine Auflistung namens **TableDefs** verfügbar (deutsch etwa: Tabellendefinitionen). Mit der Eigenschaft **Count** dieser Auflistung können Sie zunächst ermitteln, wieviele Tabellen die Datenbank enthält:

```
Debug.Print CurrentDb.TableDefs.Count
 20
```

20 Tabellen? Das sind aber viel mehr, als im Datenbankfenster beziehungsweise im Navigationsbereich angezeigt werden? Nun: Die **TableDefs**-Auflistung bezieht sich auch auf ansonsten unsichtbare Systemtabellen. Dies belegt die folgende Prozedur, in der Sie in einer **For Each**-Schleife alle Elemente der **TableDefs**-Auflistung durchlaufen.

Für den Einsatz der **For Each**-Schleife deklarieren Sie zunächst eine Variable namens **tdf** mit dem Datentyp **DAO.TableDef**. In der **For Each**-Schleife weisen Sie der hinter dem Schlüsselwort For angegebenen Variablen **tdf** nacheinander alle Elemente der Auflistung **TableDefs** zu. Die einzige Anweisung innerhalb der Schleife soll schlicht den Namen der jeweiligen Tabelle beziehungsweise Tabellendefinition ausgeben:

```
Public Sub TabellenAusgeben()
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    Set db = CurrentDb
    For Each tdf In db.TableDefs
        Debug.Print tdf.Name
    Next tdf
    Set db = Nothing
End Sub
```

Dies liefert dann Tabellennamen wie **MSysObjects**, aber auch die von selbst erstellten Tabellen wie **tblArtikel**. Mit der **TableDefs**-Auflistung lässt sich noch mehr anstellen: Sie können damit neue Tabellen erstellen, Tabellen löschen et cetera. Das **TableDef**-Objekt wiederum, das eine einzelne Tabelle beschreibt, enthält eine weitere Auflistung mit den Feldern der Tabelle und deren Eigenschaften. Auf dieses Thema gehen wir in einem späteren Artikel ein (**DAO: Tabellen verwalten per VBA**). Dort erfahren Sie nicht nur, wie Sie die Eigenschaften und Felder einer Tabelle auslesen, sondern auch, wie Sie Tabellen mithilfe des **TableDef**-Objekts per VBA erstellen können.

## Alle Abfragen auslesen

Genau wie die Definition der Tabellen einer Datenbank können Sie auch auf die Abfragen zugreifen. Dies gelingt prinzipiell genau wie beim **TableDef**-Objekt, nur dass in diesem Fall das **QueryDef**-Objekt zum Einsatz kommt. Eine Auflistung aller Abfragen einer Datenbank erhalten Sie wie in obigem Beispiel. Diesmal verwenden Sie die **QueryDefs**-Auflistung und geben den Namen einer Abfrage aus und den SQL-Ausdruck, der sich hinter der Abfrage verbirgt:

```
Public Sub AbfragenAusgeben()
    Dim db As DAO.Database
    Dim qdf As DAO.QueryDef
    Set db = CurrentDb
    For Each qdf In db.QueryDefs
        Debug.Print qdf.Name, qdf.SQL
    Next qdf
    Set db = Nothing
End Sub
```

## Weitere Auflistungen

Neben **TableDefs** und **QueryDefs** gibt es noch weitere Auflistungen und Auflistungselemente, mit denen Sie den Aufbau des Datenmodells einer Datenbank per VBA verwalten können. Wir gehen später jeweils in separaten Artikeln auf die folgenden Elemente ein:

- **Properties/Property**: Eigenschaften der Datenbank und auch der einzelnen Objekte wie Tabellen, Abfragen, Felder et cetera (siehe **DAO: Eigenschaften verwalten mit VBA**)
- **Relations/Relation**: Beziehungen zwischen den Tabellen einer Datenbank
- **Containers/Container**: Enthält Informationen zu den einzelnen Datenbankobjekten wie Tabellen, Abfragen, Formularen et cetera, kann aber auch zum Abfragen und Ändern von Datenbankeigenschaften eingesetzt werden. Mehr dazu später im Artikel **DAO: Datenbankeigenschaften per Formular**.

# 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

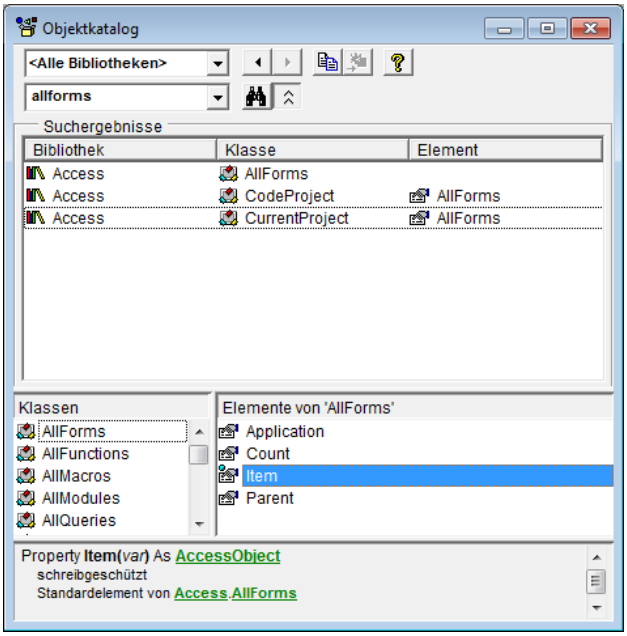



**Bild 1:** Der Objektkatalog hilft, den Typ eines Auflistungsobjekts zu ermitteln.

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
```

Access [basics] 11/2011
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Editorial

In dieser Ausgabe von **Access [basics]** gibt es wieder Access-Know-how auf 21 Seiten. Den Beginn machen die bedingten Formatierungen. Dabei handelt es sich um eine Methode, um Daten mit bestimmten Eigenschaften aus spezielle Art zu formatieren – sei es, indem Sie die Schriftart, -größe oder -farbe anpassen, den Text fett oder kursiv setzen oder auch einfach den Hintergrund farbig gestalten. Der Artikel **Bedingte Formatierung** stellt diese Technik ab Seite 2 vor.

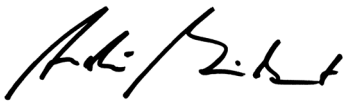
Mit Berichten lassen sich Daten wunderbar auflisten. Meist geschieht dies einspaltig, aber manchmal brauchen die Daten in der Breite so wenig Platz, dass Sie diese vielleicht auf zwei oder mehr Spalten aufteilen möchten. Dabei gibt es natürlich einiges zu beachten. Was dies ist und wie Sie schicke mehrspaltige Berichte erstellen, erfahren Sie im Artikel **Mehrspaltige Berichte** ab Seite 7.

Alle Datenbankanwendungen verwalten Texte. Diese zeigen Sie normalerweise an, bearbeiten diese manuell oder geben diese in Berichten aus. Manchmal möchten Sie Texte aber auch automatisiert manipulieren, um beispielsweise den Inhalt zweier Textfelder zu einem zusammenzuführen. Hier und in vielen anderen Fällen setzen Sie die Zeichenkettenfunktionen von Access ein. Einsatzorte gibt es genügend dafür: Sie setzen diese in Eigenschaften von Tabellen, Abfragen, Formularen und Berichten, in den Werten und Kriterien von Abfragen und auch unter VBA ein. Der Artikel **Zeichenketten-Funktionen** stellt die wichtigsten Zeichenkettenfunktionen ab Seite 12 vor.

Beim Thema Zeichenketten treten auch Zeilenumbrüche und Tabulatoren auf. Diese behandeln wir separat in einem Einseiter – lesen Sie dazu auf Seite 17 den Artikel **Zeilenumbruch und Tabulator in Access**.

Schließlich steigen Sie mit dieser Ausgabe in den VBA-gesteuerten Zugriff auf die Daten Ihrer Datenbank ein. Im Artikel **DAO: Recordsets öffnen, lesen und bearbeiten** ab Seite 18 erfahren Sie, was ein Recordset-Objekt ist, wie Sie die Daten Ihrer Tabellen dort hineinladen und diese bearbeiten. Sie erfahren aber auch, wie Sie mit den entsprechenden DAO-Methoden neue Datensätze anlegen oder bestehende Datensätze löschen. In der kommenden Ausgabe gibt es dann weitere Informationen zum Umgang mit DAO-Recordsets – dort erfahren Sie beispielsweise, wie Sie darin nach Daten suchen.

Und nun: Viel Spaß beim Lesen!



Ihr André Minhorst

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst | Fachverlag für Softwareentwicklung | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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.

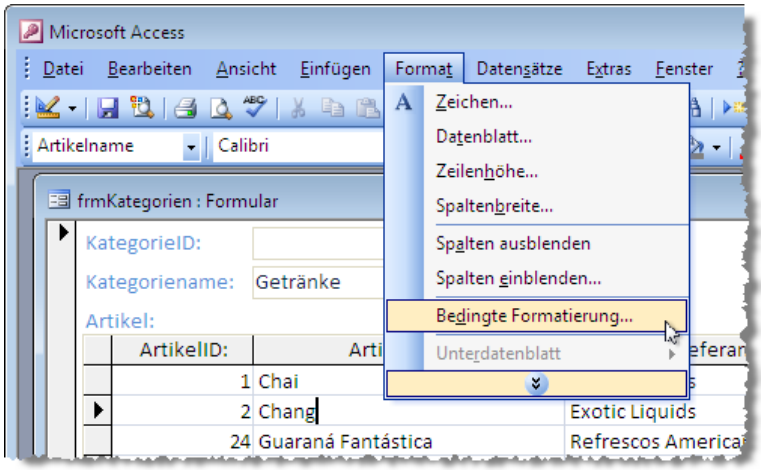



**Bild 1:** Aufrufen des Dialog zum Einstellen der bedingten Formatierung

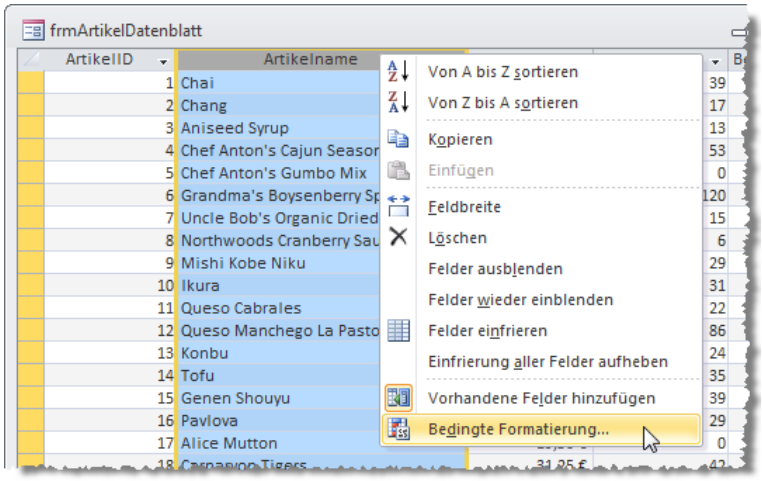



**Bild 2:** Unter Access 2010 öffnen Sie den Dialog einfach über das Kontextmenü der Spaltenüberschrift im Datenblatt.

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:

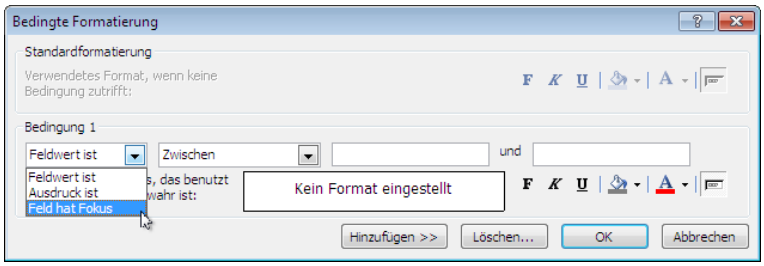



**Bild 3:** Bedingte Formatierung unter Access 2003 und älter anlegen

- **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.

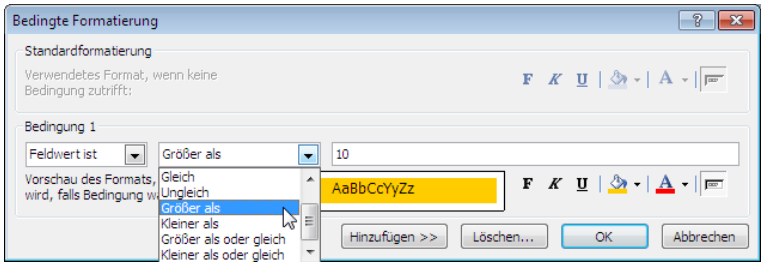



**Bild 4:** Vergleichsoperator der Bedingung

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).

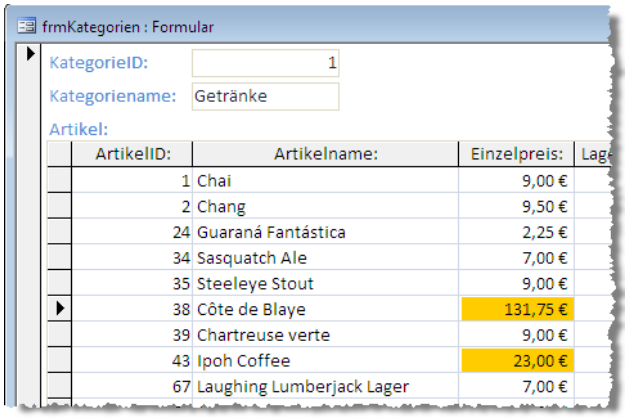



**Bild 5:** Bedingte Formatierung in Aktion

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.

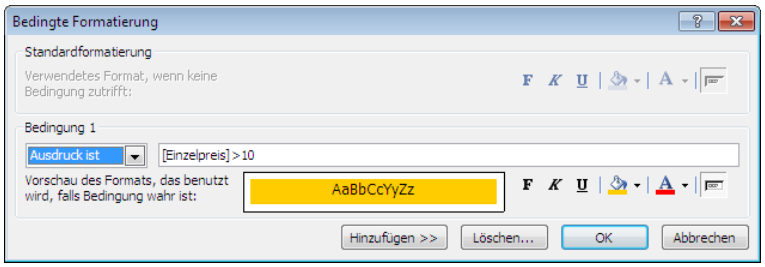



**Bild 6:** Ausdruck als Bedingung einer bedingten Formatierung

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.

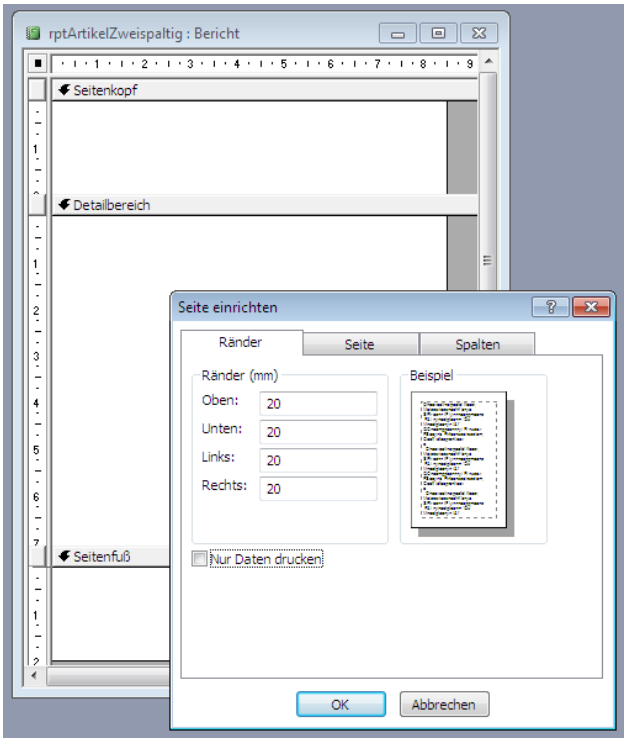



**Bild 1:** Einstellen der Seiten-Eigenschaften

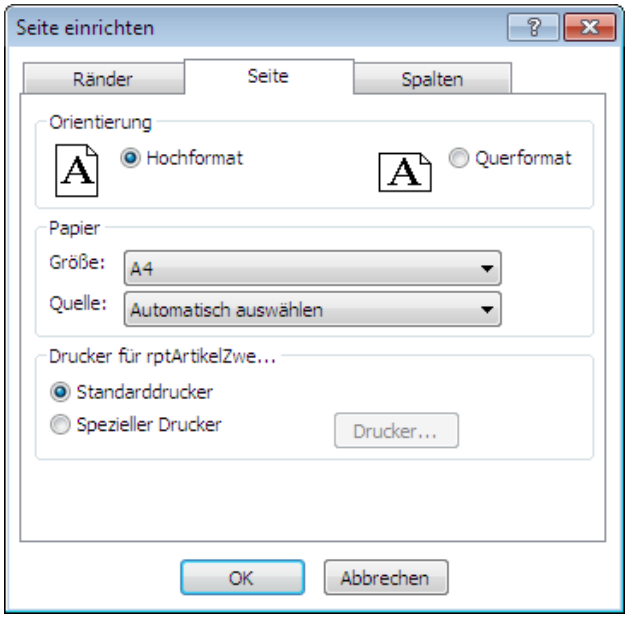



**Bild 2:** Weitere Seiteneigenschaften

## 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.

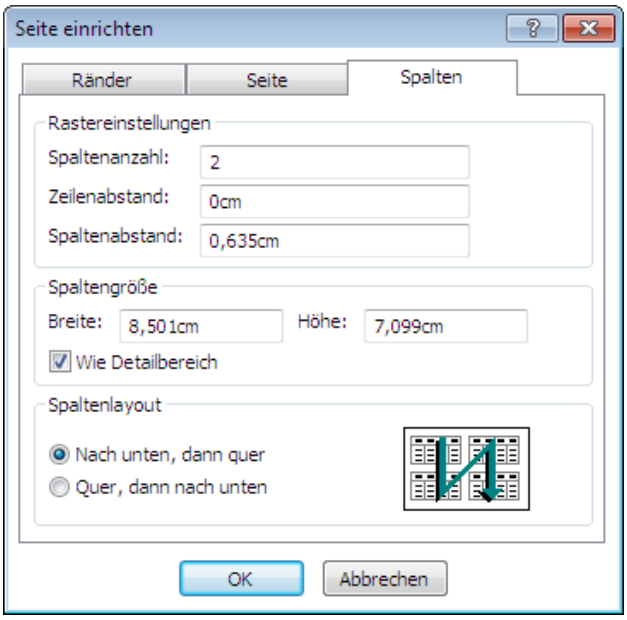



**Bild 3:** Spalten-Einstellungen des Berichts

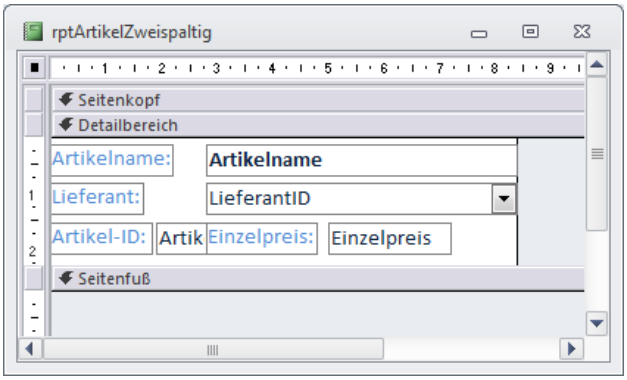



**Bild 4:** Entwurfsansicht des zweispaltigen Berichts

## 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-

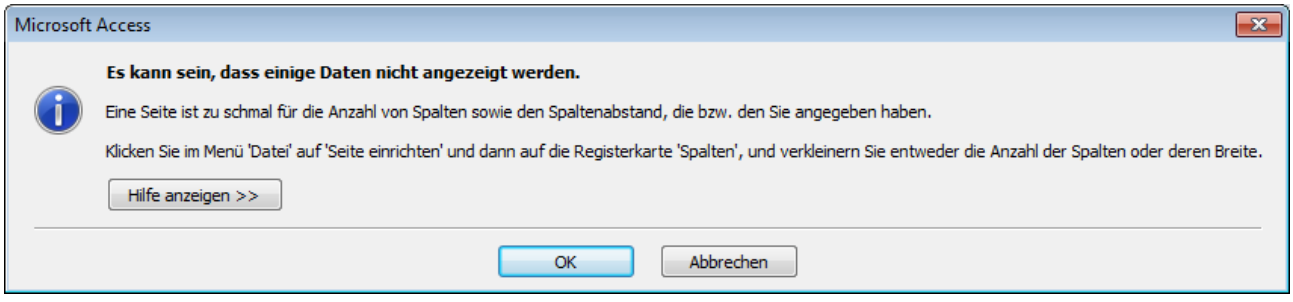



**Bild 5:** Diese Meldung erscheint, wenn der Bericht zu schmal für den anzuzeigenden Inhalt ist.

# 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

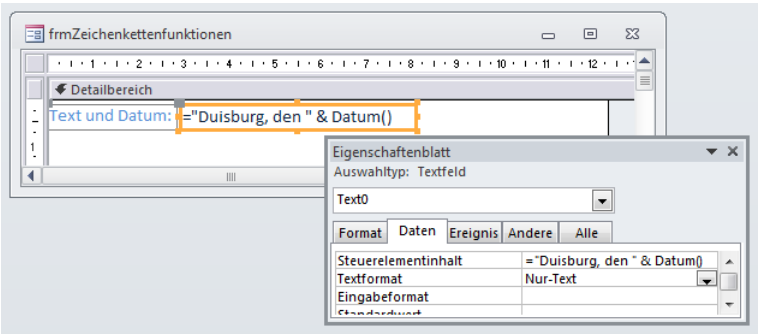



**Bild 1:** Zusammensetzen eines Literals und einer VBA-Funktion




**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

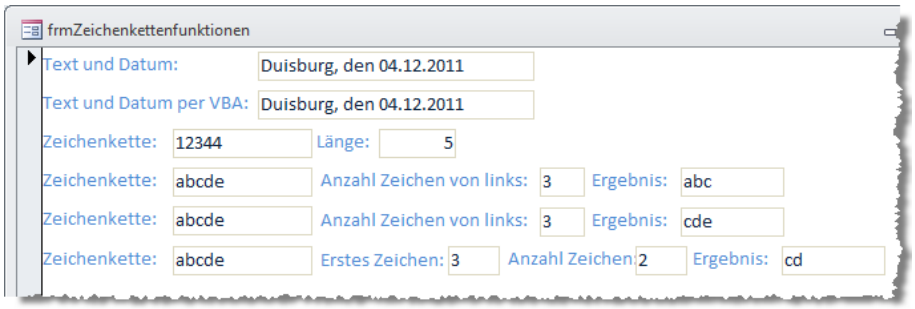



**Bild 3:** Beispiele für Zeichenkettenfunktionen in Textfeldern

# Zeilenumbruch und Tabulator in Access

**Wenn Sie einen Text in Word oder auch in ein Textfeld einer Anwendung eingeben, verwenden Sie Tasten wie die Eingabe- oder die Tabulator-Taste, um Absätze oder Einrückungen zu markieren. Wenn Sie solche Texte per VBA oder in sonstigen Ausdrücken zusammenstellen möchten, gelingt dies nicht mit der Eingabe- oder Tabulatortaste: Deren Betätigung wird im VBA-Editor schlicht wie anderswo interpretiert. Also müssen spezielle Ausdrücke her, welche die Steuerzeichen im VBA-Code vertreten und bei der Anzeige der gewünschten Texte entsprechend ersetzt werden. Dieser Artikel stellt die möglichen Konstanten unter VBA vor.**

## Zeilenumbruch

Wenn Sie in einem VBA-Ausdruck einen Zeilenumbruch einfügen möchten, benötigen Sie dazu ein entsprechendes Steuerzeichen.

Genaugenommen sind dies normalerweise sogar zwei Zeichen: Nämlich eines für die Zeilenweiterschaltung und eines für das Verschieben an den Beginn der folgenden Zeile. Für beides gibt es zwei Methoden:

- mit der **Chr**-Funktion, mit der Sie alle Zeichen, auch Steuerzeichen, abbilden können
- mit Visual-Basic-Konstanten.

Einen vollen Zeilenumbruch erhalten Sie mit der Konstanten **vbCrLf** oder mit der Kombination der beiden Zeichen **Chr(13)** und **Chr(10)**. Hier sind Beispiele für das Direktfenster:

```
Debug.Print "a" & vbCrLf & "b"
a
b
Debug.Print "a" & Chr(13) & Chr(10) & "b"
a
b
```

Beide Ausdrücke liefern exakt das gleiche Ergebnis. In der Tat erreichen Sie mit den folgenden drei Varianten zumindest im VBA-Editor ebenfalls das Gleiche:

```
Debug.Print "a" & vbCrLf & "b"
a
b
Debug.Print "a" & vbCr & "b"
a
b
Debug.Print "a" & vbLf & "b"
a
b
```

Wenn Sie Texte jedoch etwa in einer Textdatei speichern und anderswo weiter verwerten möchten, sollten Sie zuvor genauer prüfen, wie die Zeilenumbrüche dort aussehen.Und wenn Sie nicht nur einen Zeilenumbruch, sondern eine leere Zeile abbilden möchten, fügen Sie einfach ein weiteres vbCrLf hinzu.

## Tabulator

Gegebenenfalls möchten Sie nicht nur Zeilenumbrüche, sondern auch Tabulatoren in den Text einfügen. Dann können Sie die Konstante **vbTab** verwenden. Alternativ setzen Sie die **Chr**-Funktion mit dem Wert **9** ein:

```
Debug.Print "a" & vbTab & "b"
a    b
```

Im VBA-Editor wird das Tabulator-Zeichen durch eine entsprechende Anzahl Leerzeichen ersetzt, aber wenn Sie eine solche Zeichenkette in einem Textfeld ausgeben, erhalten Sie einen richtigen Tabulator-Sprung.

## Von Chr zu vb...

Wenn Sie wissen möchten, welchen Zahlenwert Sie mit der **Chr**-Funktion einsetzen müssen, um eine bestimmte **vb...**-Konstante zu ersetzen, finden Sie dies mit der **Asc**-Funktion heraus. Diese erwartet eine Zeichenkette und liefert den entsprechenden ASCII-Code zurück:

```
Debug.Print Asc(vbCr)
 13
```

Allerdings verarbeitet diese Funktion immer nur das erste Zeichen des Ausdrucks, was dieses Beispiel belegt:

```
Debug.Print Asc(vbCrLf)
 13
```

# 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.

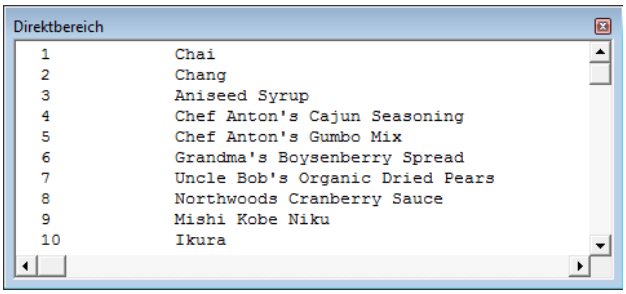



**Bild 1:** Ausgabe von Daten im Direktfenster

## 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

Access [basics] 12/2011
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Editorial

Geschafft! Das erste vollständige Jahr **Access [basics]** ist vorüber (2010 habe ich erst im Mai begonnen und nur acht Ausgaben veröffentlicht).

Ich bin wirklich sehr glücklich, dass Sie das Magazin so gut annehmen und mir soviel positives Feedback liefern. Ich hoffe auch, dass die jetzige Darstellung der Artikel all Ihre Wünsche erfüllt:

- HTML-Format
- PDF zum Ansehen und Ausdrucken
- Beispieldatenbanken zum Download

In den nächsten Wochen und Monate werde ich die älteren Ausgaben Schritt für Schritt in das neue Format übertragen, sodass diese auch in den oben angegebenen Formaten vorliegen.

Was bietet die aktuelle Ausgabe? Zunächst einmal schauen wir uns das Steuerelement Listenfeld genauer an. In diesem Fall interessiert uns die Mehrfachauswahl, die zwar schnell aktiviert ist, aber die sich leider nicht allzu einfach auslesen lässt. Wie dies funktioniert, erläutert der Artikel **Mehrfachauswahl in Listenfeldern auslesen** ab Seite 2.

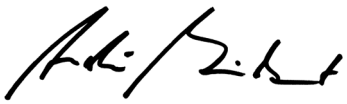
In der vorherigen Ausgabe haben wir uns erstmalig mit DAO-Recordsets beschäftigt. Dies setzen wir diesmal fort: Unter **DAO: Recordsets filtern, durchsuchen und sortieren** erfahren Sie ab Seite 6, wie Sie einmal geöffnete DAO-Recordsets durchsuchen, filtern oder sortieren oder dies gleich beim Öffnen durchführen.

Außerdem steigen wir in die SQL-Grundlagen ein: Der Artikel **SQL: SELECT-Abfragen, Teil I** erläutert ab Seite 10 die Basisformen der Auswahlabfrage. Für das Thema **SQL** haben wir eine eigene neue Kategorie namens **SQL-Grundlagen** eröffnet.

Schließlich finden Sie unter **Kunden per Bezeichnung verwalten** (siehe Seite 16) eine Lösung, mit der Sie demnächst zuverlässiger auch Kunden auswählen können, deren Namen doppelt in Ihrem Datenbestand vorliegen.

Ich hoffe, dass Sie ein schönes Weihnachtsfest im Kreise Ihrer Liebsten haben und gut ins neue Jahr kommen. Bleiben Sie gesund und zufrieden!

Und nun: Viel Spaß beim Lesen!



Ihr André Minhorst

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst | Fachverlag für Softwareentwicklung | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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).

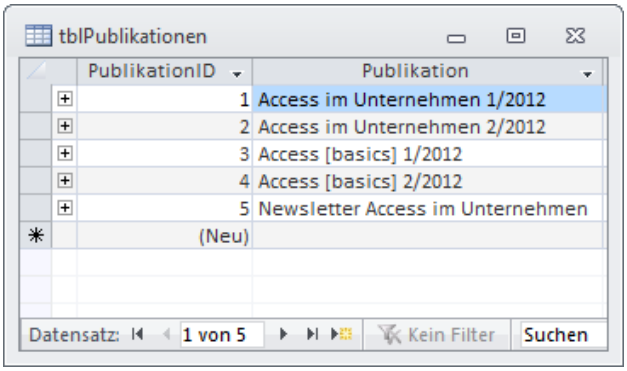



**Bild 1:** Tabelle **tblPublikationen** mit Beispieldaten

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.

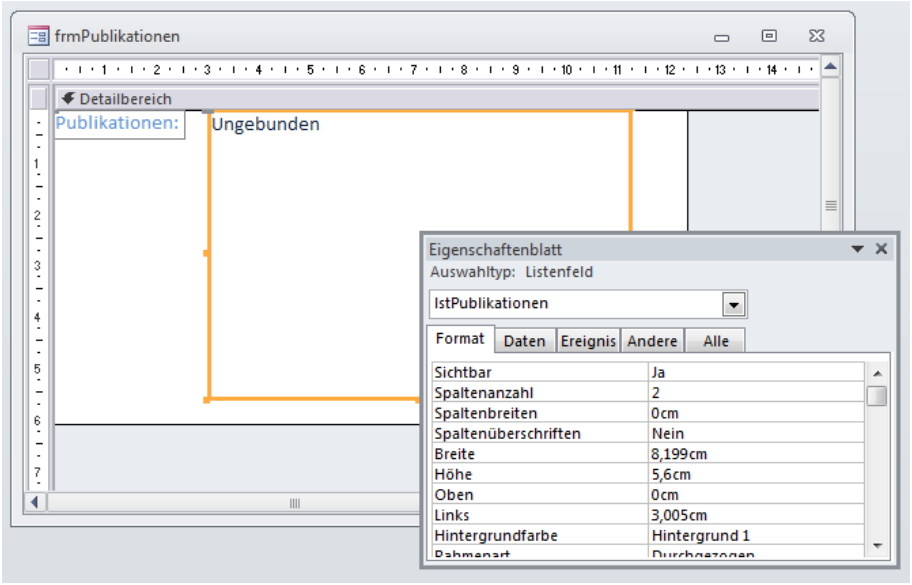



**Bild 2:** Das Formular **frmPublikationen** mit dem Listenfeld

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-

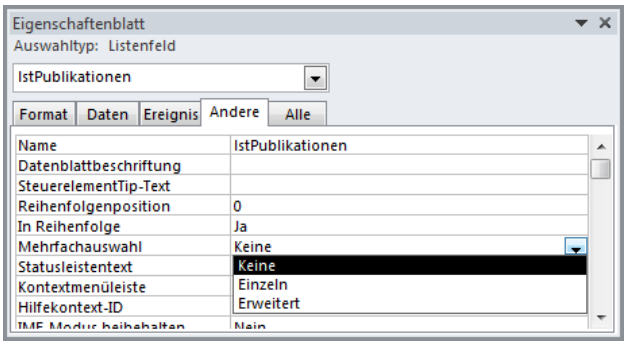



**Bild 3:** Einstellen der Eigenschaft **Mehrfachauswahl**

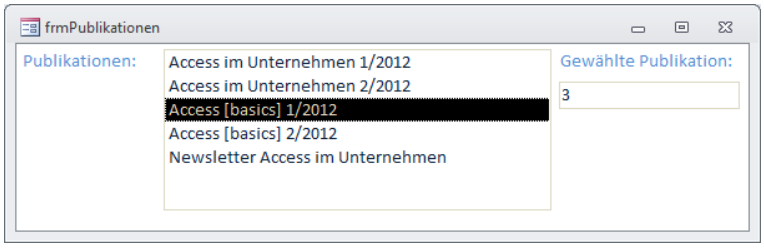



**Bild 4:** Anzeige der ausgewählten Publikation im Textfeld

# 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
```

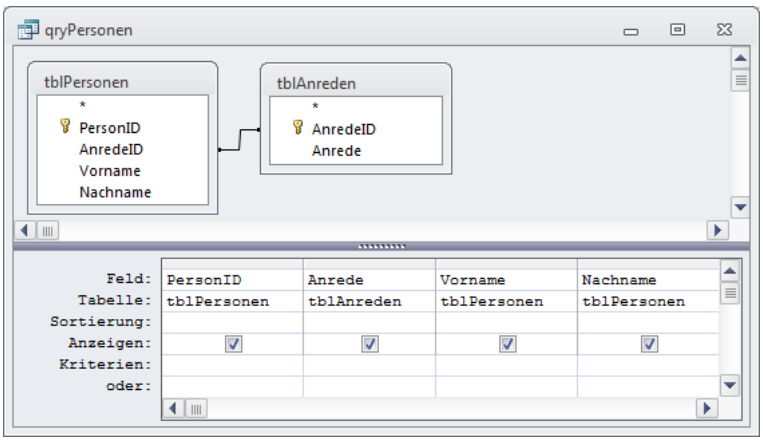



**Bild 1:** Zusammenführen von Personen und Anreden für ein Recordset

```
    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.

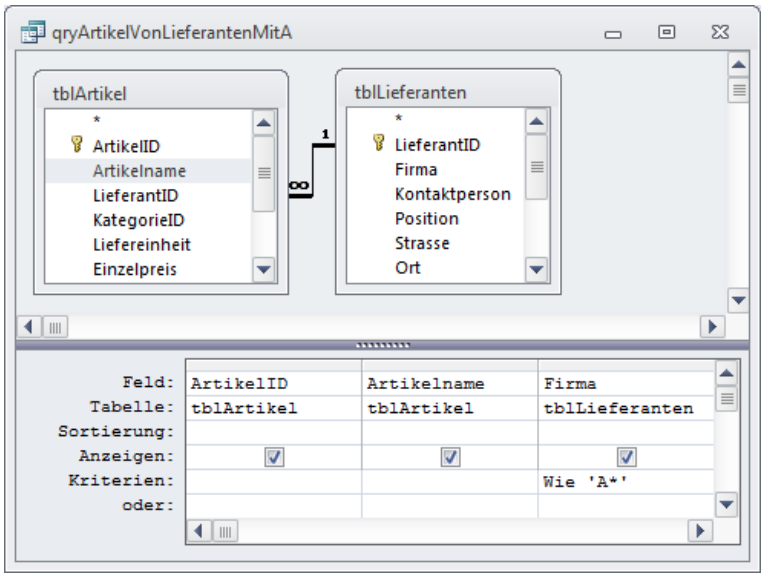



**Bild 1:** Abfrage mit zwei Tabellen und einem Kriterium in der Abfrage-Entwurfsansicht ...

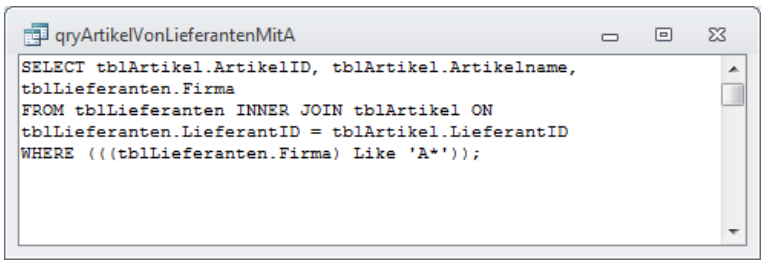



**Bild 2:** ... und in der SQL-Ansicht.

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).

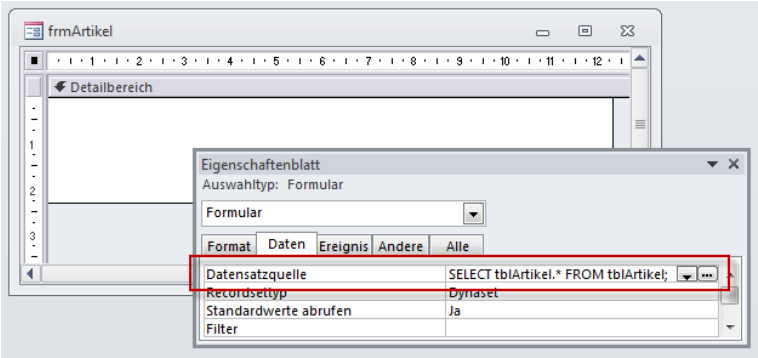



**Bild 3:** SQL-Ausdruck als Wert der Eigenschaft **Datenherkunft** eines Formulars

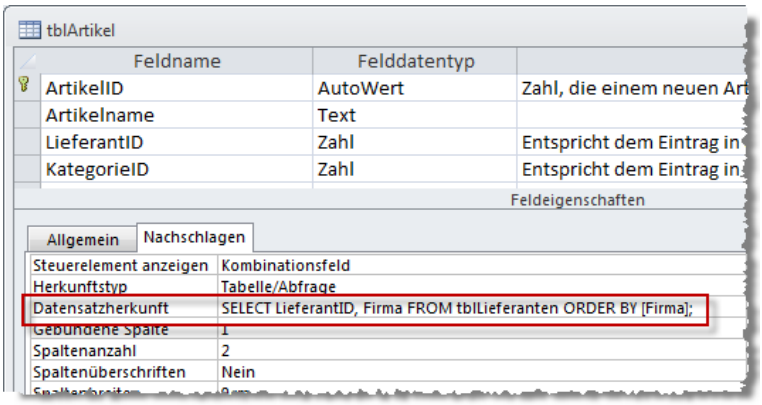



**Bild 4:** SQL-Ausdrücke in Nachschlagefeldern

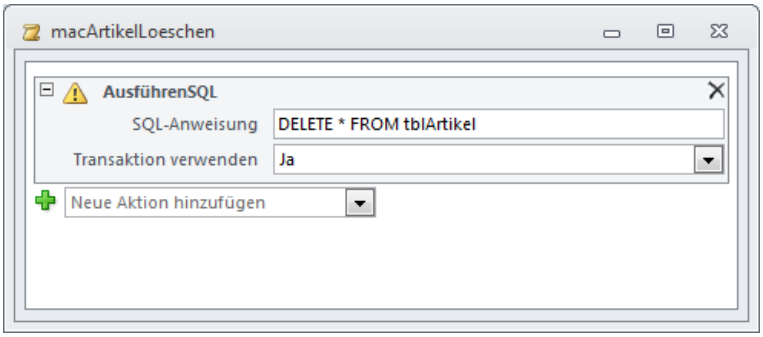



**Bild 5:** Aktionsabfrage per Makro

## 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

# 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.

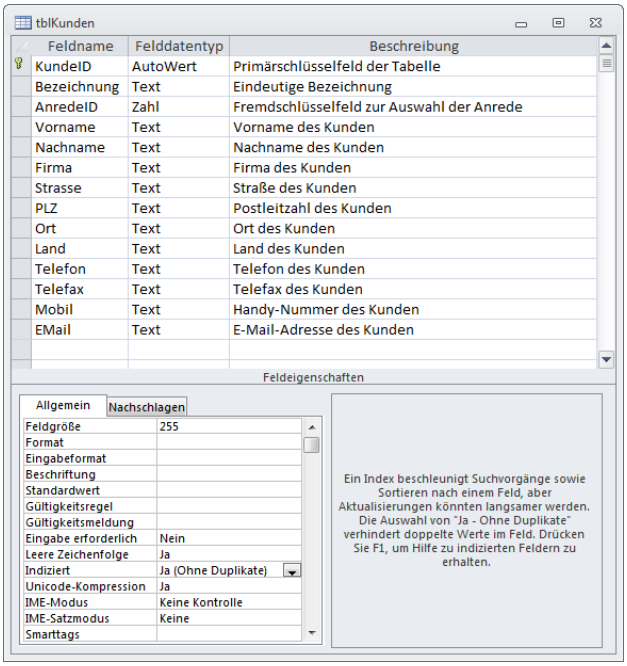



**Bild 1:** Kundentabelle mit Bezeichnungsfeld

## 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.

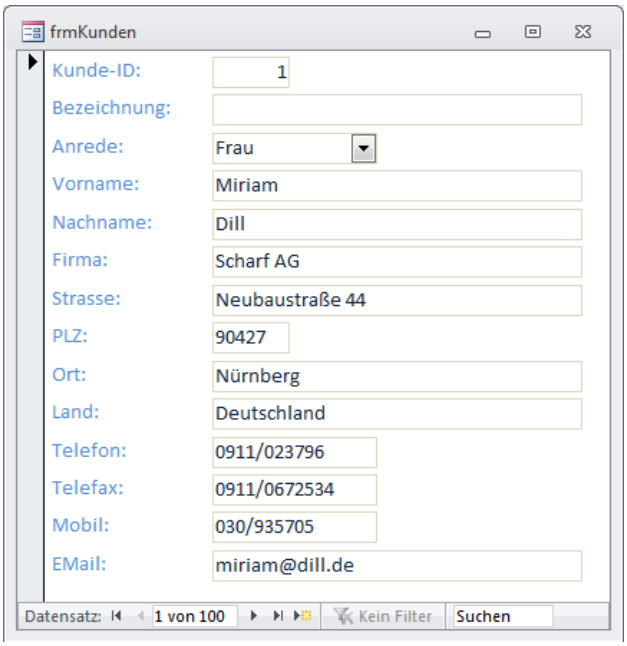



**Bild 2:** Formular zur Anzeige der Kundendaten

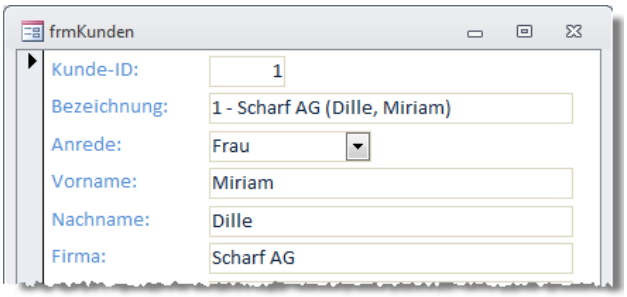



**Bild 3:** Kunde mit angepasster Bezeichnung

## 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

Access [basics] 1/2012
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Editorial

Das ist Negativ-Rekord: Mit dieser Ausgabe von **Access [basics]** liefere ich Ihnen nur schlappe drei Artikel! Dafür verteilen diese sich auf immerhin 22 Seiten, obwohl eigentlich nur 20 Seiten geplant sind. Das sind die Momente, in denen ich mich freue, dass **Access [basics]** ein reines PDF-Magazin ist und es dementsprechend nicht gedruckt werden muss. Sonst wäre es kompliziert geworden: Ich hätte die Artikel kürzen oder gar einen kompletten Artikel durch einen kürzeren Artikel ersetzen müssen, um die entsprechende Seitenzahl zu erreichen und die Druckkosten im Rahmen zu halten.

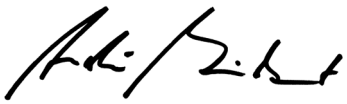
So aber macht es gar nichts aus, wenn ein Artikel mal eine Seite länger wird als geplant – ganz im Gegenteil: Sie bekommen alle Informationen, die Sie zum jeweiligen Thema benötigen.

Schauen wir doch mal rein, was die neue Ausgabe bietet! Der erste Artikel ist der zweite Teil der Artikelreihe **SQL: SELECT-Abfragen**, den wir in Ausgabe 12/2011 begonnen haben. Gleich ab Seite 2 erfahren Sie, wie Sie eine **SELECT**-Abfrage um Kriterien erweitern, mit denen Sie beispielsweise alle Artikel mit einem bestimmten Artikelnamen anzeigen können, warum der Vergleich von Feldern mit den Datentypen Text oder Datum eine Spezialbehandlung erfordert, welche Vergleichsoperatoren es gibt und so weiter.

Der zweite Artikel ist der Übeltäter, der den geplanten Rahmen dieser Ausgabe gesprengt hat: Unter dem Titel **Einfache Suchfunktion für Formulare** erfahren Sie ab Seite 8 in aller Ausführlichkeit, wie Sie die im Unterformular eines Formulars angezeigten Datensätze nach Kriterien filtern können, die Sie in entsprechende Steuerelemente im Hauptformular eingeben. Das Zusammenstellen der Filterkriterien übernimmt dabei eine VBA-Prozedur, die so konzipiert ist, dass Sie leicht Steuerelemente zum Filtern nach weiteren Feldern hinzufügen können.

Schließlich machen wir im Artikel **Arrays** ab Seite 19 mit den VBA-Grundlagen weiter. Array sind Datenfelder, mit denen Sie mehrere gleichartige Werte in einer einzigen Variablen speichern können. Der Artikel zeigt alle Möglichkeiten dieser Datenfelder: Wie fügen Sie dem Datenfeld Daten hinzu? Wie greifen Sie später auf die enthaltenen Werte zu? Wie erweitern Sie ein Datenfeld, wenn die angegebene Anzahl Elemente nicht ausreicht? In späteren Ausgaben finden Sie dann praktische Einsatzmöglichkeiten für Datenfeldern.

Nun wünsche ich einen guten Start ins neue Jahr und viel Spaß beim Lesen!



Ihr André Minhorst

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst | Fachverlag für Softwareentwicklung | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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"));
```

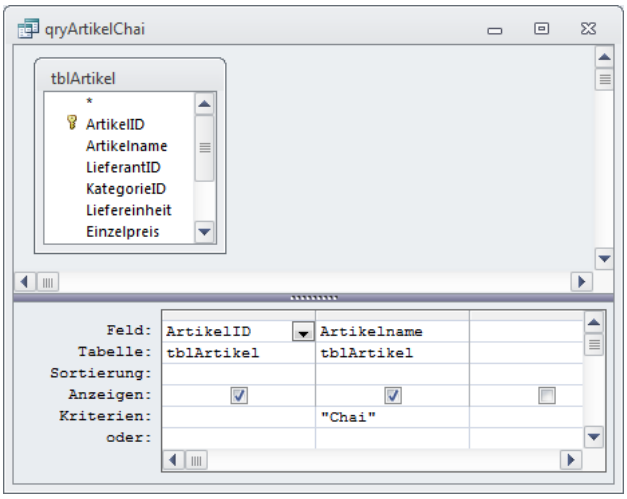



**Bild 1:** Eingeben von Abfragekriterien in der Entwurfsansicht einer Abfrage

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-

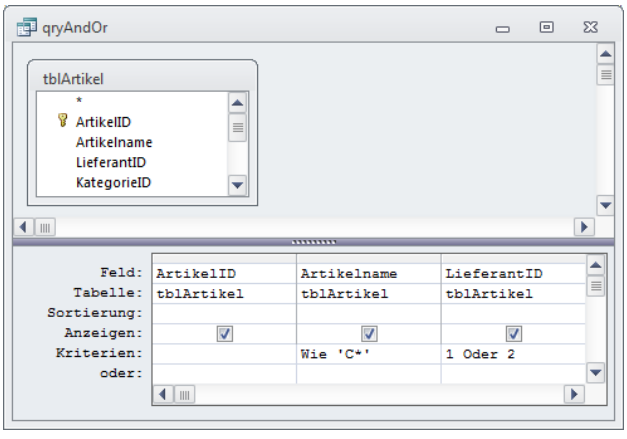



**Bild 2:** AND und OR in einer einzigen Abfrage

# 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.

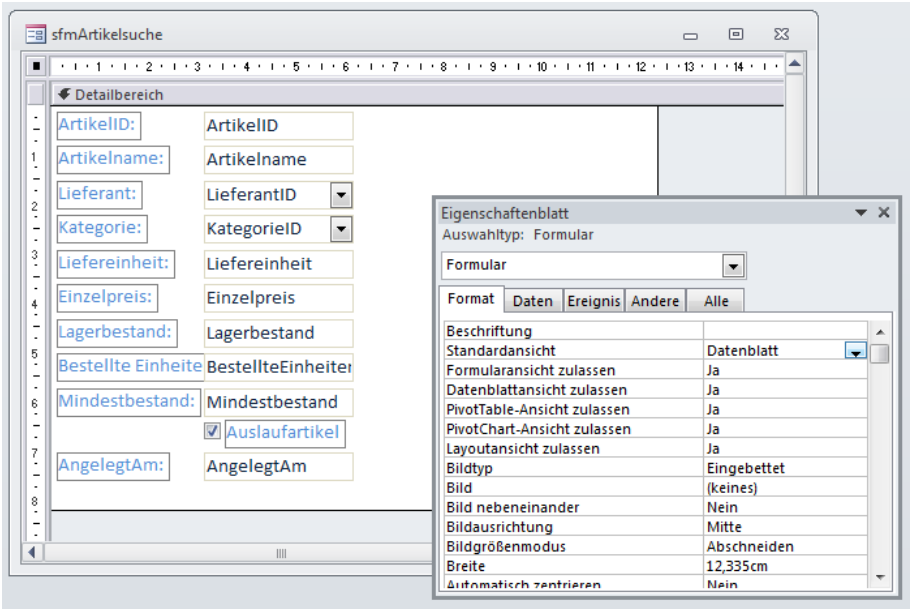



**Bild 1:** Entwurfsansicht des Unterformulars **sfmArtikelsuche**

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).

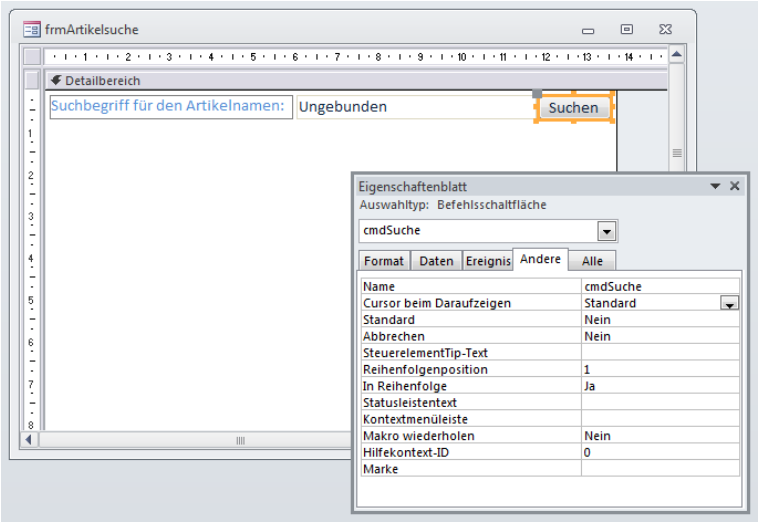



**Bild 2:** Hauptformular der Artikelsuche

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.

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.

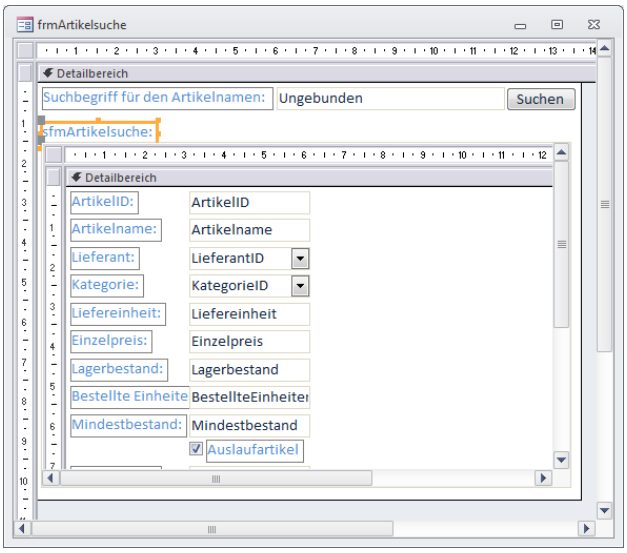



**Bild 3:** Haupt- und Unterformular in der Entwurfsansicht

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.

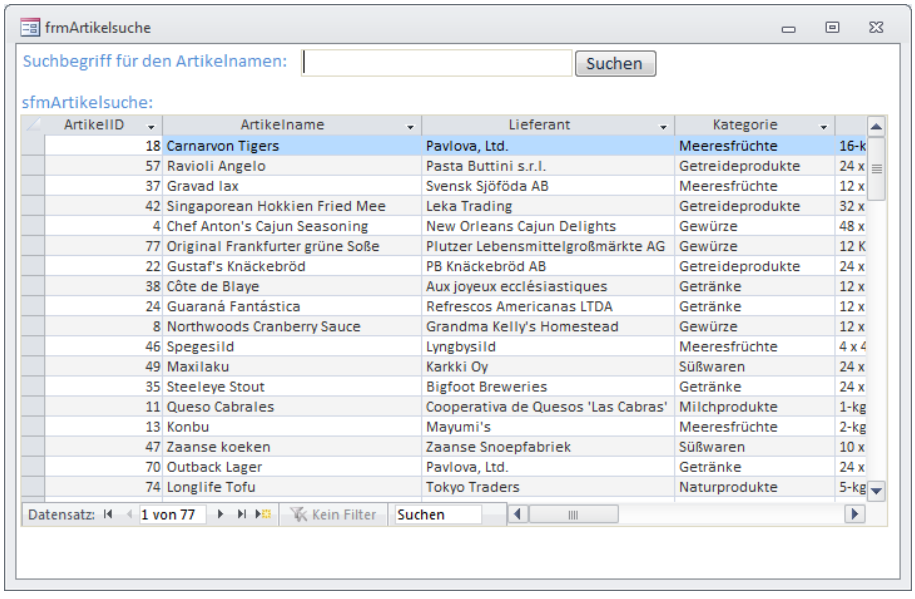



**Bild 4:** Ohne weitere Handgriffe zeigt das Unterformular einfach alle Datensätze an.

## Suchen per Filter

Die einfachste Möglichkeit, die Datensätze des Unterformulars durch die Eingabe eines Suchbegriffs einzu-

# 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.

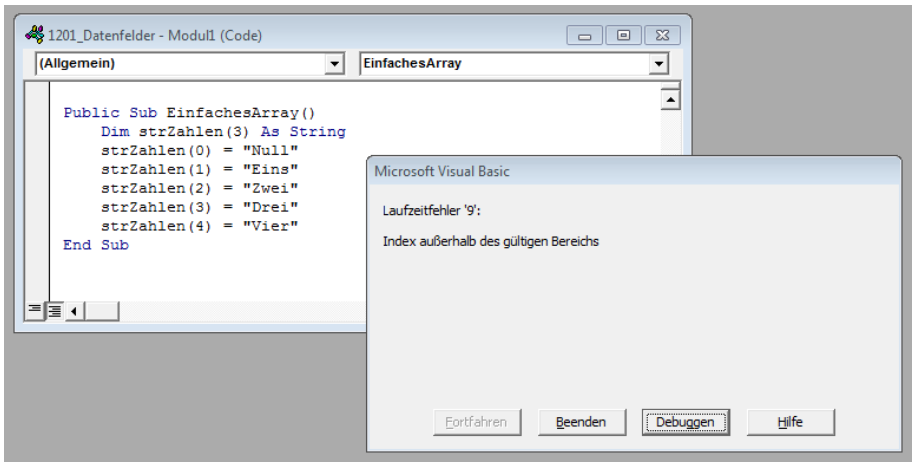



**Bild 1:** Fehler beim Versuch, einen nicht vorhandenen Eintrag im Array zu referenzieren

## 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
```

Access [basics] 2/2012
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

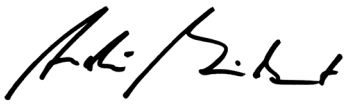
## Editorial

Mit der Einrichtung eines Editorials habe ich mir wirklich einen Bärendienst erwiesen. Eigentlich gibt es das Editorial nur, damit die erste Seite des Magazins neben dem Titel und dem Impressum nicht auch noch den Beginn des ersten Artikels der Ausgabe enthält. Ich wollte, dass Sie jeden Artikel ohne störende Elemente als Ganzes ausdrucken oder speichern konnten – also musste ich den verbleibenden Platz auf der ersten Seite anders füllen. Was eignet sich mehr dazu als ein Editorial! Zuletzt mündete das Füllen dieser guten halben Seite allerdings darin, dass ich beschrieben habe, was Sie, lieber Leser, auf den kommenden rund 20 Seiten erwartet. Sicher gut gemeint, aber diese übersichtliche Seitenmenge können Sie auch gut selbst durchblättern, um sich die aktuell interessanten Artikel rauszupicken und sich den Rest für später zu merken. Heute habe ich keine Lust, den Inhalt nochmal zu reproduzieren und schreibe mal, was ich aktuell so treibe – und wenn das für uns alle interessant ist, probiere ich das mal eine Weile aus.

Aktuell betreue ich beispielsweise ein interessantes Projekt – ein Zeitmanagement-Tool für ein kleines Unternehmen. Es bietet im wesentlichen zwei Listen, mit denen die Mitarbeiter erstens den Tagesverlauf planen und zweitens den tatsächlichen Verlauf eintragen können. Außerdem lassen sich mit dem Tool Urlaubs- und sonstige Fehlzeiten verwalten oder die Terminpläne von Mitarbeitern nebeneinanderstellen, um Termine für gemeinsame Besprechungen zu ermitteln. Auch der Import vorhandener Termine aus Outlook ist möglich: So kann das Unternehmen weiterhin Outlook zur Vereinbarung von Terminen mit seinen Kunden nutzen und diese Termine einfach in das Tool zur Tagesplanung übernehmen.

Wenn das Zeitmanagement-Tool demnächst rundläuft, biete ich es vielleicht als Produkt an. Wie bei so vielen Projekten wäre es doch schade, wenn nur ein einziger Kunde davon profitiert! Sollten Sie selbst Projekte für Kunden durchführen: Haben Sie schon einmal darüber nachgedacht, sich die Rechte zu sichern, um das fertige Produkt selbst an weitere Kunden zu veräußern? Ich halte das für eine interessante Geschäftsidee. Auf diese Weise können Sie erstens ohne großen Aufwand noch etwas Folgegeschäft generieren. Zweitens können Sie dem Auftraggeber vielleicht ein attraktiveres Angebot machen und so potentiellen Konkurrenten ein Schnippchen schlagen.

In diesem Sinne: Viel Erfolg bei Ihren anstehenden Projekten!



Ihr André Minhorst

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst | Fachverlag für Softwareentwicklung | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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.

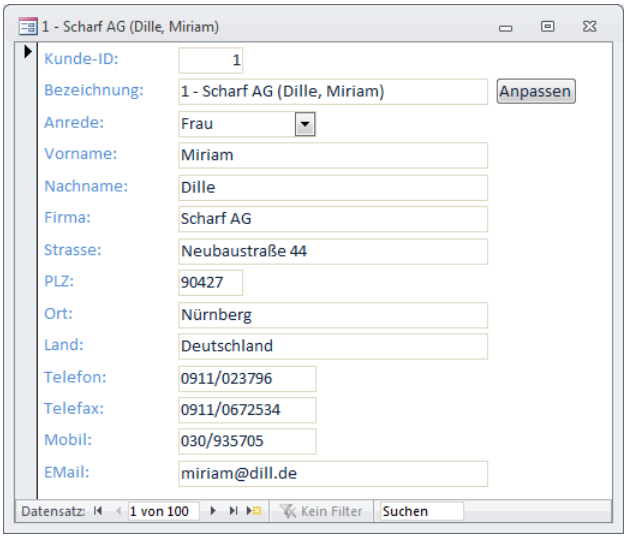



**Bild 1:** Einfache Variante: Die Kommunikationsdaten werden einfach in der Kundentabelle gespeichert.

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-

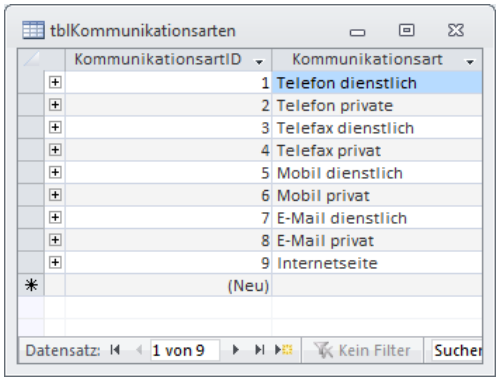



**Bild 3:** Die Tabelle **tblKommunikationsdaten** mit einigen Beispieldatensätzen

felder namens **KommunikationsartID** und **KundeID**, mit denen die ein Kunde mit einem Eintrag der Tabelle **tblKommunikationsarten** zusammengeführt wird.

tblKommunikationsdetails

| 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.

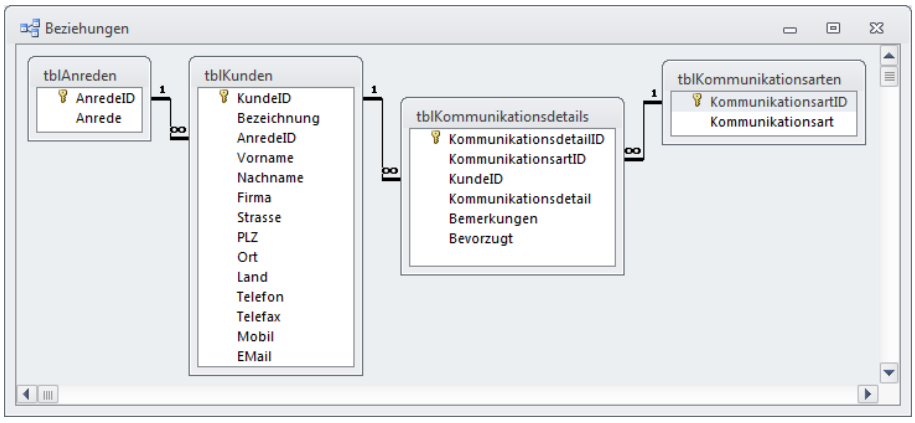



**Bild 5:** Das Datenmodell verdeutlicht die Zusammenhänge zwischen den Tabellen zur Verwaltung der Kommunikationsdaten.

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:

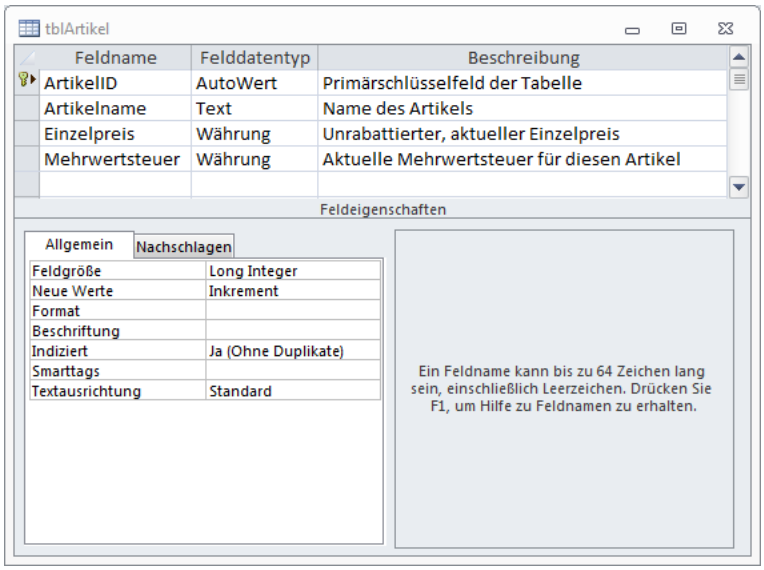



**Bild 1:** Entwurf der Tabelle **tblArtikel**

- **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.

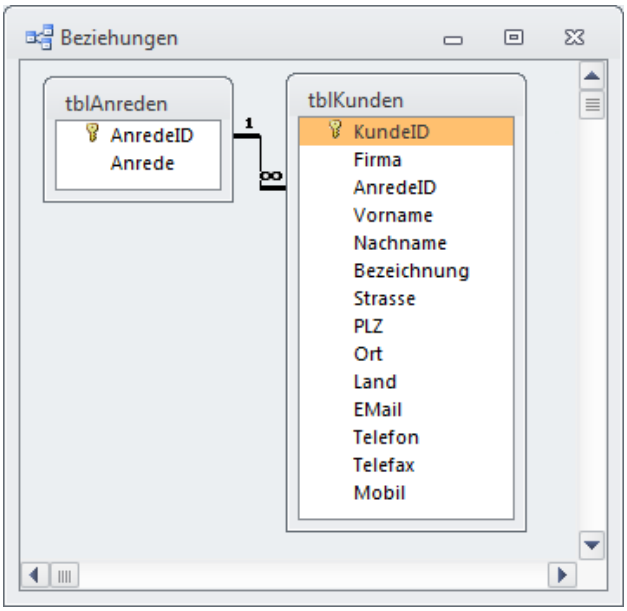



**Bild 2:** Die beiden Tabellen **tblKunden** und **tblAnreden** enthalten die Kundendaten.

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.

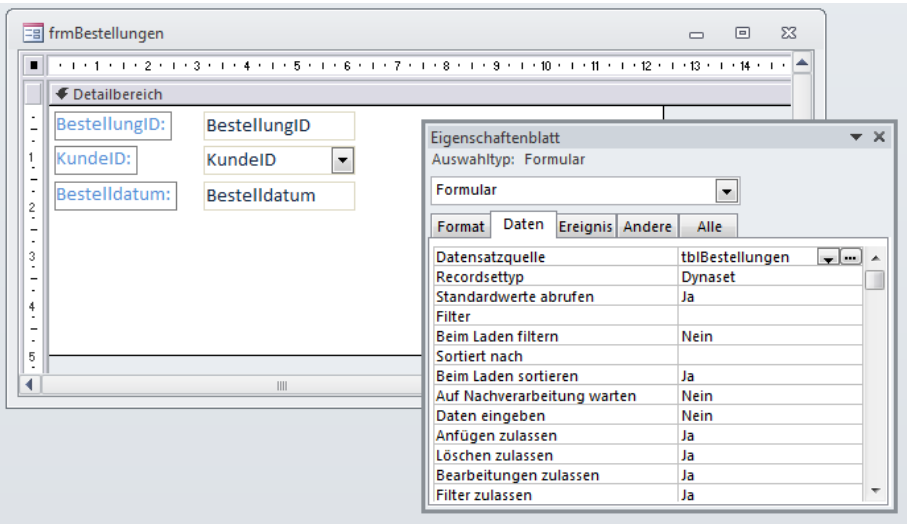



**Bild 1:** Hauptformular zur Eingabe der Bestelldaten

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

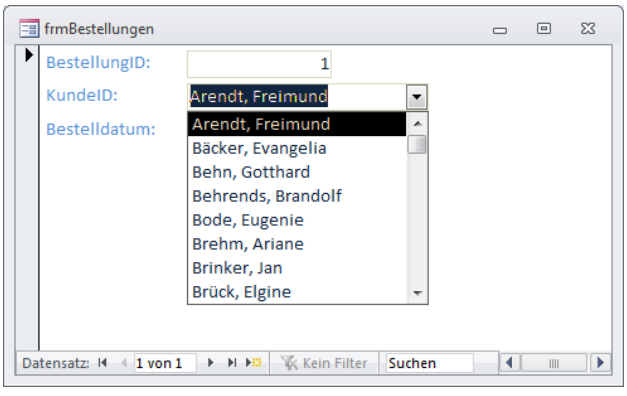



**Bild 2:** Auswahl des Kunden per Kombinationsfeld

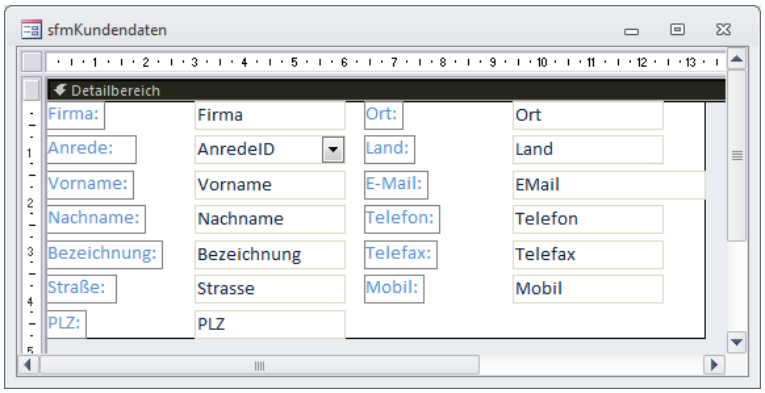



**Bild 3:** Unterformular zur Anzeige der Kundendaten

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,

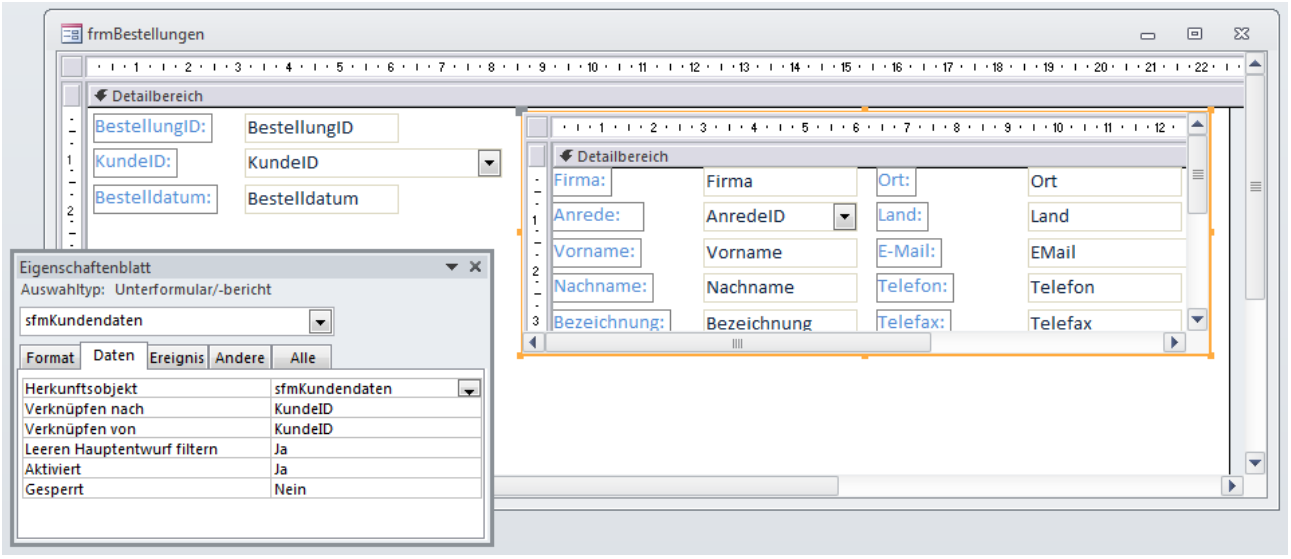



**Bild 4:** Herstellen der Verknüpfung zwischen Haupt- und Unterformular

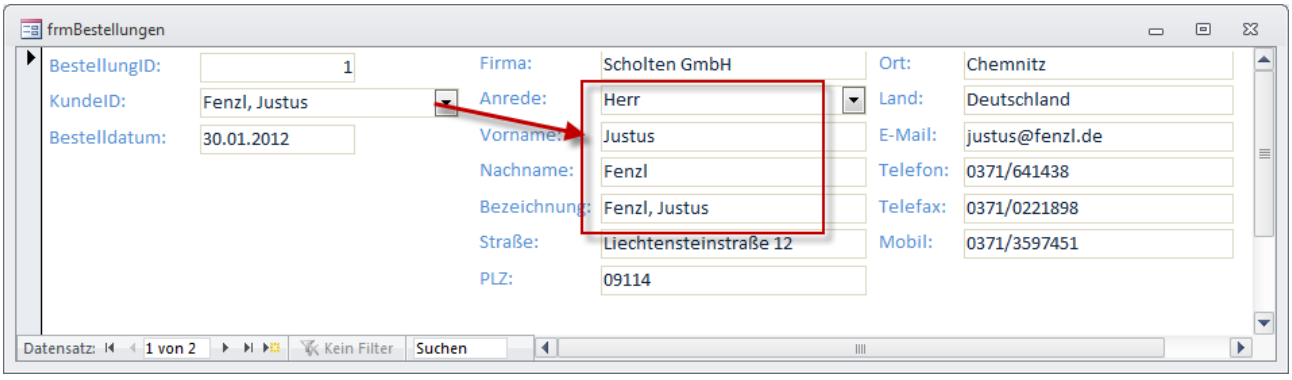



**Bild 5:** Synchronisieren des ausgewählten Kunden mit dem Unterformular **sfmKundendaten**

# 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:

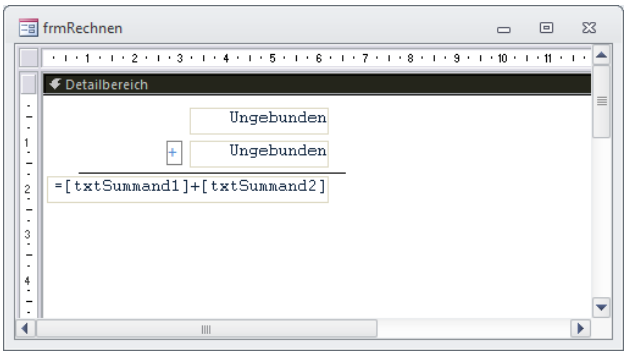



**Bild 1:** Summieren zweier Textfelder

```
=[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:

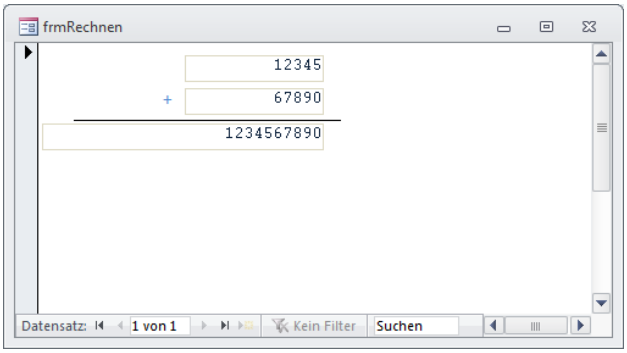



**Bild 2:** Zeichenverkettung statt Addition

```
=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).

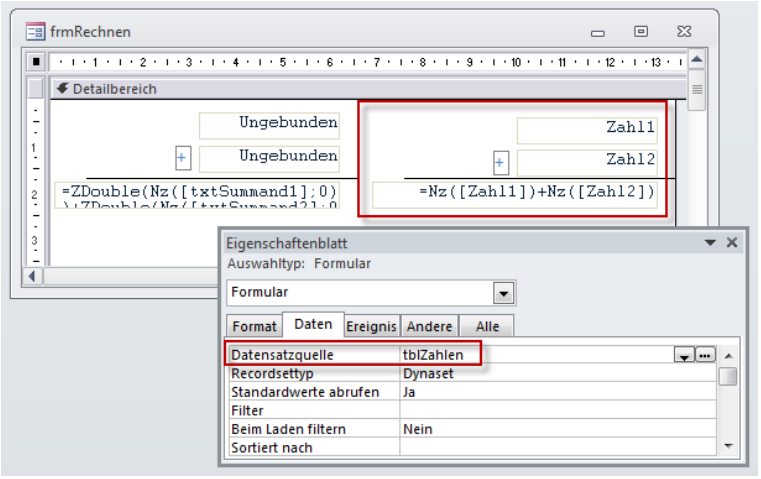



**Bild 3:** Beispiel für die Addition gebundener Felder

## 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.

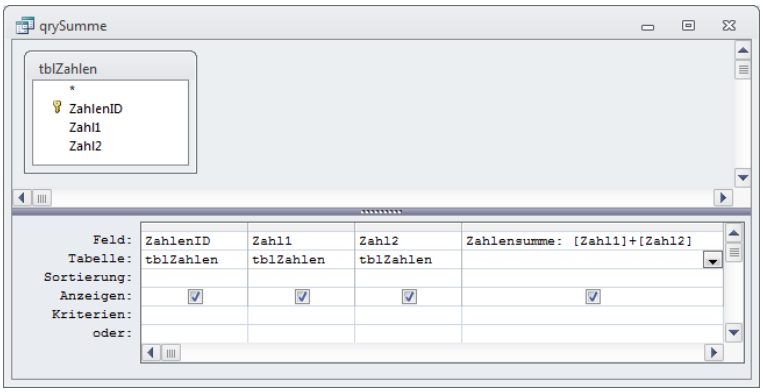



**Bild 4:** Addition per Abfrage

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.

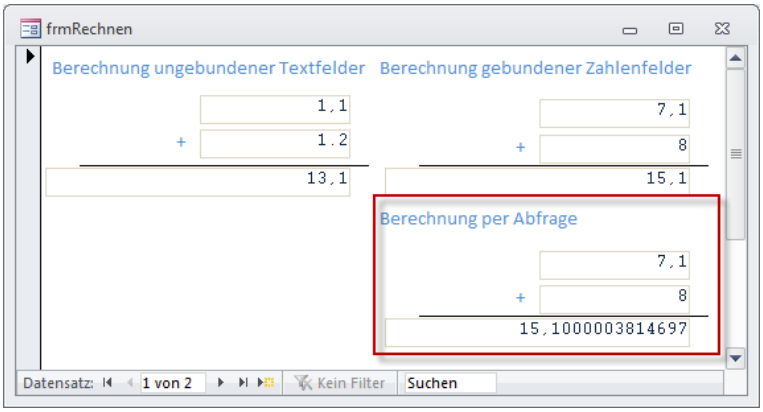



**Bild 5:** Addition per Abfrage im Formular

## 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

Access [basics] 3/2012
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Editorial

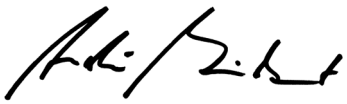
In dieser Ausgabe beleuchten wir ausnahmslos das ansonsten recht stiefmütterlich behandelte Thema Berichte. In gleich vier Artikeln erhalten Sie einige Grundlagen für das tägliche Reporting. Dabei geht es um das Rechnen in Berichten, die Gruppierung von Daten nach verschiedenen Kriterien und das Umwandeln von Berichten in PDF-Dokumente. Schließlich gibt es im Tipps-und-Tricks-Artikel noch ein paar kleine Helferlein.

Wie im letzten Editorial erwähnt, erfahren Sie hier regelmäßig, was ich in der Zwischenzeit so getrieben habe. Neben der Forschung und dem Schreiben von Artikeln für die Magazine (neben diesem hier noch Access im Unternehmen) gab es ein paar konkrete Projekte.

Da war zunächst einmal die Fortsetzung des laufenden Projekts, das nun offiziell **Time-Admin** heißt und die Planung des Tagesablaufs unterstützt. Hier zeigte sich, dass eine Anwendung, die auf dem eigenen Rechner problemlos arbeitet, ihren Dienst auf dem Kundenrechner noch lange nicht nach Plan verrichtet. Da gibt es Parallelinstallationen von Office-Paketen, Runtime-Versionen, andere Netzwerk-Gegebenheiten und vieles mehr. Da ist es schon sinnvoll, die eine oder andere Konstellation von Betriebssystem, Office-Paket und Access-Version auf virtuellen Maschinen zu betreiben, um problematische Szenarien nachzustellen und Fehler reproduzieren zu können. Für den Zugriff auf Betriebssysteme und Office-Pakete ist ein Technet-Abonnement eine gute Quelle: Sie können damit alle halbwegs aktuellen Produkte von Microsoft zu Evaluierungszwecken herunterladen und einsetzen. In manchen Fällen hilft auch das nicht mehr und man muss sich die Situation auf dem Kundenrechner ansehen. Mittlerweile braucht man dazu aber nicht mehr zum Kunden zu fahren, sondern kann per Remote-Anwendung auf dessen Rechner schauen. Hier gibt es verschiedene Produkte, zum Beispiel **GotoMeeting** oder **Teamviewer**. Die Kosten für diese Software sind überschaubar – zumindest, wenn man die alternativ anfallenden Fahrtkosten dagegenstellt.

Im Rahmen dieses Projekts habe ich außerdem damit begonnen, ein Tool zum Vergleichen von Datenmodellen zu entwickeln. Damit ist es sehr einfach, Unterschiede zwischen zwei Versionen einer Datenbank auf Tabellenebene zu ermitteln. Im zweiten Schritt soll dieses Tool es ermöglichen, die Unterschiede abzugleichen, ohne die Daten zu verlieren.

Dazu demnächst mehr an dieser Stelle!



Ihr André Minhorst

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst | Fachverlag für Softwareentwicklung | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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**.

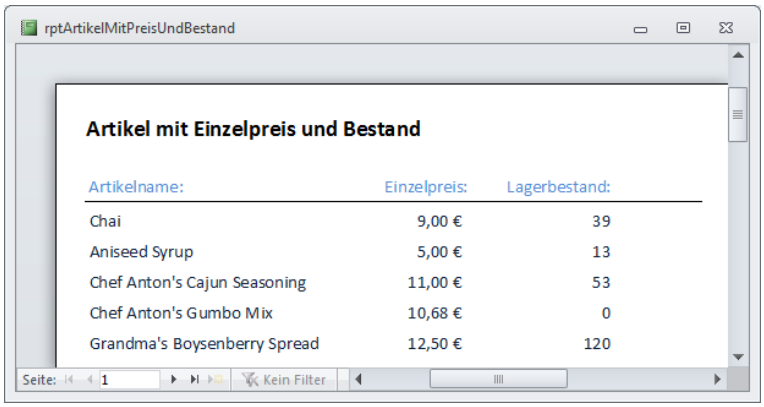



**Bild 1:** Grundlage für die Beispiele dieses Artikels: ein Bericht mit Artikeln, Einzelpreisen und Lagerbeständen.

## 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 3:** Eine weitere Zeile zeigt das Produkt aus Einzelpreis und Lagerbestand an.

## 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.

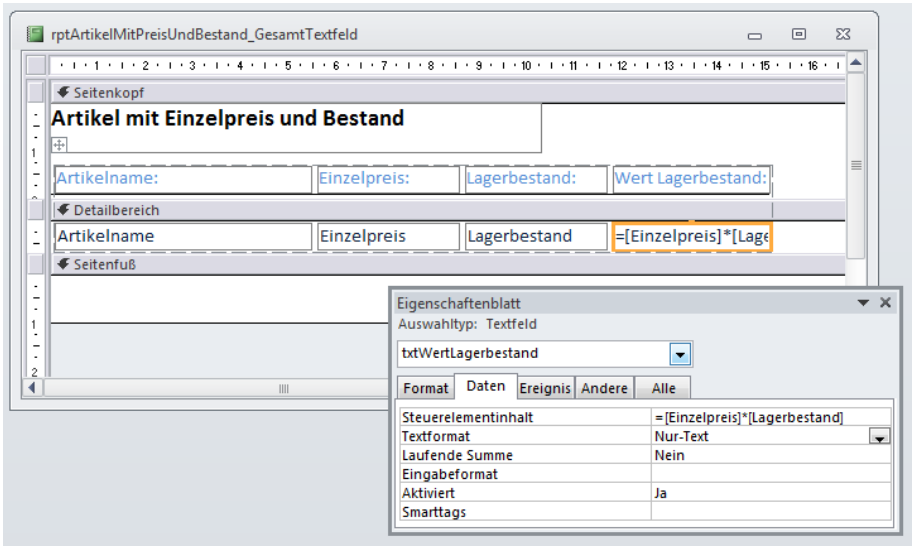



**Bild 2:** Hinzufügen eines Feldes zur Berechnung des Wertes der Artikel im Lager

## 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:

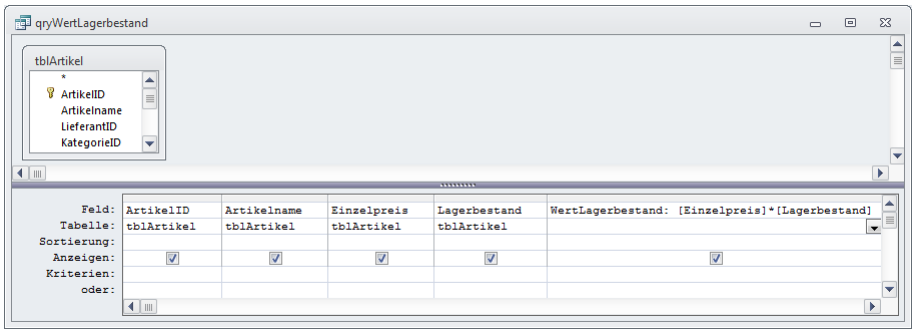



**Bild 4:** Abfrage zur Berechnung des Werts des Lagerbestands je Artikel

```
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.

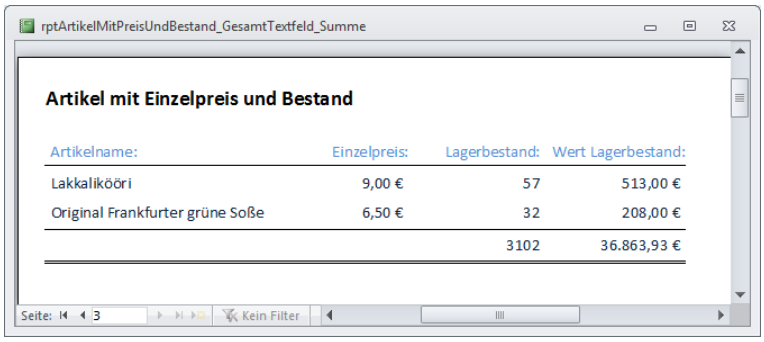



**Bild 6:** Summe der gelagerten Artikel und ihrer Preise

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.

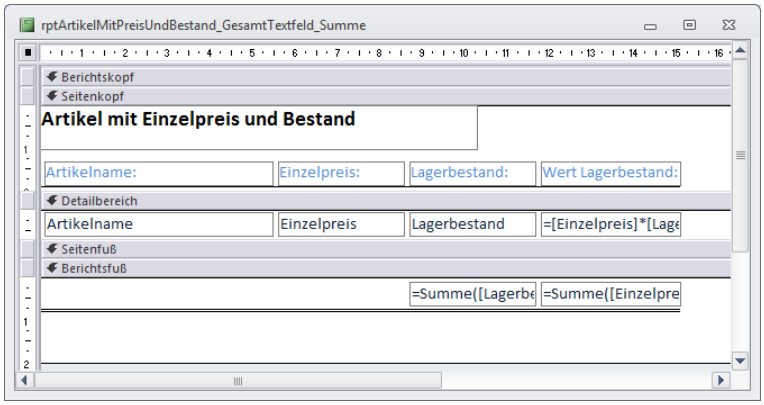



**Bild 7:** Entwurf des Berichts mit Summen im Berichtsfuß

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**.

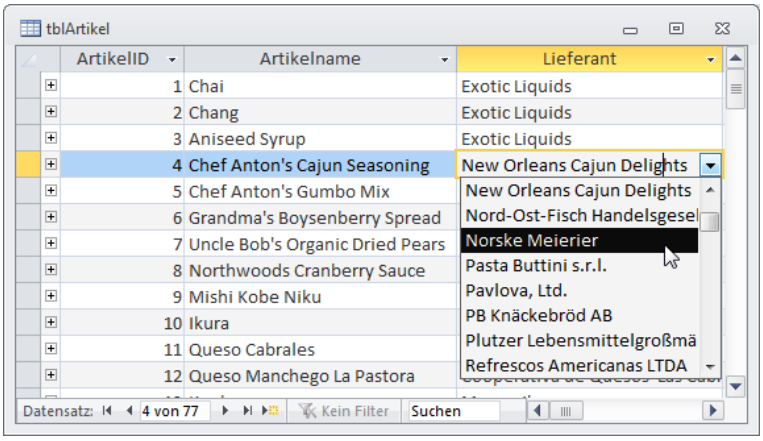



**Bild 1:** Diese Daten reichen, um Artikel nach Lieferant gruppiert darzustellen.

## 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

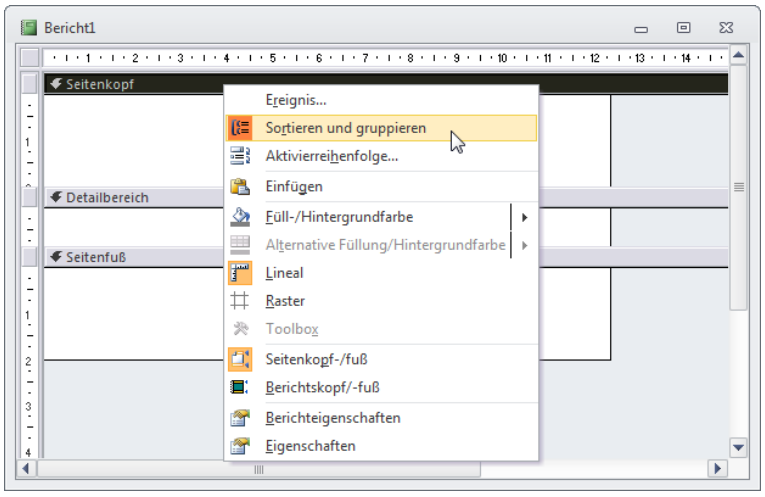



**Bild 2:** Diese Daten reichen, um Artikel nach Lieferant gruppiert darzustellen.

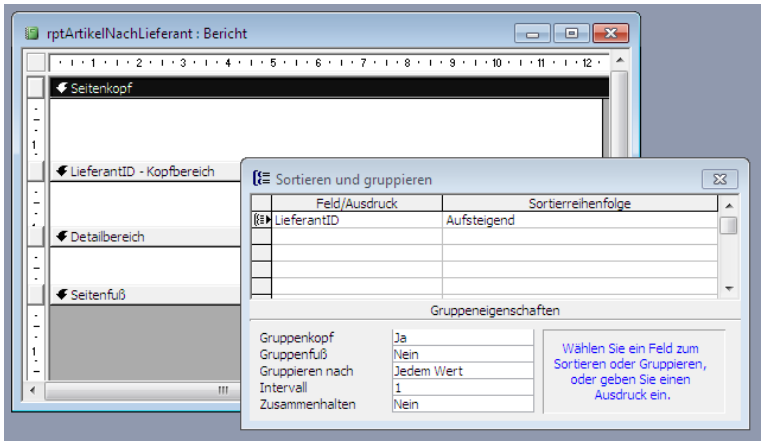



**Bild 3:** Der Dialog zum Anlegen und Anpassen von Gruppierungen unter Access 2003 und älter

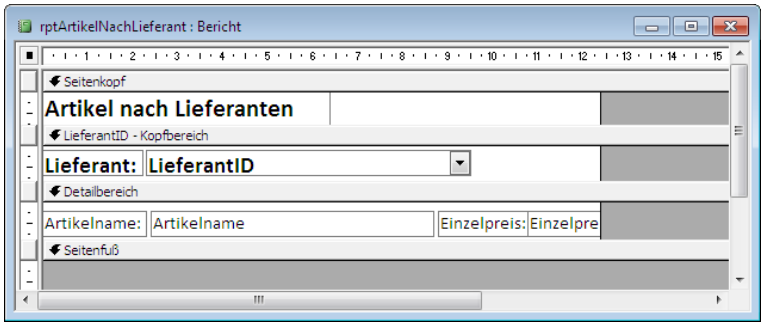



**Bild 4:** Entwurf eines Berichts mit einem einfachen Gruppenkopfbereich

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-

# 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.

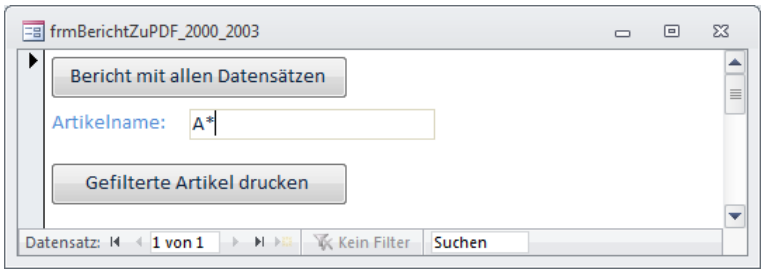



**Bild 1:** Formular zum Exportieren von Berichten im .pdf-Format

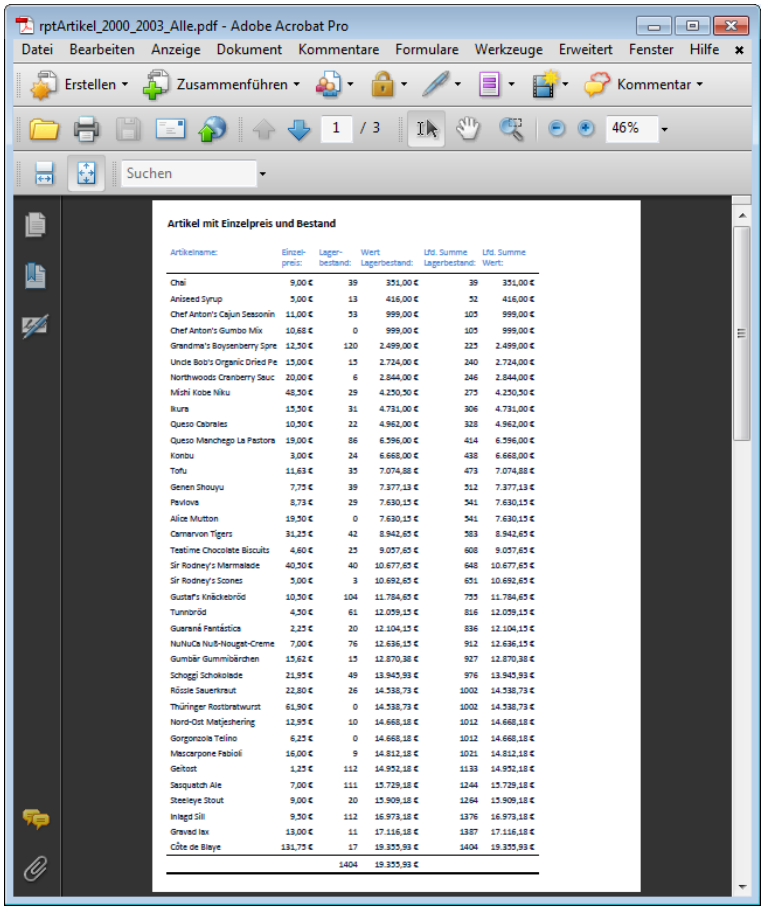



**Bild 2:** Mit Access 2000-2003 erzeugte PDF-Datei, basierende auf einem Access-Bereicht

Um die folgende Lösung einzusetzen, kopieren Sie also optimalerweise die beiden DLLs in das Verzeichnis Ihrer Anwendung und fügen dieser das Modul **mdlReportToPDF** aus unserer Beispieldatenbank hinzu.

Mit Access 2000 wurde mit dem **Snapshot**-Format (**.snp**) ein Export-Format für Berichte eingeführt. Der Konvertierungsalgorithmus basiert auf dem Ansatz, solch ein Snapshot zu erzeugen und diesen dann in eine PDF-Datei umzuwandeln.

Um dies zu erreichen, benötigen Sie prinzipiell nur einen einzigen Aufruf, der wie folgt aussieht:

```
ConvertReportToPDF "<Bericht>", , "<Dateiname>"
```

Für die Platzhalter **<Bericht>** und **<Dateiname>** fügen Sie die gewünschten Werte ein, wobei der Dateiname mit Pfad angegeben werden muss.

Die Beispieldatenbank enthält einen Bericht namens **rptArtikel** und ein Formular namens **frmBerichtZuPDF_2000_2003**. Letzterer bietet zunächst eine Schaltfläche an, mit der Sie den genannten Bericht in das PDF-Format exportieren können (siehe Bild 1). Diese Schaltfläche löst die folgende Prozedur aus:

```
Private Sub cmdAlleDatensaetze_Click()
    ConvertReportToPDF "rptArtikel", , _
        CurrentProject.path _
        & "\rptArtikel_2000_2003_Alle.pdf"
End Sub
```

Das Ergebnis überzeugt: Die PDF-Datei sieht genauso aus wie der Originalbericht (siehe Bild 2).

## Bericht mit WhereCondition exportieren

Leider gibt es einen kleinen Wermutstropfen: Die Funktion **ConvertReportToPDF** bietet keinerlei Möglichkeit, die Datenherkunft des Berichts anzupassen, um beispielsweise nur bestimmte Datensätze auszugeben.

Das ist jedoch kein Problem, da Sie mit **ConvertReportToPDF** auch bereits geöffnete Berichte ausgeben können. Das bedeutet, dass Sie den Bericht zunächst mit der **DoCmd.OpenReport**-Methode öffnen und die Datenherkunft dabei mit dem **WhereCondition**-Parameter filtern. Anschließend rufen Sie die **ConvertReportToPDF**-Methode auf und schließen den Bericht wieder. Wenn Sie nicht wollen, dass der Bericht in der Zwischenzeit angezeigt wird, können Sie die Anzeige mit dem Parameter **WindowMode** unterbinden. Als Beispiel für diesen Anwendungsfall finden Sie im Formular **frmBerichtZuPDF_2000_2003** zwei weitere Steuerelemente: Mit dem Textfeld geben Sie ein Filterkriterium für den Artikelnamen an, mit der Schaltfläche **Gefilterte Artikel drucken** lösen Sie die Ereignisprozedur aus Listing 1 aus.

# Tipps und Tricks zu Berichten

**Berichte bringen einige Eigenschaften mit sich, die man kennen sollte, wenn man optisch ansprechende und funktionale Berichte erstellen möchte. Diese stellen wir im vorliegenden Artikel vor. Sie erfahren hier beispielsweise, wie Sie Duplikate ausblenden, wenn mehrere Datensätze hintereinander teilweise gleiche Daten enthalten. Oder wie Sie den Raum für leere Textfelder freigeben, damit der übrige Inhalt nach oben rutschen kann. Außerdem zeigen wir Ihnen, wie Sie einen Seitenwechsel an der gewünschten Stelle unterbringen.**

## Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1203_TippsUndTricksZuBerichten.mdb**.

## Duplikate ausblenden

Wenn Sie in einem Bericht Daten in tabellarischer Form anzeigen, was ja meistens der Fall ist, kann es vorkommen, dass bestimmte Daten in jeder Zeile erscheinen. Ein Beispiel hierfür ist die Auflistung von Artikeln eines Lieferanten.

Der Beispielbericht aus Bild 1 zeigt, wie eine Liste von Artikeln unter Angabe des Lieferanten ohne weiteres Zutun aussieht – also wenn der Bericht schlicht und einfach die Werte der Datenherkunft anzeigt (in diesem Fall die Abfrage **qryArtikelMitLieferant**).

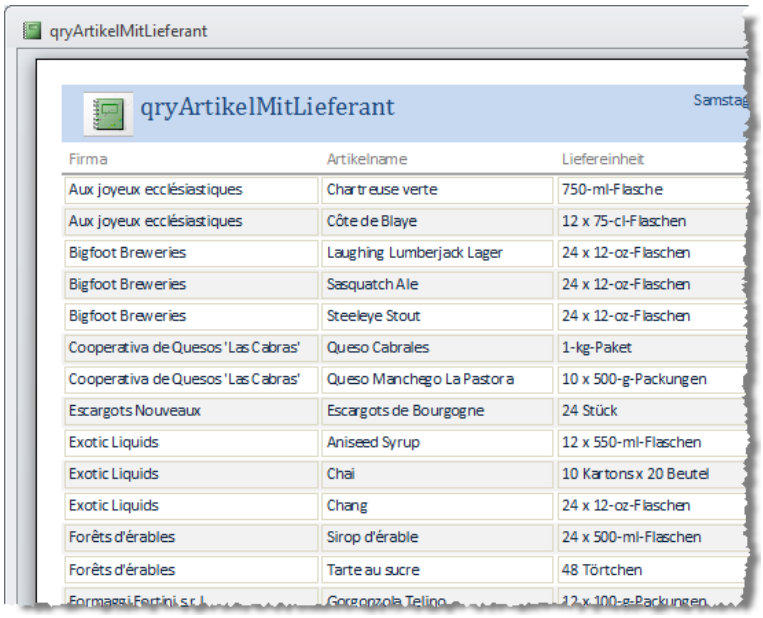



**Bild 1:** Lieferant und Artikel mit doppelt angegebenen Lieferanten

Die Datensätze sind zunächst nach dem Namen des Lieferanten und dann nach dem Namen des Artikels sortiert. Dies führt dazu, dass der Name des Lieferanten mehrfach untereinander angezeigt wird.

Um die Artikel eines Lieferanten deutlicher voneinander abzugrenzen, werden wir den Lieferantennamen nur noch beim ersten Auftreten anzeigen, in den übrigen Zeilen wird dieser ausgeblendet.

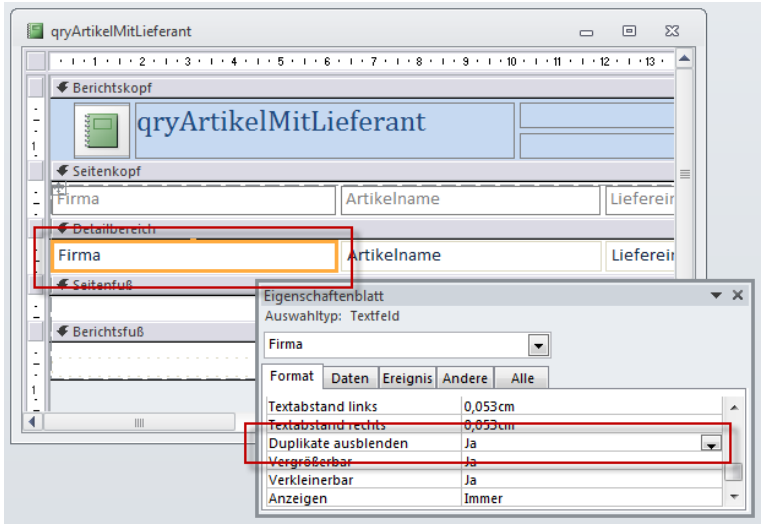



**Bild 2:** Einstellen der Eigenschaft **Duplikate ausblenden** auf den Wert **Ja**

Der Aufwand hierfür hält sich in Grenzen: Sie brauchen nur einmal in den Entwurfsbereich des Berichts zu wechseln (am schnellsten mit der Tastenkombination **Strg + Umschalt + ,** – ab Access 2007 zweimal zu betätigen), das Steuerelement zur Anzeige des Lieferanten zu markieren (Feldname **Firma**), mit der Taste **F4** die Eigenschaften einzublenden und die Eigenschaft **Duplikate ausblenden** auf den Wert **Ja** einzustellen (siehe Bild 2). Anschließend wechseln Sie mit **Strg + Umschalt + .** (ab Access 2007 zweimal betätigen) wieder zur Seitenansicht und erhalten die Darstellung aus Bild 3.

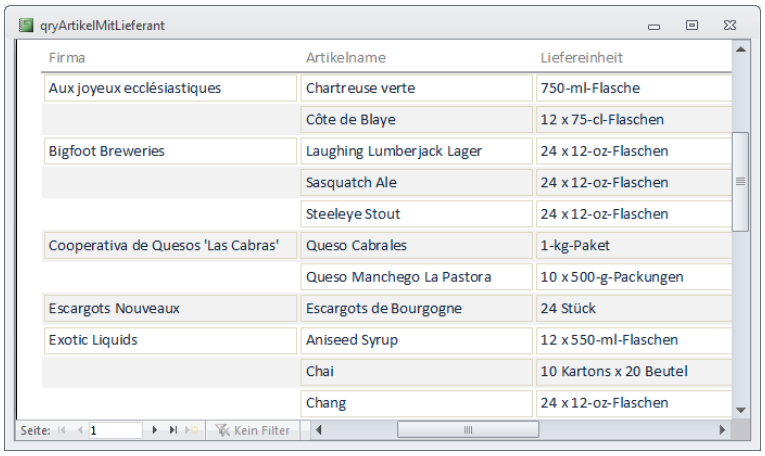



**Bild 3:** Artikelliste mit einfacher Anzeige des jeweiligen Lieferanten

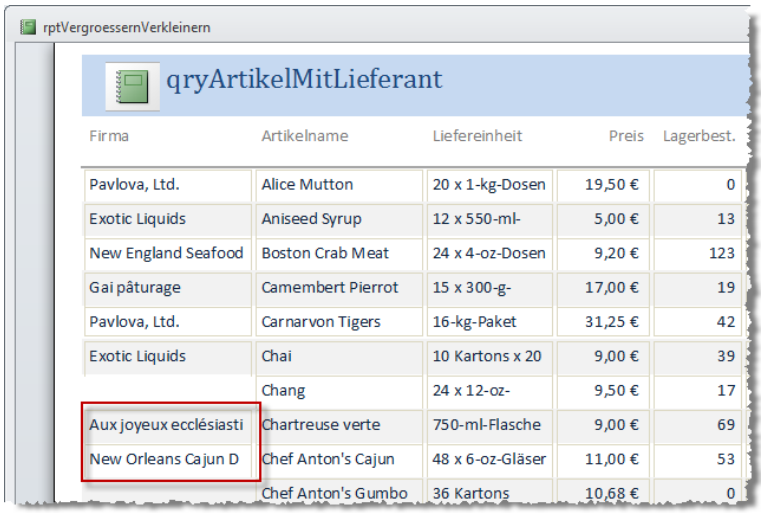



**Bild 4:** Zu lange Texte werden abgeschnitten.

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

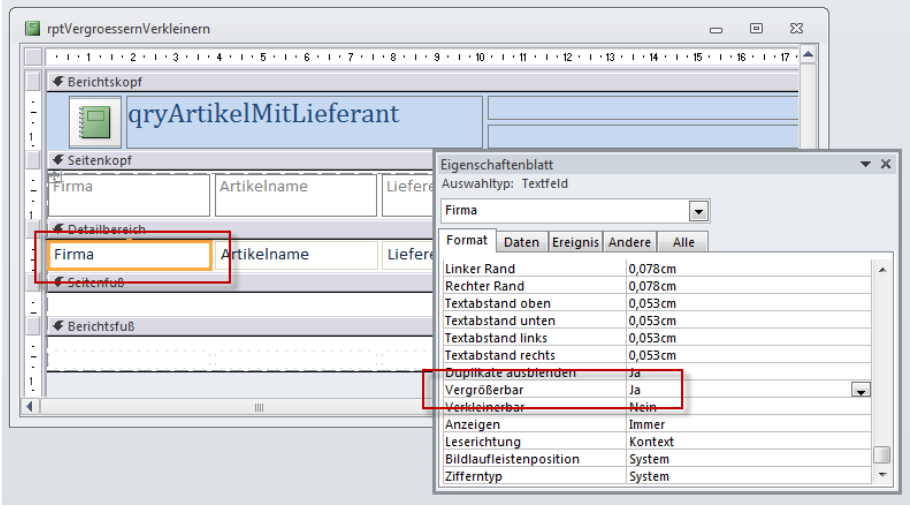



**Bild 5:** Zu lange Texte werden abgeschnitten.

Access [basics] 4/2012
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Editorial

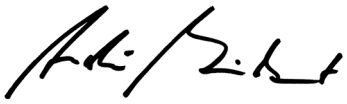
Die aktuelle Ausgabe widmet sich zum größten Teil der Formatierung von Daten. Sie erfahren, wie Sie Daten wie Datums- oder Währungsangaben in Formularen und Berichten formatieren, wie Sie die Format-Funktion von VBA einsetzen oder wie Sie Daten abhängig vom Wert hervorheben können. Außerdem zeigen wir, wie Sie die Mehrfachauswahl in Listenfeldern speichern und wiederherstellen können und starten eine Artikelreihe zum Thema Formularereignisse.

Und um das Editorial aufzufüllen, berichte ich mal wieder aus dem Leben eines Access-Entwicklers. Ein Kollege leitete mir einen Kunden weiter, der eine Anbindung seiner Access-Anwendung an den Webservice von UPS benötigte, um Label für den Versand von rund 1.000 Sendungen pro Monat direkt selbst drucken zu können – nach Erfassung im System von UPS, damit diese die Lieferung schnellstmöglich abholen können. Ein sehr interessantes Projekt: Erstens habe ich mittlerweile einige Erfahrung gesammelt, was den Zugriff auf die verschiedenen Webservices angeht. Dabei habe ich mittlerweile Anbindungen an eBay, Amazon, Paypal und Twitter programmiert, und jetzt auch noch für UPS.

Für **Access [basics]** kommt dieses Thema noch ein wenig zu früh, denn es gibt noch so viele Grundlagen, die hier behandelt werden wollen. Aber sollten Sie mal selbst eine Anbindung an einen Webservice programmieren wollen oder müssen – Sie wissen ja jetzt, wo Sie einen Experten finden. ;-)

Sollten Sie Interesse an Publikationen zu diesen Themen haben, finden Sie Einiges in **Access im Unternehmen** (**www.access-im-unternehmen.de**).

Was ist noch geschehen seit der letzten Ausgabe? Nun: Ich plane ein Buch, das die Entwicklung einer kompletten Anwendung beschreiben wird. Die Anwendung wird eine kombinierte Kunden-/Artikeldatenbank werden, mit der man sowohl seine Kunden, die Kommunikation mit diesen sowie Artikel und Bestellungen abwickeln kann. Leider habe ich noch keine einzige Seite geschrieben, was sich aber jederzeit ändern kann. Ich halte Sie an dieser Stelle auf dem Laufenden!



Ihr André Minhorst

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst | Fachverlag für Softwareentwicklung | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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**.

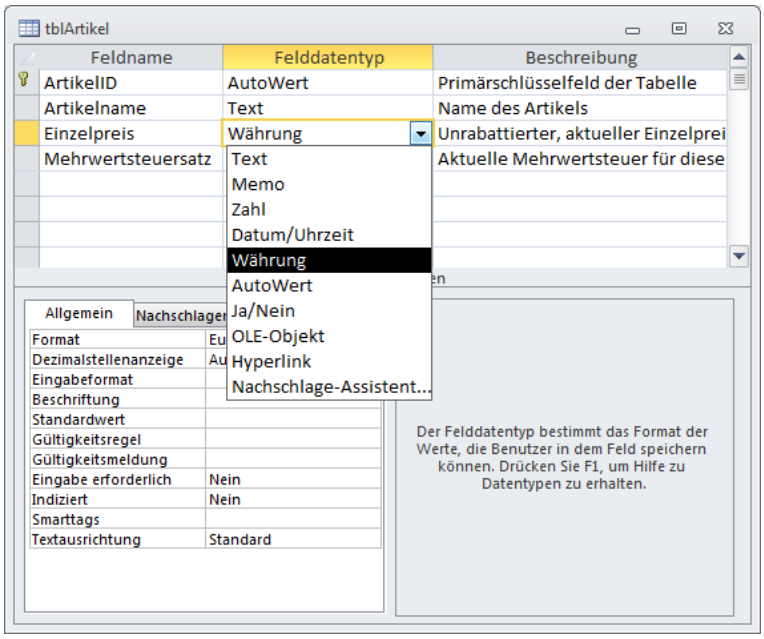



**Bild 1:** Die Auswahl des Felddatentyps stellt Feldeigenschaften wie Datentyp, Feldgröße und Format ein.

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

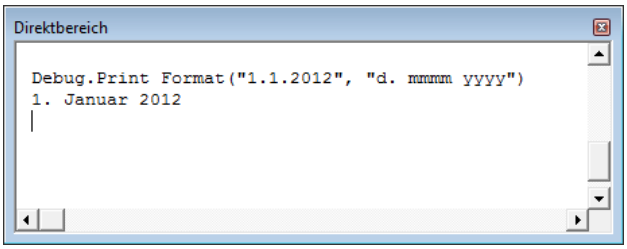



**Bild 2:** Testen der Format-Funktion im Direktfenster

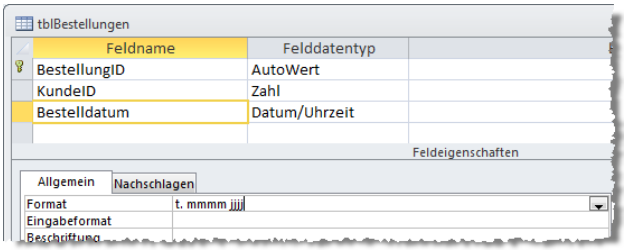



**Bild 3:** Die **Format**-Eigenschaft im Tabellenentwurf ...

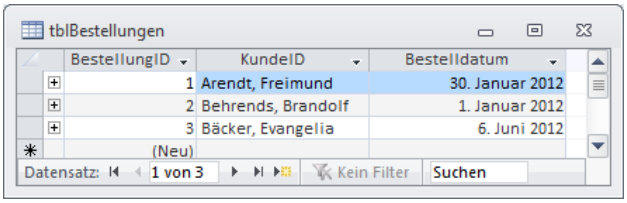



**Bild 4:** ... und in der Datenblattansicht.

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).

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 1:** Datenblatt mit verschiedenen Farben

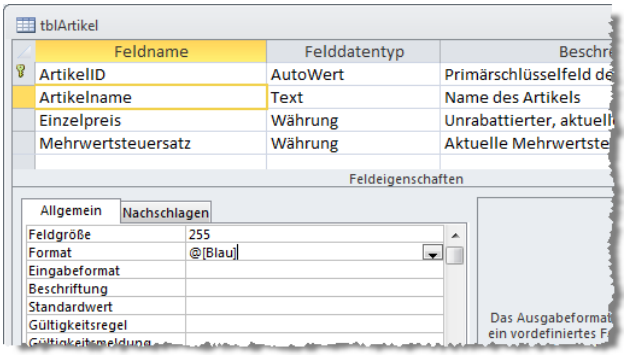



**Bild 2:** Format für eine farbige Spalte

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

Steuerelementen in Formularen und Berichten einstellen. Wenn Sie diese in der **Format**-Funktion verwenden, wird die Farbangabe ignoriert.

## Bedingte Farbzuweisung

Die **Format**-Eigenschaft erlaubt es, mehrere vom jeweiligen Wert abhängige Formatierungen zu platzieren. Diese bis zu vier Formatierungen werden angewendet, wenn die folgenden Werte auftreten:

- positive Zahl
- negative Zahl
- **0**
- **NULL**

Nehmen wir an, Sie möchten positive Zahlen schwarz, negative Zahlen rot und die Zahl **0** grün darstellen (siehe Bild 3). Dann definieren Sie folgenden Ausdruck für die **Format**-Eigenschaft (siehe Bild 4):

```
0;-0[Rot];0[Grün]
```

Die Formatierung arbeitet genau wie erwartet. Das Verhalten ändert sich allerdings, wenn Sie nur zwei Formatierungen angeben, also beispielsweise so:

```
0;-0[Rot]
```

In diesem Fall wird die Formatierung für die positiven Zahlen auch für **0** angewendet.

Auch bei **Format**-Ausdrücken mit mehr als einer Formatierung können Sie keine benannten Formatierungen verwenden.

## Platzhalter für leere Felder ausgeben

In der Praxis profitieren Sie mit und ohne farbige Gestaltung von der Möglichkeit, mehrere Formatierungen zu definieren. Wenn ein Feld beispielsweise noch leer ist, aber unbedingt ausgefüllt werden soll, können Sie einen entsprechenden Text als vierte Formatierung hinterlegen. Dies wird dann eingeblendet, wenn der Wert des Feldes **NULL** ist.

Um also beispielsweise in einer Tabelle mit Kundendaten für das Feld **Vorname** den Ausdruck **<Vorname eingeben>** in blauer Farbe anzuzeigen, wenn das

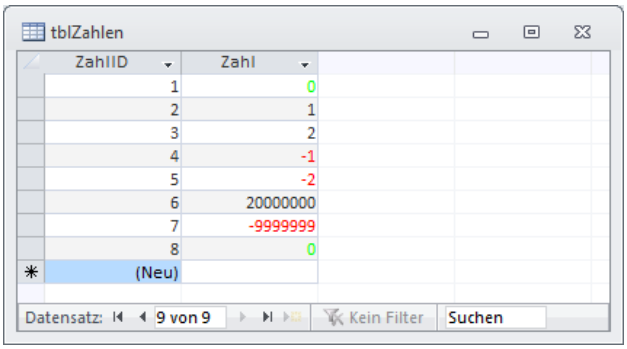



**Bild 3:** Verschiedene Farben innerhalb einer einzigen Zeile

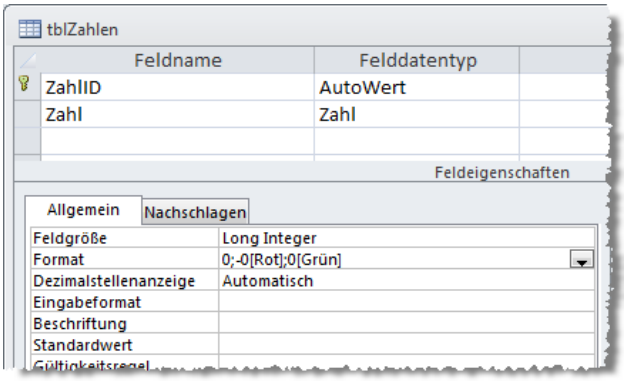



**Bild 4:** Formatierung für positive und negative Zahlen und 0




**Bild 5:** Für leere Felder wird ein Platzhaltertext angezeigt.

Feld leer ist, geben Sie den folgenden Ausdruck für die **Format**-Eigenschaft an:

```
@;;"<Vorname eingeben>"[Blau]
```

Das Ergebnis sieht dann wie in Bild 5 aus. Richtig interessant wird dieses Feature dann, wenn Sie das Feld in einem Formular anzeigen. Der Benutzer sieht dann direkt in entsprechender Farbe, wo noch Handlungsbedarf besteht.

# 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**.

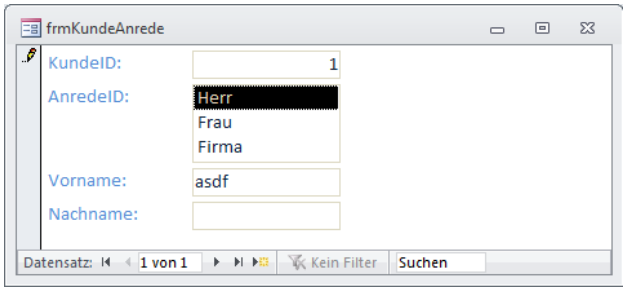



**Bild 1:** Lookup-Daten per Listenfeld auswählen

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

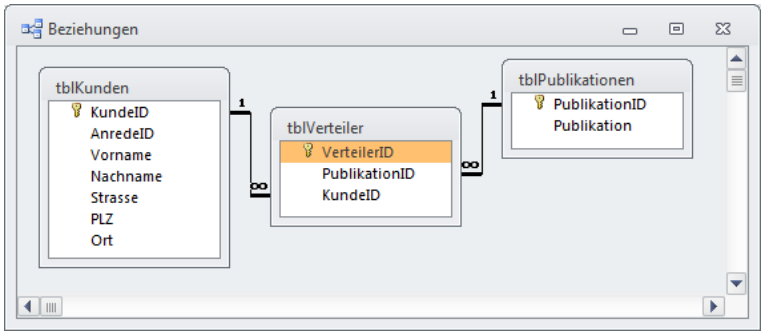



**Bild 2:** Datenmodell für den Publikationsverteiler

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).

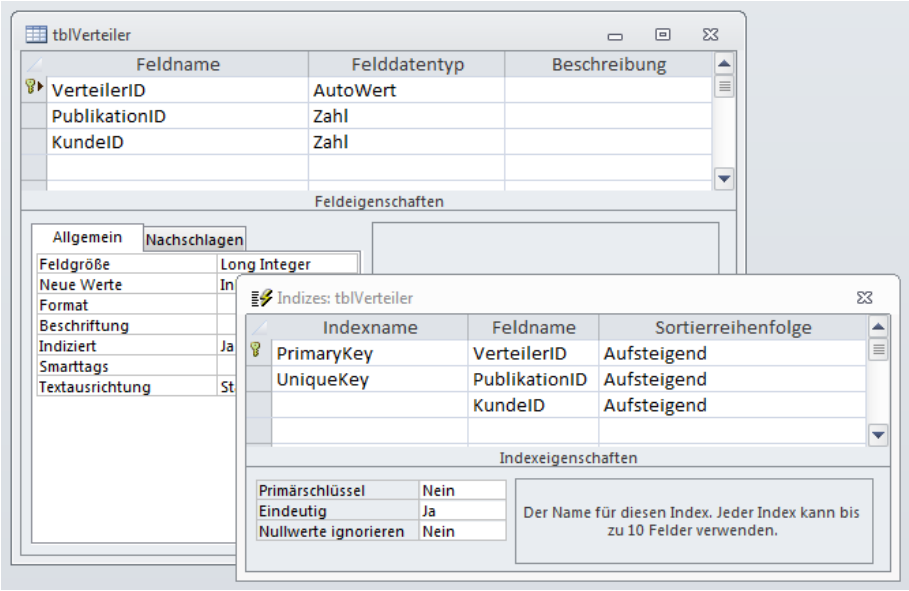



**Bild 3:** Eindeutiger Index für die Kombination aus Kunde und Publikation

## 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.

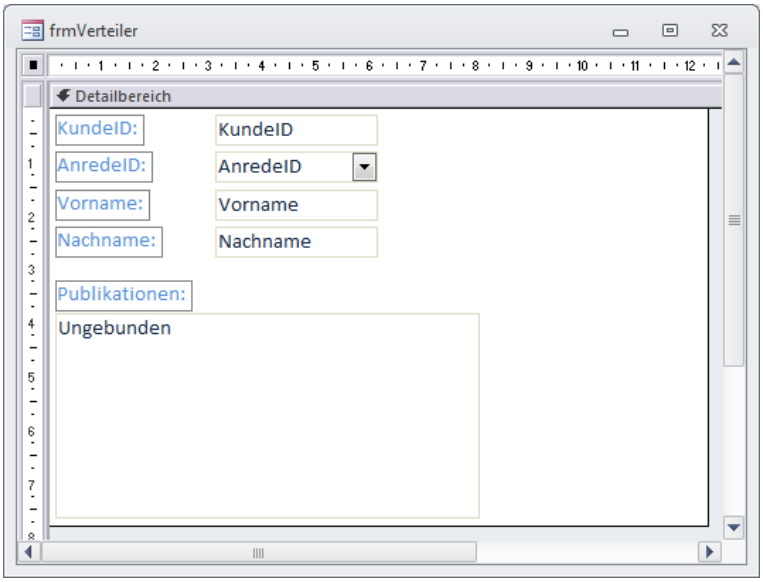



**Bild 4:** Entwurf des Formulars zum Festlegen der Verteilerlisten

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

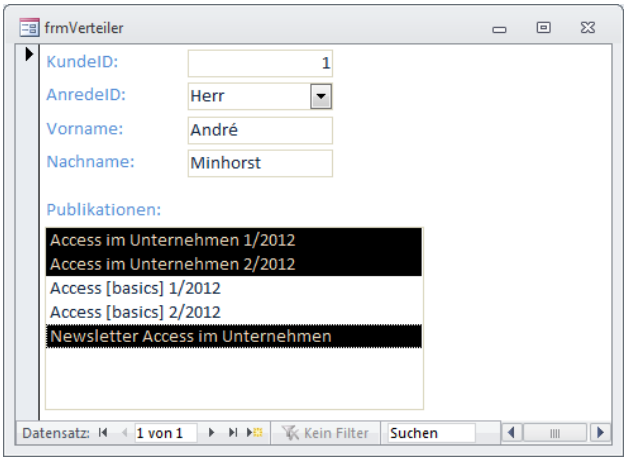



**Bild 5:** Das Formular zum Festlegen von Verteilerlisten in der Formularansicht

# 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.

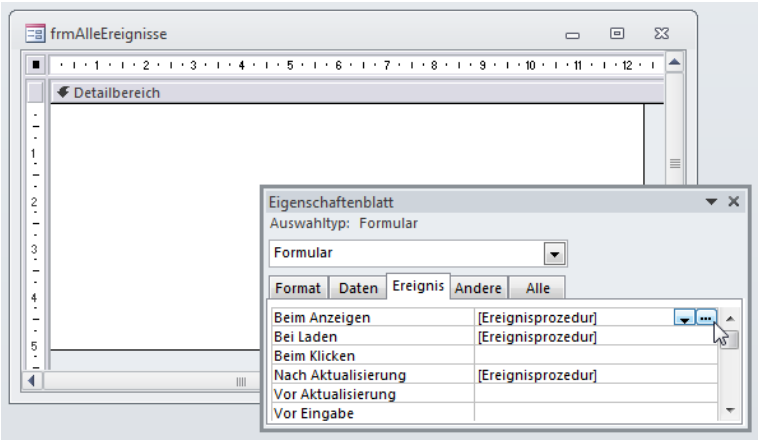



**Bild 1:** Anlegen von Ereignissen von der Benutzeroberfläche aus

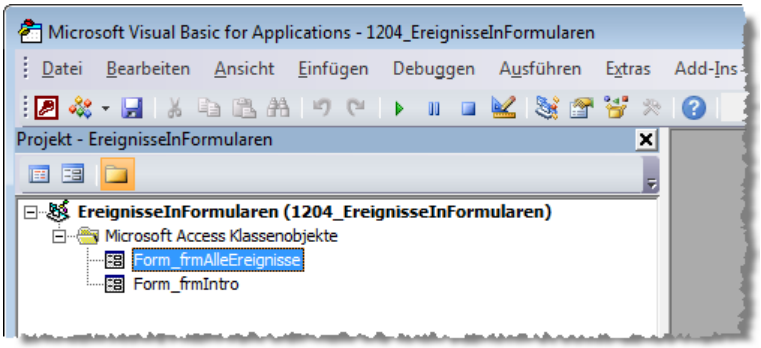



**Bild 2:** Das Klassenmodul im Projekt-Explorer

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.

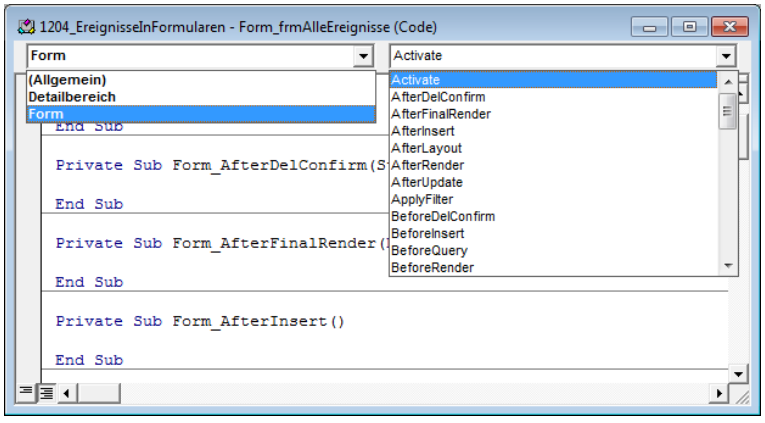



**Bild 3:** Anlegen von Ereignisprozeduren per VBA-Editor

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.

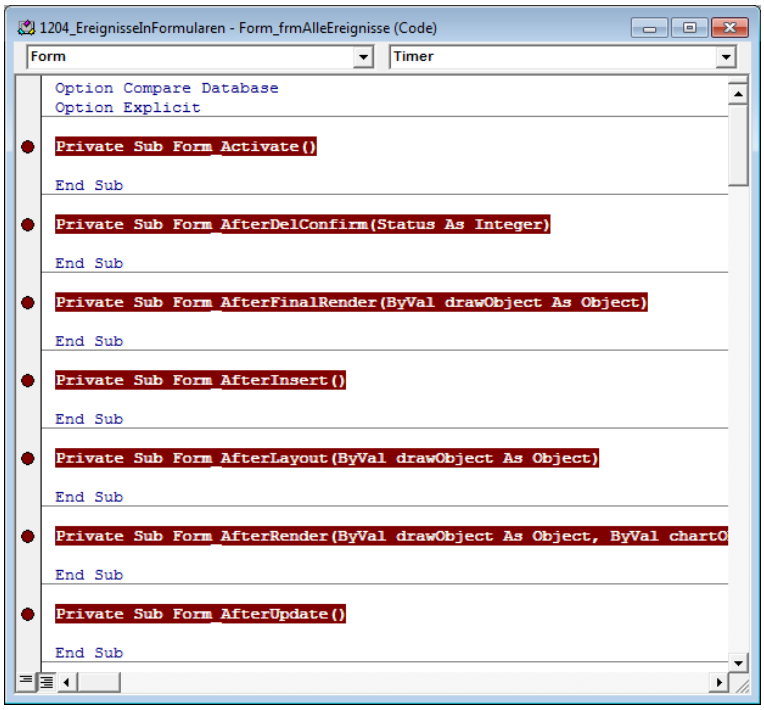



**Bild 4:** Haltepunkte zu Ereignisprozeduren hinzufügen

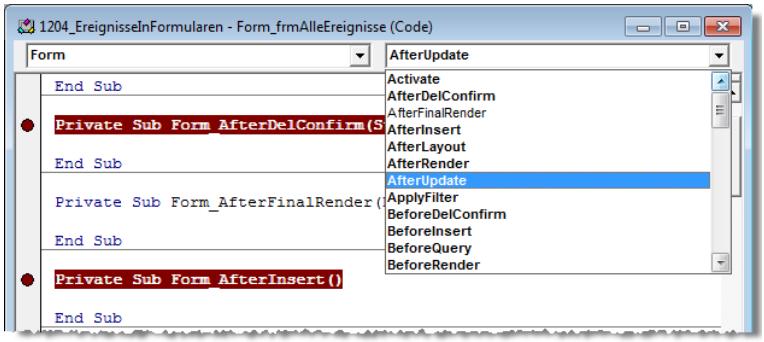



**Bild 5:** Implementierte Ereignisse werden in der Liste gekennzeichnet.

## 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).

Access [basics] 5/2012
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Editorial

Die aktuelle Ausgabe liefert einiges Know-how zum Thema Abfragen. Sie erfahren, wie Sie das Ergebnis einer Abfrage gruppieren und damit Daten aggregieren. Außerdem zeige ich Ihnen, wie Sie Kreuztabellenabfragen erstellen und somit praktisch zweidimensionale Gruppierungen und Berechnungen durchführen können – also beispielsweise alle Umsätze der Mitarbeiter pro Kategorie ermitteln. Diese Ausgabe liefert außerdem Informationen darüber, was Null-Werte eigentlich sind und wie sich diese von leeren Zeichenketten und der Zahl **0** unterscheiden. Im Praxisteil erfahren Sie, wie eine Schnellsuche erstellen und ein wenig Programmierkenntnisse liefere ich auch noch: Dabei erfahren Sie, welche Ereignisse beim Öffnen von Formularen ausgelöst werden und was Sie damit anfangen können.

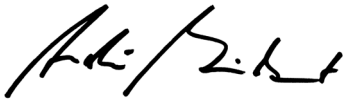
Und was gibt es bei mir Neues? Nun: Zunächst einmal standen die Arbeiten an der neuen Ausgabe meines anderen Magazins, **Access im Unternehmen**, an. Auch hier gibt es wieder jede Menge interessante Artikel – die Ausgabe erscheint Anfang Juni.

Danach gingen die Arbeiten am **Ribbon-Admin 2010** weiter, dem Tool für die Anpassung des Ribbons unter Access 2007 und 2010. Damit ist auch direkt die wichtigste erwähnt: Das Add-In kümmert sich fortan auch um Datenbankanwender, die noch (oder schon?) mit Access 2007 arbeiten. Außerdem lässt sich die Anwendung nun auch in englischer Sprache bedienen, wobei ich noch an der Übersetzung feilen muss.

Und wenn ein Bild 1.000 Worte sagt – was sagt dann ein Video? Unter diesem Motto habe ich ein paar Videos online gestellt, welche die Arbeit mit dem **Ribbon-Admin 2010** erklären. Dabei kam das Tool **Camtasia-Recorder** zum Einsatz. Und ich muss sagen: Mit diesem Tool macht es wirklich viel Spaß, Online-Tutorials zu erstellen. Diese habe ich anschließend auf Youtube veröffentlicht und von der Produktseite zum **Ribbon-Admin 2010** verlinkt. Wer mal reinschauen möchte, findet die Links unter **http://ribbon-admin.de/index.php/Dokumentation.html**.

Während ich die aktuellen Ausgaben von **Access im Unternehmen** und **Access [basics]** fertigstellt, standen die Arbeiten an meinem neuen Buch natürlich still. Das wird sich allerdings kurzfristig ändern!

In diesem Sinne: Viel Spaß beim Lesen!



Ihr André Minhorst

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst | Fachverlag für Softwareentwicklung | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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

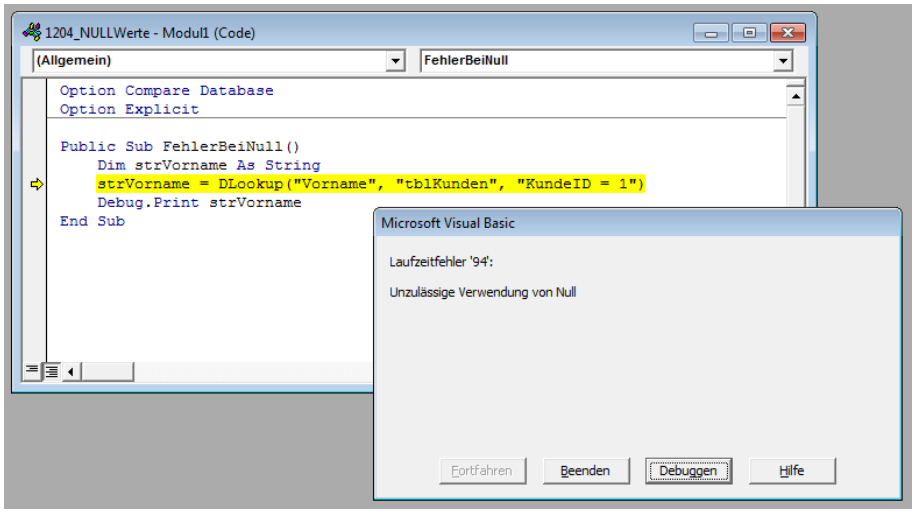



**Bild 1:** Fehler beim Versuch, einer String-Variablen einen NULL-Wert zuzuweisen

als Parameter den zu prüfenden Ausdruck und liefert den Wert **True** oder **False** zurück:

```
IsNull(<Ausdruck>)
```

Bei der Prüfung eines Feldwertes setzen Sie die Funktion etwa so ein:

```
Public Sub AufNullPruefen()
    Dim strVorname As String
    If IsNull(DLookup("Vorname", "tblKunden", _
            "KundeID = 1")) = True Then
        strVorname = ""
    Else
        strVorname = DLookup("Vorname", _
            "tblKunden", "KundeID = 1")
    End If
    Debug.Print strVorname
End Sub
```

## IsNull und IIf

Nun ist das schon ein recht umfangreiches Konstrukt, nur um mal eben den Wert eines Feldes in eine Variable zu schreiben. Sie können dies auch mit einer einzigen Zeile erledigen, wenn Sie die **IIf**-Funktion hinzunehmen:

```
strVorname = IIf(IsNull(DLookup("Vorname", _
    "tblKunden", "KundeID = 1")), "", _
    DLookup("Vorname", "tblKunden", "KundeID = 1"))
```

Der erste Parameter der **IIf**-Funktion enthält dabei den zu prüfenden Ausdruck, der zweite das Ergebnis für den Fall, dass der Ausdruck **True** ist und der dritte das Ergebnis für den Fall, dass der Ausdruck **False** ist.

## Die Nz-Funktion

Die Zeile ist immer noch recht lang und unübersichtlich. Zum Glück gibt es für diesen Fall eine spezielle Funktion namens **Nz**. Diese erwartet mindestens einen Parameter, und zwar den zu prüfenden Ausdruck. Liefert dieser Ausdruck nicht den Wert **Null**, gibt **Nz** einfach den Ausdruck zurück.

Ist der Ausdruck aber **Null**, hängt der Rückgabewert von verschiedenen Faktoren ab:

- Wenn Sie einen Ausdruck oder ein Literal als zweiten Parameter angegeben haben, liefert **Nz** den Wert des zweiten Parameters zurück.

- Wenn der zweite Parameter leer ist und das Ergebnis beispielsweise als Ausdruck innerhalb einer Abfrage zum Einsatz kommt, liefert die Funktion eine leere Zeichenkette zurück (**""**).

- Wenn der zweite Parameter leer ist und Sie weisen das Ergebnis einer Variablen zu, wird das Ergebnis in Abhängigkeit vom Datentyp der Variablen ermittelt. Wenn Sie eine Zahlen-Variable verwenden, liefert **Nz** den Wert **0**, sonst eine leere Zeichenkette (**""**).

Dies verdeutlichen die folgenden Beispiele. Im ersten Fall weisen wir das Ergebnis einer Variablen zu, deren Datentyp eine Zahl erwartet:

```
Dim intTest As Integer
intTest = Nz(DLookup("Vorname", "tblKunden", _
    "KundeID = 1"))
```

**intTest** enthält anschließend den Wert **0**. Beim zweiten Beispiel verwenden wir eine String-Variable als Ziel:

```
Dim strTest As String
strTest = Nz(DLookup("Vorname", _
    "tblKunden", "KundeID = 1"))
```

**strTest** ist nachher mit einer leeren Zeichenkette gefüllt. Wenn Sie sichergehen wollen, dass **Nz** auf jeden Fall eine leere Zeichenkette liefert, wenn der Ausdruck den Wert **Null** ergibt, geben Sie als zweiten Parameter eine leere Zeichenkette an:

```
strTest = Nz(DLookup("Vorname", "tblKunden", _
    "KundeID = 1"), "")
```

Gleiches gilt für den Fall, dass statt dem Wert **Null** die Zahl **0** zurückgeliefert werden soll:

```
Nz(DLookup("KundeID", "tblKunden", _
    "Vorname = 'xyz'"), 0)
```

## Alternative Werte zurückgeben

Der zweite Parameter kann jedoch auch mit beliebigen Werten gefüllt werden, also auch mit anderen Zahlen als **0** oder mit Zeichenketten.

## Null-Wert speichern

Wenn Sie damit rechnen, dass eine Funktion oder ein Ausdruck den Wert **Null** zurückliefert, kann es na-

# 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.

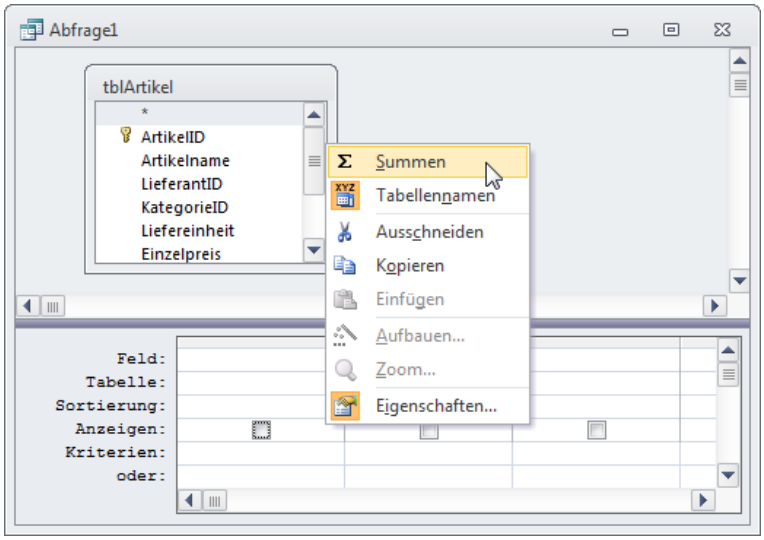



**Bild 1:** Einblenden der Zeile zum Festlegen der Gruppierungseigenschaften

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).

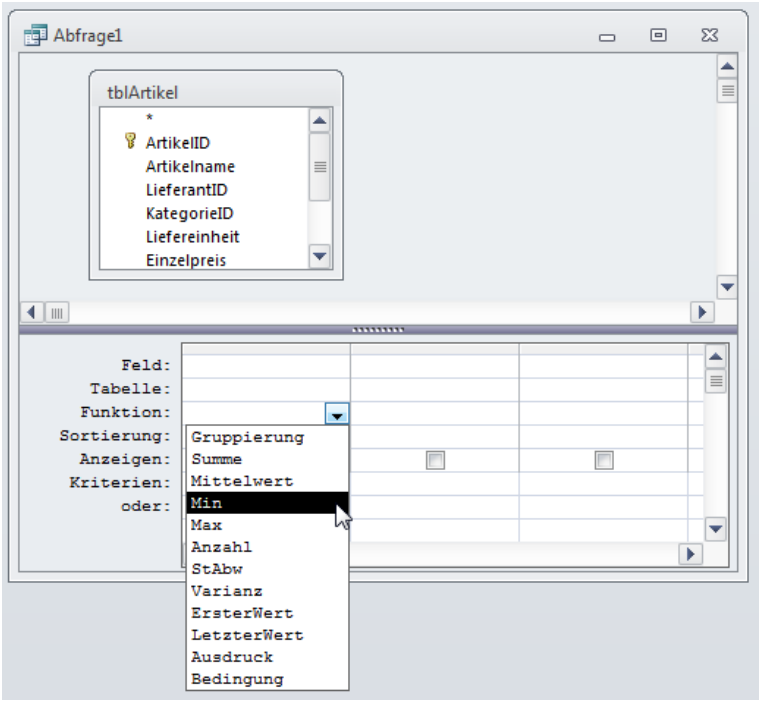



**Bild 2:** Auswählen einer Gruppierungsfunktion

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

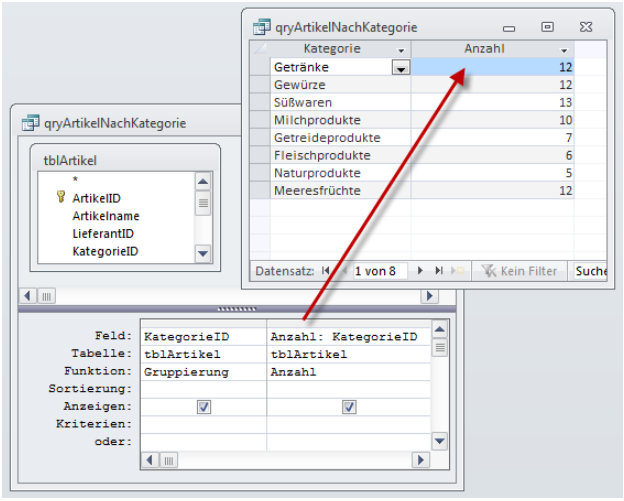



**Bild 5:** Finetuning der Spaltenüberschrift einer Aggregatspalte

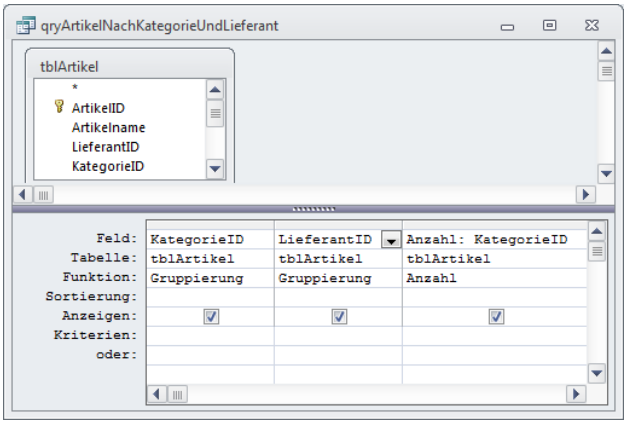



**Bild 6:** Entwurf einer Abfrage mit zwei Gruppierungskriterien

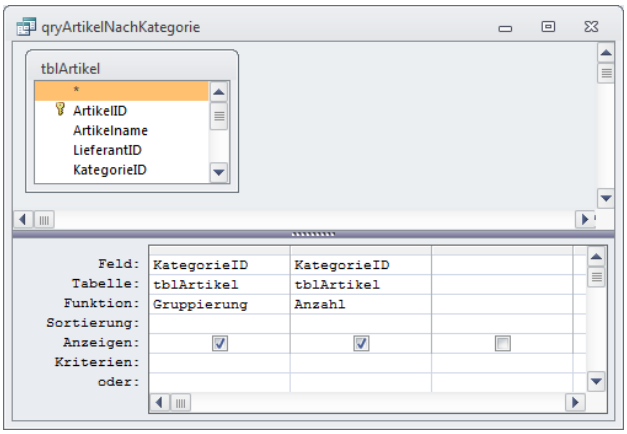



**Bild 3:** Hinzufügen zweier Felder zum Ausgaben des Kategorienamens und der Anzahl der Artikel je Kategorie

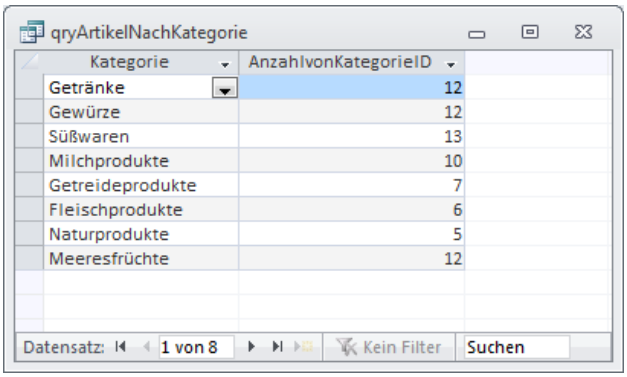



**Bild 4:** Ausgabe der Artikel je Kategorie

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.

tes Beispiel dafür, dass Sie auch mehrere Funktionen gleichzeitig in einer Abfrage einsetzen können.

## Bestellungen nach Monat und Jahr

Im folgenden Beispiel soll die Anzahl der Bestellungen nach Monat und Jahr ausgegeben werden. Dazu ziehen Sie das Feld **Bestelldatum** der Tabelle **tblBestellungen** zwei Mal in das Entwurfsraster der Abfrage und das Feld **BestellungID** einmal. Für das Bestelldatum legen Sie die folgenden beiden Ausdrücke fest:

```
Jahr: Jahr([Bestelldatum])
Monat: Monat([Bestelldatum])
```

Für das Feld **BestellungID** wählen Sie in der Zeile Funktion den Eintrag **Anzahl** aus. Den Entwurf und das Ergebnis finden Sie in Bild 10.

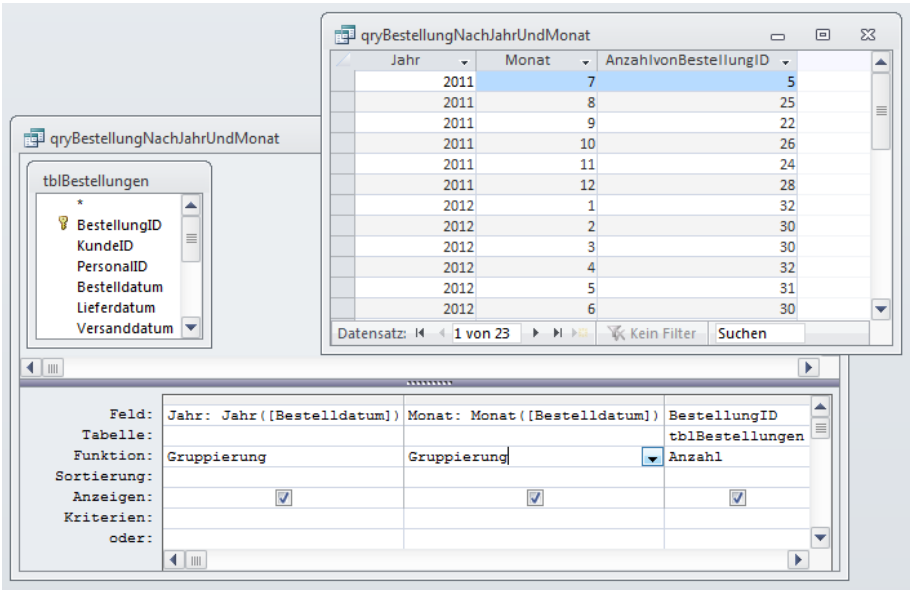



**Bild 10:** Anzahl der Bestellungen nach Jahr und Monat

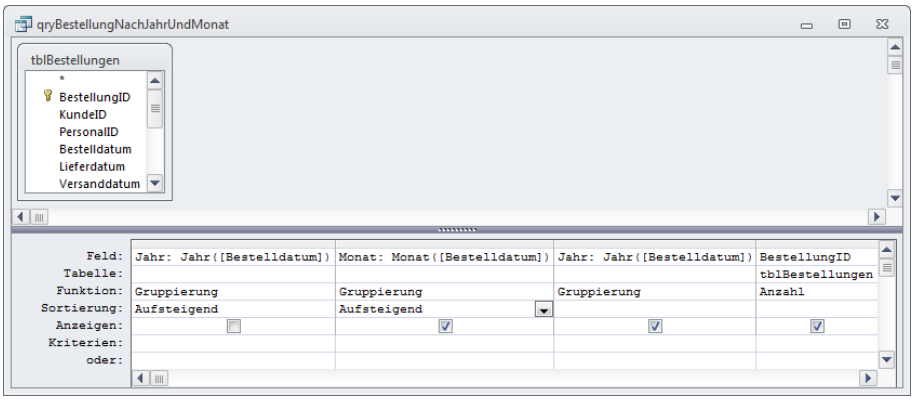



**Bild 11:** Anzahl der Bestellungen nach Monat und Jahr

Nun nehmen wir an, Sie möchten die Spalte mit dem Monat vorn und die Spalte mit dem Jahr erst als zweite Spalte anzeigen. Dann reicht es nicht aus, die beiden zu vertauschen, denn dann würde die Abfrage zunächst nach den Monaten und erst dann nach den Jahren sortieren (**1/2010**, **1/2011**, **2/2010** ...). Um die Reihenfolge für dieses Beispiel richtig einzustellen, müssen Sie die Spalte für das Jahr kopieren und einfügen, dann an vor die erste Spalte verschieben und diese schließlich ausblenden. Erst dann zeigt Access die benötigten Ansicht an (siehe Bild 11).

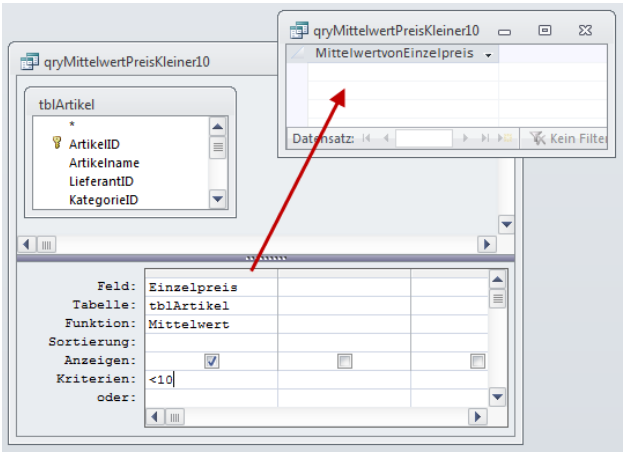



**Bild 12:** Durchschnitt aller Artikel, die weniger als zehn Euro kosten?

## Kriterien in Gruppierungen

Der Umgang mit Kriterien in Gruppierungen ist nicht ganz trivial. Angenommen, Sie möchten den Durchschnittswert aller Artikel mit einem Preis unter 10 Euro ermitteln.

Dann würden Sie vermutlich das Feld **Einzelpreis** in das Entwurfsraster ziehen, die Spalte **Funktion** aktivieren und dort den Eintrag **Mittelwert** auswählen. Außerdem legen Sie für dieses Feld noch das Kriterium **<10** fest. Das Ergebnis aus Bild 12 überrascht jedoch ein wenig: Es ist schlicht leer.

Die Lösung ist eigentlich naheliegend: Das Kriterium wirkt sich nicht auf die Datensätze aus, die zur Ermittlung des Mittelwerts herangezogen werden, sondern auf den ermittelten Wert. Der liegt über zehn Euro, also gibt die Abfrage keinen Datensatz zurück.

Also fügen Sie das Feld **Einzelpreis** ein zweites Mal zum Entwurfsraster hinzu, wählen den Eintrag **Bedingung** aus der Zeile **Funktion** aus und tragen den Ausdruck **<10** als Kriterium ein.

Dies liefert schließlich das korrekte Ergebnis (siehe Bild 13). Außerdem hat Access automatisch das Kontrollkästchen in der Spalte **Anzeigen** geleert. Bedingungsfelder werden im Ergebnis von Gruppierungsabfragen nicht eingeblendet.

Der Unterschied zwischen den beiden Varianten ist, dass im ersten Fall erst die Aggregatfunktion ausgeführt wird und dann erst das Kriterium berücksichtigt wird – im zweiten Fall läuft dies genau umgekehrt.

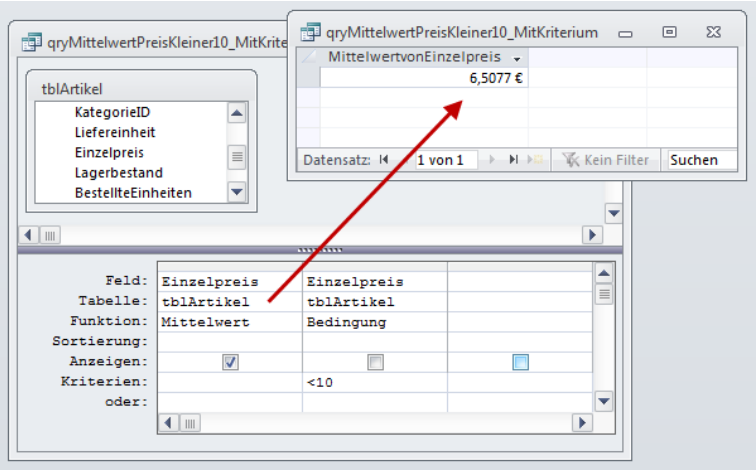



**Bild 13:** Korrekte Ermittlung des Durchschnitts

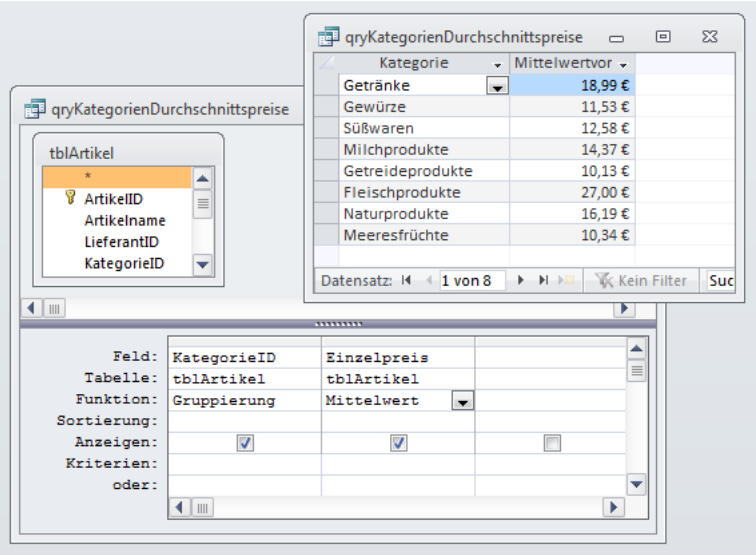



**Bild 14:** Mittelwerte der Einzelpreise für mehrere Kategorien

## Aggregate auf Gruppierungen anwenden

Das nächste Beispiel zeigt, wie Sie den Durchschnittspreis der Artikel für alle Kategorien ausgeben.

Hier übernehmen Sie auch wieder das Feld **KategorieID** aus der Tabelle **tblArtikel** in die Abfrage und fügen das Feld **Einzelpreis** hinzu. Dieses statten Sie mit der Funktion **Mittelwert** aus und erhalten das Ergebnis aus Bild 14.

## Vorsicht Null-Werte!

Beim Ermitteln des Ergebnisses von Aggregatfunktionen werden **Null**-Werte im untersuchten Feld nicht berücksichtigt. Das heißt beispielsweise, dass der Mittelwert nur auf Basis der Werte ermittelt wird, die nicht **Null** sind.

Das ist bei Mittelwerten, Summen et cetera auch in Ordnung. Aufpassen müssen Sie jedoch, wenn Sie die Anzahl von Datensätzen ermitteln. Sie dürfen dann nicht einfach irgendein Feld mit der Funktion **Anzahl** versehen, sondern sollten eines wählen, das für jeden Datensatz einen Wert enthält. Dies ist etwa bei Primärschlüsselfeldern mit Autowert immer der Fall.

Wenn Sie natürlich nur die Anzahl der Datensätze ermitteln möchten, die in einem bestimmten Feld einen Wert enthalten, müssen Sie dieses mit der Funktion **Anzahl** ausstatten.

# 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

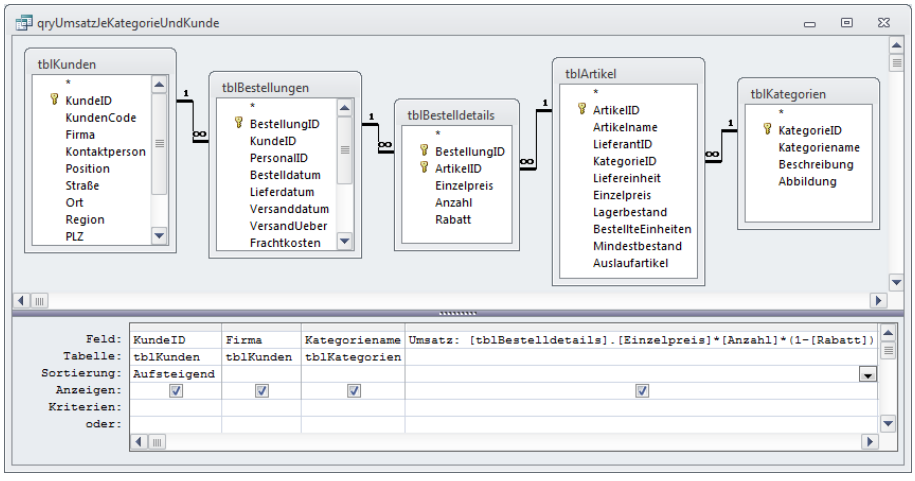



**Bild 3:** Abfrage als Vorbereitung auf die Kreuztabelle

## 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

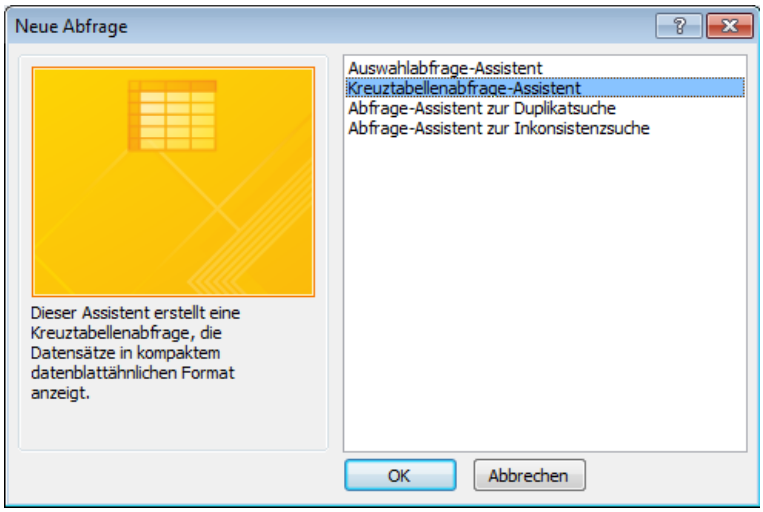



**Bild 4:** Start des Assistenten

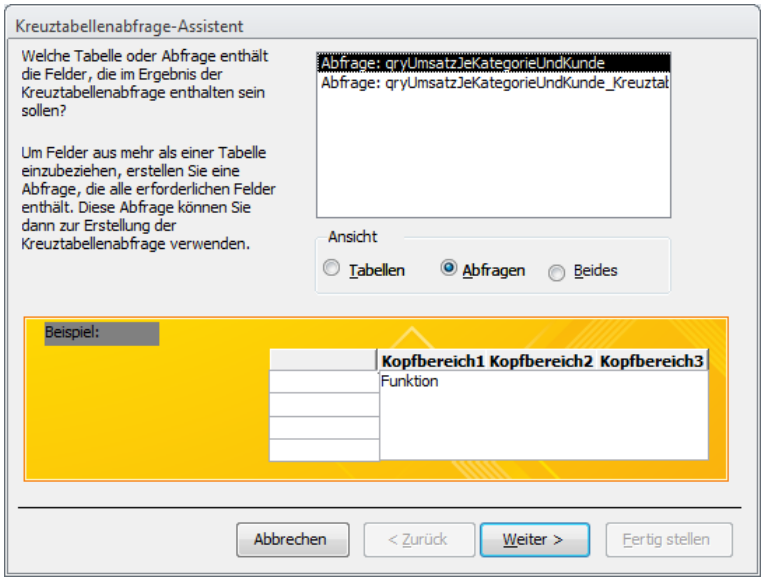



**Bild 5:** Auswählen der Quellabfrage

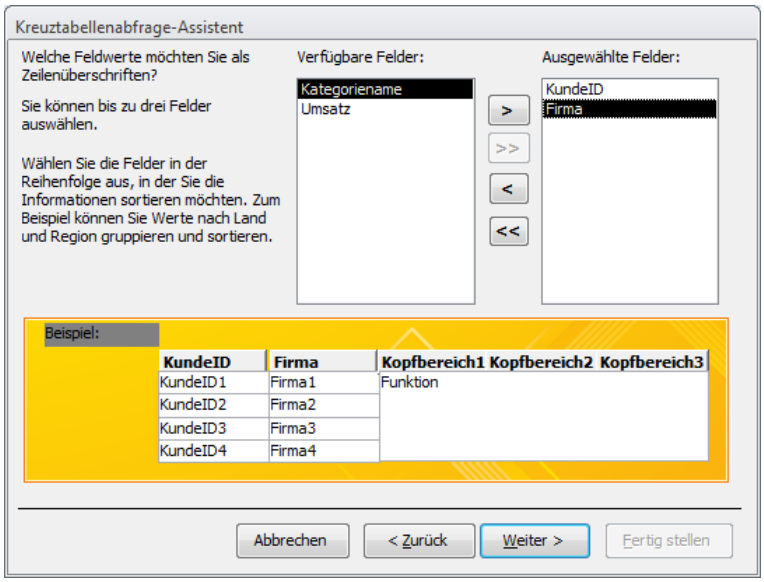



**Bild 6:** Festlegen des Feldes für die Zeilenköpfe

# 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.

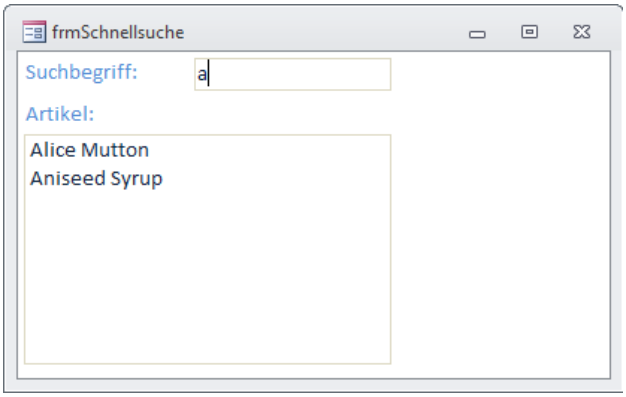



**Bild 1:** Schnellsuche im Listenfeld

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

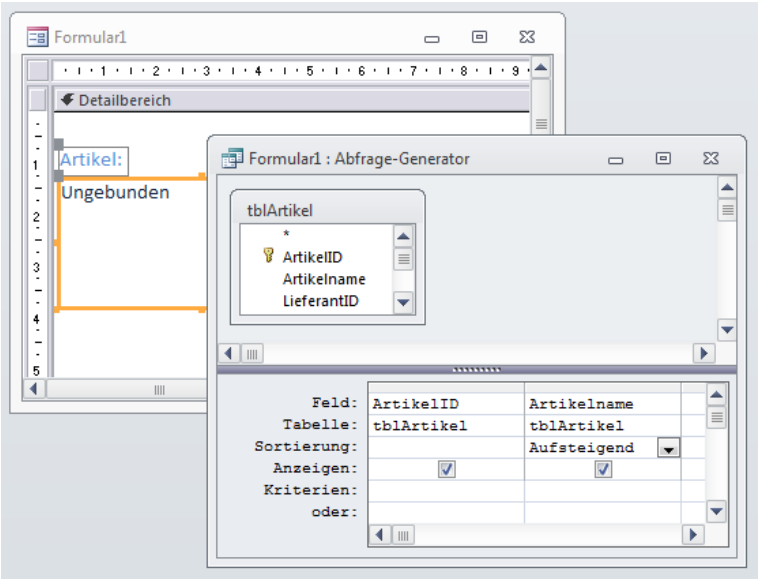



**Bild 2:** Datensatzherkunft des Listenfeldes

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.

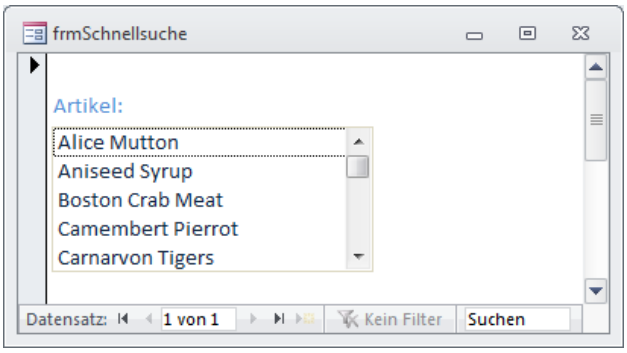



**Bild 3:** Das Listenfeld wird wie gewünscht gefüllt.

## 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.

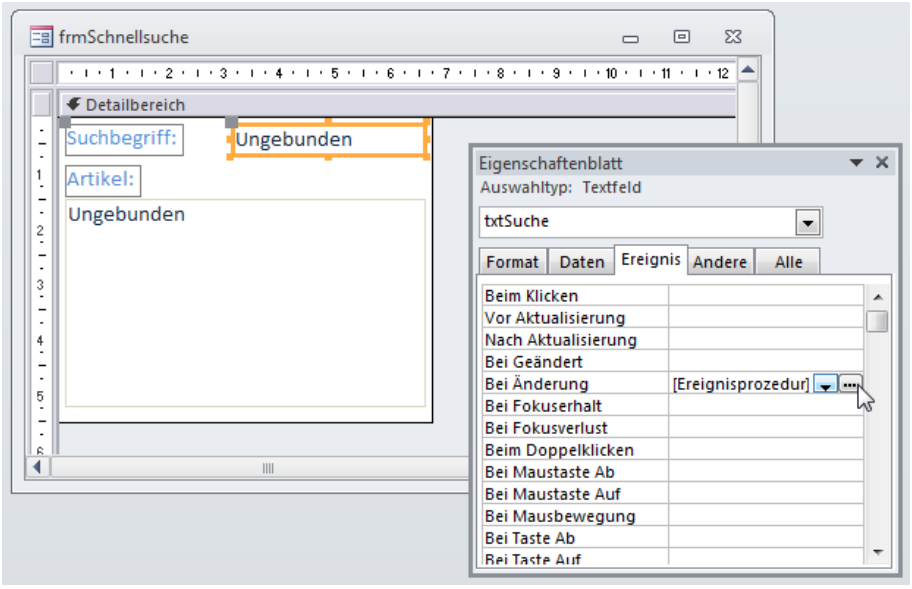



**Bild 4:** Anlegen der Ereignisprozedur, die durch das Ereignis **Bei Änderung** ausgelöst wird.

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.

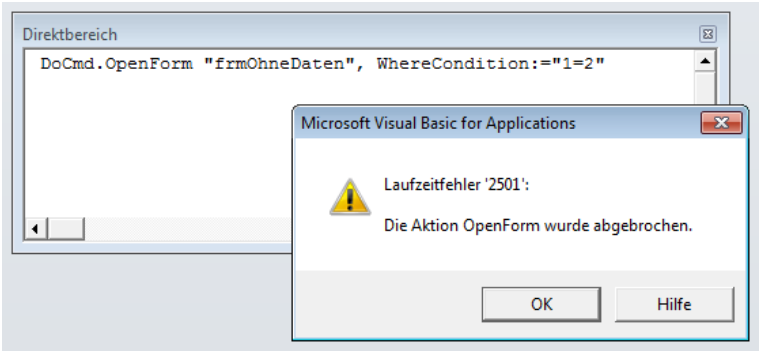



**Bild 1:** Fehler beim Abbruch des Öffnungsvorgangs

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-

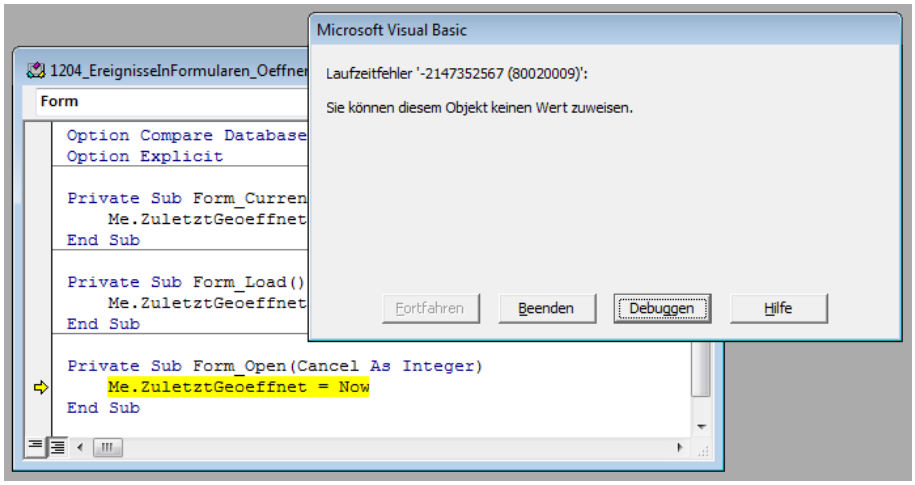


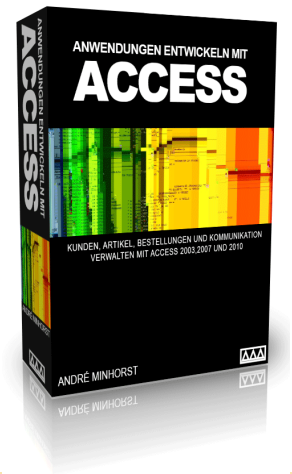

**Bild 2:** Beim Öffnen ist noch keine Änderung der Daten möglich.

Access [basics] 6/2012
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Editorial

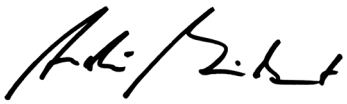
In dieser Ausgabe dreht es sich gleich in zwei Artikeln um die Behandlung von Dateien. Im ersten Artikel, **Dateibefehle**, erfahren Sie, welche Anweisungen VBA für den Umgang mit Dateien bereithält. Ein paar davon verwenden wir gleich im Artikel **Textdateien** weiter: Dort zeigen wir, wie Sie Textdateien erstellen, mit Daten füllen, auslesen und so weiter. Der Artikel **Kreuztabellen nach Datum** liefert noch einige Ergänzungen zur vorherigen Ausgabe. Wenn Sie Kreuztabellen mit Datumsangaben als Spaltenüberschriften verwenden möchten, gibt es nämlich einige Besonderheiten. Und schließlich liefert der Artikel **Rechnungsberichte** Know-how rund um die Erstellung von Rechnungen auf Basis von Access Berichten.

Uns sonst? Ich bin natürlich gerade mit der Fertigstellung des bereits angekündigten Buchs **Anwendungen entwickeln mit Access** beschäftigt. Die Webseite ist fertig und kann unter **http://book.minhorst.com** besichtigt werden. Dort können Sie das Buch auch schon bestellen! Es wird wohl am 20. Juli 2012 fertiggestellt werden und ist dann als eBook downloadbar. Allerdings ist auch eine gedruckte Version geplant, die Sie gegen einen geringen Aufpreis gegenüber dem Preis für das eBook bestellen können. Also: Wenn Sie es noch nicht getan haben, schauen Sie doch einmal auf der Seite zum Buch vorbei.




Außerdem muss ich zugestehen, dass ich vor lauter Arbeit ganz übersehen habe, dass **Access [basics]** mittlerweile zwei Jahre alt geworden ist! Das ist auch der Grund, warum es diese Ausgabe für alle Interessierten kostenlos zum Download gibt. Ich muss schon sagen, dass ich mich sehr über Ihren Zuspruch in den letzten zwei Jahren freue! Sie lesen nicht nur eifrig mit, sondern liefern mir auch viel positives Feedback. Und ich konnte schon einige Ihrer Anregungen in neue Artikel umsetzen. Sollen Sie Artikelwünsche haben, von denen Sie denken, dass diese gut in dieses Magazin passen würden, melden Sie sich doch einfach unter info@access-basics.de und teilen mir Ihren Wunsch mit!

So, nun muss ich weiter – es gibt noch einige Seiten im Buch, die mit Access-Know-how gefüllt werden sollen. Viel Spaß beim Lesen der neuen Ausgabe!



Ihr André Minhorst

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst | Fachverlag für Softwareentwicklung | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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:

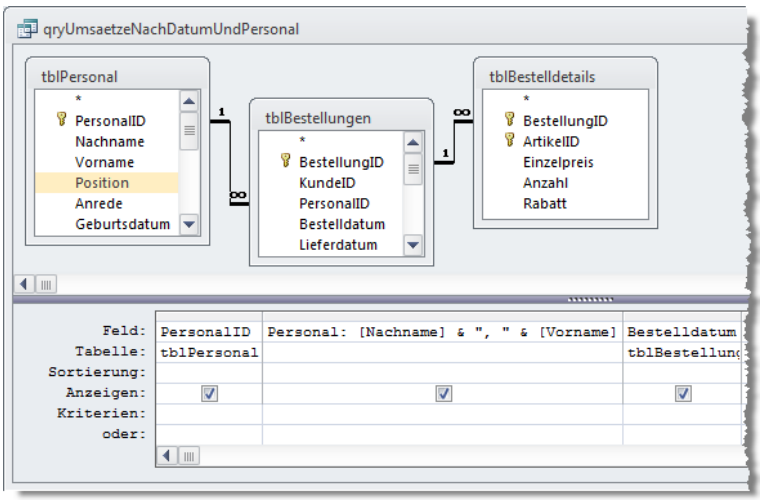



**Bild 1:** Umsätze nach Datum und Personal

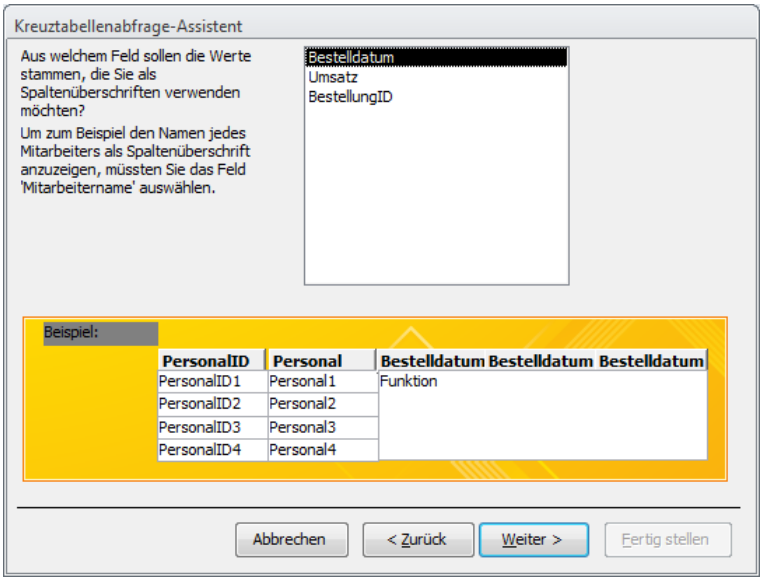



**Bild 2:** Auswahl des Feldes **Datum**

```
UmsatzFormatiert:
Format([Umsatz];"Festkommazahl")
```

Das Ergebnis finden Sie in Bild 4 vor. Auf den ersten Blick sieht das Ergebnis gut aus. Allerdings zeigt die Abfrage nur zwölf Monate ohne Jahresangabe an. Das wäre noch kein Problem, wenn die Tabelle **tblBestellungen** nur Bestellungen eines einzigen Jahres enthielte – sie enthält aber Daten aus mehreren Jahren. Die Kreuztabellenabfrage fasst also einfach alle Daten der einzelnen Monate zusammen – ohne Rücksicht auf die Jahre.

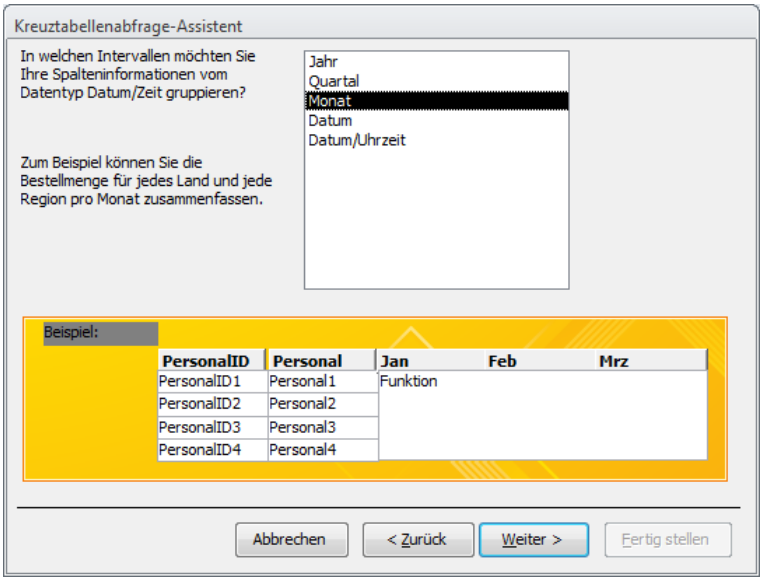



**Bild 3:** Festlegen des Datumsintervalls

## Nach Jahren und Monaten gruppieren

Daraus resultieren zwei Aufgaben: Erstens sollen die Spaltenköpfe nicht nur den Monat anzeigen, sondern auch das Jahr, und zweitens müssen die Daten dementsprechend etwas feiner gruppiert werden. Dazu ist ein Blick in den Entwurf der Abfrage nötig. Für ein wenig mehr Flexibilität bauen wir die Abfrage neu auf. Dazu legen Sie eine neue Abfrage an, ziehen die Tabellen **tblPersonal**, **tblBestellungen** und **tblBestelldetails** in den Entwurf.

Ändern Sie den Abfragetyp mit dem Kontextmenüeintrag **Abfragetyp|Kreuztabellenabfrage**. Ziehen Sie dann das Feld für den Zeilenkopf in das Entwurfsraster, nämlich **PersonalID**. Stellen Sie für dieses Feld den Wert der Zeile **Kreuztabelle** auf **Zeilenüberschrift** ein. Fügen Sie ein weiteres Feld als Zeilenüberschrift hinzu, das den folgenden Ausdruck enthält:

```
Personal: [Nachname] & ", " & [Vorname]
```

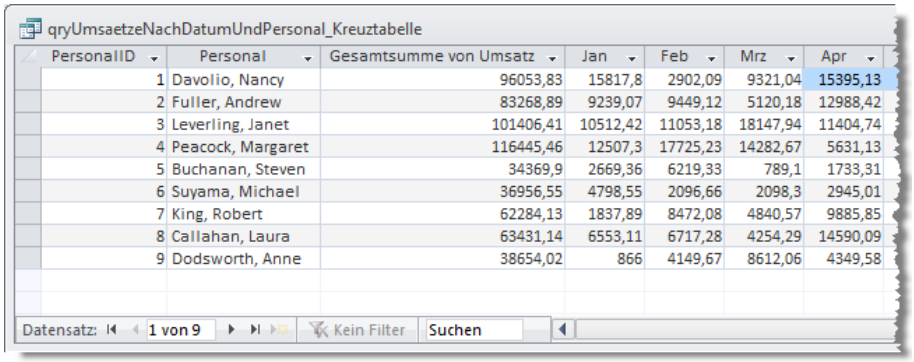
Das Bestelldatum integrieren Sie durch den folgenden Ausdruck in der Abfrage:

```
Bestellung: Monat([Bestelldatum]) & "/" &
Jahr([Bestelldatum])
```

Dies ermittelt das Datum in der Form **1/2012**. Stellen Sie den Wert in der Zeile **Kreuztabelle** auf **Spaltenüberschrift** ein. Den Umsatz berechnen wir mit der bekannten Formel:

```
Umsatz: [Einzelpreis]*[Anzahl]*(1-[Rabatt])
```

Wählen Sie für dieses Feld den Eintrag **Wert** in der Zeile Kreuztabelle aus und stellen Sie in der Zeile **Funktion** den Wert **Summe** ein. Den Entwurf der Abfrage finden Sie in Bild 5. Ein Wechsel in die Datenblattansicht liefert zunächst das gewünschte Bild. Bei genauerem Hinsehen fällt jedoch auf, dass die Spaltenüberschriften alphabetisch sortiert werden und nicht etwa nach dem Datum (siehe Bild 6). Die richtige Reihenfolge erhalten Sie nun am einfachsten über das Eintragen der Spaltenköpfe in die Eigenschaft **Fixierte Spaltenüberschriften**:


qryUmsaetzeNachDatumUndPersonal_Kreuztabelle

| PersonalID | Personal | Gesamtsumme von Umsatz | Jan | Feb | Mrz | Apr |
|---|---|---|---|---|---|---|
| 1 | Davolio, Nancy | 96053,83 | 15817,8 | 2902,09 | 9321,04 | 15395,13 |
| 2 | Fuller, Andrew | 83268,89 | 9239,07 | 9449,12 | 5120,18 | 12988,42 |
| 3 | Leverling, Janet | 101406,41 | 10512,42 | 11053,18 | 18147,94 | 11404,74 |
| 4 | Peacock, Margaret | 116445,46 | 12507,3 | 17725,23 | 14282,67 | 5631,13 |
| 5 | Buchanan, Steven | 34369,9 | 2669,36 | 6219,33 | 789,1 | 1733,31 |
| 6 | Suyama, Michael | 36956,55 | 4798,55 | 2096,66 | 2098,3 | 2945,01 |
| 7 | King, Robert | 62284,13 | 1837,89 | 8472,08 | 4840,57 | 9885,85 |
| 8 | Callahan, Laura | 63431,14 | 6553,11 | 6717,28 | 4254,29 | 14590,09 |
| 9 | Dodsworth, Anne | 38654,02 | 866 | 4149,67 | 8612,06 | 4349,58 |

Datensatz: 1 von 9 | Kein Filter | Suchen

**Bild 4:** Umsatz der einzelnen Mitarbeiter nach Monaten

```
"7/2011";"8/2011";"9/2011";"
10/2011";"11/2011";"12/2011"
;"1/2012";"2/2012";"3/2012";
"4/2012";"5/2012";"6/2012";"
7/2012";"8/2012";"9/2012";"1
0/2012";"11/2012";"12/2012";
"1/2013";"2/2013";"3/2013";"
4/2013";"5/2013"
```

Danach zeigt die Abfrage die Datumsangaben in der richtigen Reihenfolge an.

# 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"
```

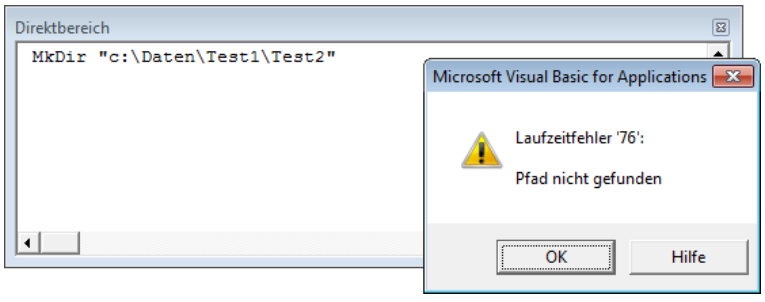



**Bild 1:** Fehler beim Versuch, zwei Verzeichnisse auf einmal zu erstellen

## 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).

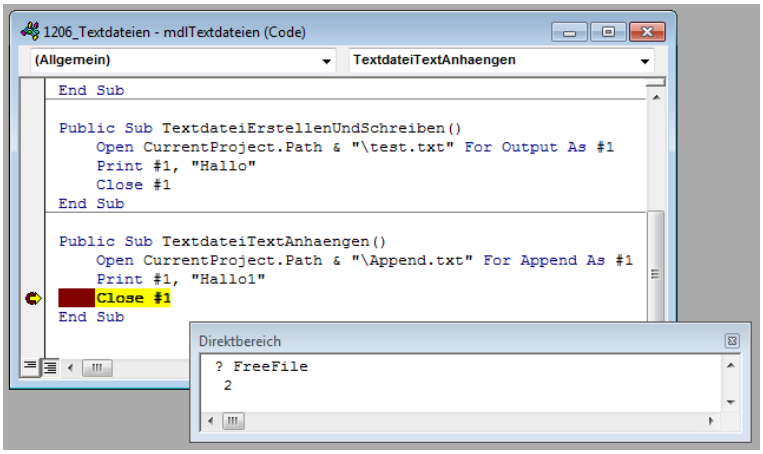



**Bild 1:** Aufruf von **FreeFile** während bereits eine Datei geöffnet ist

## 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.

# 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.

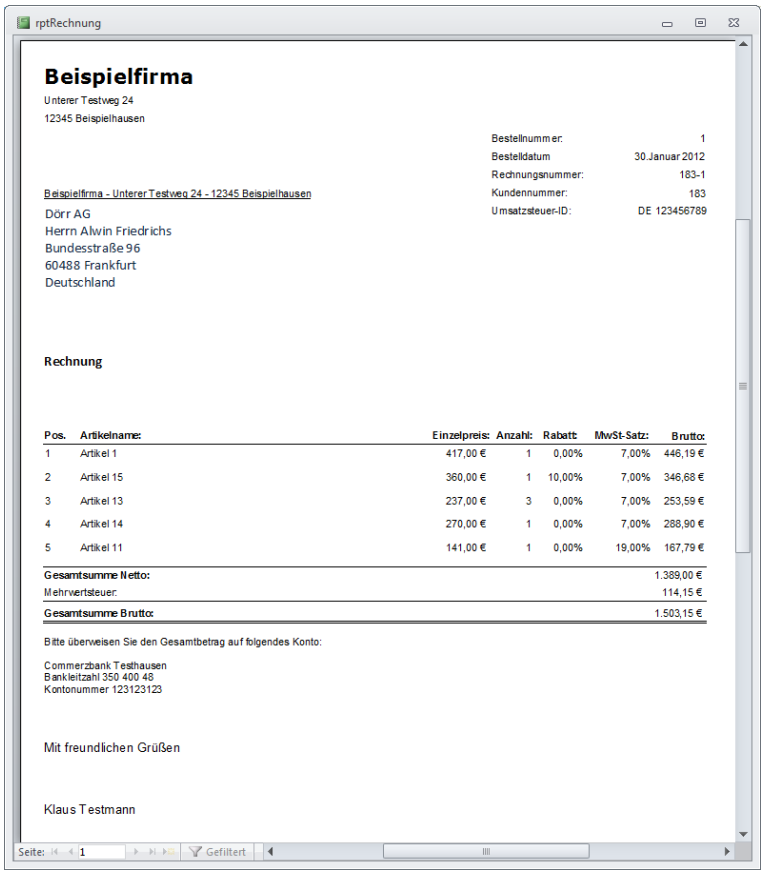



**Bild 2:** Der Bericht in der Vorschauansicht

## 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:

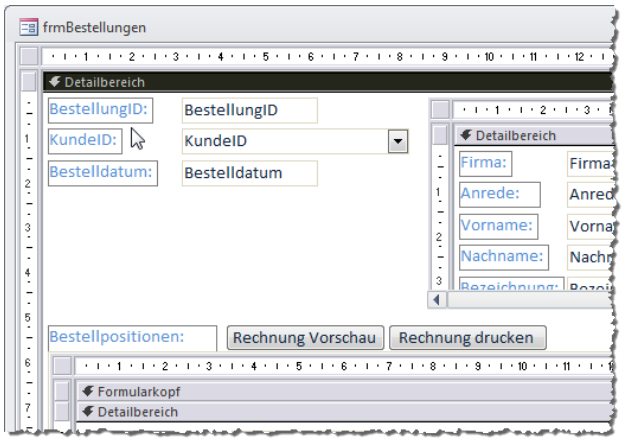



**Bild 1:** Aufruf des Berichts vom Bestellformular aus

```
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.

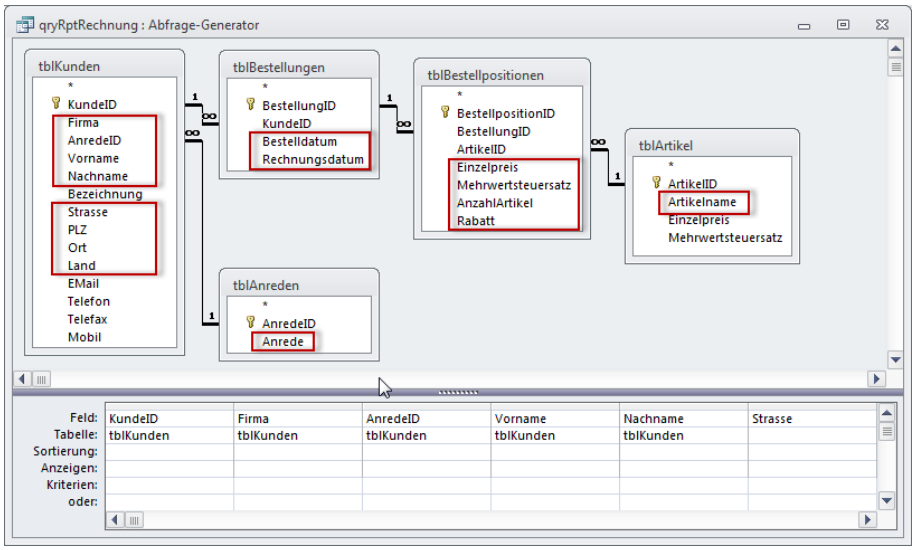



**Bild 3:** Datenherkunft des Berichts

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]
```

Access [basics] 7/2012
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Editorial

Diese Ausgabe sollte eigentlich schon vor einem Monat erscheinen, aber diverse Gründe (unter anderem Urlaub, dringende Projekte und mein neues Buch) führten dazu, dass das Zeitfenster, in dem die neue Ausgabe erstellt werden sollte, sich mehr und mehr schloss – und die Ausgabe somit quasi ausgefallen ist. Aber keine Angst: Natürlich holen wird das wieder auf, und zwar sobald wie möglich. Als kleine Wiedergutmachung für die Verzögerung erhalten alle Abonnenten einen kostenlose Ausgabe von Access im Unternehmen und somit zwei Monate Zugang zum umfangreichen Onlinearchiv dieses Magazins. Den Download finden Sie unter **http://www.access-im-unternehmen.de/download**. Verwenden Sie für die Ausgabe 4/2012 bitte die Zugangsdaten **html/ansicht**. Die aktuelle Ausgabe 7/2012 von **Access [basics]** beschäftigt sich mit den folgenden Themen:

- Die Dir-Funktion: Mit dieser Funktion greifen Sie auf die Dateien und Unterverzeichnisse eines Ordners zu und können diese sowohl auslesen als auch ihre Existenz prüfen.

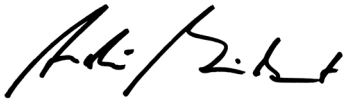
- Binärzahlen in VBA-eigenschaften: Einige VBA-Eigenschaften übergeben mit einer einzigen Zahl gleich mehrere Informationen. Dieser Artikel zeigt, wie Sie diese in Binärzahlen eingebetteten Daten auslesen können.

- **Kombinations- und Listenfelder mit Wertliste füllen**: Neben Tabellen und Abfragen können Sie auch semikola-separierte Listen als Datensatzherkunft für diese Steuerelemente nutzen. Am Beispiel der Dateien und Unterverzeichnisse eines Ordners zeigen wir, wie dies funktioniert.

- **Kombinations- und Listenfelder per Callback füllen**: Um die Möglichkeiten zum Füllen dieser Steuerelemente zu vervollständigen, schauen wir uns die Variante mit der Callback-Funktion. Dies erlaubt ein sehr flexibles füllen von Kombinationsfeld- und Listenfelder.

- **Das ListView-Steuerelement – Grundlagen**: Das ListView-Steuerelement ist eine Art Mischung aus Datenblattansicht und Listenfeld. Es bietet einige interessante Möglichkeiten, ist aber auch aufwendiger zu programmieren. In dieser Artikelreihe erfahren Sie alles, was Sie zum Thema ListView-Steuerelement wissen müssen.

Und nun: Viel Spaß beim Lesen der neuen Ausgabe!



Ihr André Minhorst

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst | Fachverlag für Softwareentwicklung | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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**.

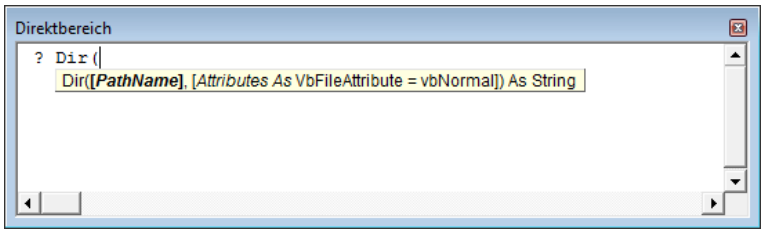



**Bild 1:** IntelliSense zeigt die Parameter als optional an

## 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-

nisnamen inklusive eines angehängten Backslash angeben. Der Aufruf sieht dann so aus:

```
? Dir("c:\Beispiel\")
```

Möchten Sie auch Verzeichnisnamen in die Suche einbeziehen, fügen Sie wiederum den Parameter **vbDirectory** hinzu:

```
? Dir("c:\Beispiel\", vbDirectory)
```

## Weitere Dateien ermitteln

Die **Dir**-Funktion liefert mit einem festen Parameter immer das gleiche Ergebnis. Sie können also zehnmal den Aufruf **? Dir("c:\Beispiel")** absetzen und erhalten immer das gleiche Ergebnis. Wenn Sie weitere Ergebnisse für das gleiche Verzeichnis erhalten möchten, verwenden Sie einfach nur die **Dir**-Anweisung:

```
? Dir()
```

## Alle Elemente eines Verzeichnisses ausgeben

Wenn Sie alle Elemente eines Verzeichnisses ausgeben möchten, verwenden Sie eine Prozedur mit einer **Do While**-Schleife wie im folgenden Beispiel:

```
Public Sub DateienAusgeben(strVerzeichnis As String)
    Dim strDatei As String
    If Not Right(strVerzeichnis, 1) = "\" Then
        strVerzeichnis = strVerzeichnis & "\"
    End If
    strDatei = Dir(strVerzeichnis, vbDirectory)
    Do While Len(strDatei) > 0
        Debug.Print strDatei
        strDatei = Dir
    Loop
End Sub
```

Diese Prozedur erwartet die Übergabe des zu untersuchenden Verzeichnisses, wahlweise mit oder ohne abschließendem Backslash (****).Die **If...Then**-Bedingung prüft, ob das Verzeichnis am Ende mit einem Backslash-Zeichen ausgestattet ist und ergänzt dies gegebenenfalls. Danach ruft die Prozedur die Dir-Anweisung einmalig mit dem angegebenen Verzeichnisnamen auf und bezieht dabei auch Verzeichnisse ein. Ist die ermittelte Zeichenkette nicht leer, startet eine **Do While**-Schleife, die den Namen des zuletzt gefundenen Eintrags ausgibt und mit einem einfachen Aufruf von **Dir** das nächste Element ermittelt. Auf diese Weise werden alle im Verzeichnis enthaltenen Elemente durchlaufen und ausgegeben.

Wenn Sie nun auch noch wissen möchten, welcher der gefundenen Einträge ein Unterverzeichnis repräsentiert und welches eine Datei, dann verwenden Sie die **GetAttr**-Prozedur. Diese liefert einen Zahlenwert, welcher den Attributen der als Parameter angegebenen Datei entspricht. Der Ausdruck **GetAttr(strVerzeichnis & strDatei)** liefert beispielsweise den Wert **16**, wenn es sich bei dem Element um ein Verzeichnis handelt. Sie können diesen Wert jedoch nicht einfach mit dem Wert **16** vergleichen, denn **GetAttr** könnte auch den Wert **17**, **18** oder andere Werte liefern, wenn dafür nicht nur das Attribut **vbDirectory** festgelegt ist. Wenn Sie den Zahlenwert jedoch per **And** mit dem Wert **16** verknüpfen, liefert dieser den Wert **16** zurück, wenn die **16** enthalten ist und **0**, falls nicht (wie dies genau funktioniert, erfahren Sie im Artikel **Binärzahlen in VBA-Eigenschaften**).

Die Erweiterung der Prozedur DateienAusgeben prüft jedenfalls mit GetAttr, ob das Attribut vbDirectory für das Element definiert wurde und gibt gegebenenfalls ein führendes Backslash-Zeichen zur Markierung von Verzeichnissen aus:

```
If (GetAttr(strVerzeichnis & strDatei) And _
        vbDirectory) = vbDirectory Then
    Debug.Print "\" & strDatei
Else
    Debug.Print strDatei
End If
```

## Dir und Rekursion

Gegebenenfalls möchten Sie einmal rekursiv alle Unterverzeichnisse und Dateien eines Verzeichnisses ausgeben. Dazu müssten sich die Prozedur **DateienAusgeben** rekursiv selbst aufrufen, und zwar mit dem aktuell gefundenen Element als Wert des Parameters **strVerzeichnis**. Dies funktioniert deshalb nicht, weil die **Dir**-Funktion ja beim rekursiven Aufruf zwangsläufig mit einem neuen Verzeichnisnamen initialisiert werden muss. Wenn also ein Verzeichnis ein Unterverzeichnis enthält, müssten Sie zunächst das Unterverzeichnis untersuchen.und würden dann die folgenden Elemente mit **Dir()** aufrufen. Da **Dir** aber mittlerweile mit dem Unterverzeichnis als Parameter verwendet wurde, liest es nicht automatisch die folgenden Elemente des übergeordneten Verzeichnisses ein.

# 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:

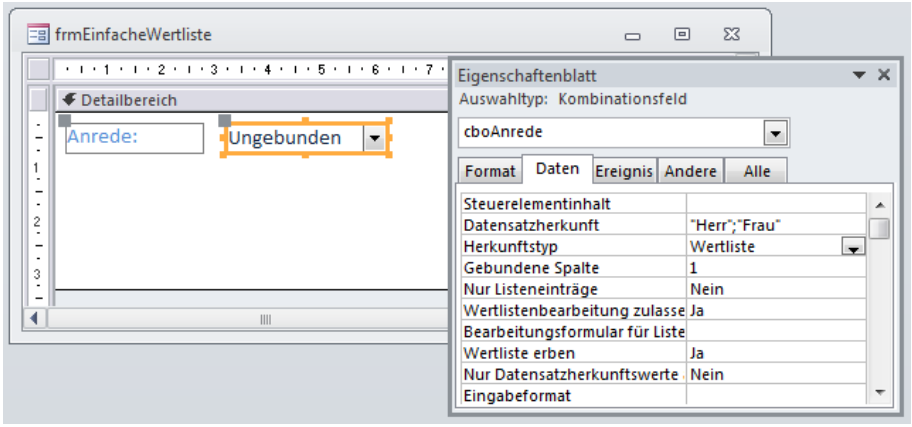



**Bild 1:** Kombinationsfeld mit einer einfachen Wertliste als Datensatzherkunft

```
"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:

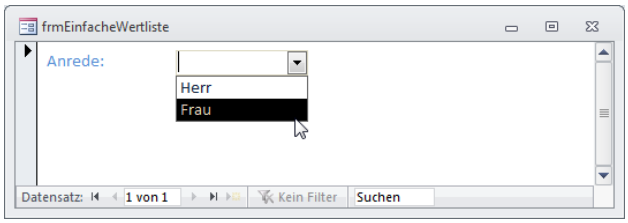



**Bild 2:** Auswahl eines Eintrags der Wertliste

```
? 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.

Wenn Sie nun mit folgendem Ausdruck den Wert des gewählten Eintrags ermitteln, liefert dies den Wert **1** oder **2** statt der Einträge **Herr** oder **Frau**:

```
? Forms!frmEinfacheWertliste!cboAnredeM
itID
```

## Wertliste per VBA zusammenstellen

Manche Konstellationen verlangen nach einer dynamischen Zusammenstellung des Ausdrucks, den Sie der Eigenschaft Datensatzherkunft zuweisen.

Grundsätzlich ist dies kein Problem: Sie brauchen nur eine entsprechende Schleife zu durchlaufen und die anzuzeigenden Werte zu einer durch Semikola getrennten Zeichenkette zusammenfügen.

Verwenden wir doch ein praktisches Beispiel, um dies zu demonstrieren. Ein Formular wie aus Bild 3 soll es ermöglichen, alle in dem im Textfeld **txtVerzeichnis** enthaltenen Unterverzeichnisse und Dateien anzuzeigen.

Zur Anzeige verwenden wir diesmal statt eines Kombinationsfeldes ein Listenfeld – dies ist in diesem Falle übersichtlicher.

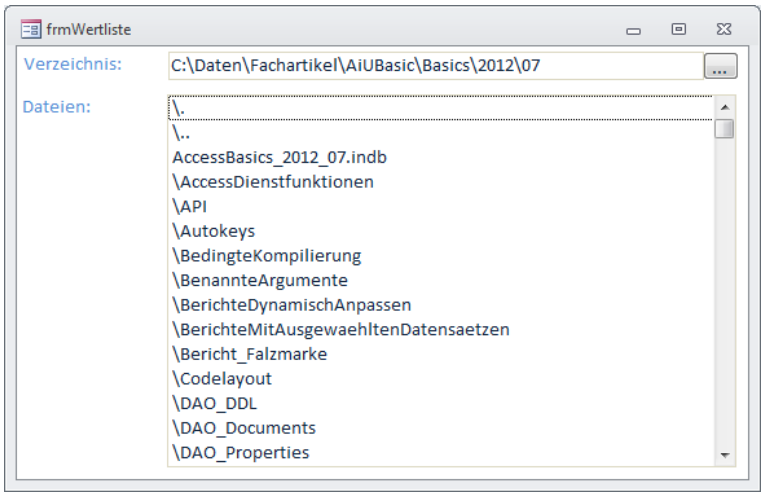



**Bild 3:** Dynamisch zusammengestellte Wertliste

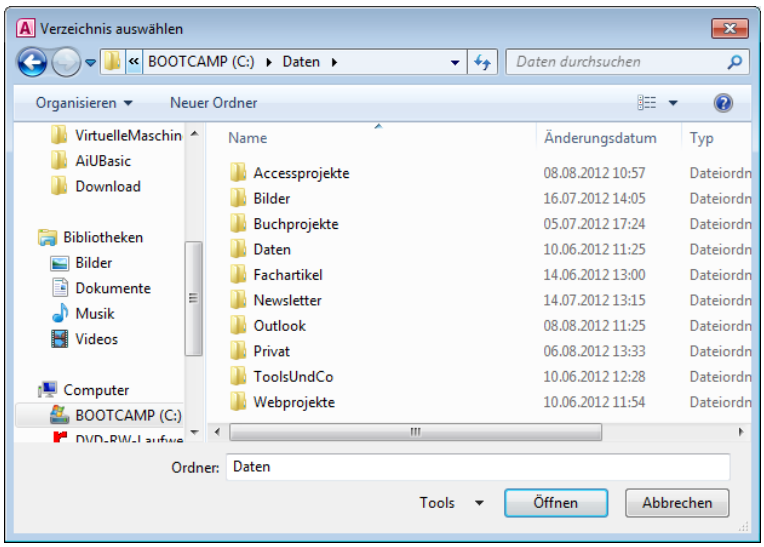



**Bild 4:** Dialog zur Auswahl eines Verzeichnisses

Damit Sie das Verzeichnis, dessen Unterverzeichnisse und Dateien angezeigt werden sollen, leicht auswählen können, fügen Sie neben dem Textfeld **txtVerzeichnis** eine Schaltfläche namens **cmdVerzeichnisAuswaehlen** hinzu.

Diese löst die folgende Prozedur aus:

```
Private Sub cmdVerzeichnisAuswaehlen_Click()
    Me!txtVerzeichnis = _
        OpenPathName(CurrentProject.Path)
    Me!lstDateien.RowSource = _
        DateienEinlesen(Me!txtVerzeichnis)
End Sub
```

Die erste Zeile ruft die Funktion **OpenPathName** auf, die einen Verzeichnisauswahl-Dialog wie in Bild 4 anzeigt. Die Funktion erwartet ein Startverzeichnis als Parameter und liefert den Namen des ausgewählten Verzeichnisses zurück.

Die zweite Anweisung erledigt das, was Sie im vorherigen Beispiel noch über das Eigenschaftsfenster erledigt haben: Sie weist der Eigenschaft **RowSource** (das ist das VBA-Pendant zum Eintrag **Datensatzherkunft** im Eigenschaftsfenster) die anzuzeigende Wertliste zu. Und diese wird hier mithilfe einer eigenen VBA-Funktion ermittelt.

Diese Funktion heißt **DateienEinlesen** und sieht wie in Listing 1 aus.

Die Funktion erwartet den Namen des zu untersuchenden Verzeichnisses als Parameter. Im ersten Schritt liest sie mit der **Dir**-Funktion unter Angabe des zu verwendenden Verzeichnisses das erste im Verzeichnis enthaltene Objekt ein. In der folgenden **Do While**-Schleife wird dies solange wiederholt, bis **Dir()** keinen Wert mehr zurückliefert – dies bedeutet, dass es keinen Eintrag mehr gibt.

# 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

# 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.

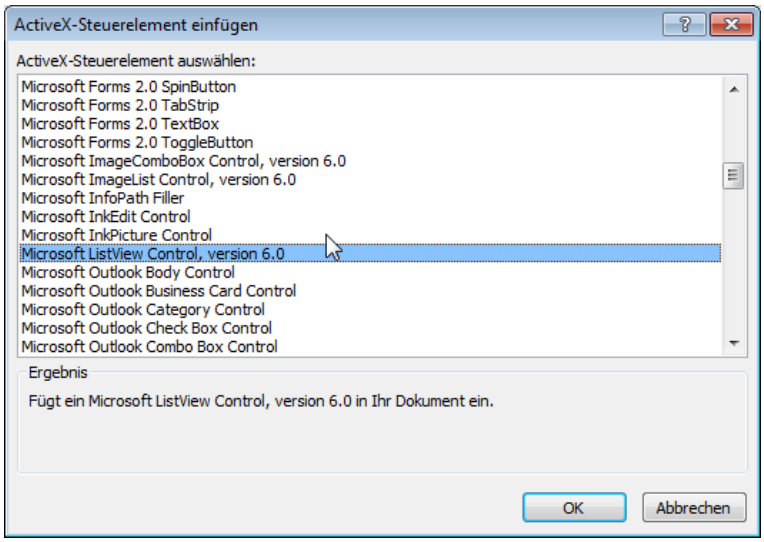



**Bild 1:** Auswählen des ListView-Steuerelements

## 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**.

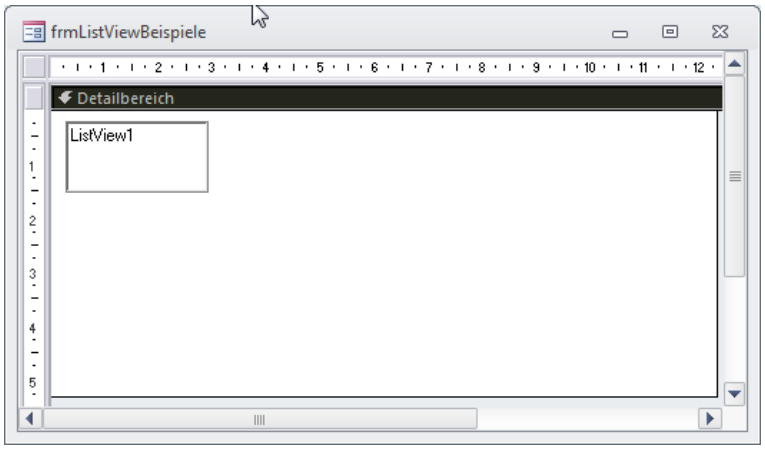



**Bild 2:** Ein ListView-Steuerelement direkt nach dem Hinzufügen

## 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.

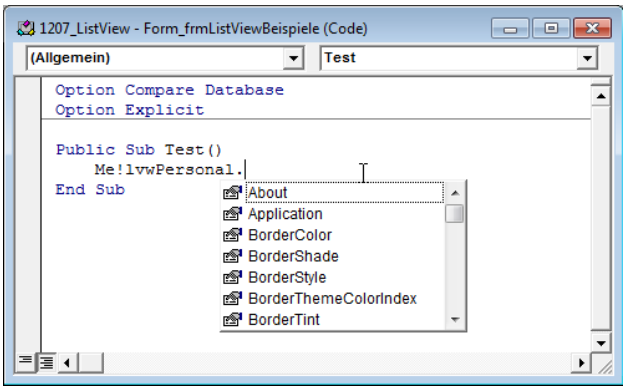



**Bild 3:** Standardeigenschaften von ActiveX-Steuerelementen

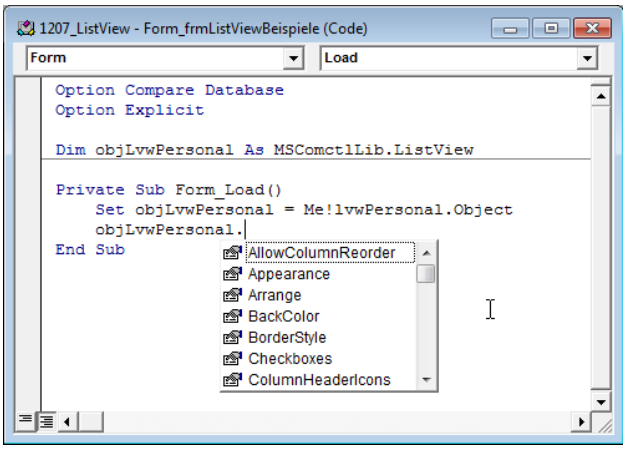



**Bild 4:** Spezielle Eigenschaften des ListView-Steuerelements

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

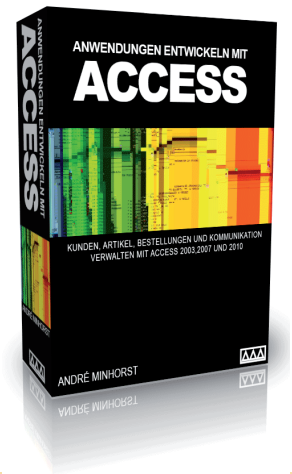
Zunächst betrachten wir die Wirkung verschiedener Werte für den Parameter **Index**. Dieser Index beginnt

Access [basics] 8/2012
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Editorial

Passiert es Ihnen auch schonmal, dass Sie den zeitlichen Aufwand für ein Projekt unterschätzen? Dann bin ich ja in guter Gesellschaft. In meinem speziellen Fall ging es um mein neues Buch **Anwendungen entwickeln mit Access**.

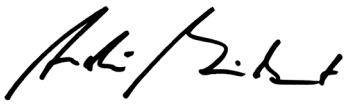
Eigentlich war ich schon vor einigen Wochen mit dem Thema durch. Das Buch schien perfekt zu sein! Dann allerdings kamen einige Meldungen von Access-Einsteigern, die das Buch bestellt und die Vorabversion als PDF gelesen hatten. Diese besagten, dass man es als Einsteiger an manchen Stellen ganz schön schwer habe – vor allem wenn es darum geht, wo denn der im Buch beschriebene Code nun landen soll. In einem Standardmodul? In einem Klassenmodul? Im Modul des Haupt- oder des Unterformulars? Und wie legt man dieses überhaupt an, wenn es noch nicht existiert? Ich bin diesen Lesern wirklich sehr dankbar, denn sie haben es mir ermöglicht, das Buch auch für Einsteiger noch viel interessanter zu gestalten. In den ersten Kapiteln habe ich noch einige Änderungen vorgenommen, um die einzelnen Schritte verständlicher zu machen. Außerdem habe ich beschlossen, die hinteren Kapitel, in denen es doch etwas schneller zur Sache geht, in ausführlicherer Form als PDF-Dokument zu veröffentlichen.




Das gedruckte Buch ist jetzt allerdings fertig – unabhängig davon, ob die für alle Käufer verfügbare eBook-Variante noch erweitert wird. Die Arbeit am Buch hat mir jedenfalls geholfen, den Fokus bei der Beschreibung von Vorgehensweisen wieder etwas mehr in Richtung Einsteiger zu rücken. Ich hoffe, dass auch Sie als Leser von **Access [basics]** davon profitieren.

Wer das Buch noch nicht kennt, kann sich unter **http://book.minhorst.com** gern die Leseproben ansehen. Im Gegensatz zur gängigen Praxis gibt es dort nicht nur das Inhaltsverzeichnis und ein freies Kapitel, sondern die ersten rund zehn Seiten eines jeden Kapitels zum freien Download. Schauen Sie es sich einfach einmal an – ich bin sicher, dass das Buch auch für Sie interessant ist.

Und nun: Viel Spaß mit der neuen Ausgabe!



Ihr André Minhorst

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst Verlag | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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.

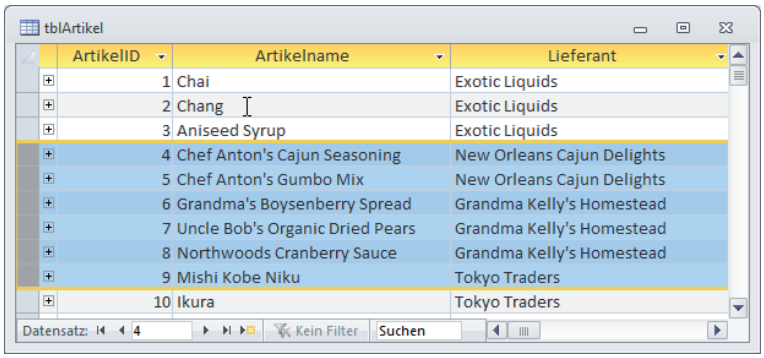



**Bild 3:** Kopieren nur einiger Zeilen einer Access-Tabelle

## 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.

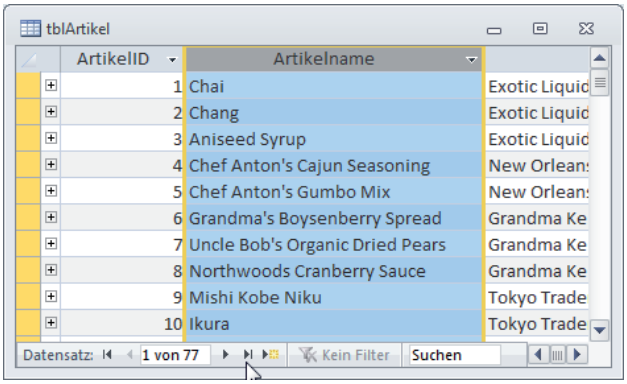



**Bild 4:** Kopieren einer einzelnen Spalte

## 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.

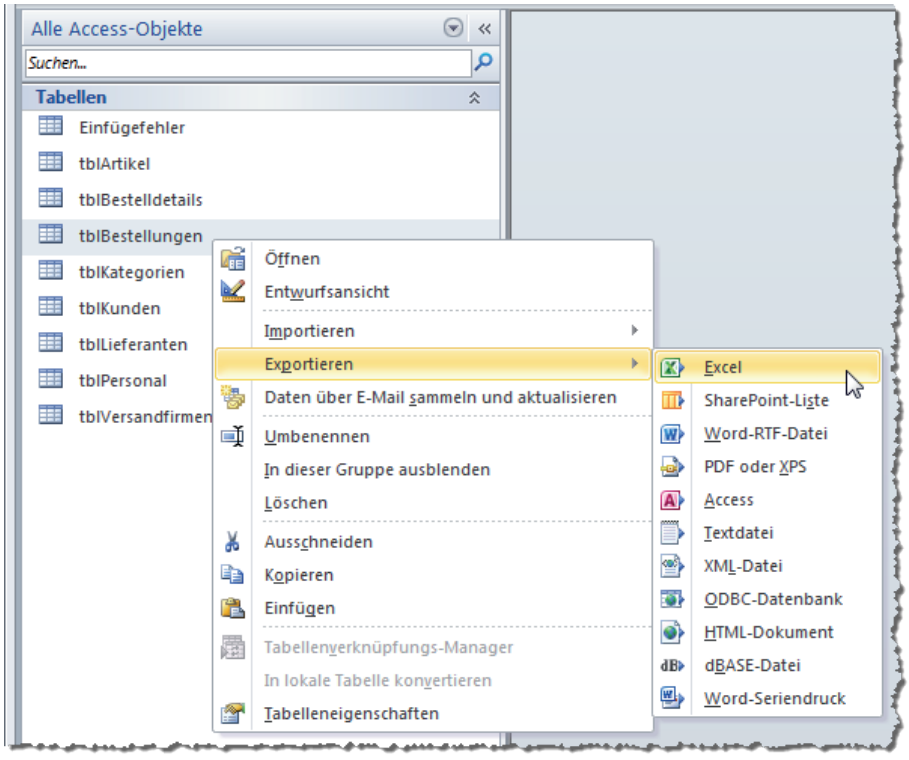



**Bild 5:** Exportieren einer kompletten Tabelle von Access nach Excel

Auch mit dieser Methode können Sie einzelne Datensätze speichern. Dazu öffnen

# 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.

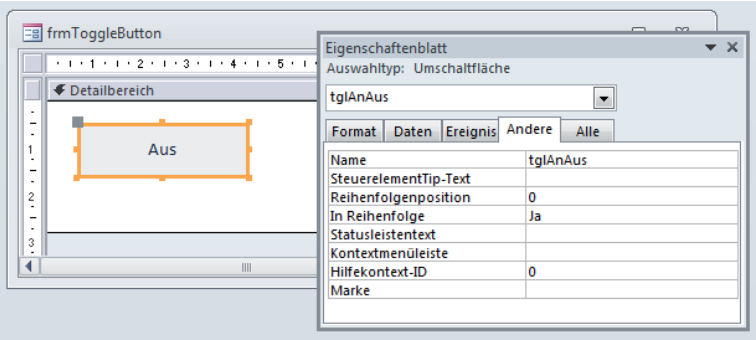



**Bild 1:** Eine einfache Umschaltfläche

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**.

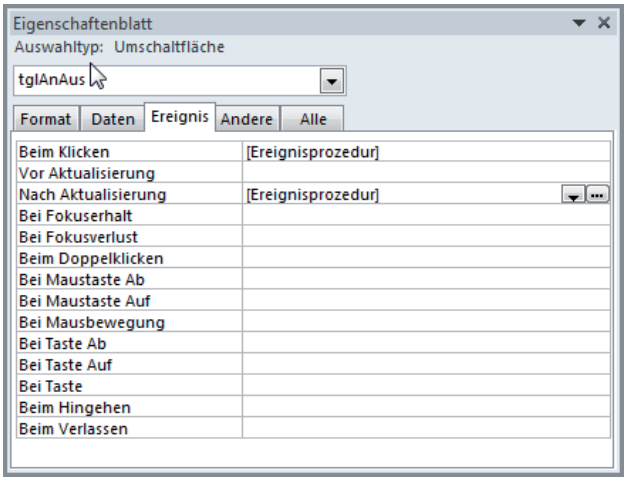



**Bild 2:** Ereigniseigenschaften der Umschaltfläche

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).

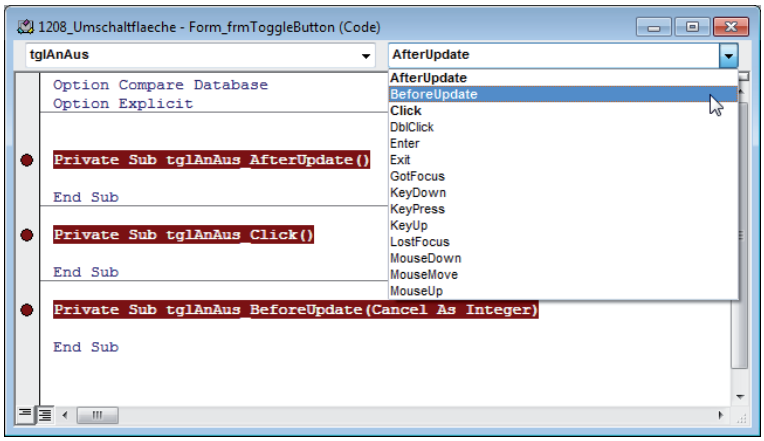



**Bild 3:** Wichtige Ereignisprozeduren einer Umschaltfläche

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 herunterdrückt, wird die Schaltfläche **cmdSchaltflaeche** aktiviert. Wenn die Umschaltfläche gelöst wird, deaktiviert dies die Umschaltfläche. Die **If...Then**-Abfrage untersucht dazu den Ausdruck **Me!tblAnAus**

# 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.

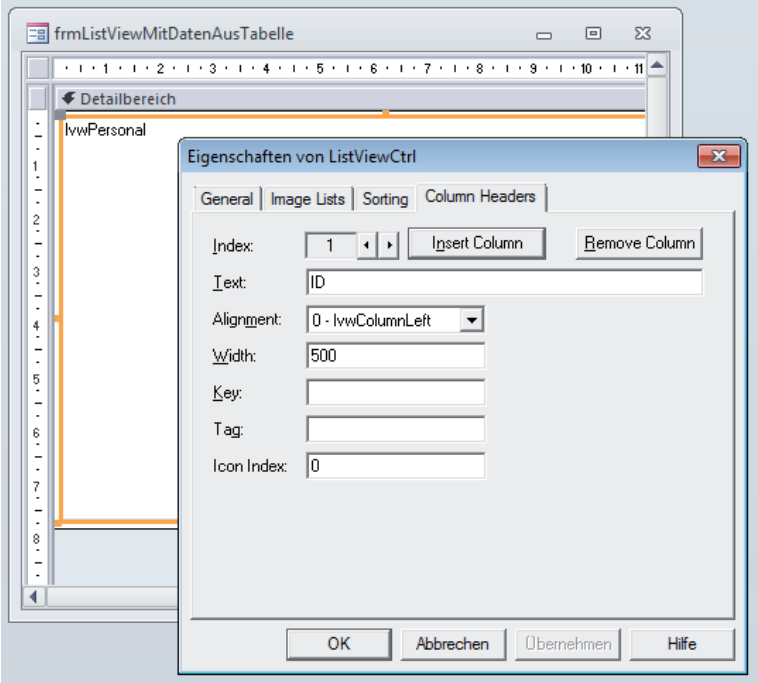



**Bild 1:** Einstellen der Spalten im ListView-Steuerelement

## 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

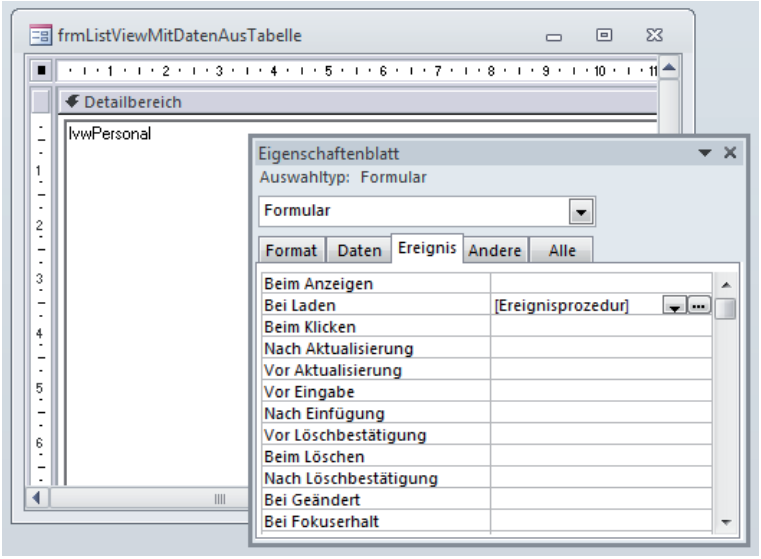



**Bild 3:** Hinzufügen einer Ereignisprozedur für das **Beim Laden**-Ereignis des Formulars mit dem **ListView**-Steuerelement

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:

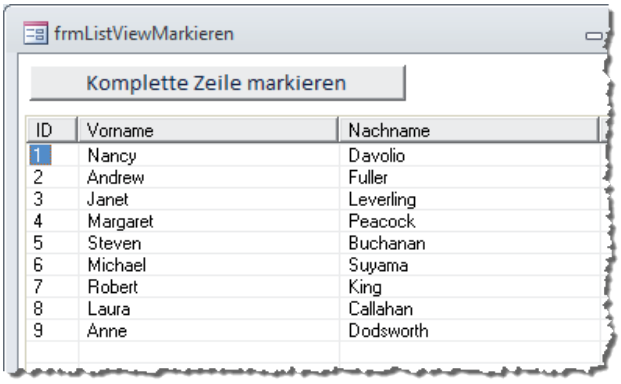



**Bild 1:** Einstellen der Spalten im ListView-Steuerelement

```
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:

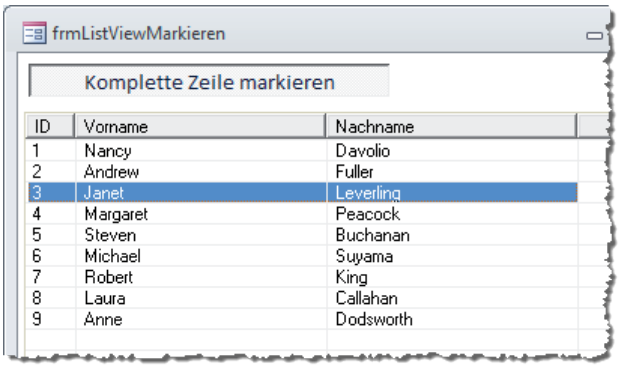



**Bild 2:** Markieren der kompletten Zeile im **ListView**

```
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-**

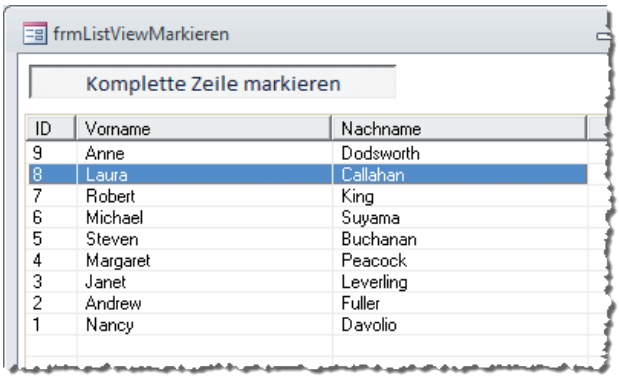



**Bild 3:** Datensätze in umgekehrter Reihenfolge

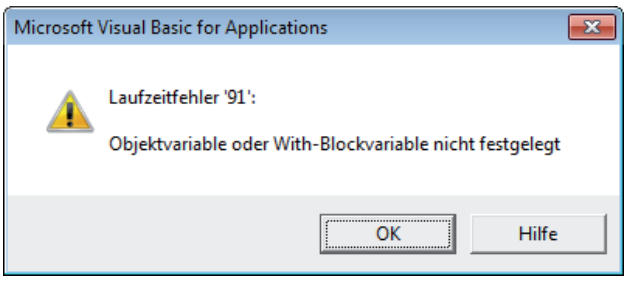



**Bild 4:** Fehler beim Zugriff auf **SelectedItem**, wenn kein Datensatz markiert ist

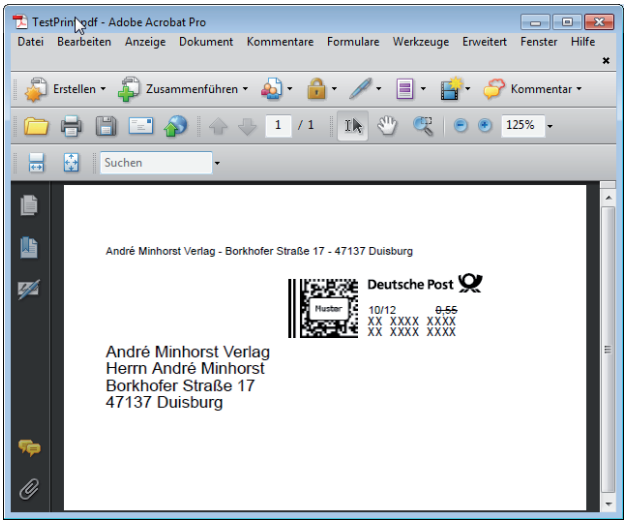
Access [basics] 9/2012
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Editorial

Ufff... endlich geschafft. Nein, nicht die neue Ausgabe von **Access [basics]** – die ist natürlich auch fertig. Aber vor allem freue ich mich, dass ich endlich mein neues Buch **Anwendungen entwickeln mit Access** in den Händen halten und vor allem an alle Leser schicken konnte, die es bereits bestellt haben. Und der damit verbundene Aufwand ist nicht zu unterschätzen. Es beginnt schon damit, die komplette Lieferung von Büchern zu lagern, dann alle bestellten Exemplare mit einem kleinen Präsent zu versehen und in einem entsprechenden Karton zu verpacken.

Noch interessanter wird es, wenn es um die Adressierung und die Frankierung geht. Bei mehreren hundert bestellten Büchern muss man sich genau überlegen, wo man den einen oder anderen Handgriff sparen kann, und wenn man statt eines Adressaufklebers und einer Briefmarke gleich beides auf einen Aufkleber bekommt, spart dies schon einige Arbeit. Wie Sie dies mithilfe der Deutschen Post und Access hinbekommen, hätte ich gern schon in dieser Ausgabe berichtet, aber dafür war es zu spät. Dafür können Sie sich in der kommenden Ausgabe auf die Vorgehensweise zum Erstellen von frankierten Adressaufklebern wie in der Abbildung freuen.

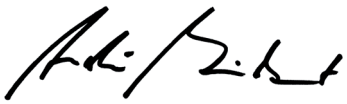
In diesem Heft erfahren Sie, wie Sie anwendungsweiter Tastenkombinationen nutzen können, wie Sie




Verknüpfungen zwischen Access und Excel herstellen und wie Sie ganz einfach Beispieldaten per Assistent in Ihre Tabellen füllen.

Außerdem lernen Sie, wie Sie Daten direkt im ListView-Steuerelement ändern und was es mit den beiden Begriffen Early Binding und Late Binding auf sich hat.

Und nun: Viel Spaß mit der neuen Ausgabe!



Ihr André Minhorst

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst Verlag | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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.

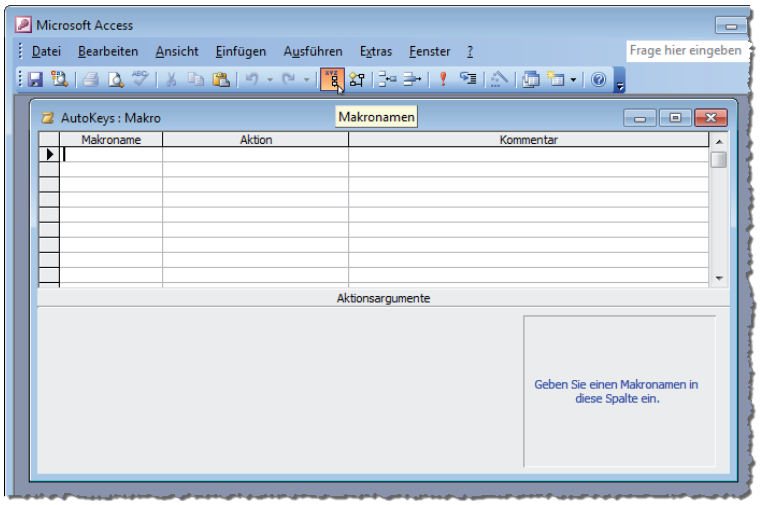



**Bild 1:** Anlegen eines neuen **AutoKeys**-Makros

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**)

- **Aktion**: Meldung (mit den Werten **Sie haben die Tastenkombination Strg + Umschalt + P gedrückt** für den Parameter **Meldung** und **AutoKeys-Meldung** für den Parameter **Titel**).

Wenn Sie das Makro nun mit der Tastenkombination **Strg + S** speichern, können Sie gleich einmal die Tastenkombination **Strg + Umschalt + P** ausprobieren. Selbst bei geöffnetem Makro-Entwurf erscheint die erwartete Meldung (siehe Bild 2).

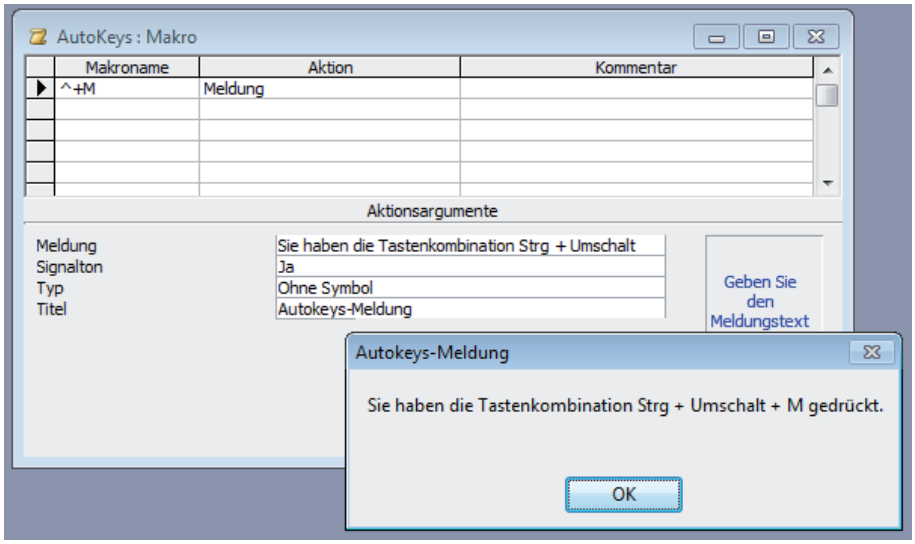



**Bild 2:** **AutoKeys**-Makro zum Anzeigen einer Meldung per Tastenkombination

## AutoKeys-Makro unter Access 2007

In Access 2007 erstellen Sie das Makro auf etwas andere Weise. Dort wählen Sie im Ribbon den Eintrag **Erstellen|Andere|Makro|Makro**. Nach dem Erscheinen des neuen Makros in der Entwurfsansicht gehen Sie wie oben beschrieben vor.

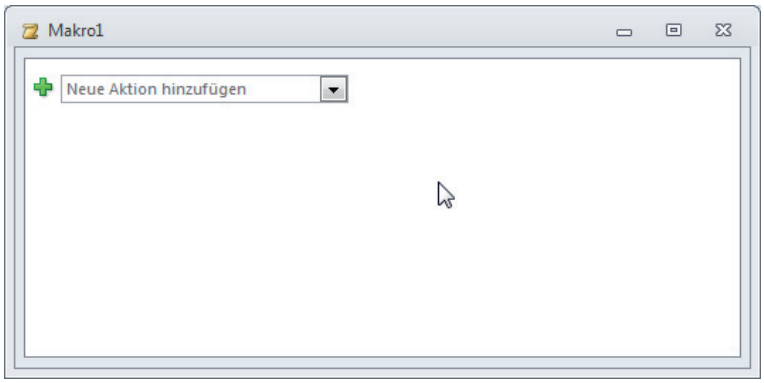



**Bild 3:** Neues Makro in der Entwurfsansicht von Access 2010

## AutoKeys-Makro unter Access 2010

Unter Access 2010 heißt der Ribbon-Befehl zum Erstellen eines neuen Makros **Erstellen|Makros und Code|Makro**. Hier erwartet Sie allerdings nicht der von älteren Versionen her bekannte Makro-Entwurf, sondern eine komplett neue Ansicht (siehe Bild 3).

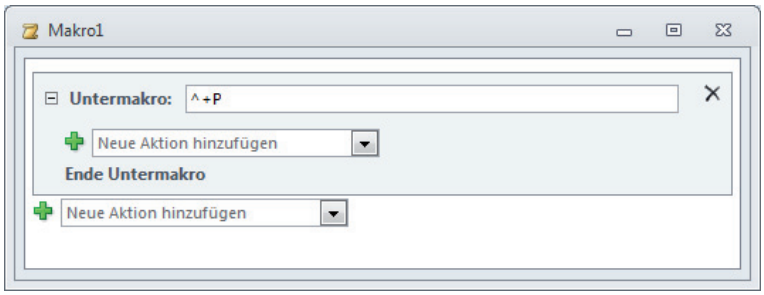



**Bild 4:** Einfügen eines Untermakros

Sie können hier nicht wie in älteren Access-Versionen eine Spalte mit der Überschrift **Makroname** einblenden. Unter Access 2010 werden Makros hierarchisch aufgebaut, wobei ein Untermakro durch die Auswahl des Eintrags **Untermakro** im Kombinationsfeld mit dem Text **Neue Aktion hinzufügen**.

Als Namen des neuen Makros geben Sie auch hier den Code für die Tastenkombination ein, also **^+P** (siehe Bild 4).

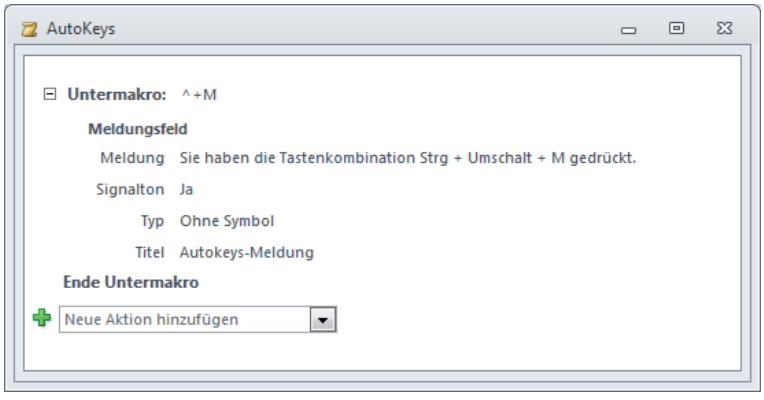



**Bild 5:** Das **AutoKeys**-Makro unter Access 2010

# 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.

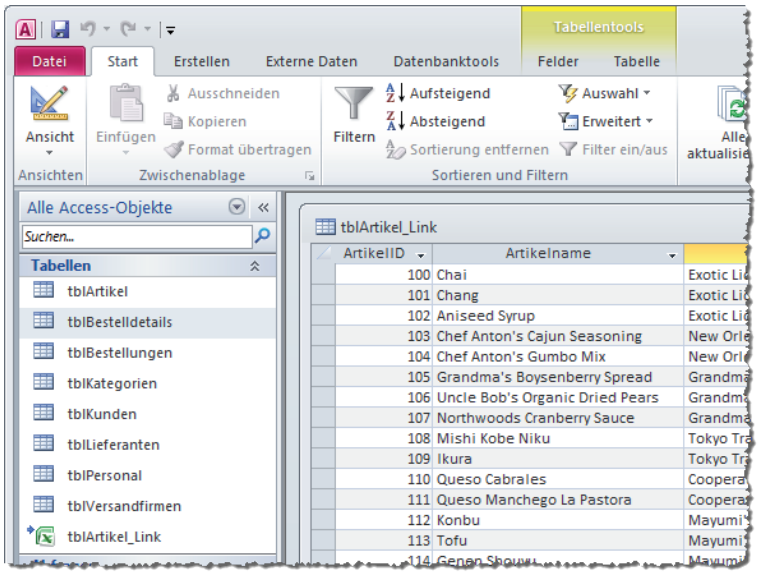



**Bild 1:** Verknüpfte Tabelle im Navigationsbereich und in der Datenblattansicht

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-

nishierarchie verwenden wie ich, werden Sie beim Zugriff auf die Daten der verknüpften Excel-Tabelle eine Fehlermeldung ernten (siehe Bild 2). Diese besagt, dass das Objekt **<Tabellenname>$** nicht gefunden werden konnte (im Beispiel **tblArtikel$**).

In einem solchen Fall bemühen Sie den Tabellenverknüpfungs-Manager. Diesen Starten Sie unter Access 2003 mit dem Menübefehl **Extras|Datenbank-Dienstprogramme|Tabellenverknüpfungs-Manager**, unter Access 2007 mit dem Ribbon-Eintrag **Datenbanktools|Datenbanktools|Tabellenverknüpfungs-Manager** und unter Access 2010 mit dem Ribbon-Eintrag **Externe Daten|Importieren und Verknüpfen|Tabellenverknüpfungs-Manager** (siehe Bild 3).

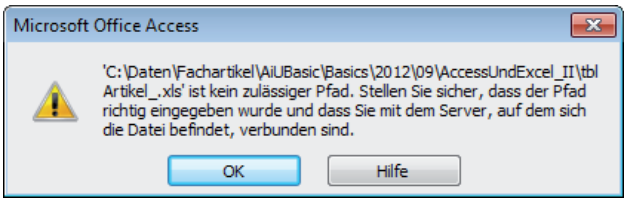



**Bild 2:** Fehler beim Versuch, auf eine Excel-Tabelle in einem nicht auffindbaren Excel-Dokument zuzugreifen

Der Dialog selbst sieht in den verschiedenen Access-Versionen gleich aus, und zwar so wie in Bild 4. Hier wählen Sie die Verknüpfung aus, deren Quelldatei nicht aktuell ist, und bewirken mit einem Klick auf die Schaltfläche **OK** die Anzeige eines **Datei öffnen**-Dialogs zur Auswahl der Quelldatei der Verknüpfung.

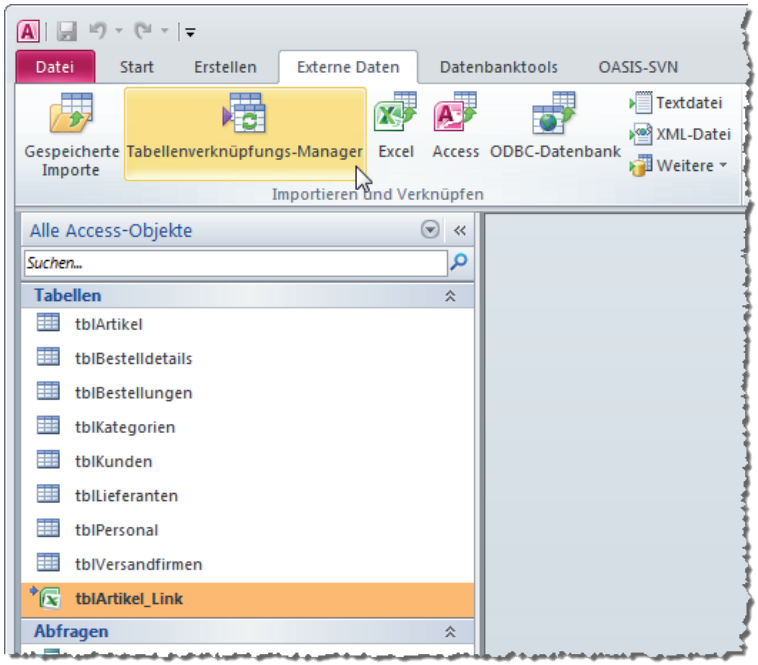



**Bild 3:** Aufrufen des Tabellenverknüpfungs-Managers unter Access 2010

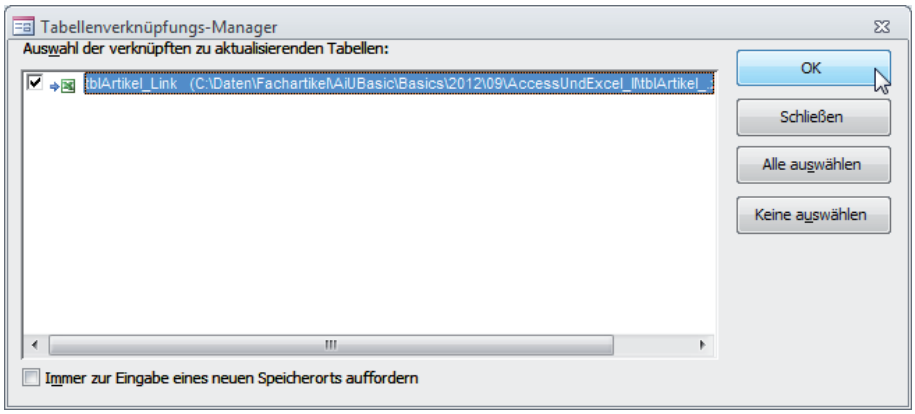



**Bild 4:** Sollte eine markierte Verknüpfung keine passende Datei finden, wird nach dem Klick auf **OK** ein entsprechender **Datei öffnen**-Dialog zur Auswahl der Datei angezeigt.

Anschließend können Sie wieder auf die in der zugrunde liegenden Excel-Tabelle enthaltenen Daten zugreifen.

## Verknüpfen von Access-Tabellen in Excel

Andersherum können Sie auch die Daten einer Access-Tabelle in Excel anzeigen. Unter Excel 2003 starten Sie die notwendigen Schritte mit einem Mausklick auf den Menüeintrag **Daten|Externe Daten importieren|Daten importieren**. Unter Excel 2007 finden Sie diesen Befehl im Ribbon unter **Daten|Externe Daten abrufen|Aus Access**. Der gleiche Ribbon-Eintrag ist auch unter Excel 2010 verfügbar.

Dieser Befehl öffnet zunächst einen **Datei öffnen**-Dialog, mit dem Sie zunächst die Quelldatei auswählen können. Anschließend erscheint ein weiterer Dialog mit der Überschrift **Tabelle auswählen**, mit dem Sie die Quelltabelle festlegen (siehe Bild 5).

In einem weiteren Dialog mit dem Titel **Daten importieren** wählen Sie aus, ob die Daten in einer Tabelle, in einem PivotTable-Bericht oder in einem PivotChart- und PivotTable-Bericht landen sollen (siehe Bild 6). Mit den möglichen Ergebnissen experi-

# 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-

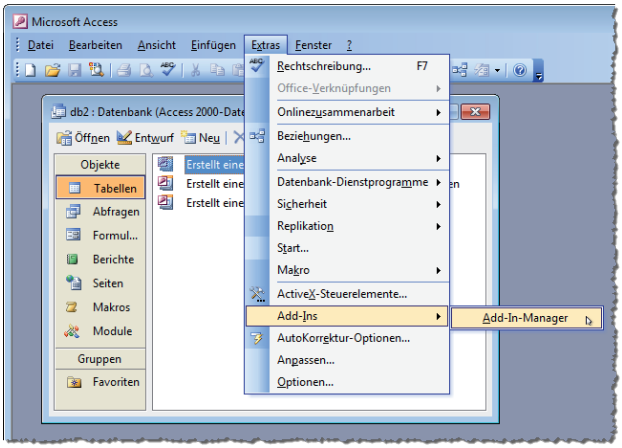



**Bild 1:** Installieren eines Access-Add-Ins unter Access 2003

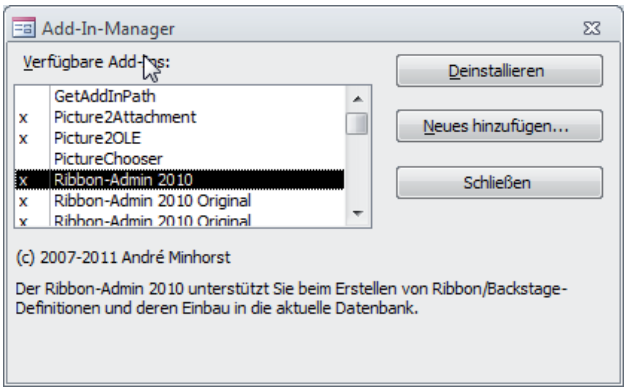



**Bild 2:** Der Add-In-Manager in Aktion

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**.

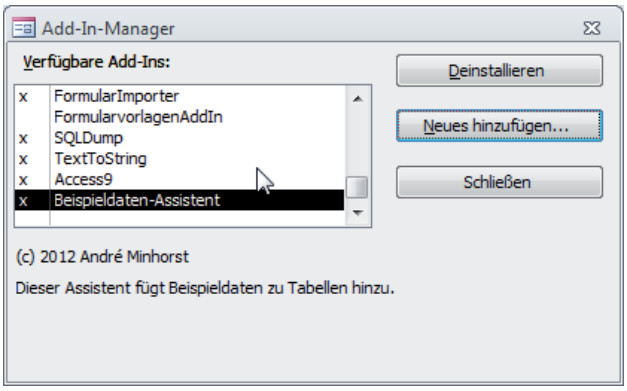



**Bild 3:** Ein frisch hinzugefügtes Access-Add-In

- 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.

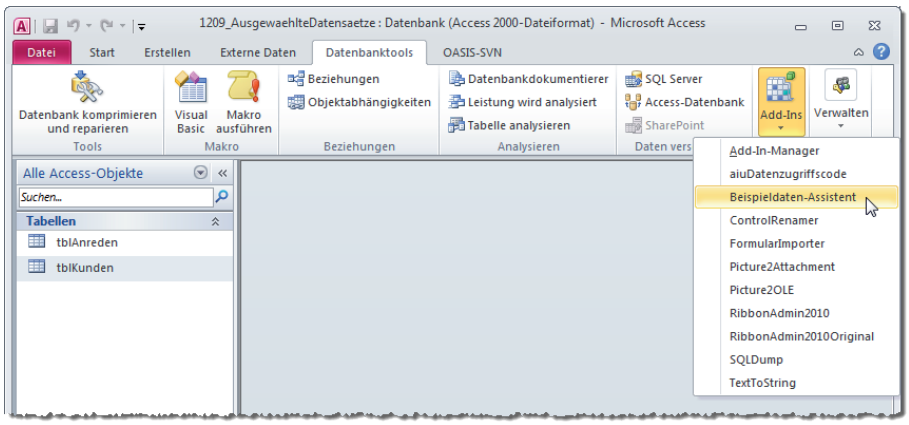



**Bild 4:** Aufruf des Add-Ins über die Liste der installierten Add-Ins

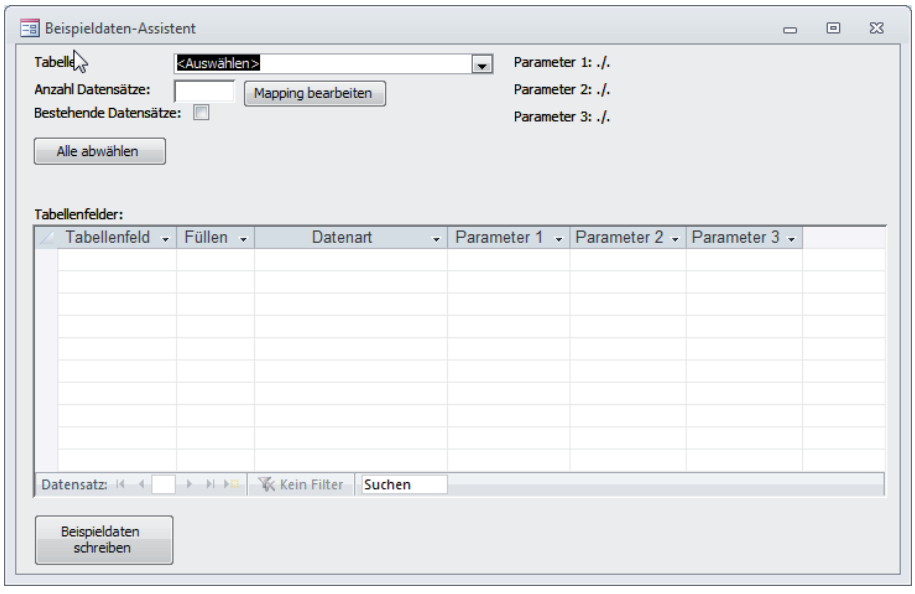



**Bild 5:** Beispieldaten-Assistent gleich nach dem Aufruf

## 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).

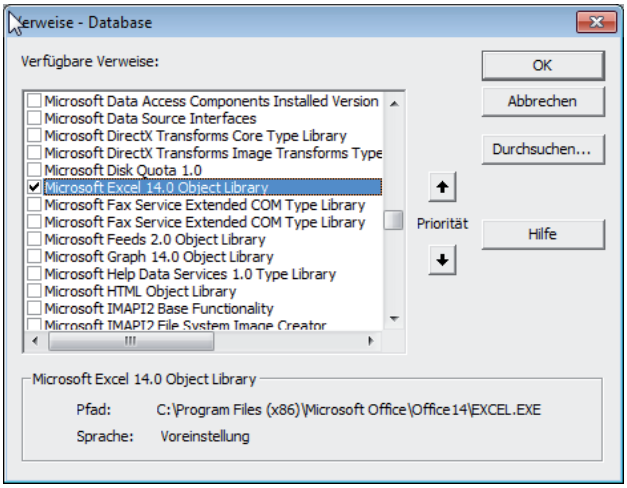



**Bild 1:** Hinzufügen eines Verweises auf die Excel-Bibliothek

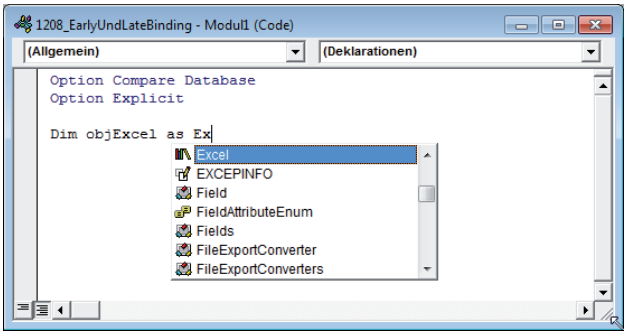



**Bild 2:** Beim Early Binding stehen die Elemente der referenzierten Bibliothek per IntelliSense zur Auswahl bereit

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.

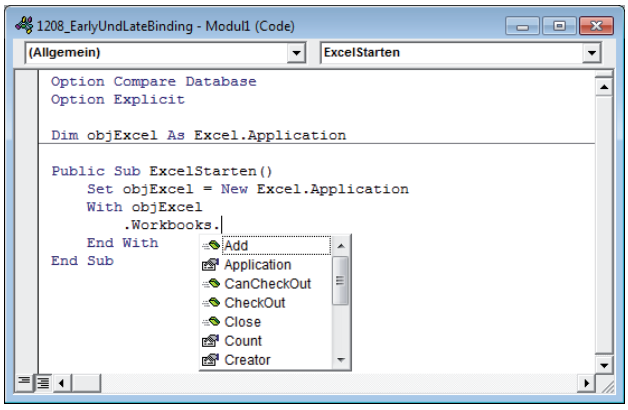



**Bild 3:** Auch die Eigenschaften und Methoden der Objektvariablen werden per IntelliSense angezeigt.

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).

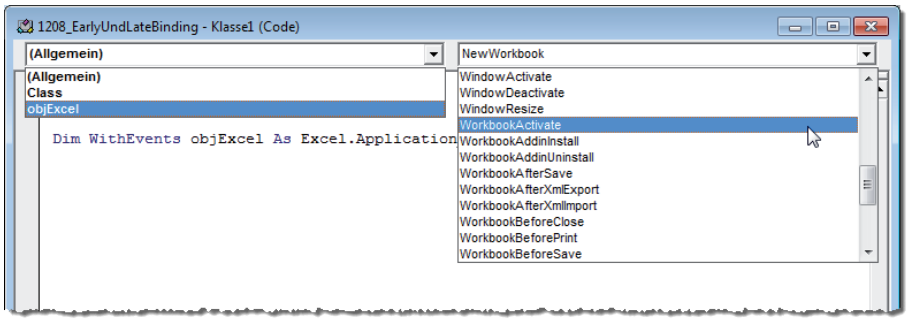



**Bild 4:** Einsatz von Ereignissen eines mit **WithEvents** deklarierten Objekts

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).

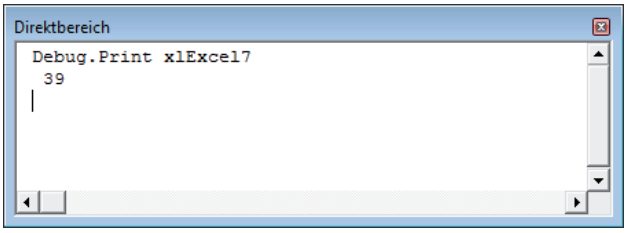



**Bild 5:** Ausgabe des tatsächlichen Wertes einer Konstanten der Excel-Bibliothek

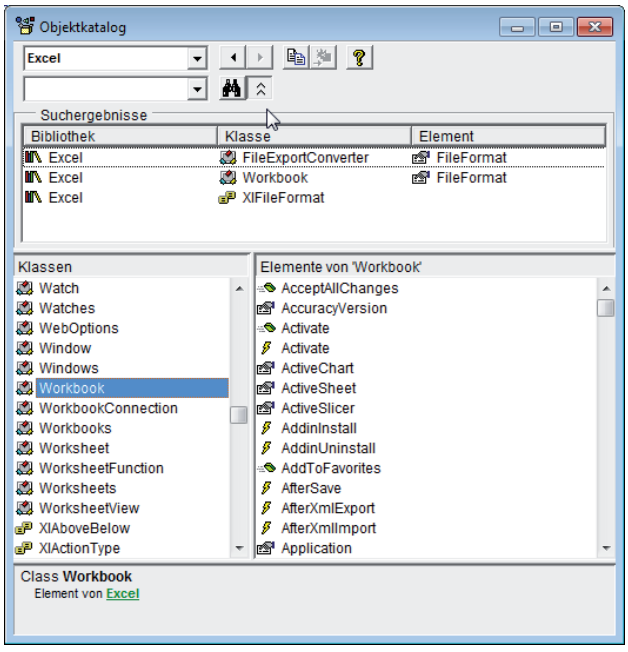



**Bild 6:** Der Objektkatalog bietet Einblick in alle Elemente, Eigenschaften, Methoden und Ereignisse von Objekten

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.

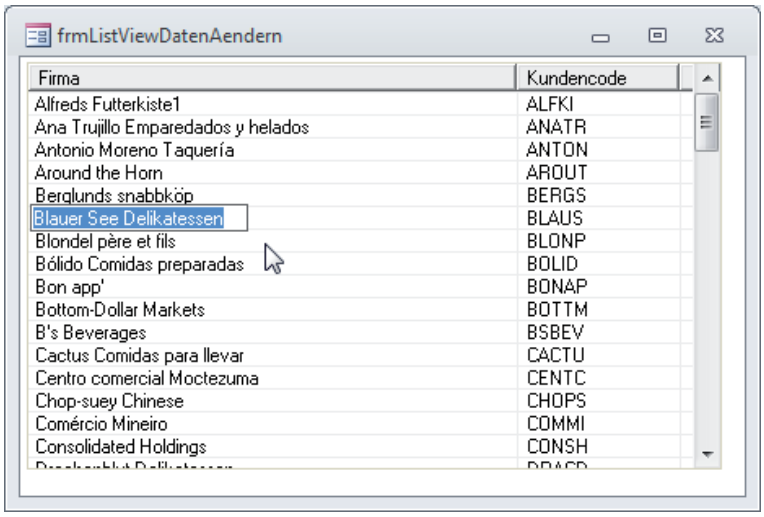



**Bild 1:** Ändern der in der ersten Spalte angezeigten Daten

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.

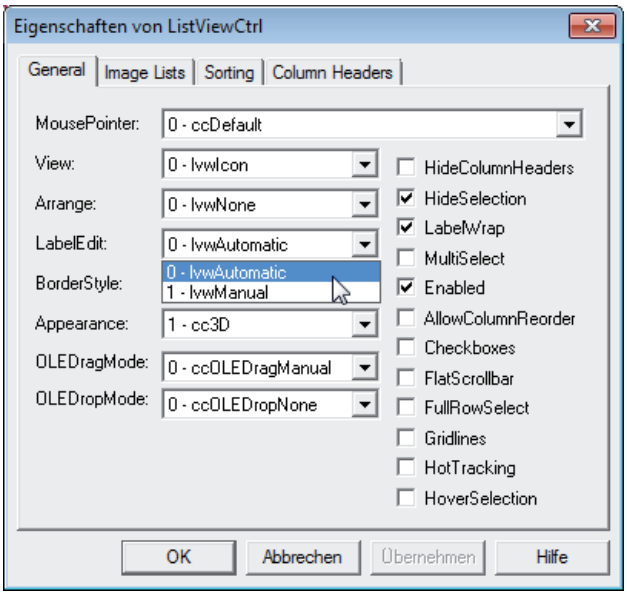



**Bild 2:** Eigenschaft für das Verhalten beim Ändern der ersten Spalte

## 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-

Access [basics] 10/2012
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Editorial

Wie in der vorherigen Ausgabe angekündigt, kümmeren wir uns im aktuellen Heft um ein Angebot der Deutschen Post namens **Internetmarke**. Damit können Sie auf Basis von **.cvs**-Dateien beispielsweise Etiketten mit Adresse und Porto ausdrucken.

Sie benötigen lediglich eine entsprechende Adressentabelle, eine Abfrage, welche die Adressen in einer Form aufbereitet, die durch die Webseite der Deutschen Post eingelesen werden kann, eine daraus abgeleitete **.csv**-Datei und ein paar Mausklicks auf **http://www.post.de**.

Wie all dies funktioniert, erfahren Sie im Artikel Internetmarke mit Access nutzen ab Seite 13.

Zuvor kümmern wir uns um die Möglichkeiten, Datensätze individuell auszuwählen. Sie haben ja mittlerweile einige Möglichkeiten kennengelernt, um Daten mithilfe von Abfragen zu filtern. Dort müssen Sie allerdings Kriterien definieren. Manchmal gibt es solche Kriterien aber nicht.

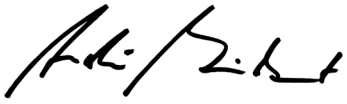
Das könnte zum Beispiel der Fall sein, wenn Sie die Empfänger eines schönen Weihnachtspräsents auswählen. Da geht es oberflächlich vielleicht um den Umsatz, den Sie mit einem Kunden gemacht haben, aber vielleicht haben Sie auch Kunden, mit denen Sie besonders gern zusammenarbeiten und denen Sie daher ein besonderes Präsent zukommen lassen wollen. Da hilft nur eins: das individuelle Auswählen der gewünschten Datensätze per Mausklick.

Die technischen Möglichkeiten zu diesem Thema finden Sie im Artikel **Datensätze individuell auswählen** ab Seite 2, den Rest müssen Sie erledigen!

Schließlich zeige ich Ihnen unter dem Titel **Synchrone Unterformulare** ab Seite 8, wie Sie zwei Unterformulare innerhalb eines Hauptformulars synchronisieren. Und zwar so, dass das zweite Unterformular nur solche Daten anzeigt, die zu dem im ersten Unterformular markierten Datensatz passen.

Schließlich wünsche ich Ihnen einen guten Start in die Vorweihnachtszeit. Gönnen Sie sich einen Bummel auf dem Weihnachtsmarkt, genießen Sie das eine oder andere Heißgetränk und laben Sie sich an den gebotenen Köstlichkeiten. ;-)

Vor allem aber: Viel Spaß mit der neuen Ausgabe!



Ihr André Minhorst

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst Verlag | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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.

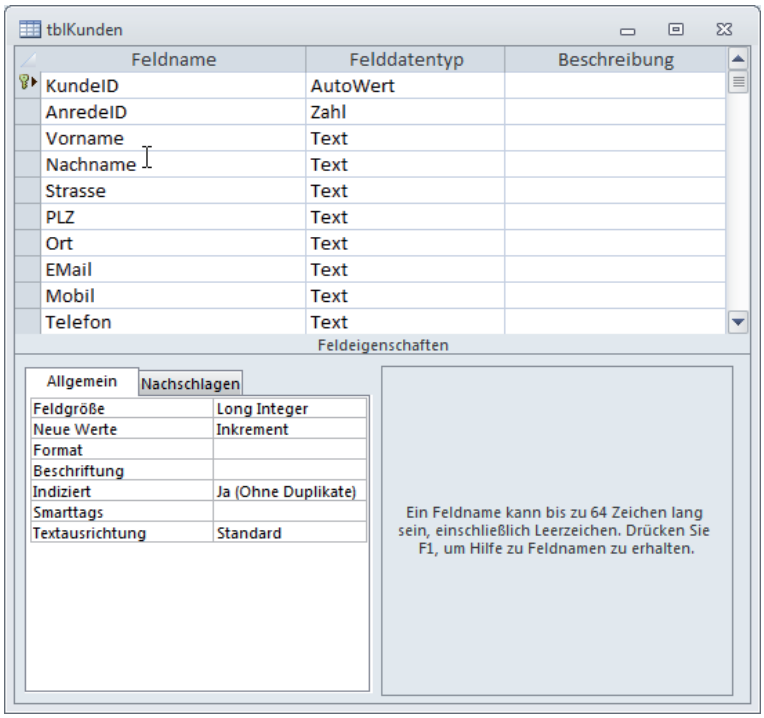



**Bild 1:** Beispieltabelle für die Auswahl von Datensätzen

## 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.

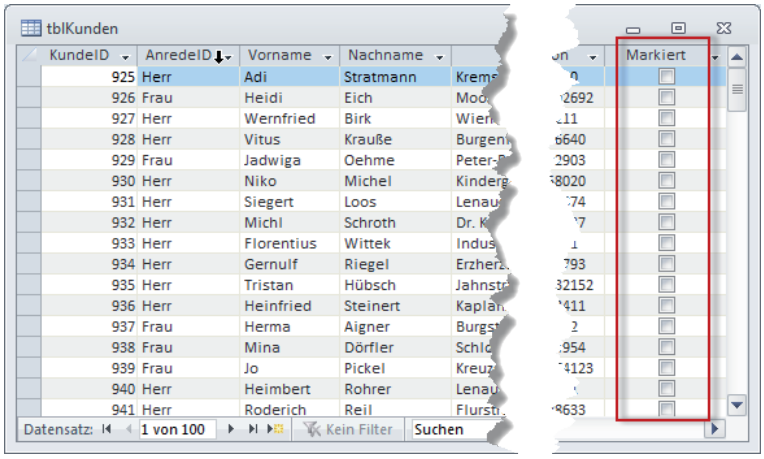



**Bild 2:** Datenblattansicht mit **Markiert**-Feld

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.

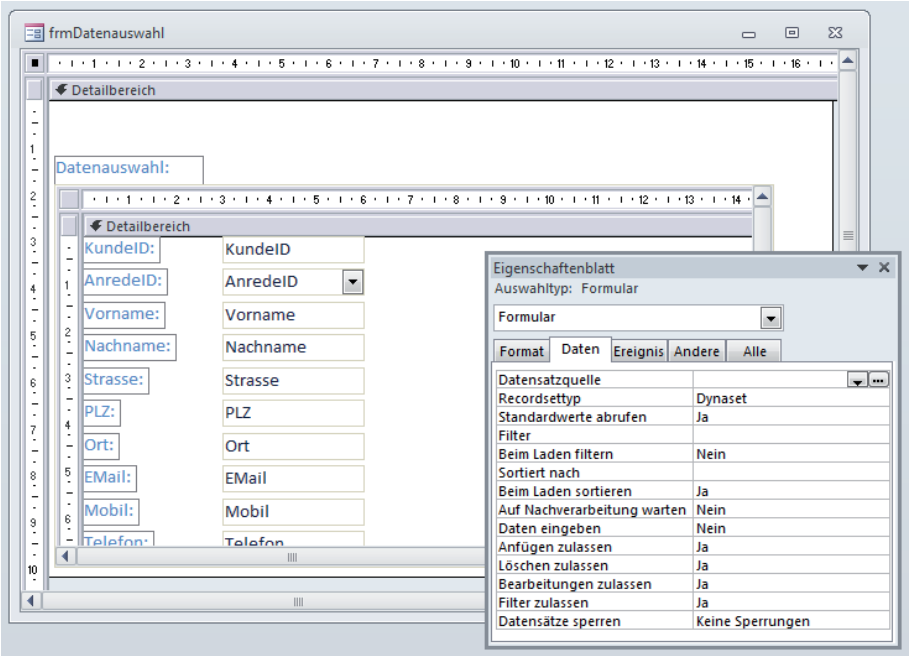



**Bild 3:** Haupt- und Unterformular der Beispieldatenbank

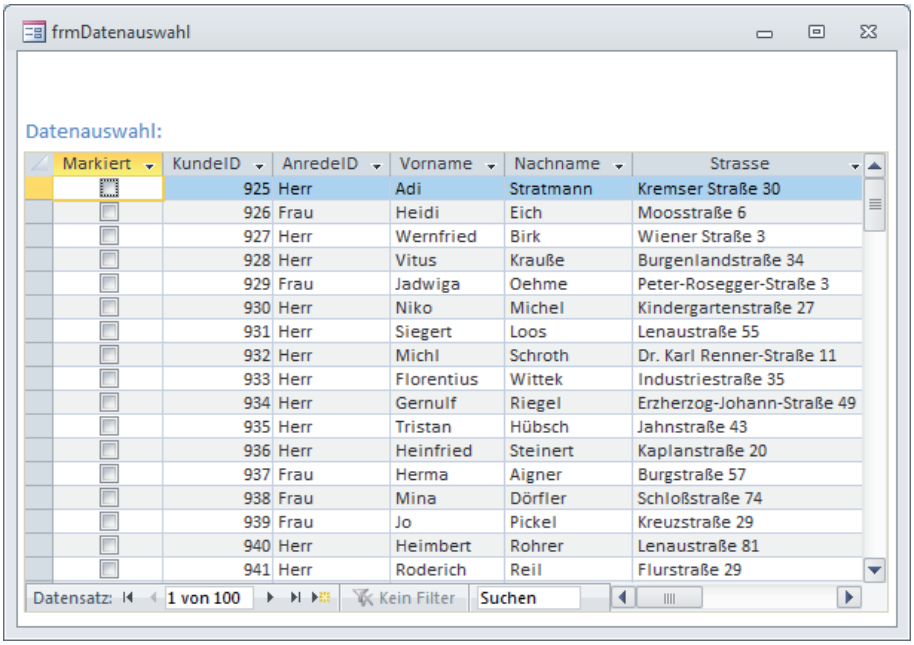



**Bild 4:** Anzeige der auszuwählenden Datensätze im Formular **frmDatenauswahl**

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.

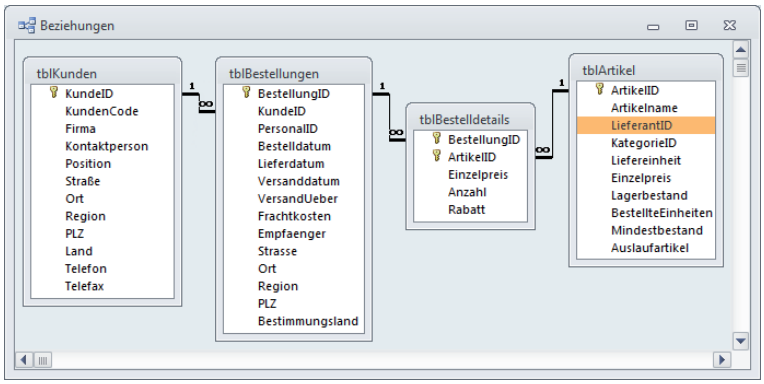



**Bild 1:** Diese Tabellen sollen im Formular angezeigt werden.

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.

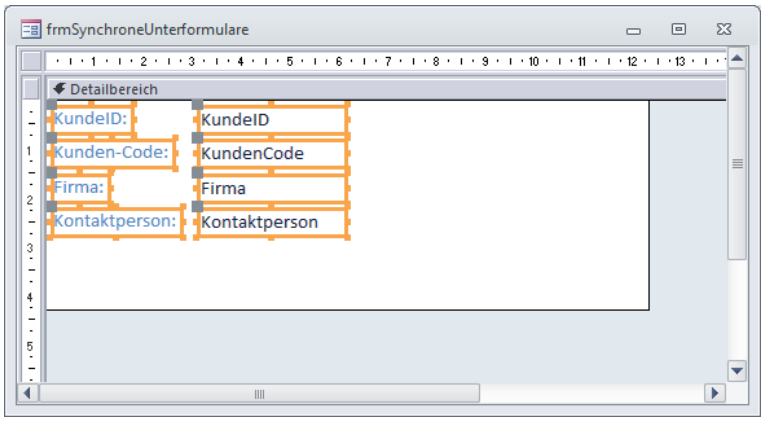



**Bild 2:** Ausgangsbasis: Das Formular **frmSynchroneUnterformulare** mit Daten aus der Tabelle **tblKunden**

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.

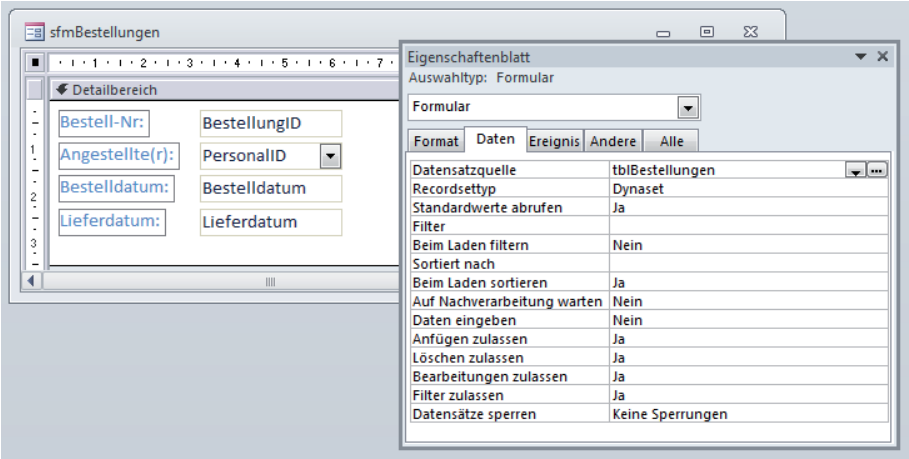



**Bild 3:** Das erste Unterformular zeigt die Daten der Tabelle **tblBestellungen** an.

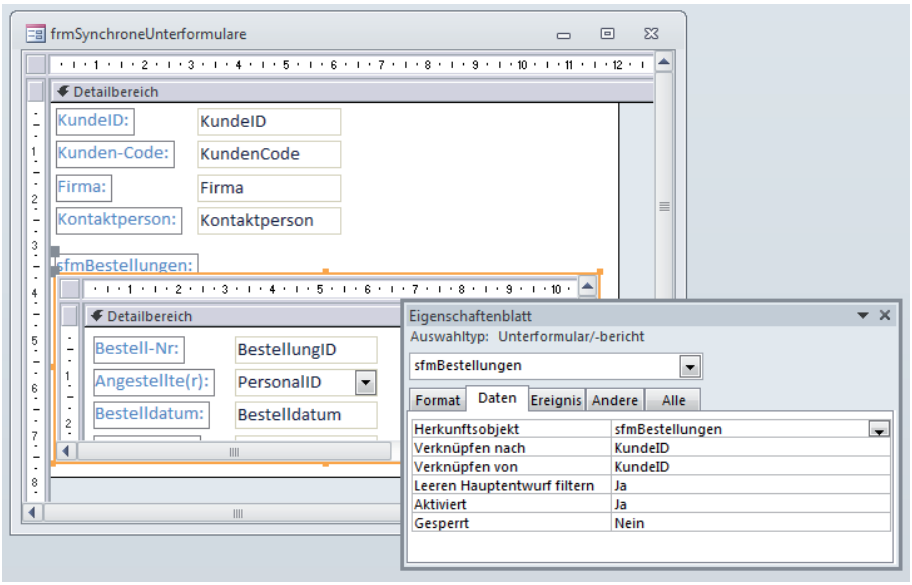



**Bild 4:** Einbau des ersten Unterformulars

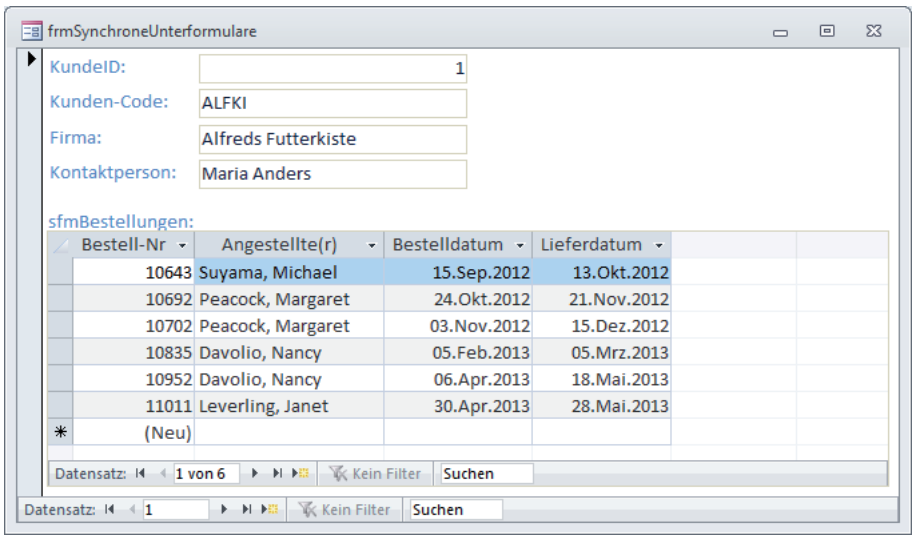



**Bild 5:** Haupt- und Unterformular in Aktion

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.

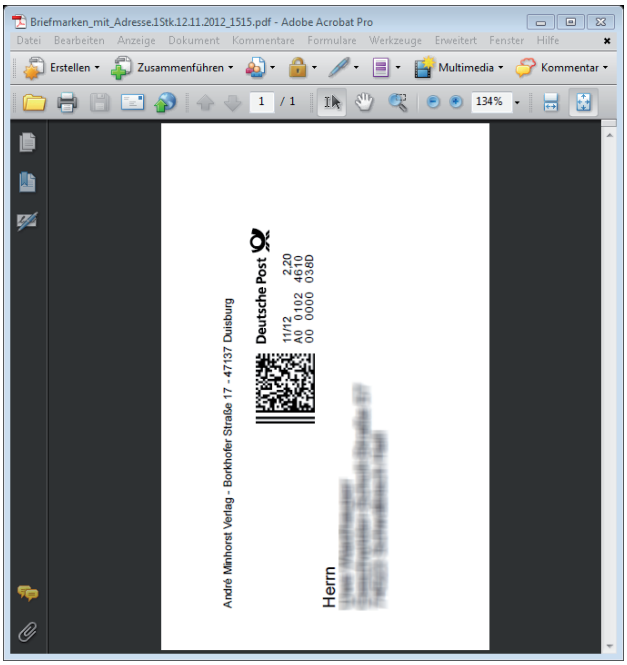



**Bild 1:** Beispiel für ein Adressetikett

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**.

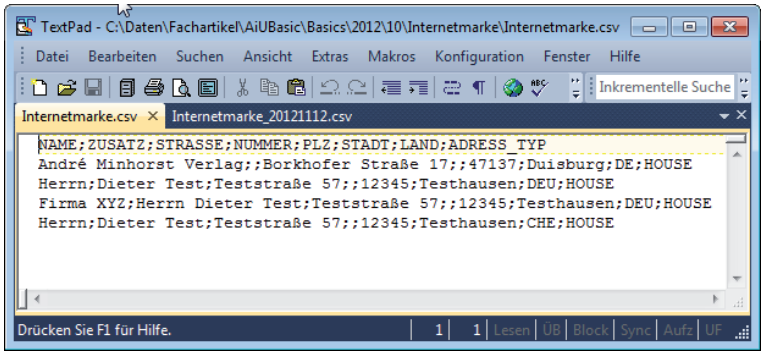
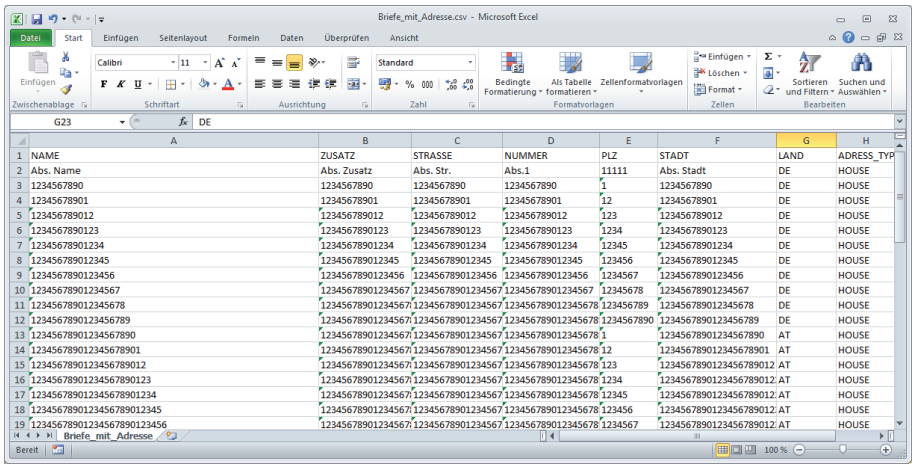



**Bild 2:** .csv-Datei für den Upload von Adressdaten

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**.

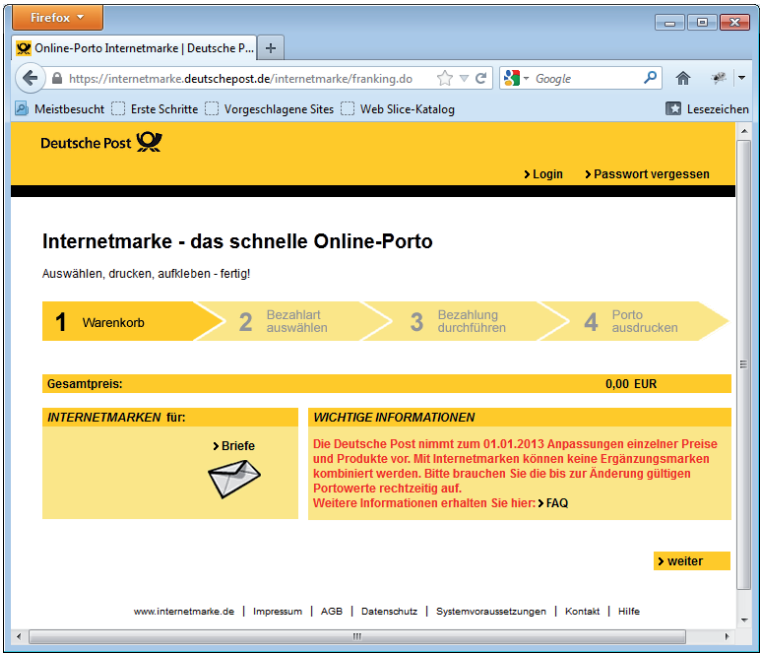



**Bild 4:** Startpunkt beim Erstellen von Internetmarken

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).

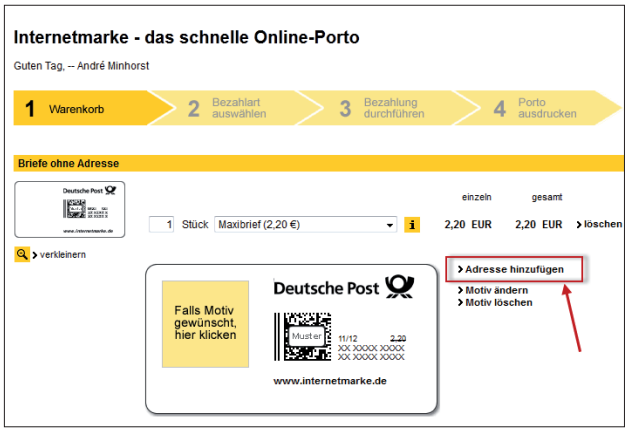



**Bild 5:** Einblenden der Befehle zum Hinzufügen von Adressen

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).

Access [basics] 11/2012
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Editorial

Es geht in die Zielgerade! Noch eine Ausgabe – die Nummer 12/2012 – und dann geht es in die verdienten Weihnachtsferien.

In der vorliegenden Ausgabe finden Sie noch einmal interessante Themen. Im ersten Beitrag stellen wir die Grundagen zum Registersteuerelement vor. Dieses Steuerelement hilft, auf überfüllten Formularen Platz zu schaffen oder auch einfach nur verschieden Ebenen von Steuerelementen ein- oder auszublenden und gleichzeitig die Editierbarkeit im Formularentwurf zu erhalten (**Das Registersteuerelement**, ab Seite 2).

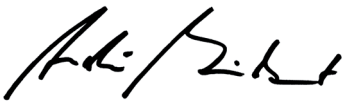
Der zweite Artikel setzt die Artikelreihe rund um das ListView-Steuerelement fort. Dieses kann durchaus auch ein Kontrollkästchen vor jeder Zeile einblenden. Damit aktivieren oder deaktivieren Sie Einträge temporär, oder Sie füllen diese mit den Daten eines Ja/Nein-Feldes der zugrunde liegenden Tabelle und speichern Änderungen, die der Benutzer daran vornimmt, wieder in der Datenbank. Alles zum Umgang mit Kontrollkästchen im Steuerelement erfahren Sie unter dem Titel **ListView-Steuerelement: Kontrollkästchen nutzen** ab Seite 7.

Wer bereits die eine oder andere Aktionsabfrage unter Access erstellt hat, möchte vielleicht auch wissen, wie er diese per VBA-Anweisung ausführen kann. Dies erläutert der Artikel **Aktionsabfragen per VBA** ab Seite 10.

Formulare, Berichte und Steuerelemente wie das Listenfeld oder das Kombinationsfeld füllen Sie am einfachsten mit einer Tabelle oder Abfrage als Datenherkunft beziehungsweise Datensatzherkunft. Gelegentlich sollen diese Datenquellen jedoch in Abhängigkeit von Daten im Formular eingeschränkt werden. Unter dem Titel **Dynamische Datenherkunft mit Formularbezug** zeigen wir Ihnen ab Seite 13, wie Sie die Datenherkunft unter dem Einsatz von Parametern flexibel gestalten.

Schließlich folgt der zweite Teil der Artikelreihe zum Thema **Internetmarke mit Access** (ab Seite 17). Wir bohren die Techniken, die Sie in der vorherigen Ausgabe von **Access [basics]** kennengelernt haben, auf und erweitern diese. Auf diese Weise erstellen Sie die Exportdatei nun per Mausklick und wählen die zu exportierenden Adressen komfortabel aus.

Und nun: Viel Spaß mit der neuen Ausgabe!



Ihr André Minhorst

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst Verlag | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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).

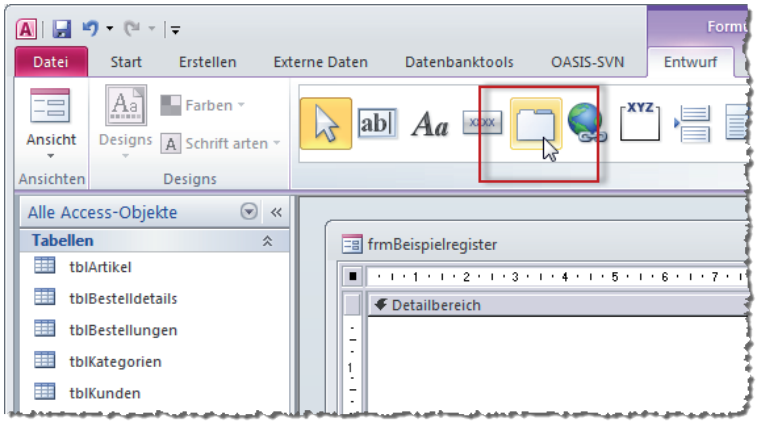



**Bild 1:** Einfügen eines Registersteuerelements

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.

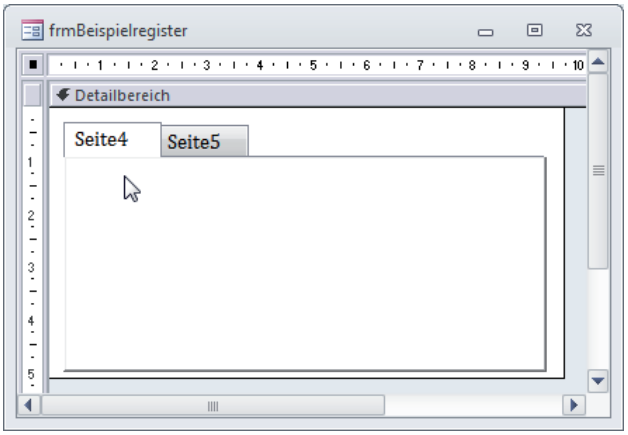



**Bild 2:** Ein frisch angelegtes Registersteuerelement

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.

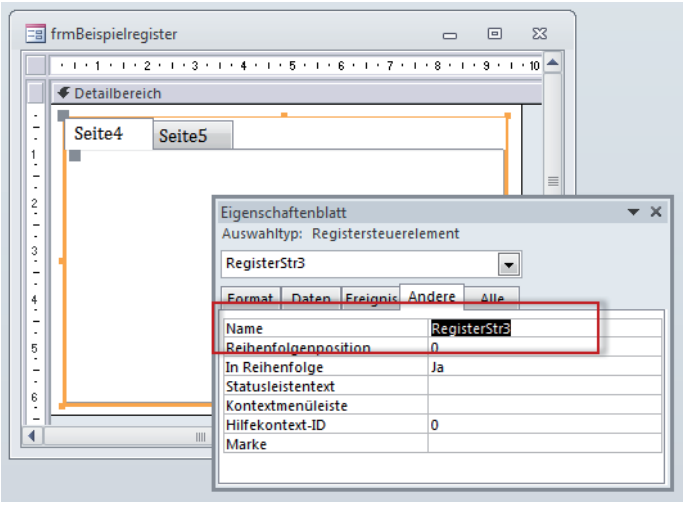



**Bild 3:** Einstellen des Namens des Registersteuerelements

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.

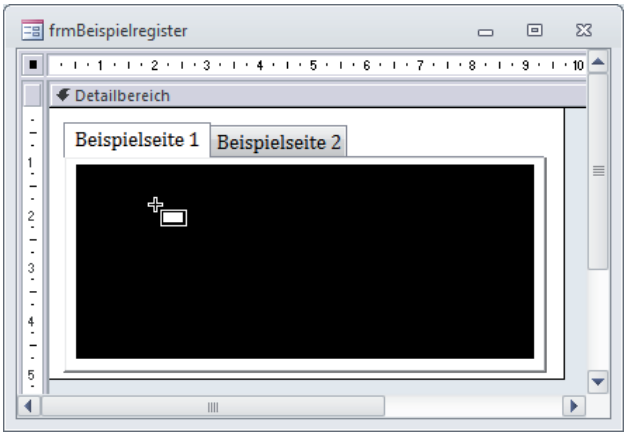



**Bild 4:** Hinzufügen eines Steuerelements

## 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.

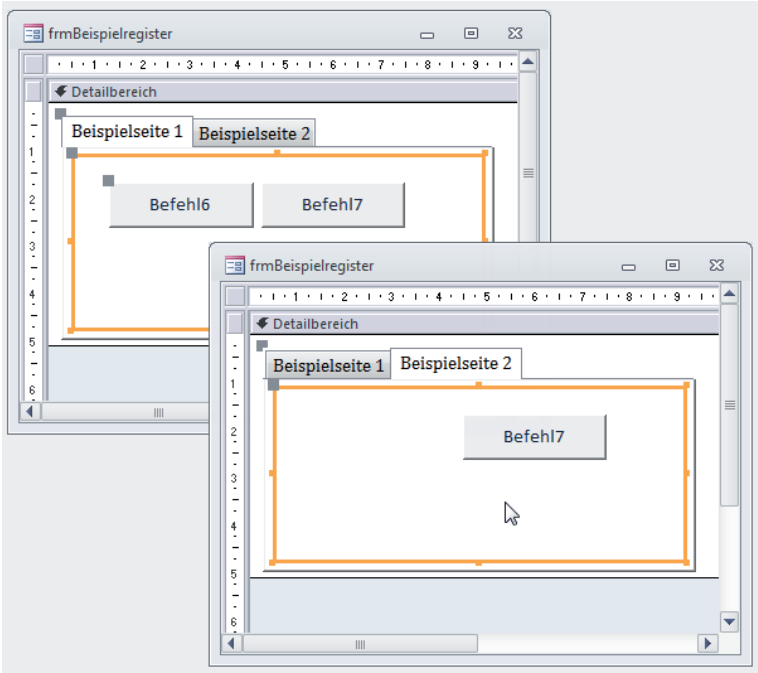



**Bild 5:** Die Schaltfläche liegt über dem Registersteuerelement, nicht darauf.

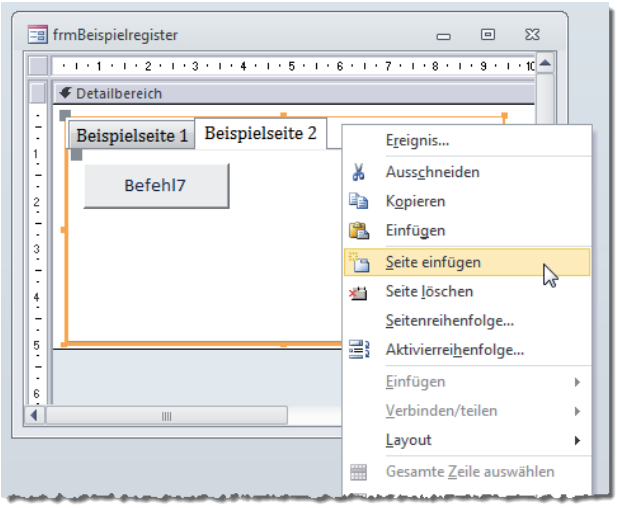



**Bild 6:** Hinzufügen einer Registerseite

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
```

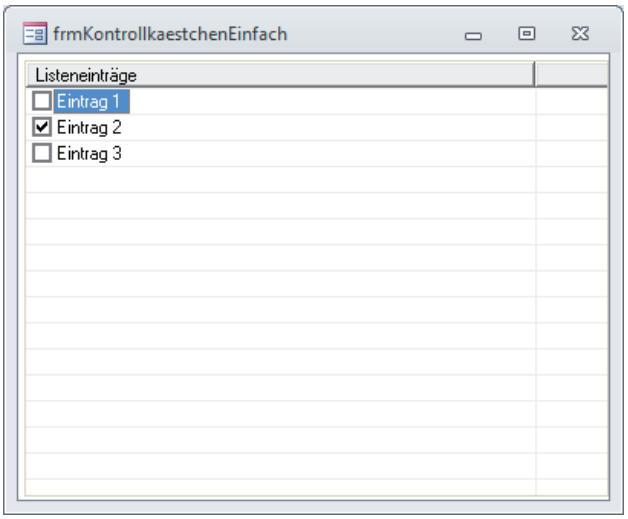



**Bild 1:** Kontrollkästchen und erstes Feld in einer Spalte

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:

```
Set objListItem = .ListItems.Add(, , "Eintrag 1")
```

Damit können Sie dann schließlich die **Checked**-Eigenschaft auf den Wert **True** einstellen, um den Eintrag anzuhaken:

```
objListItem.Checked = True
```

Um den Eintrag wieder zu deaktivieren, stellen Sie die gleiche Eigenschaft auf **False** ein:

```
objListItem.Checked = False
```

## Checkbox-Zustand ermitteln

Wie finden Sie nun heraus, ob die Checkbox eines Eintrags angehakt ist oder nicht? Im Direktfenster gelingt dies beispielsweise wie folgt, wenn Sie nur den Index des betroffenen Eintrags kennen (der Index beginnt mit der Zahl **1**):

```
? Forms!frmKontrollkaestchenEigeneSpalte!
lvwKontrollkaestchen.ListItems(2).Checked
```

Diese Anweisung prüft beispielsweise die **Checked**-Eigenschaft des zweiten Eintrags des **ListView**-Steuerelements.

## Kontrollkästchen in eigener Spalte

Vielleicht gefällt Ihnen die Ansicht nicht, in der die Checkbox in der gleichen Spalte wie der erste Wert angezeigt werden. In diesem Fall reservieren Sie eine eigene Spalte für das Kontrollkästchen, sodass das Ergebnis wie in Bild 2 aussieht (siehe auch Formular **frmKontrollkaestchenEigeneSpalte** in der Beispieldatenbank). Dort wurde der Code der Prozedur **Form_Load** nur am Ende angepasst:

```
.ColumnHeaders.Add , , "Markiert", 750
.ColumnHeaders.Add , , "Listeneinträge", 5000
.ListItems.Clear
Set objListItem = .ListItems.Add()
objListItem.Checked = True
objListItem.ListSubItems.Add , , "Eintrag 1"
Set objListItem = .ListItems.Add()
objListItem.ListSubItems.Add , , "Eintrag 2"
Set objListItem = .ListItems.Add()
objListItem.ListSubItems.Add , , "Eintrag 3"
```

Der Trick ist einfach: Sie legen einfach ein Element für die erste Spalte an, ohne einen Text einzutragen. Dadurch zeigt das **ListView**-Steuerelement das Kontrollkästchen in einer eigenen Spalte und den Text erst in der nächsten Spalte an.

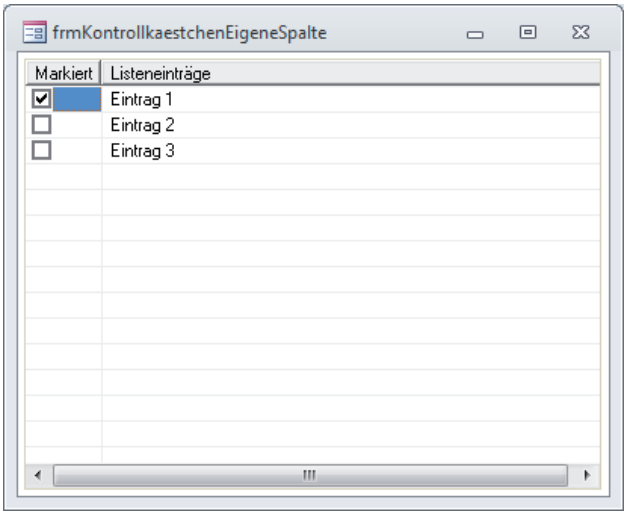



**Bild 2:** Kontrollkästchen in eigener Spalte

Beachten Sie, dass Sie den Text somit nicht mehr über das **ListView**-Steuerelement anpassen können.

## Kontrollkästchen mit Daten füllen

Wenn Sie ein **ListView**-Steuerelement mit den Daten aus einer Tabelle wie etwa **tblArtikel** füllen möchten und das Kontrollkästchen soll einen der Werte der Tabelle repräsentieren, gelingt dies wie im Beispielformular **frmListViewKontrollkaestchenDaten**.

Dort soll die erste Spalte den Wert des **Ja/Nein**-Feldes Auslaufartikel der Tabelle aufnehmen und später die Möglichkeit bieten, diesen Wert über die Kontrollkästchen anzupassen.

Im Klassenmodul des Formulars **Form_frmListViewKontrollkaestchenDaten** wird im Modulkopf wiederum eine Objektvariable zum Aufnehmen eines Verweises auf das **ListView**-Steuerelement eingefügt – diesmal allerdings mit dem Schlüsselwort **WithEvents**, da wir später noch eines der Ereignisse des Steuerelements nutzen möchten:

```
Dim WithEvents objLvwArtikel As MSComctlLib.ListView
```

Die Form_Load-Methode, die durch das Ereignis Beim Laden des Formulars ausgelöst wird, füllt diese Variable mit dem Verweis und stellt die Eigenschaften ein.

Außerdem fügt sie zwei Spaltenköpfe hinz, deren Beschriftungen **Auslaufartikel** und **Artikelname** lauten:

# 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**.

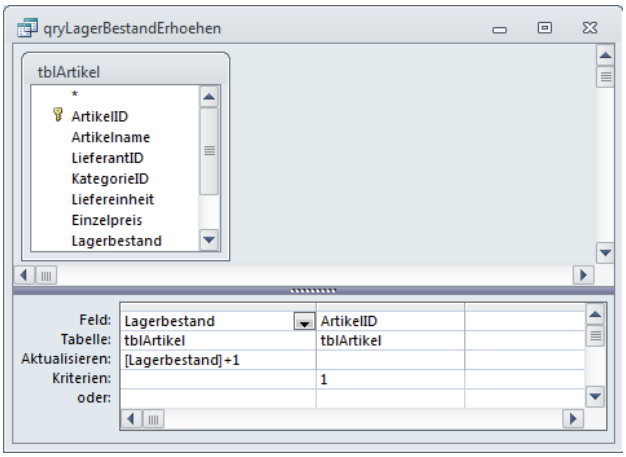



**Bild 1:** Aktionsabfrage zum Erhöhen des Lagerbestandes eines Artikels um 1

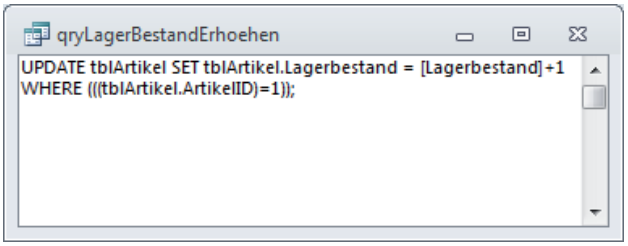



**Bild 2:** SQL-Ansicht der Abfrage **qryLagerbestandErhoehen**

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

funktioniert. Den Aufruf und das Ergebnis sehen Sie in Bild 4.

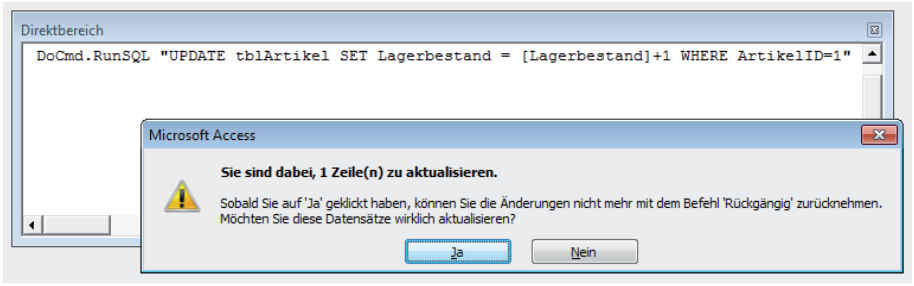



**Bild 3:** Hinweis auf den anstehenden Löschvorgang

## Meldungen an- und ausschalten

Die beim Ausführen von Aktionsabfragen erscheinenden Meldungen können Sie deaktivieren. Da Sie ohnehin gerade im Direktfenster arbeiten, können Sie dies per VBA erledigen:

```
DoCmd.SetWarnings False
```

Mit dieser Anweisung schalten Sie die Meldungen wieder an:

```
DoCmd.SetWarnings True
```

Sie können diese Option auch in den Access-Optionen einstellen – etwa wie in Bild 5 unter Access 2010. Die entsprechende Option heißt **Aktionsabfragen**.

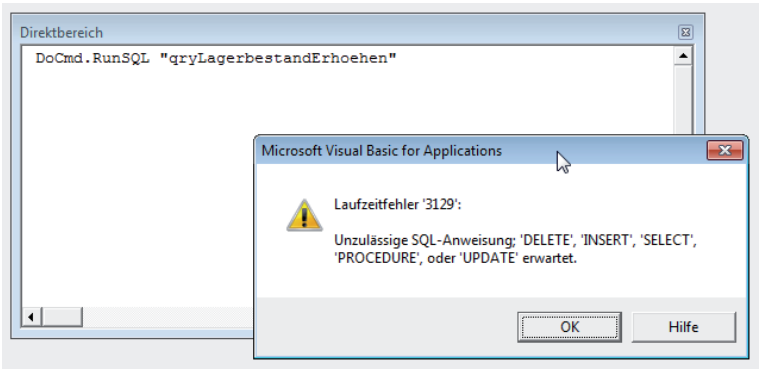



**Bild 4:** Fehler beim Ausführen einer gespeicherten Abfrage mit **DoCmd.RunSQL**

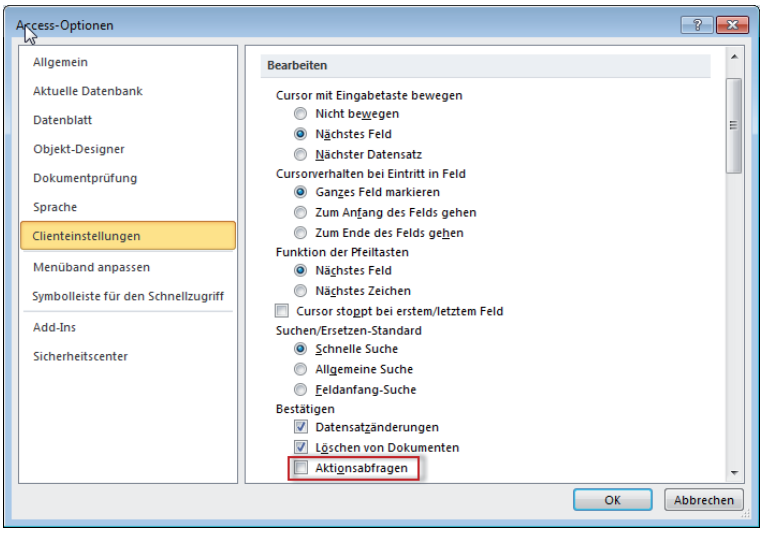



**Bild 5:** Aktivieren und Deaktivieren der Warnmeldungen

## Ausführen mit Execute

Angenommen, Sie haben bereits eine Aktionsabfrage etwa zum Löschen eines Datensatzes unter dem Namen **qryArtikelLoeschen** gespeichert. Dann können Sie den Namen der Abfrage mit der **Execute**-Methode auch direkt als Parameter angeben – im Gegensatz zur **RunSQL**-Methode. Dies sieht etwa im Direktfenster wie folgt aus:

```
CurrentDB.Execute "qryLagerbestandErhoehen"
```

Als Objekt kommt hier **CurrentDB** zum Zuge, dass einen Verweis auf das **Database**-Objekt für die aktuelle Datenbank liefert.

Ein wesentlicher Unterschied zur **DoCmd.RunSQL**-Anweisung lässt sich hier auch erkennen: es gibt keine Warnmeldung.

## Fehler erkennen

Wenn Sie der **DoCmd.RunSQL**-Methode eine fehlerhafte SQL-Anweisung übergeben – also beispielsweise eine, die sich auf eine nicht vorhandene Tabelle bezieht –, erhalten Sie eine entsprechende Fehlermeldung (siehe Bild 6).

Genau die gleiche Fehlermeldung erhalten Sie auch, wenn Sie eine fehlerhafte SQL-Abfrage mit der **Execute**-Methode des **Database**-Objekts ausführen.

Wenn jedoch die Syntax des SQL-Statements korrekt ist, aber ein Fehler etwa durch die Verletzung von Restriktionen auftritt, behandeln beide Methoden dies unterschiedlich. Ein solcher Fehler tritt beispiels-

# 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.

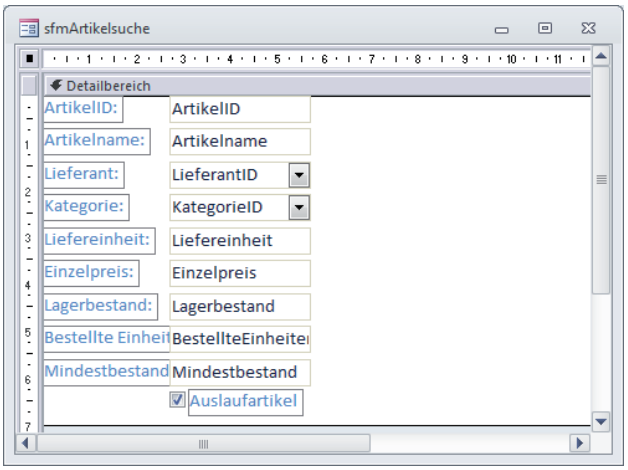



**Bild 1:** Zu filterndes Unterformular

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
```

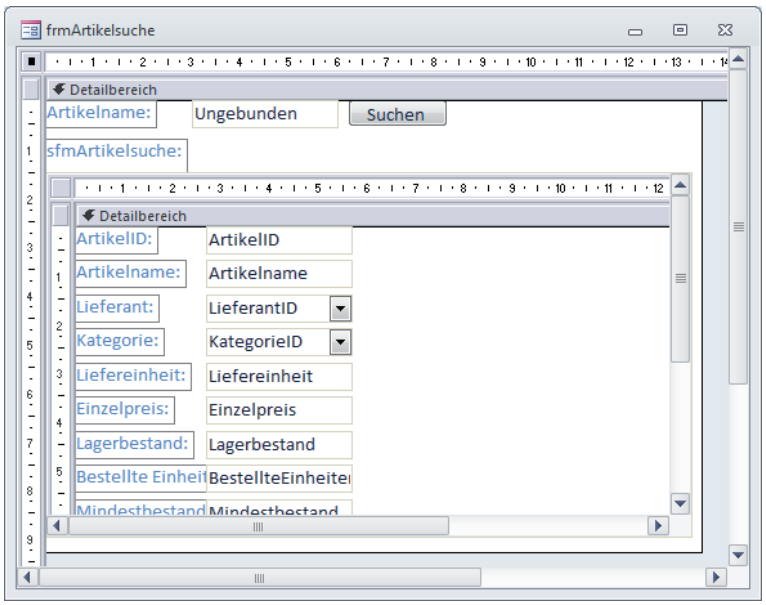



**Bild 2:** Haupt- und Unterformular in der Entwurfsansicht

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.

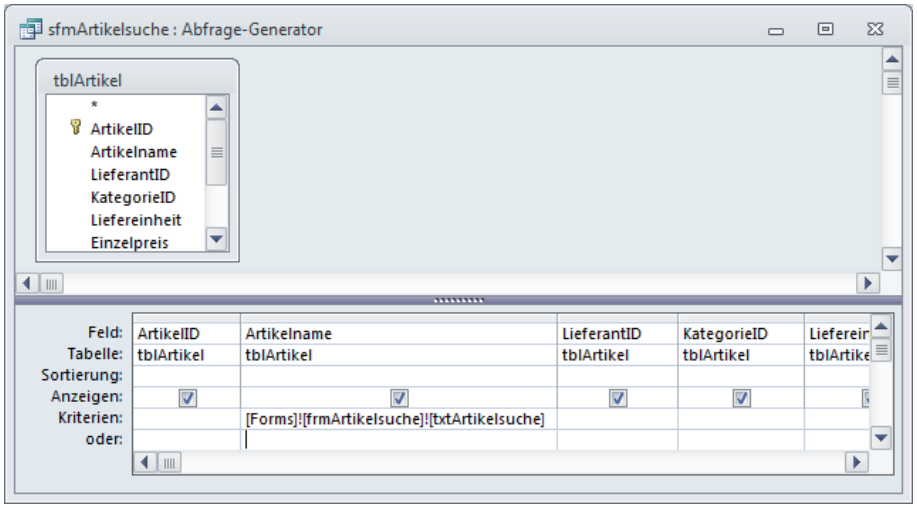



**Bild 3:** Neue Datenherkunft des Unterformulars

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.

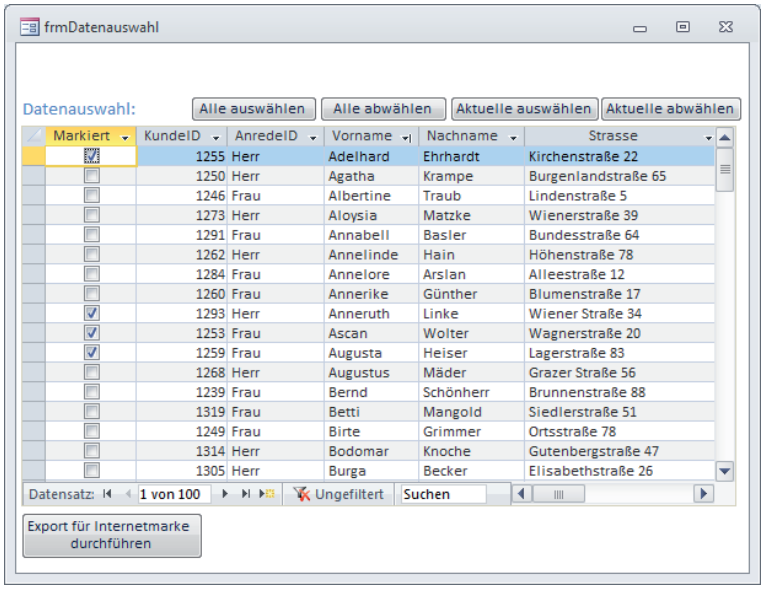



**Bild 1:** Ausgangspunkt für das Erstellen von Internetmarken

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.

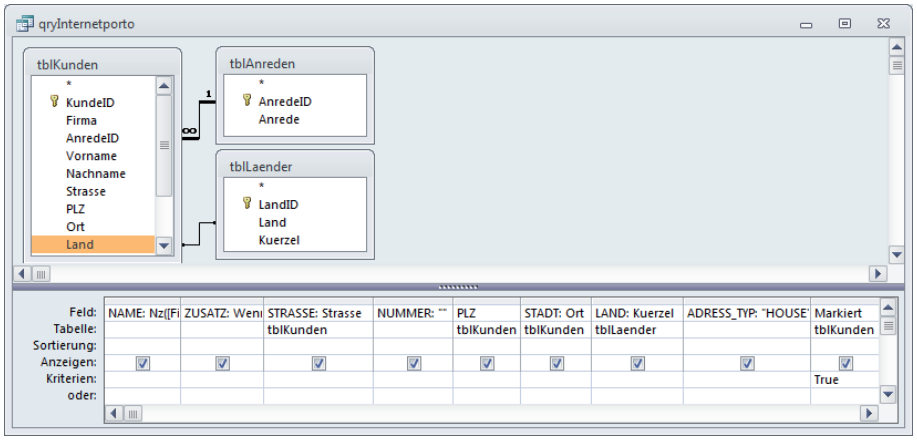



**Bild 2:** Einfügen des Kriteriums, das nur die ausgewählten Datensätze anzeigt

## 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.

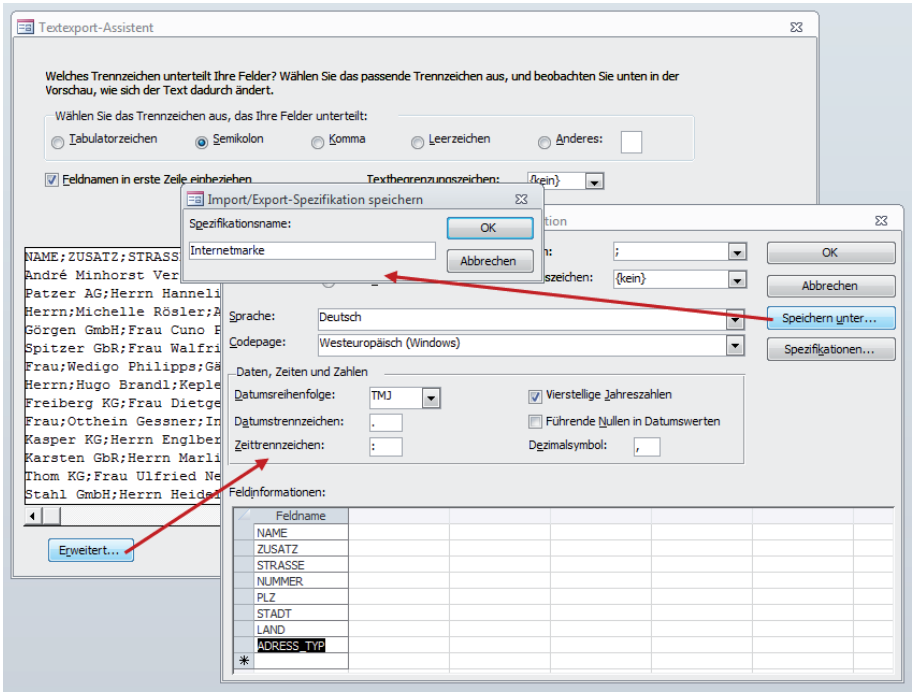



**Bild 3:** Speichern der Export-Spezifikation

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

Access [basics] 12/2012
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Editorial

Was für ein Jahr! In 2012 habe ich das erste Buch im eigenen Verlag veröffentlicht und der Zuspruch war unglaublich groß. Nicht nur, dass ich während des Schreibens bereits sehr viel Feedback und Anregungen zu den vorab veröffentlichten Kapiteln erhalten habe. Nein, Sie haben das Buch sogar gekauft – zumindest einige von Ihnen! Und ich denke, dass Sie damit eine gute Ergänzung zu **Access [basics]** erhalten, wo ich ja eher auf die Grundlagen der Datenbankentwicklung mit Microsoft Access eingehe.

Das Schreiben eines solchen Buches kostet natürlich auch Zeit, und dadurch ist zwischenzeitlich eine komplette Ausgabe um etwa einen Monat nach hinten verschoben worden. Mit der Ausgabe 12/2012 erhalten Sie pünktlich zum Jahresende die zwölfte Ausgabe für dieses Jahr – und als kleine Entschädigung für die zwischenzeitliche Verzögerung gibt es diesmal ein paar Seiten mehr.

Ich hoffe, Sie können den enthaltenen Artikeln einiges Know-how für Ihre eigene Arbeit mit Access entnehmen!

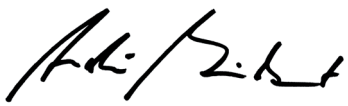
Schauen wir nun auf den Inhalt der aktuellen Ausgabe von **Access [basics]**. Der erste Artikel heißt Registersteuerelement in der Praxis und zeigt an einigen Beispielen, wie Sie das Registersteuerelement programmieren und praktisch einsetzen können (ab Seite 2).

Was wäre eine Datenbank ohne Änderungen der enthaltenen Daten? Meist erledigen Sie dies über die Benutzeroberfläche, aber manchmal sollen Daten auch automatisiert geändert werden. Dabei helfen Grundlagen zu Aktionsabfragen unter SQL. Der Artikel **SQL-Grundlagen: Aktionsabfragen** liefert ab Seite 9 das entsprechende Know-how.

Und wenn wir schon beim Thema Abfragen sind, schauen wir uns in einem weiteren Artikel an, wie Sie per VBA auf die Abfragen in einer Datenbank zugreifen. Dabei erstellen Sie Abfragen, löschen diese wieder und statten diese mit Parametern aus, um beispielsweise flexible Kriterien einsetzen zu können. Mehr zu diesem Thema unter **DAO: Querydefs und Parameterabfragen unter VBA** ab Seite 13.

Schließlich geht es mit der Artikelserie zum Thema ListView-Steuerelement allmählich auf die Zielgerade: Im Artikel **ListView-Steuerelement: Drag and Drop, Teil I** erfahren Sie ab Seite 17, wie Sie Daten von einem ListView-Steuerelement zum nächsten ziehen und so m:n-Beziehungen verwalten können.

Und nun: Viel Spaß mit der neuen Ausgabe!



Ihr André Minhorst

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst Verlag | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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.

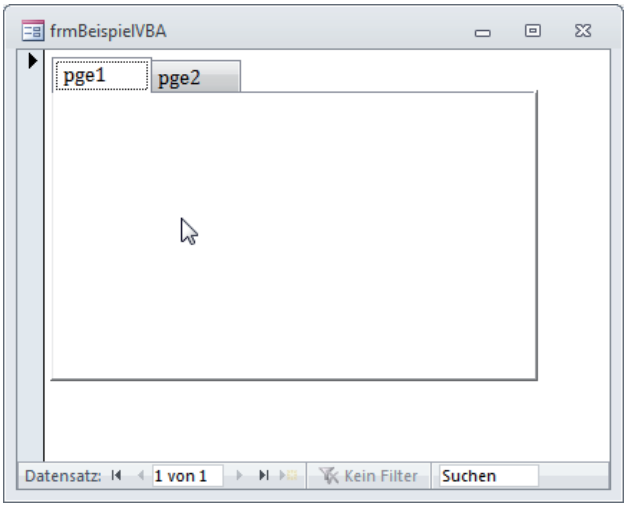



**Bild 1:** Beispielregistersteuerelement

## 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).

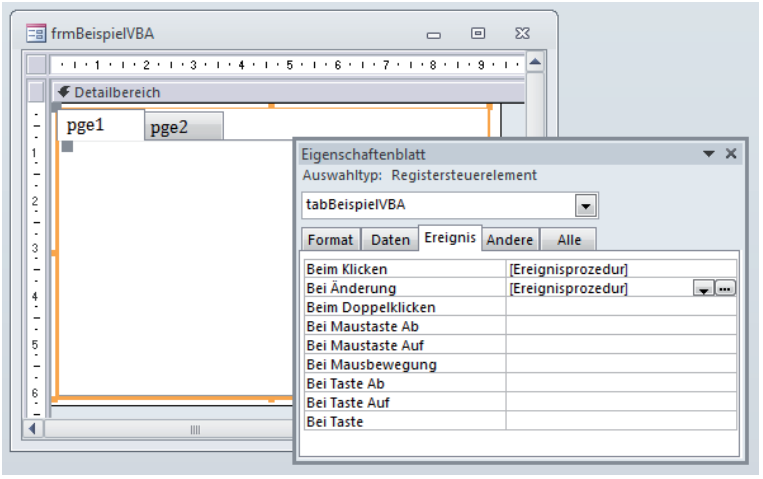



**Bild 4:** Anlegen von Ereignisprozeduren für die Ereigniseigenschaften des Registersteuerelements

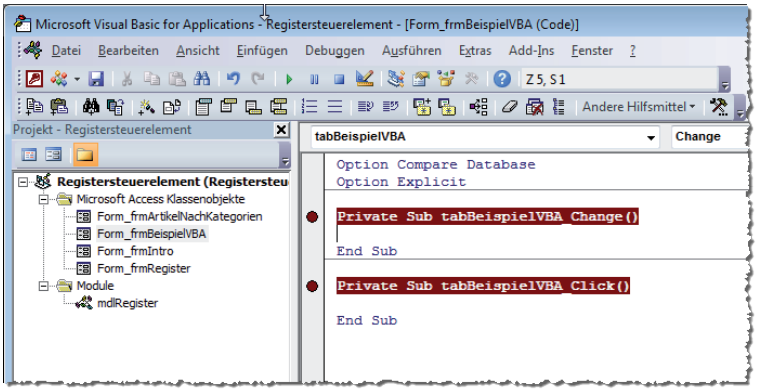



**Bild 2:** Ereignisprozeduren mit Haltepunkten

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.

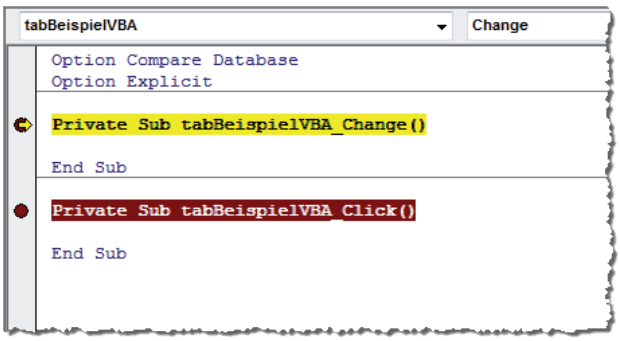



**Bild 3:** Nur diese Prozedur wird beim Wechsel der Registerseite ausgelöst.

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
```

# 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
```

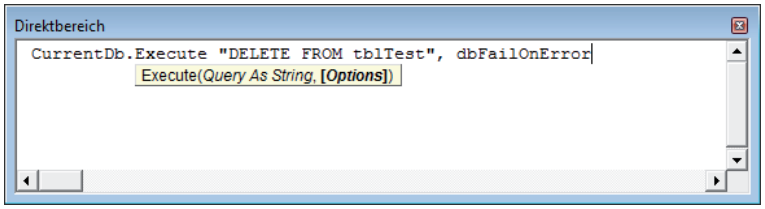



**Bild 1:** Absetzen einer SQL-Anweisung per Direktfenster

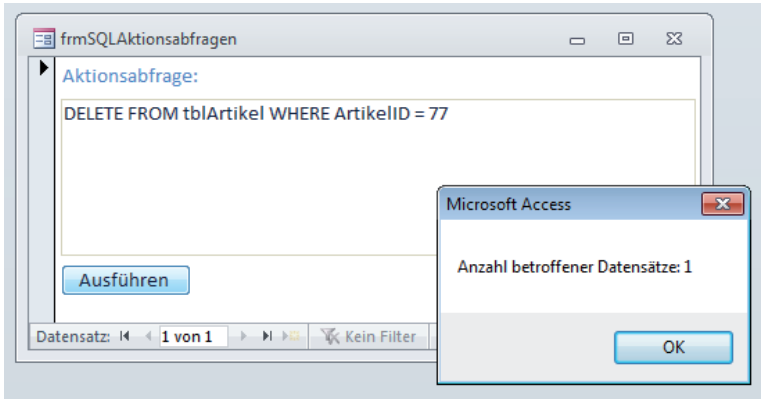



**Bild 2:** Hilfsformular zum Ausführen von SQL-Aktionsabfragen

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.

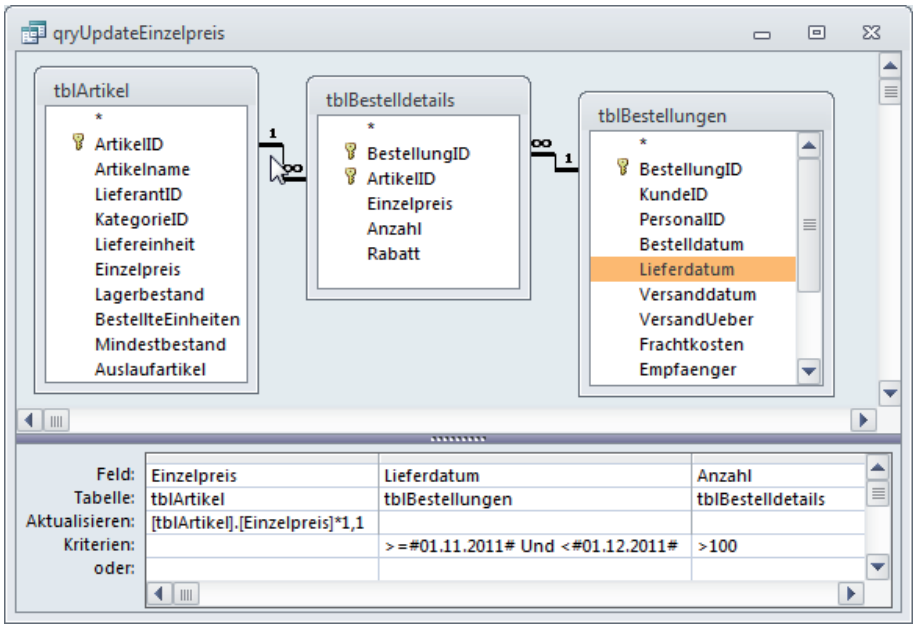



**Bild 3:** Komplizierte Abfrage im Entwurf

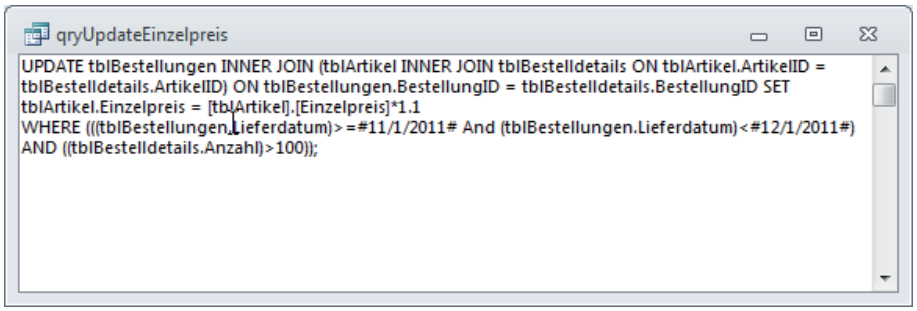



**Bild 4:** Komplizierte Abfrage – unter SQL noch verwirrender

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).

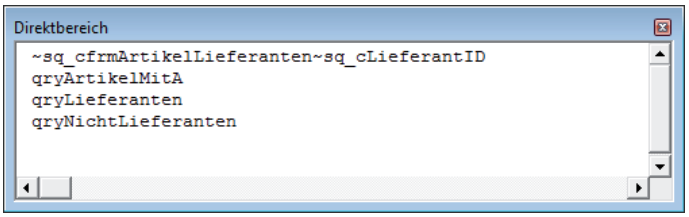



**Bild 1:** Ausgabe aller Abfragen der Datenbank

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.

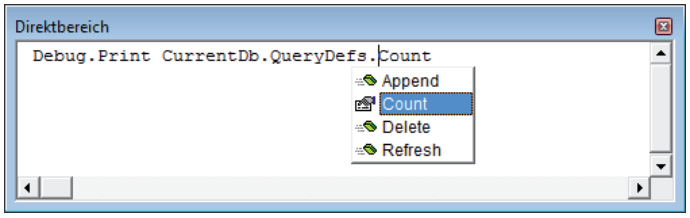



**Bild 2:** Abfrage der Anzahl der **QueryDef**-Objekte

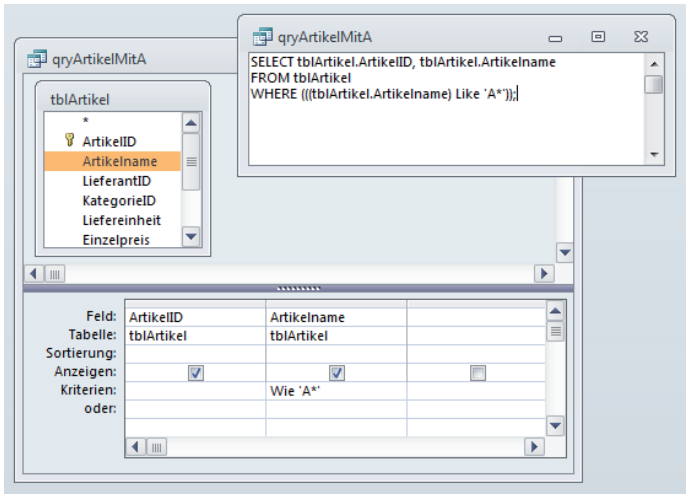



**Bild 3:** SQL-Ansicht einer Abfrage

## 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.

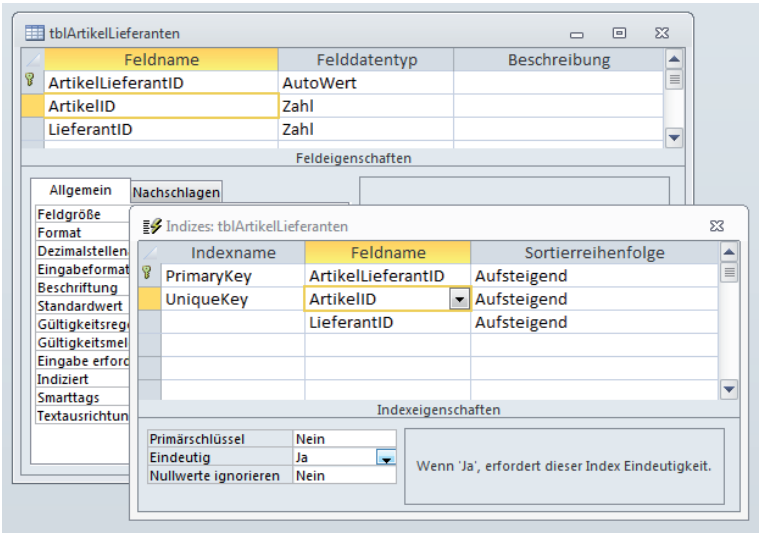



**Bild 1:** Tabelle zur Herstellung einer m:n-Beziehung

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.

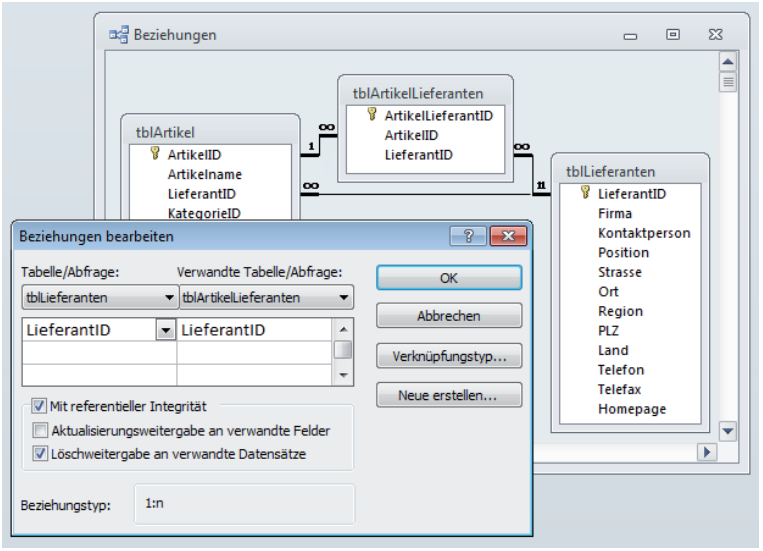



**Bild 2:** Festlegen der Beziehungen zwischen den Tabellen der m:n-Verknüpfung

## 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.

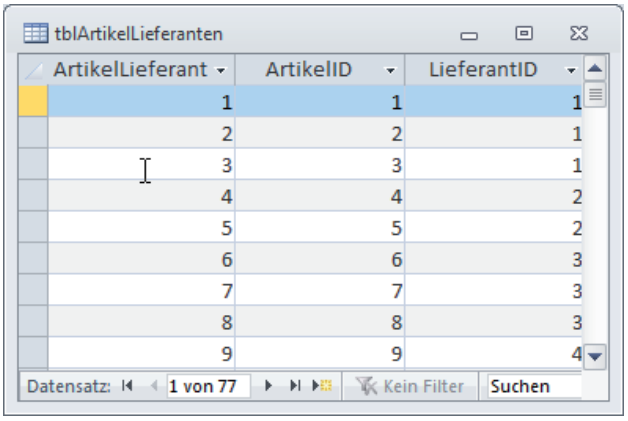



**Bild 3:** Die Tabelle **tblArtikelLieferanten** mit den Standardlieferanten eines jeden Artikels

## 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).

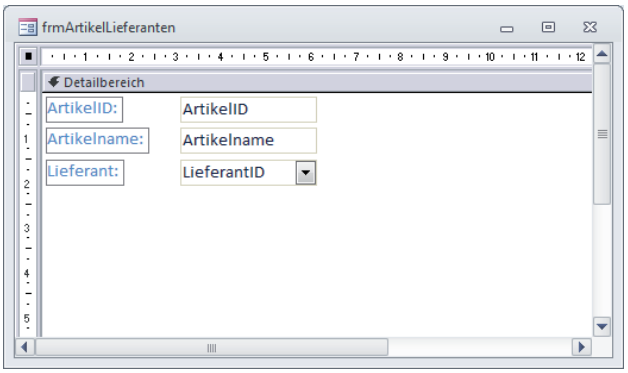



**Bild 4:** Das Formular zeigt einige Daten der Tabelle **tblArtikel** an.

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.

Access [basics] 1/2013
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Editorial

**Access [basics]** geht nun ins vierte Jahr – im Mai 2009 fiel der Startschuss dieses Magazins. Damit entwachsen wir allmählich den Kinderschuhen, und auch die Themen werden erwachsener. Das Magazin hat bereits einige Grundlagen beleuchtet, sodass hier und da auch mal ein fortgeschritteneres Thema behandelt wird. Dennoch wird es in den folgenden Ausgaben weiterhin Artikel mit grundlegenden Themen geben.

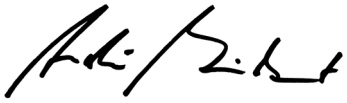
Welche dies sind, dürfen Sie übrigens gern mitbestimmen – schicken Sie mir einfach Anregungen zu den Grundlagen, die Sie interessieren, an meine E-Mail-Adresse **info@access-basics.de**. Ich freue mich auf Ihre Vorschläge!

Die vorliegende Ausgabe liefert drei Artikel, von denen gleich zwei konkrete Lösungen bieten. Im ersten Artikel namens **Programmieren mit Arrays** ab Seite 2 geht es mit dem VBA-Grundlagen weiter. Dieser Artikel behandelt eine spezielle Form der Variablen, mit der Sie gleich mehrere gleichartige Elemente in einer einzigen Variablen speichern und über den Index auf diese Elemente zugreifen können.

Der Artikel **Reihenfolge individuell einstellen** behandelt ab Seite 7 ein Thema, auf das jeder Programmierer einmal stößt: Daten sollen sortiert dargestellt werden, aber die Felder der zugrunde liegenden Tabellen liefern kein konkretes Kriterium, nach dem man sortieren könnte. Dabei kann es beispielsweise um die Priorität von Aufgaben, aber auch um andere individuelle Sortierungen gehen. Der Artikel zeigt, wie Sie die Reihenfolge mithilfe eines zusätzlichen Feldes in der entsprechenden Tabelle festlegen und diese Reihenfolge beispielsweise per Mausklick auf eine Schaltfläche ändern.

Unter dem Titel **Feiertage verwalten** erfahren Sie ab Seite 15 schließlich, wie Sie die Feiertage ermitteln, die in einem bestimmten Zeitraum liegen – beispielsweise alle Feiertage für das Jahr 2013. Im ersten Teil dieser Artikelreihe lernen Sie zunächst das Datenmodell dieser Lösung kennen sowie ein Formular, mit dem Sie die für die Ermittlung der Feiertage nötigen Basisdaten verwalten.

Und nun: Viel Spaß mit der neuen Ausgabe!



Ihr André Minhorst

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst Verlag | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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
```

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.

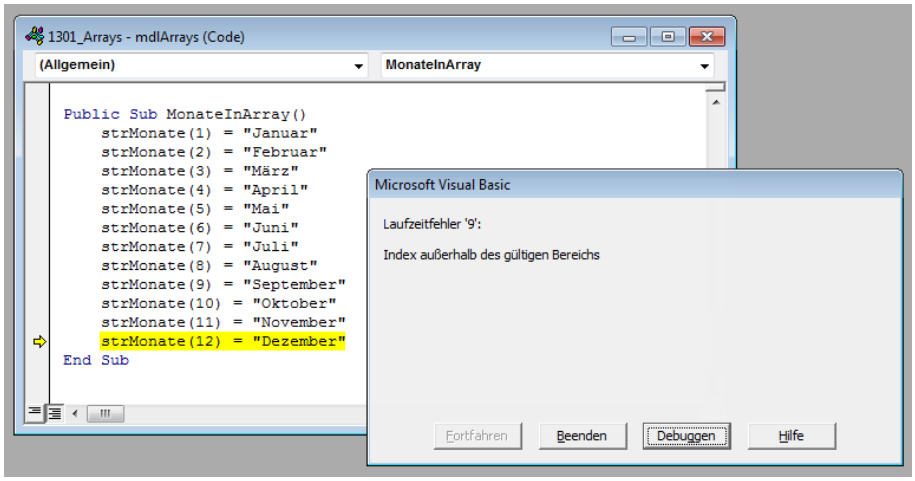



**Bild 1:** Versuch, ein Array mit einem zu großen Indexwert zu füllen

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).

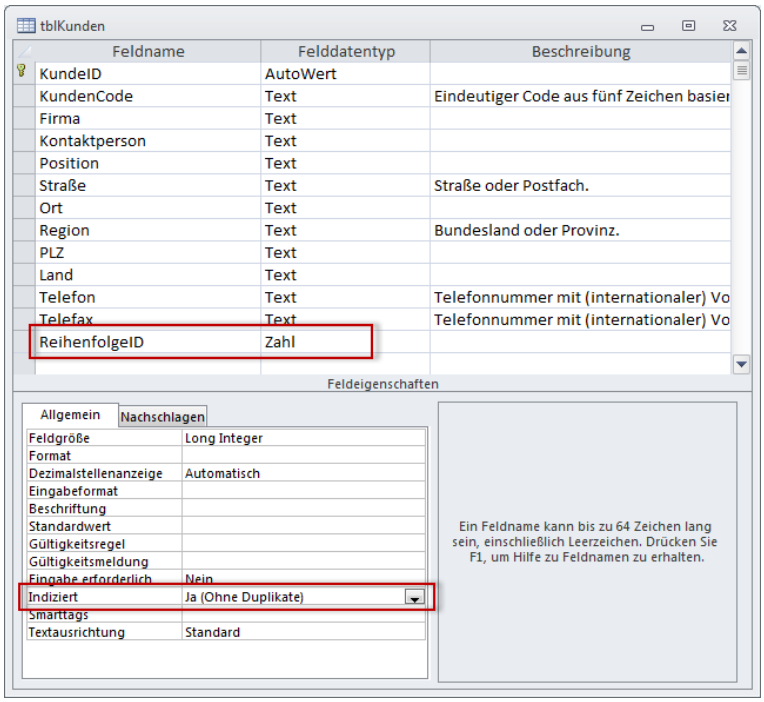



**Bild 1:** Hinzufügen eines Feldes für die Definition einer individuellen Reihenfolge

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?

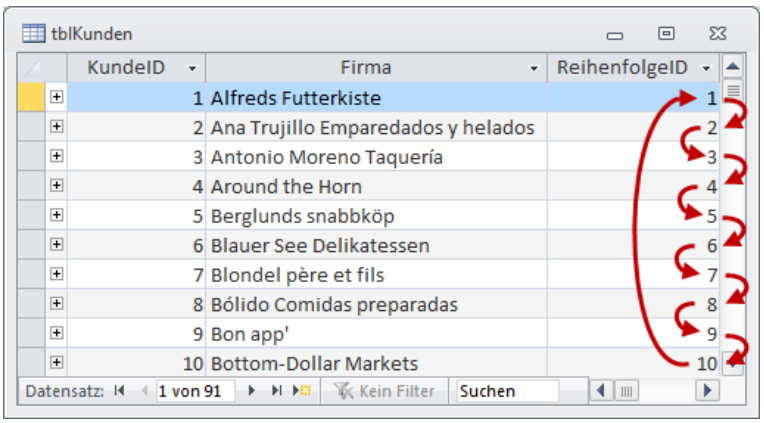



**Bild 2:** Änderungen im Feld **ReihenfolgeID** beim Verschieben des zehnten Eintrags an die erste Position

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.

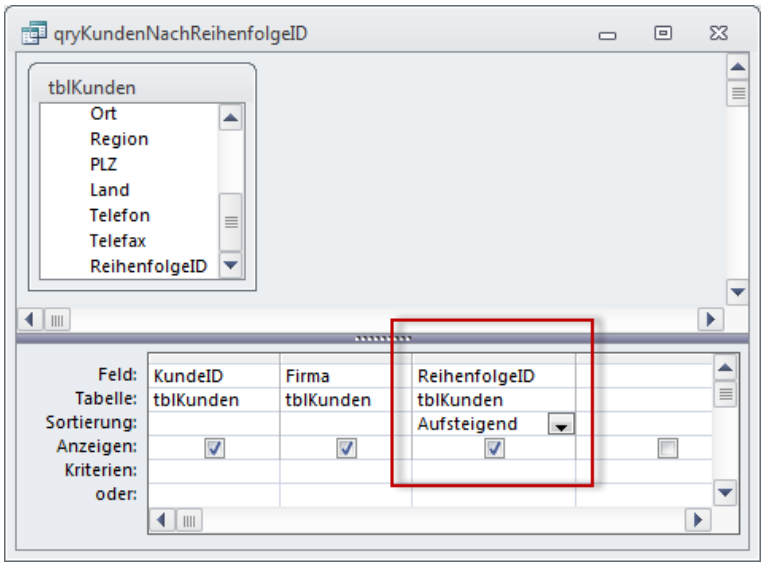



**Bild 3:** Abfrage zur Sortierung der Kunden nach dem Wert des Feldes **ReihenfolgeID**

## 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**.

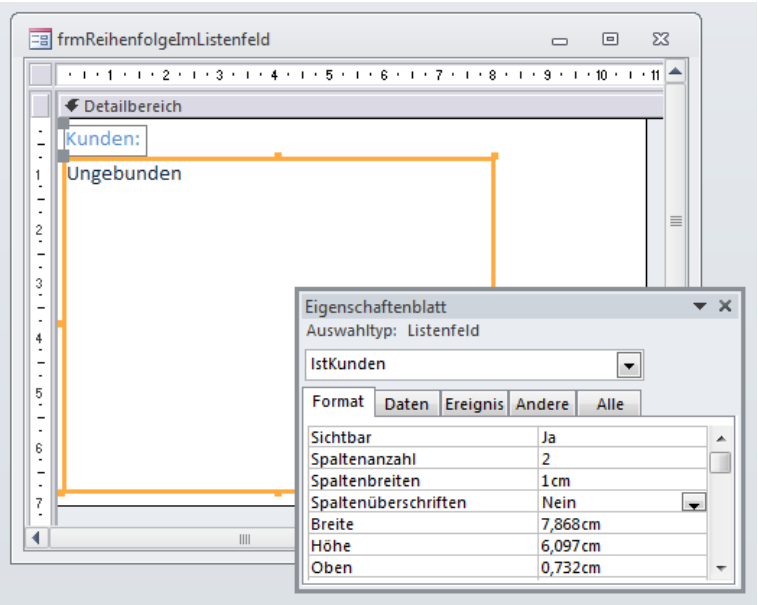



**Bild 4:** Listenfeld mit den nach dem Feld **ReihenfolgeID** sortierten Datensätzen in der Entwurfsansicht

## 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-

# 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.

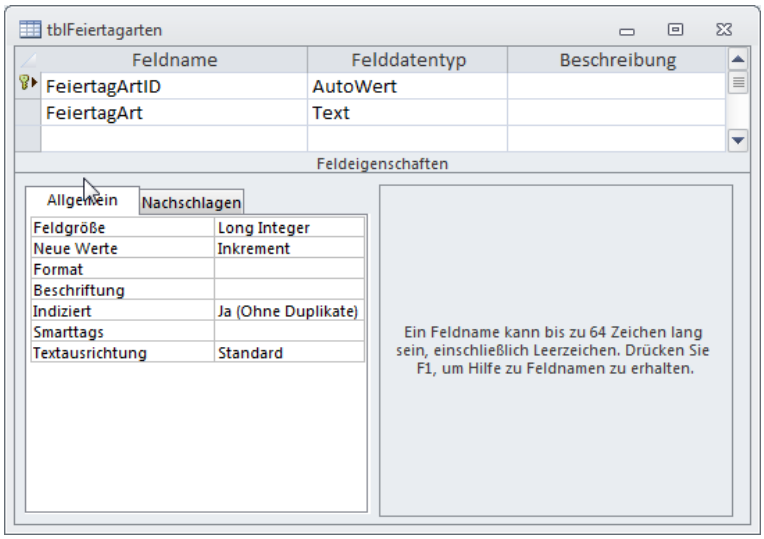



**Bild 1:** Tabelle der Feiertagsarten in der Entwurfsansicht

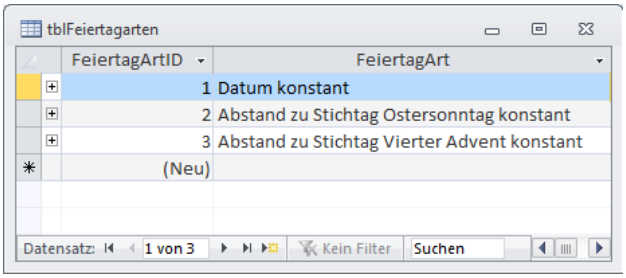



**Bild 2:** Tabelle der Feiertagsarten mit allen Werten

## 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.

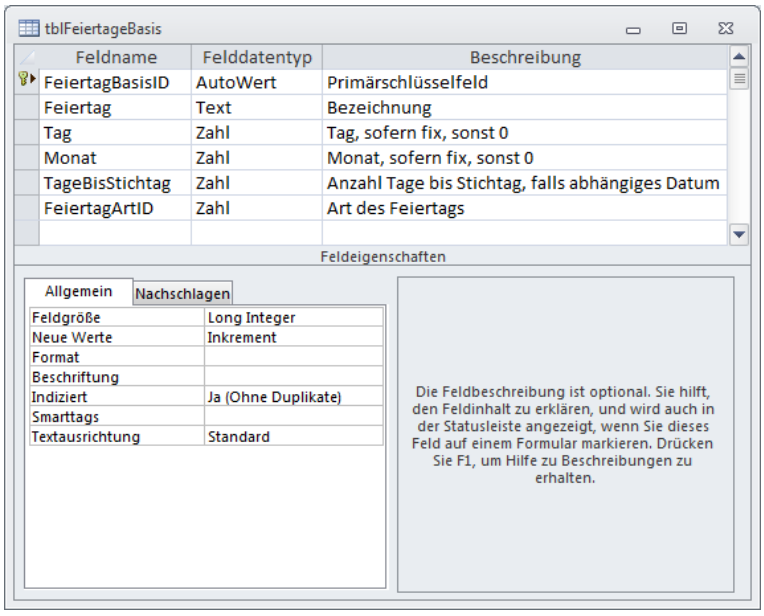



**Bild 3:** Tabelle der Feiertag-Basisdaten in der Entwurfsansicht

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**.

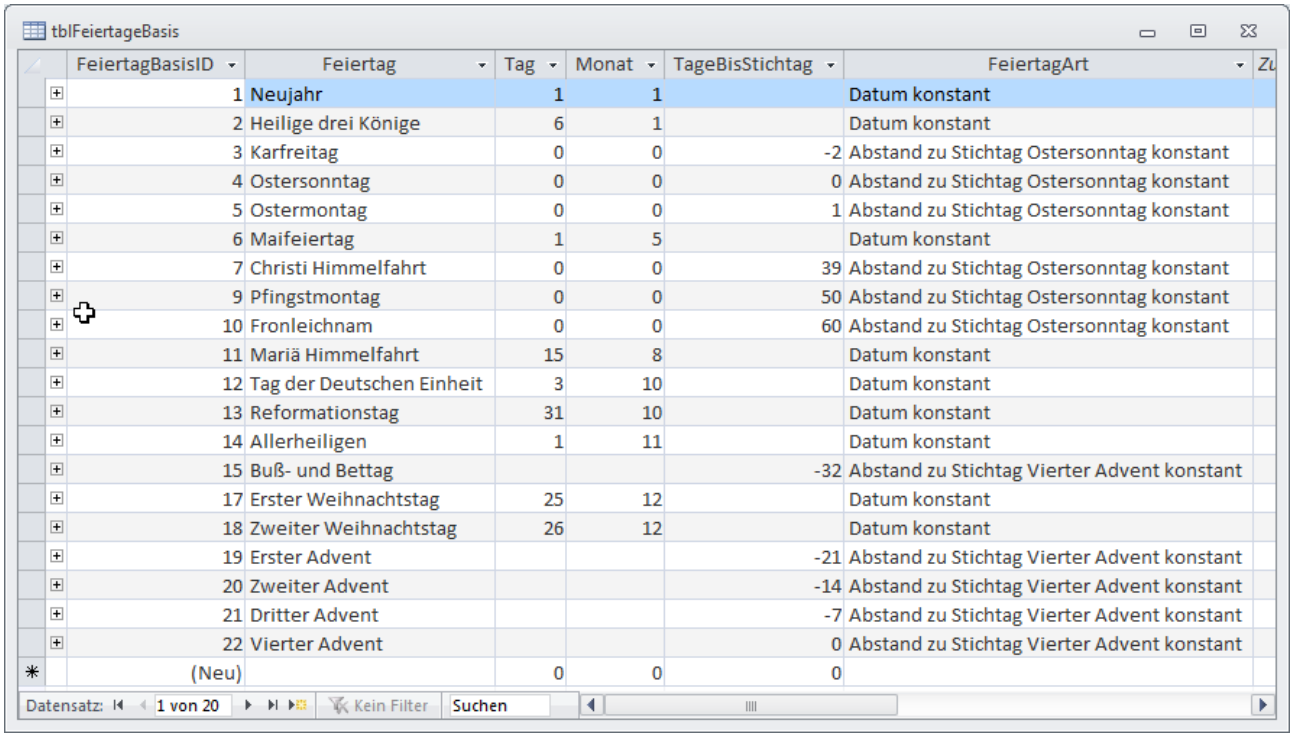



**Bild 4:** Tabelle der Feiertag-Basisdaten in der Entwurfsansicht

Access [basics] 2/2013
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

## Editorial

Wissen Sie, was das Schönste beim Schreiben von Fachbeiträgen ist? Man lernt immer noch etwas dazu! Gut: Da ich nun bereits eine Reihe Artikel für verschiedene Magazine geschrieben und entsprechend viele Beispiele und Lösungen programmiert habe, nimmt die Anzahl der neuen Erkenntnisse nicht mehr so schnell zu.

Dennoch findet ich immer wieder neue Tricks, die mir den Programmiereralltag erleichtern oder entdecke alte Techniken für mich wieder. Bei dieser Ausgabe passierte das beim Artikel **Debugging im VBA-Editor** (ab Seite 2). Das Debuggen von Code ist mein tägliches Brot, aber dennoch habe ich bislang eine wichtige Funktion ignoriert: den Prozedurschritt.

Dabei durchläuft man alle Zeilen einer Prozedur, wechselt aber nicht zu den von dort aufgerufenen Routinen! Wenn man viele kleine Hilfsprozeduren nutzt, ist es nämlich sonst sehr störend, wenn man diese auch immer noch durchlaufen muss – obwohl man deren Funktion bereits kennt und dieser vertrauen kann.

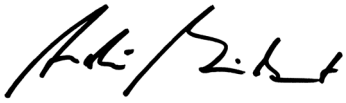
Unter dem Titel **ListView-Steuerelement: Drag and Drop, Teil II** erfahren Sie ab Seite 7, wie die Einträge eines **ListView**-Steuerelements per Drag and Drop individuell sortieren können. Ziehen Sie ein Element auf ein anderes, nimmt das gezogene Element den Platz des Zielelements ein und alle dazwischen liegenden Elemente werden um eine Position nach oben under unten verschoben – je nachdem, wohin Sie das Element gezogen haben.

Außerdem kümmern wir uns in **Feiertage ermitteln, Teil 2** um die Funktionen zum Berechnen der flexiblen Feiertage, deren Datum vom Ostersonntag oder vom vierten Advent abhängen (ab Seite 15).

Und in der Abteilung **Tipps und Tricks** liefert diese Ausgabe ab Seite 20 gleich drei kleine Helferlein für den täglichen Einsatz.

Sie erfahren dort, wie Sie Ereignisprozeduren ganz schnell vom Eigenschaftsfenster aus anlegen können, wie Sie die Namen von Wochentagen oder von Monaten mit einfachen Ausdrücken ermitteln und wie Sie leere Textfelder mit Platzhaltern versehen – diese werden nur angezeigt, wenn das Feld noch keinen Inhalt hat. Der Clou: Sie können dabei sogar andere Farben als die aktuelle Textfarbe verwenden!

Und nun: Viel Spaß mit der neuen Ausgabe!



Ihr André Minhorst

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst Verlag | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | Telefon: 0203/4495577 | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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-

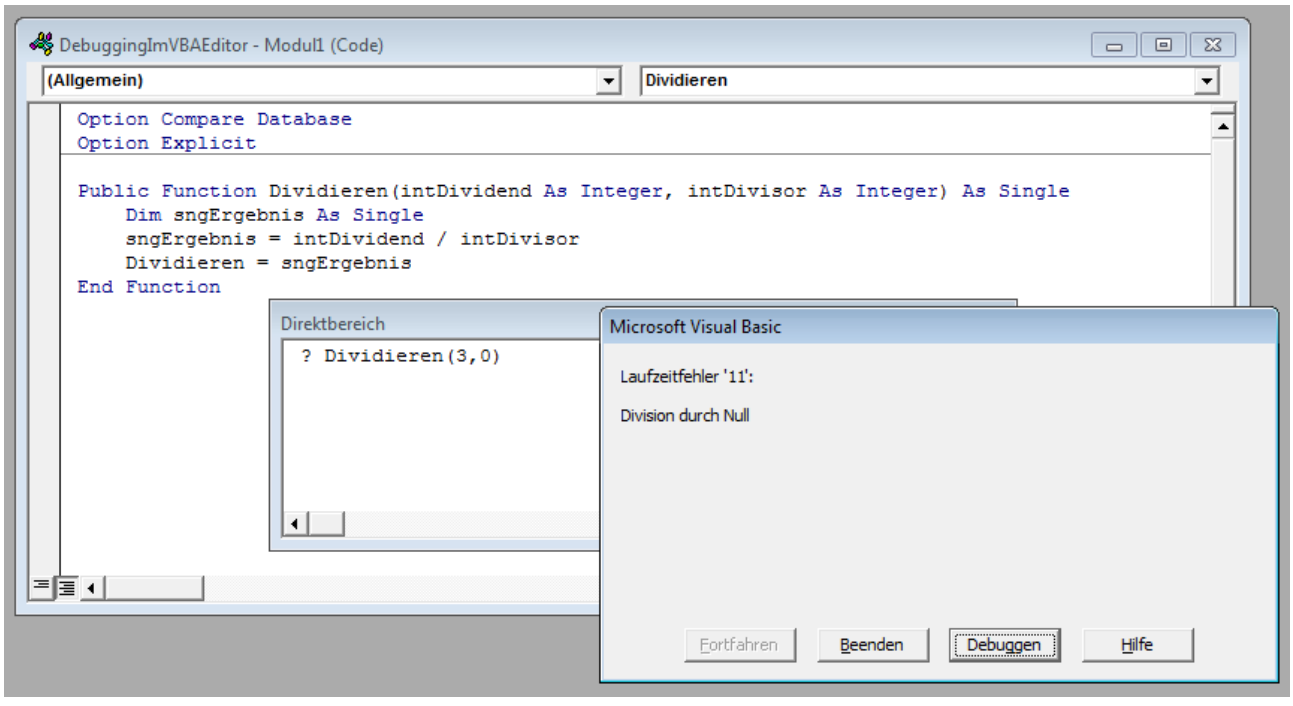



**Bild 1:** Laufzeitfehler mit der Möglichkeit, zu debuggen

lermeldung aus Bild 1. Dort finden Sie unter anderem eine Schaltfläche mit der Beschriftung **Debuggen**, mit der Sie in den Debugging-Modus einsteigen können.

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.

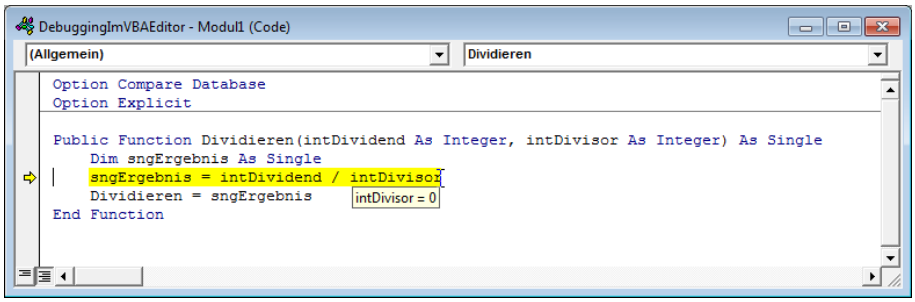



**Bild 2:** Debugging eines Laufzeitfehlers

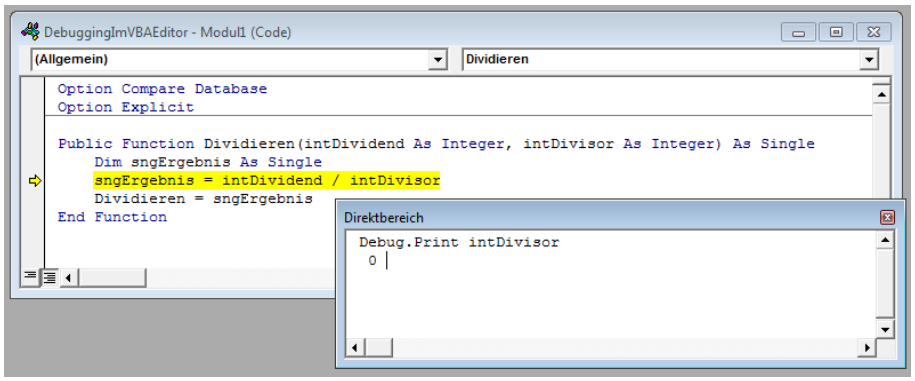



**Bild 3:** Variablenwerte über das Direktfenster ermitteln

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.

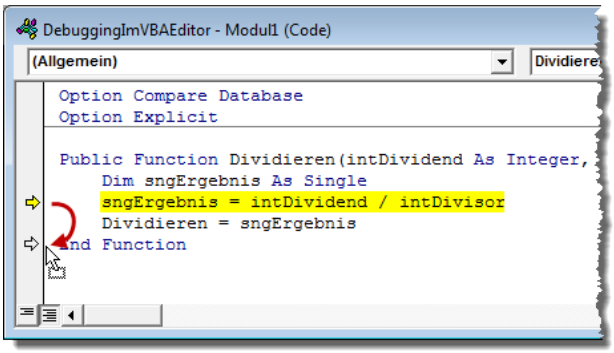



**Bild 4:** Aktuelle Zeile ändern

# 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

# 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

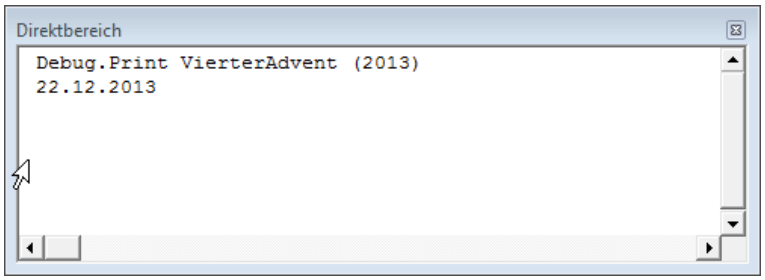



**Bild 2:** Test der Funktion zum Ermitteln des vierten Advents

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

# Tipps und Tricks

**In dieserm Artikel finden Sie drei kleine Tipps für die tägliche Arbeit mit Access. Zuerst erfahren Sie, wie Sie Ereignisprozeduren sehr schnell anlegen können. Dann lernen Sie, wie Sie per VBA oder auch in Ausdrücken den Namen eines Tages oder eines Monats ausgeben können. Schließlich schauen wir uns noch an, wie Sie in leeren Textfeldern Hinweistexte vorblenden können.**

## Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1302_TippsUndTricks.mdb**.

## Ereignisprozeduren schneller anlegen

Wenn Sie oft Ereignisprozeduren anlegen, ärgern Sie sich vielleicht darüber, dass Sie immer erst den Wert **[Ereignisprozedur]** für die entsprechende Ereigniseigentschaft auswählen und dann auf die Schaltfläche mit den drei Punkten (...) klicken müssen, damit der VBA-Editor mit dem Prozedurrumpf der Ereignisprozedur erscheint.

Oder Sie klicken mit der rechten Maustaste auf ein Steuerelement und wählen den Eintrag **Ereignis...** aus, um gleich das standardmäßig verwendete Ereignis anzulegen – dann erscheint zunächst noch der Dialog zur Auswahl des entsprechenden Werkzeugs (siehe Bild 1).

Das geht auch schneller, wenn Sie die Einstellung für eine spezielle Option ändern:

- Unter Access 2003 und älter: Öffne Sie den Optionen-Dialog mit dem Menüeintrag Extras|Optionen und wechseln Sie dort zur Registerseite **Formulare/Berichte**. Aktivieren Sie die Option **Ereignisprozedur immer verwenden** (siehe Bild 2).

- Unter Access 2007: Klicken Sie auf den Office-Button und dann auf die Schaltfläche **Access-Optionen**. Im Optionen-Dialog wechseln Sie zum Bereich **Objekt-Designer** und aktivieren die Option **Entwurfsansicht für Formulare/Berichte|Immer Ereignisprozeduren verwenden** (siehe Bild 3).

- Unter Access 2010: Hier aktivieren Sie den Optionen-Dialog mit dem Ribbon-Eintrag **Datei|Optionen**, der Rest verläuft wie unter Access 2007.

Nach dieser kleinen Änderung brauchen Sie nur noch auf die Schaltfläche mit den drei Punkten zu klicken beziehungsweise den Kontextmenü-Befehl **Ereignis...** des betroffenen Steuerelements aufzuru-

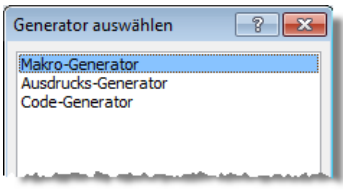



**Bild 1:** Beim Anlegen einer Ereignisprozedur wählen Sie zunächst den Typ aus.

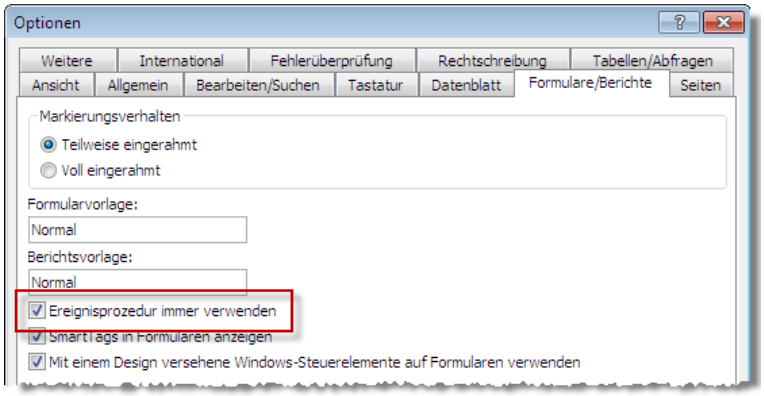



**Bild 2:** Wenn Sie diese Option aktivieren, erstellen Sie Ereignisprozeduren mit einem Mausklick.

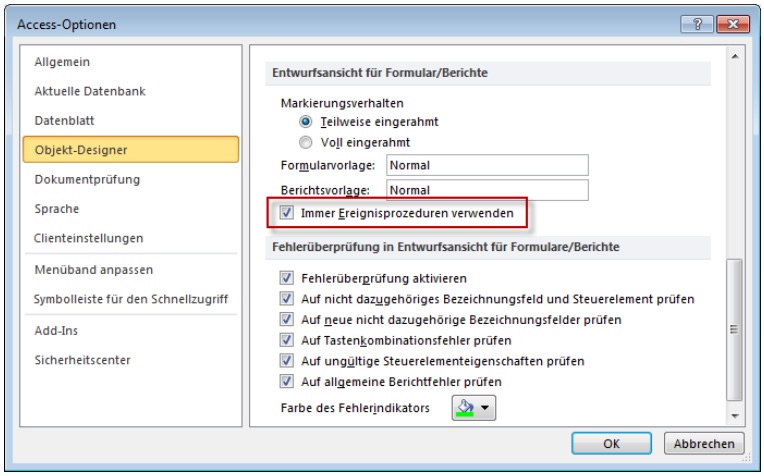



**Bild 3:** Auch unter Access 2007 und jünger erstellen Sie Ereignisprozeduren mit dieser Option blitzschnell.

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.

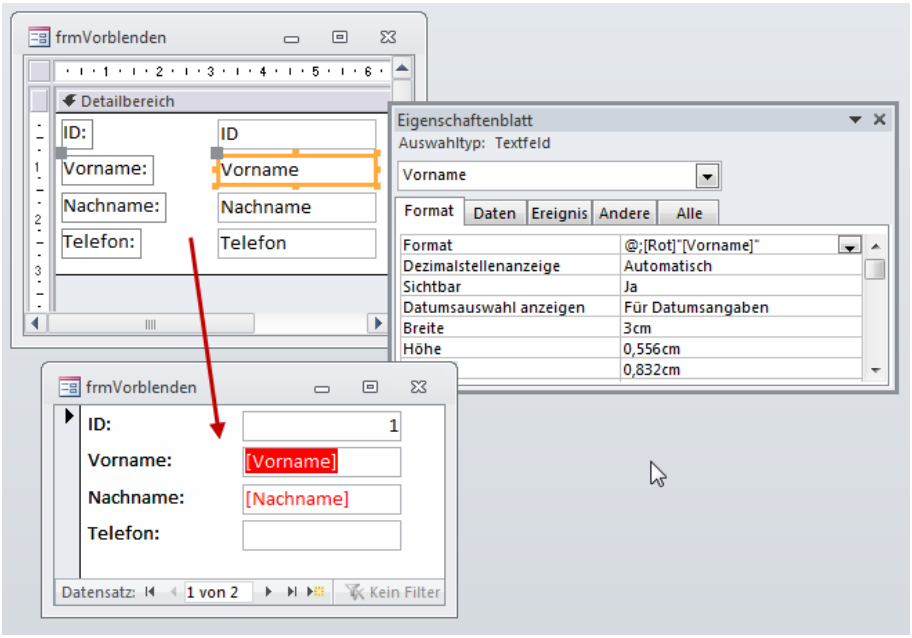



**Bild 4:** Einstellung zum Vorblenden eines Wertes in leeren Feldern

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.

Access [basics] 3/2013
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

# In dieser Ausgabe:

**Steuerelementlayouts mit Access 2007 und jünger – Seite 2**

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.

**Balkendiagramme mit Bordmitteln – Seite 9**

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.

**Haupt- und Unterformular synchron – Seite 15**

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.

**Debugging im VBA-Editor, Teil 2 – Seite 18**

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.

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst Verlag | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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).

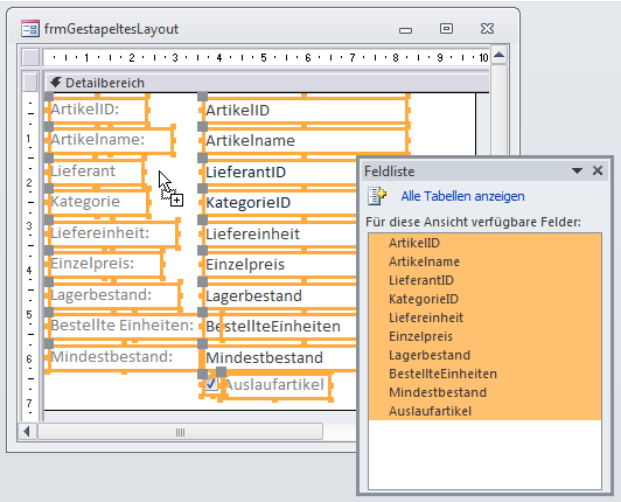



**Bild 1:** Einfügen der Felder in das Formular

## 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).

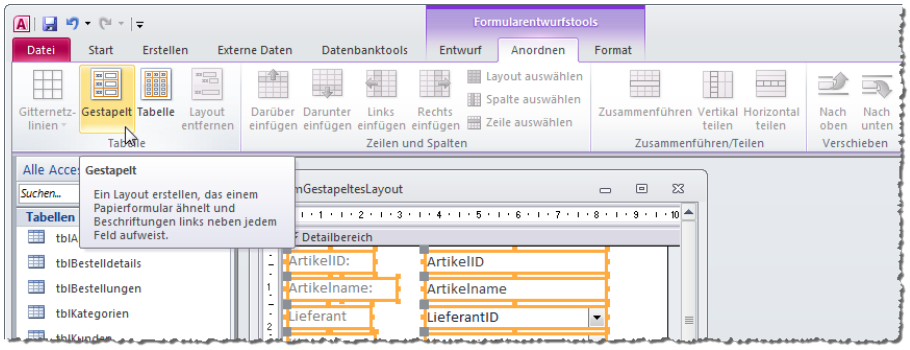



**Bild 2:** Einfügen der Felder in das Formular

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-

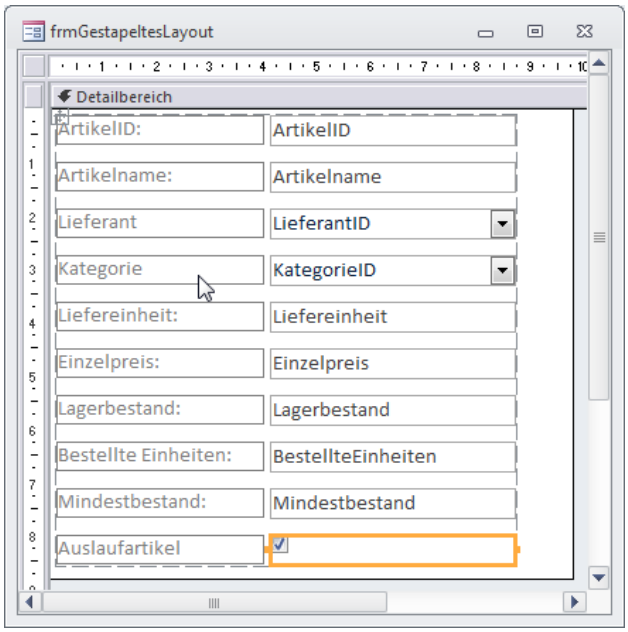



**Bild 3:** Die Steuerelemente nach dem Hinzufügen des Layouts **Gestapelt**.

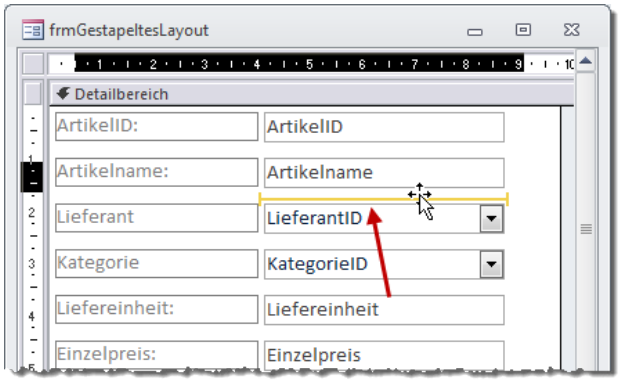



**Bild 4:** Verschieben eines Steuerelements per Drag and Drop

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

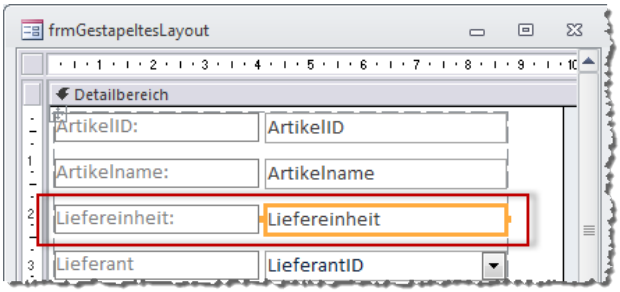



**Bild 5:** Drag and Drop verschiebt das Bezeichnungsfeld gleich mit und platziert die unter dem Ziel liegenden Steuerelemente um eine Position nach unten.

# 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, "|")
||||||||||
```

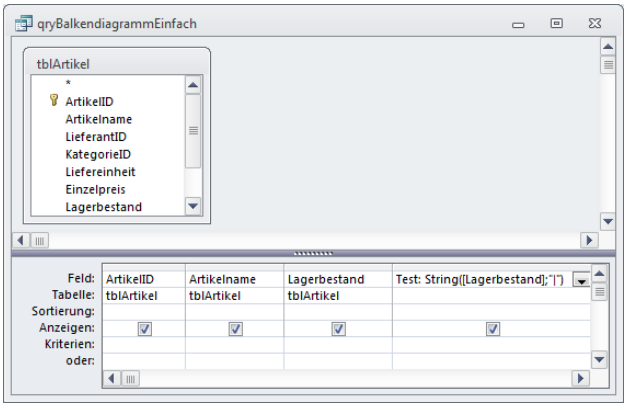



**Bild 2:** Entwurfsansicht einer Abfrage zur Darstellung eines einfachen Balkendiagramms

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

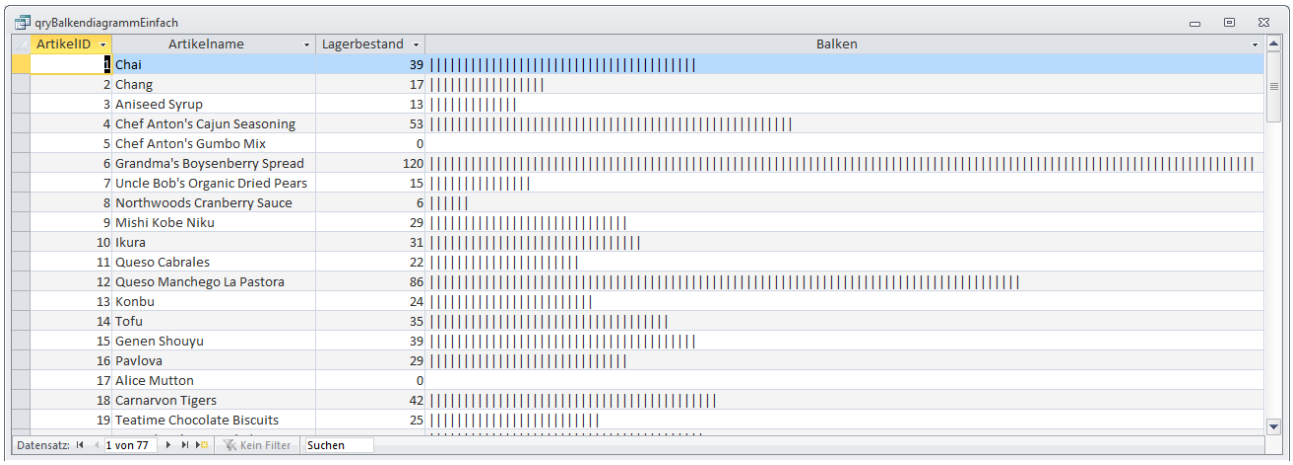



**Bild 1:** Einfaches Balkendiagramm direkt in der Abfrage

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**.

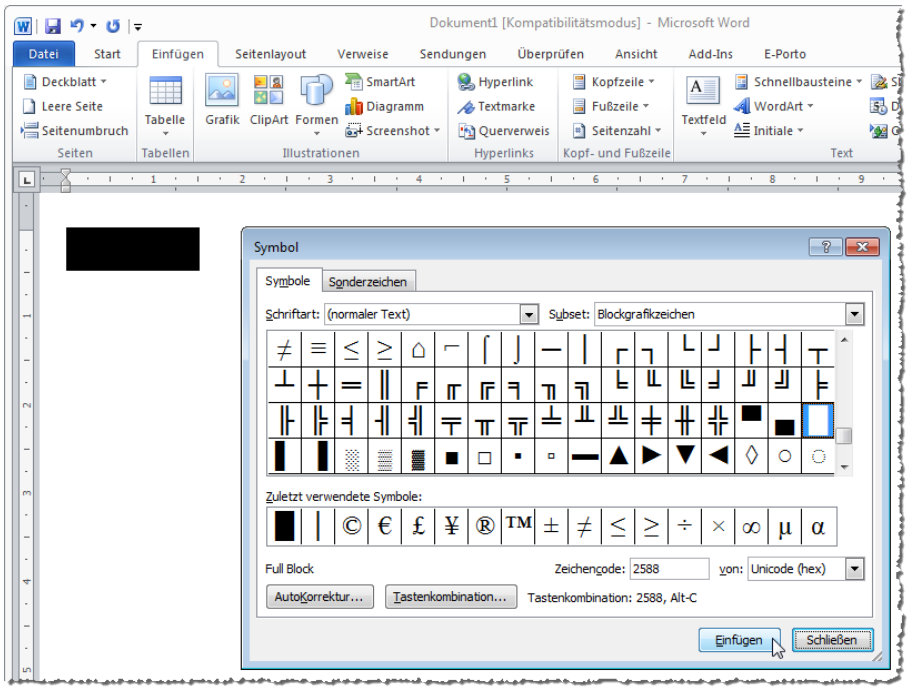



**Bild 3:** Sonderzeichen suchen mit Word

## 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.

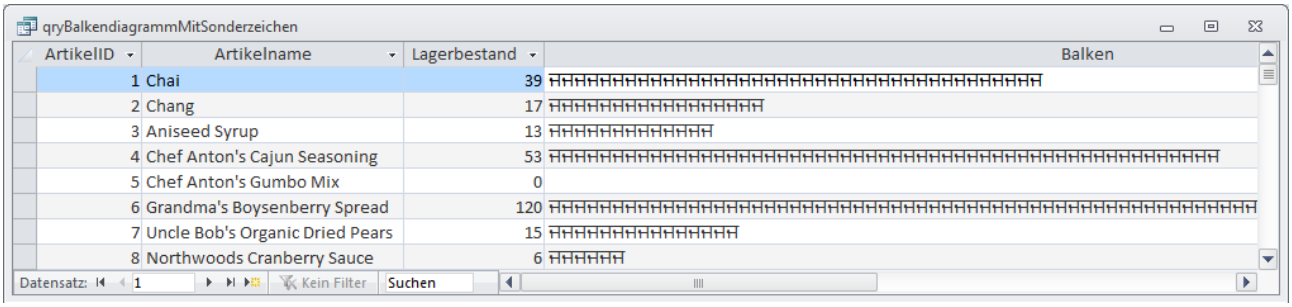



**Bild 4:** Die Abrage liefert das falsche Sonderzeichen

# 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).

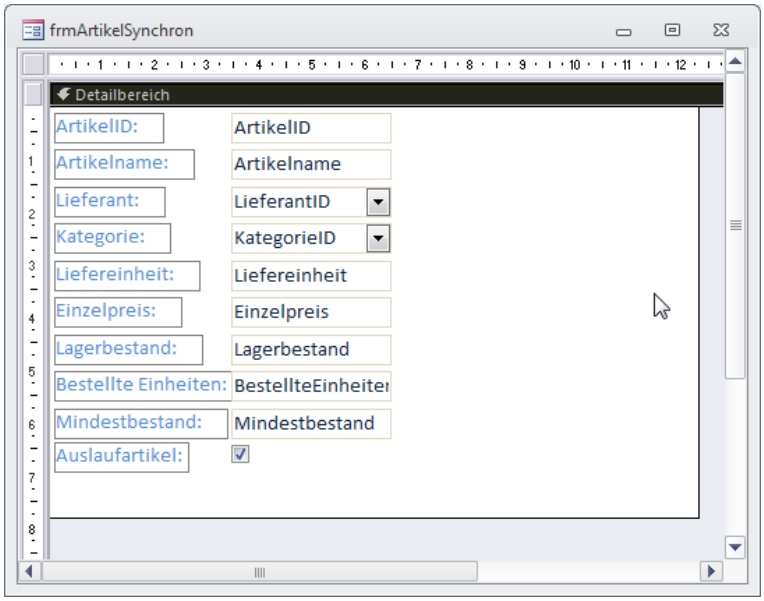



**Bild 1:** Hauptformular mit Artikeldetails

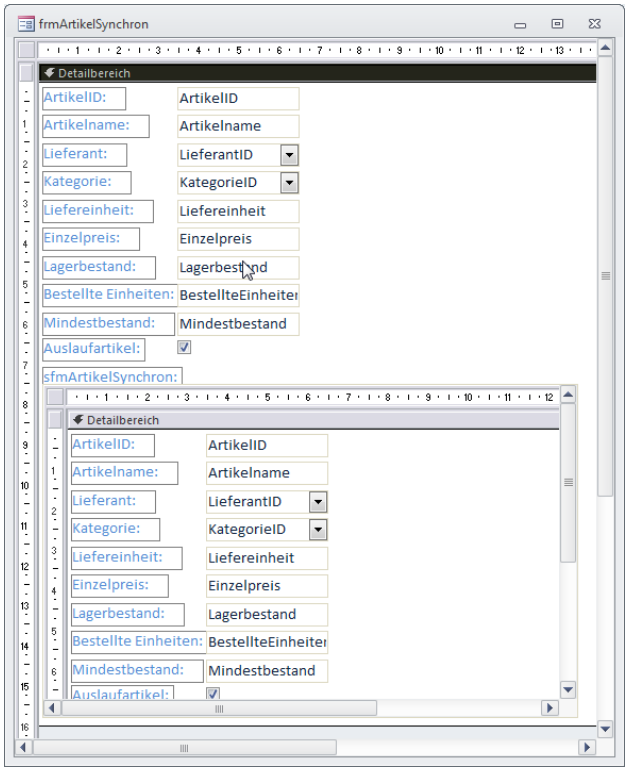



**Bild 2:** Haupt- und Unterformular

## 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.

## Haupt- und Unterformular zusammenführen

Das Ergebnis sieht in der Entwurfsansicht wie in Bild 2 aus. Wenn Sie nun in die Formularansicht wechseln, sieht das Ergebnis wie in Bild 3 aus. Das ist noch nicht exakt das gewünschte Ergebnis, denn das Unterformular zeigt jeweils nur den Datensatz an, der auch im Hauptformular erscheint. Sie können sogar durch die Datensätze im Hauptformular navigieren, das Unterformular wird immer auf den Datensatz des Hauptformulars eingestellt.

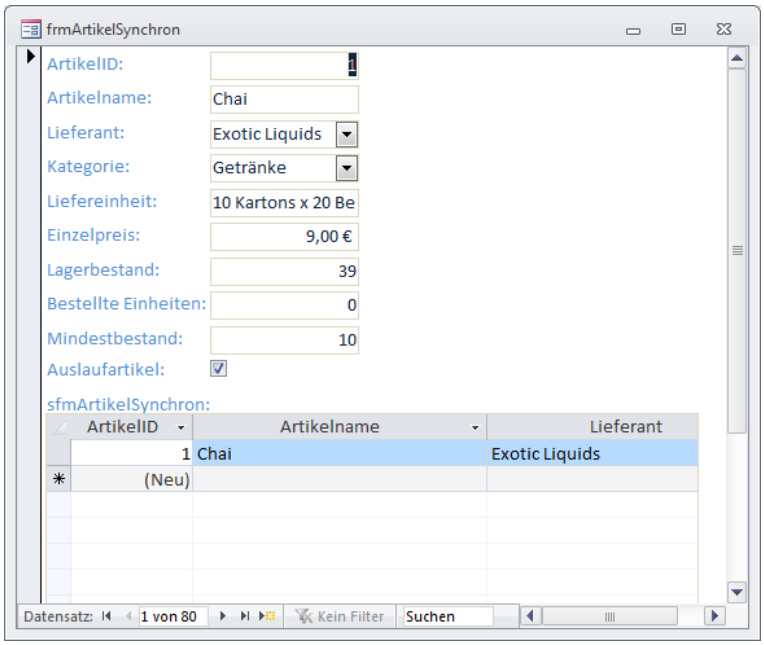



**Bild 3:** Formularansicht, erster Versuch

Warum ist dies so? Weil wir das Unterformular direkt in das Hauptformular gezogen haben. Access prüft dann normalerweise automatisch, ob es Felder in den Datenherkünften von Haupt- und Unterformular gibt, zwischen denen eine Beziehung existiert und trägt diese für die beiden Eigenschaften **Verknüpfen von** und **Verknüpfen nach** ein. Dadurch soll das Unterformular normalweise nur die Datensätze anzeigen, die mit dem Datensatz im Hauptformular verknüpft sind.

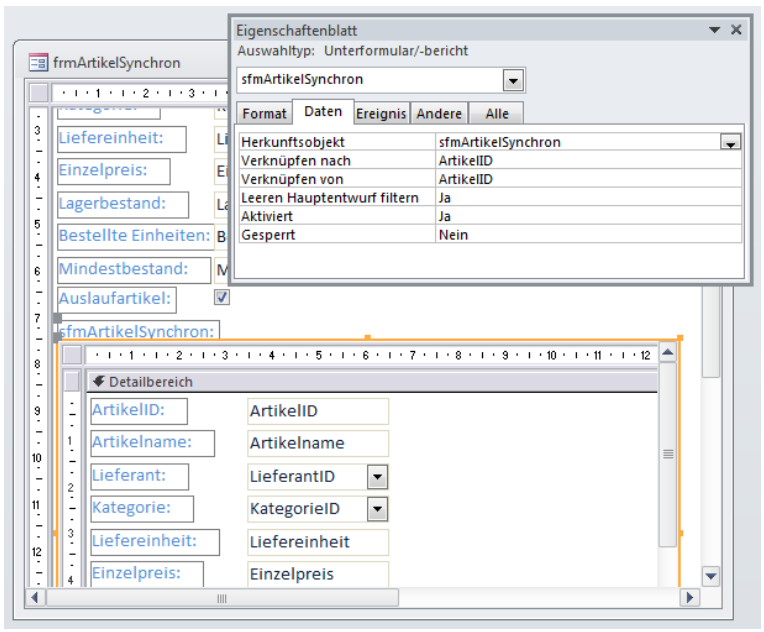



**Bild 4:** Der Grund für die unerhoffte Synchronisierung

Interessanterweise geschieht dies auch, wenn Haupt- und Unterformular die gleiche Tabelle als Datenherkunft verwenden. In diesem Fall trägt Access für die Eigenschaften **Verknüpfen von** und **Verknüpfen nach** jeweils den Namen des Primärschlüsselfeldes der Datenherkünfte ein, also **KundeID** (siehe Bild 4).

Dies bewirkt den nicht erwünschten Effekt: Das Unterformular wird jeweils so gefiltert, dass nur der Datensatz erscheint, dessen Wert im Feld **KundeID** mit dem entsprechenden Wert im Hauptformular identisch ist.

## Synchronisierung anpassen

Der erste Schritt, die Synchronisierung wie gewünscht herzustellen, ist das Leeren der Eigenschaften **Verknüpfen von** und **Verknüpfen nach**. Danach erhalten wir ein Haupt- und ein Unterformular, die völlig unabhängig voneinander sind. Nun soll das Hauptformular allerdings zu dem im Unterformular ausgewählten Wert springen und umgekehrt. Ohne VBA-Code lässt sich dies nicht erreichen, aber die Lösung ist relativ einfach – wenn auch nicht intuitiv. Normalerweise würde man davon ausgehen, dass hier zwei Ereignisprozeduren nötig sind, die beim Anzeigen eines neuen Datensatzes im Haupt- und im Unterformular ausgelöst werden und das jeweils andere Formular auf den aktuellen Datensatz einstellen.

Sie würden dann für das Ereignis **Beim Anzeigen** des Hauptformulars die folgende Ereignisprozedur hinterlegen:

# 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.

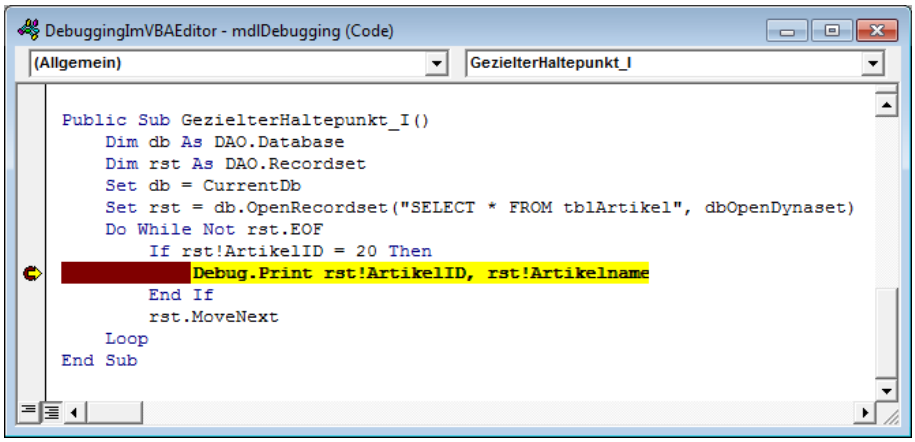



**Bild 1:** Anhalten des Codes bei Erreichen eines bestimmten Wertes

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

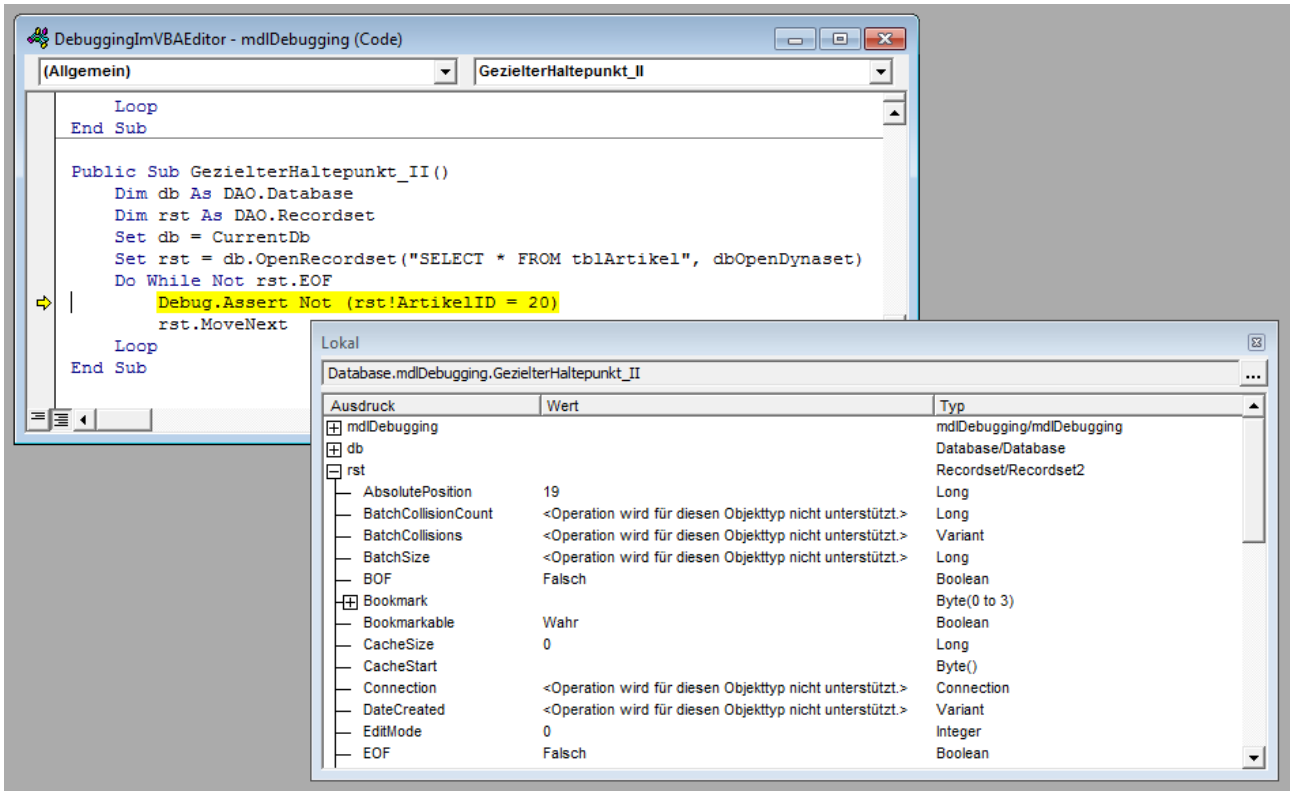



**Bild 2:** Das Lokal-Fenster liefert eine Reihe Informationen.

Access [basics] 4/2013
www.access-basics.de
Das Access-Magazin für alle, die schnell von 0 auf 100 wollen

# In dieser Ausgabe:

**Filterkriterien für Formulare, Teil I – Seite 2**

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.

**Textfelder: Value, OldValue und Text – Seite 10**

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.

**Änderungen protokollieren, Teil I – Seite 14**

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.

**Berichte per E-Mail verschicken, Teil I – Seite 19**

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.

## Impressum

Access [basics] wird monatlich herausgegeben von:
André Minhorst Verlag | Borkhofer Straße 17 | 47137 Duisburg

Die hier veröffentlichten Texte sind urheberrechtlich geschützt. Übersetzung und Vervielfältigung bedürfen der ausdrücklichen schriftlichen Genehmigung des Verlages. Sämtliche Veröffentlichungen in Access [basics] erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

André Minhorst Fachverlag für Softwareentwicklung übernimmt für beschriebene oder zum Download bereitstehende Programme weder Gewähr noch Haftung, außer für Vorsatz oder grobe Fahrlässigkeit. Bezugspreise erfahren Sie auf www.access-basics.de.

Redaktion:
André Minhorst (V.i.S.d.P) | E-Mail: info@access-basics.de | Internet: www.access-basics.de

Geschäftsführung, Herstellung, Text- und Schlussredaktion, Layout von Magazin und Webseite: André Minhorst

Autor: André Minhorst

**ISSN: 2190-8761**

# 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.

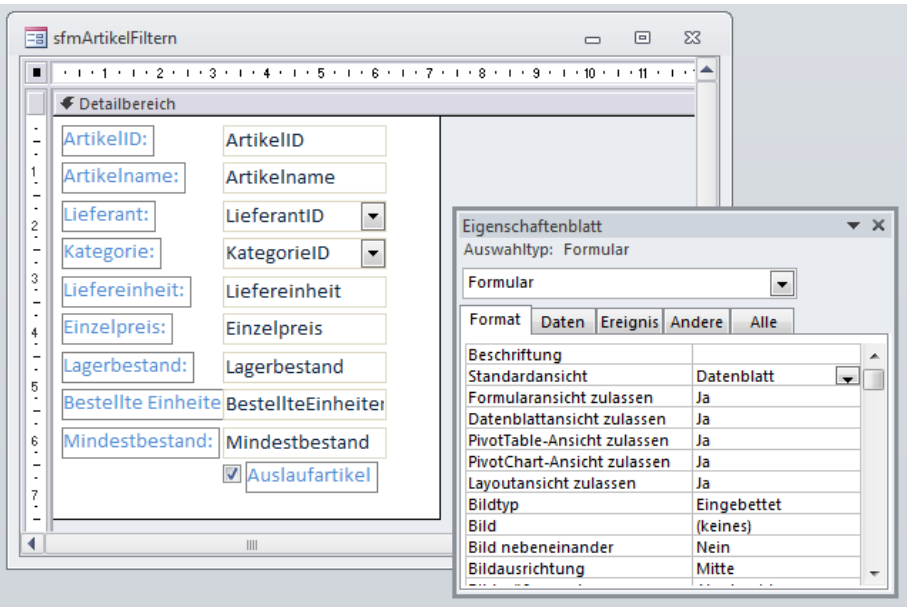



**Bild 1:** Entwurf des Unterformulars mit den zu filternden Daten

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.

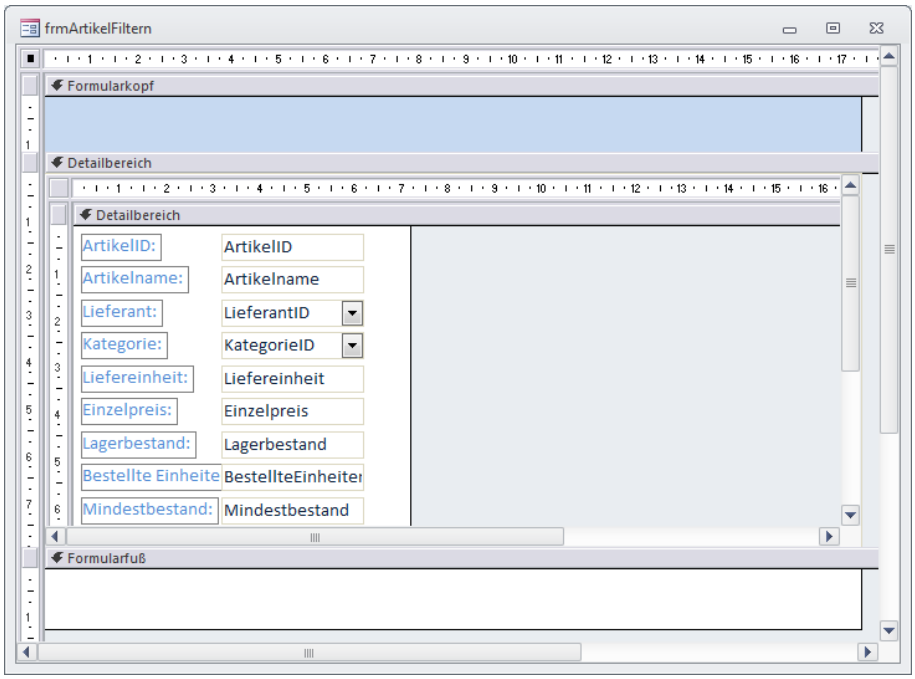



**Bild 2:** Haupt- und Unterformular in der Entwurfsansicht

## 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.

# 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.

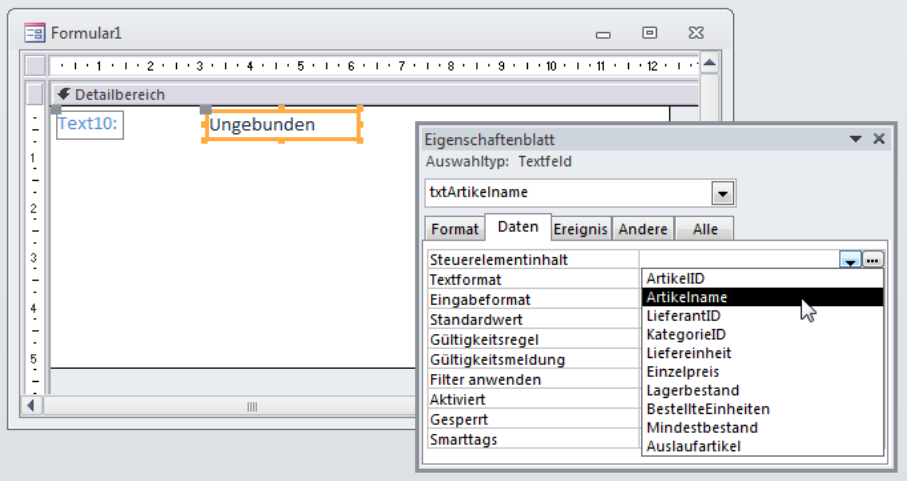



**Bild 1:** Zuweisen eines Feldes als Steuerelementinhalt

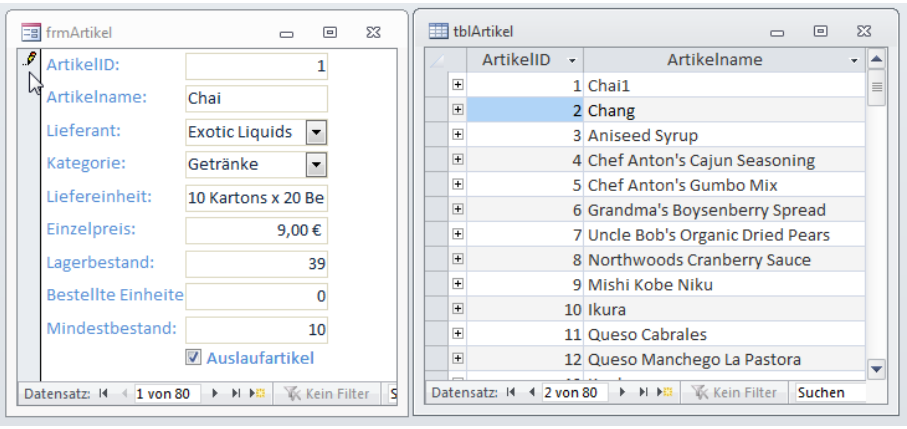



**Bild 2:** Wann wird der geänderte Wert gespeichert?

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
```

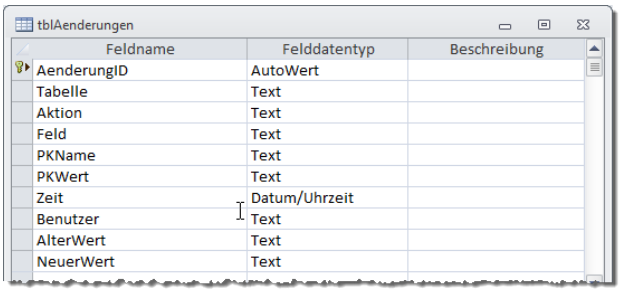



**Bild 1:** Die Tabelle **tblAenderungen** in der Entwurfsansicht

```
        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).

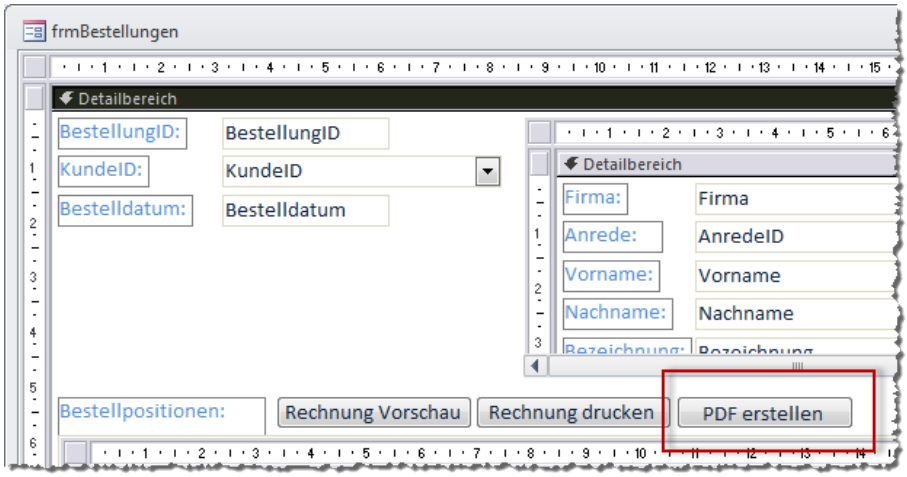



**Bild 1:** Neue Schaltfläche zum Erstellen eines PDF-Dokuments auf Basis der Rechnung

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).

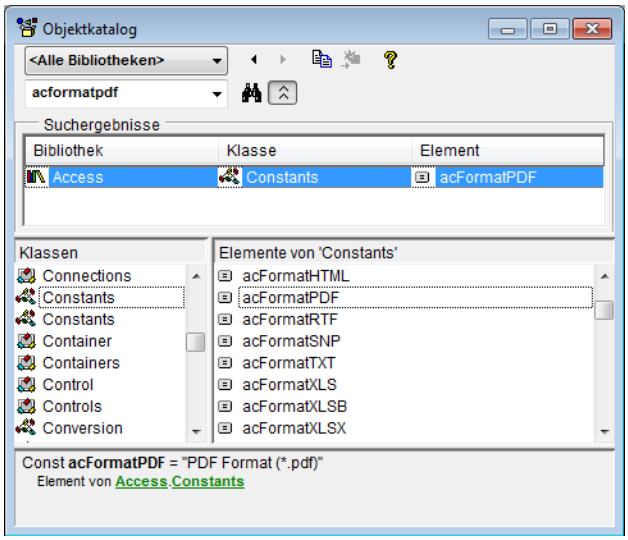



**Bild 2:** Neue Schaltfläche zum Erstellen eines PDF-Dokuments auf Basis der Rechnung

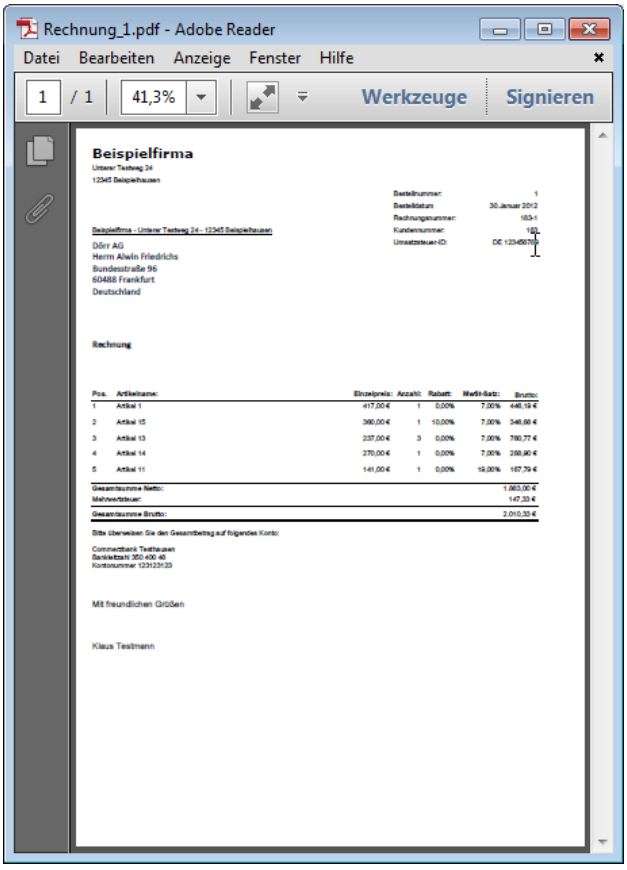



**Bild 3:** Ein Access-Bericht im PDF-Format

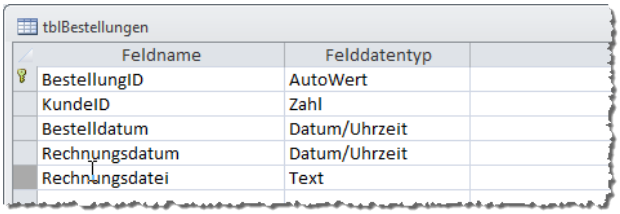
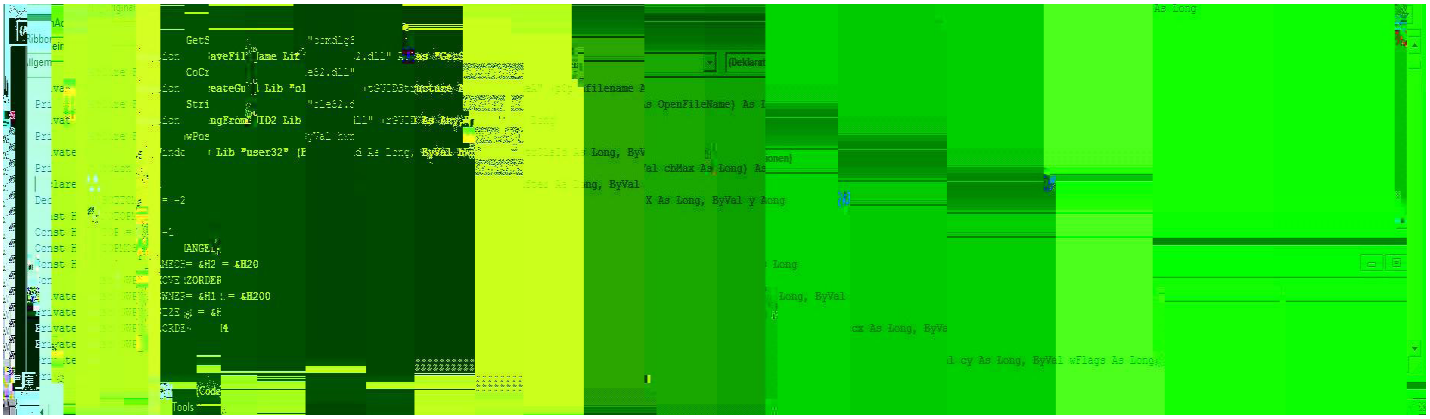



**Bild 4:** Feld zum Speichern der Rechnungsdatei

Ausgabe 05/2013

# ACCESS BASICS

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



## In diesem Heft:

### FILTERKRITERIEN FÜR FORMULARE, TEIL II

DAS FILTERN VON DATEN IST EINE DER STÄRKEN VON ACCESS. DER ZWEITE TEIL DER ARTIKELSERIE ZEIGT, WIE SIE DATEN NACH DATUM, ZAHLEN UND JA/NEIN-FELDERN FILTERN.

SEITE 3

### MAILS MIT OUTLOOK SCHICKEN

MICROSOFT OUTLOOK IST FÜR VIELE ACCESS-NUTZER DAS MITTEL DER WAHL, UM E-MAILS ZU VERSENDEN. LERNEN SIE DIE WICHTIGSTEN TECHNIKEN KENNEN.

SEITE 3

### BERICHTE PER E-MAIL VERSENDEN, TEIL II

ERSTELLEN SIE PDF-DOKUMENTE AUS BERICHTEN UND VERSENDEN SIE DIESE ALS ANHANG EINER E-MAIL – UND DAS ALLES MIT VBA UND OUTLOOK!

SEITE 3

André Minhorst Verlag

# Filterkriterien für Formulare, Teil II

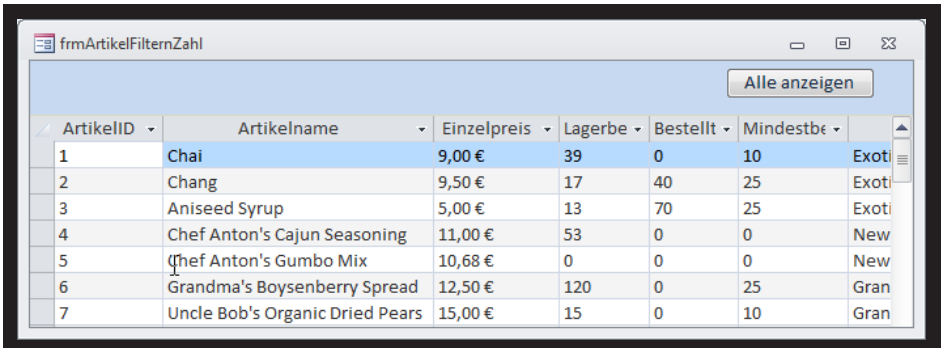
Was helfen Artikel-, Kunden-, Bestell- und sonstige Daten, wenn Sie diese zwar schön in einem Formular darstellen, aber diese nicht entsprechend filtern können? In diesem Artikel zeigen wir Ihnen, wie Sie verschiedene individuelle Filterkriterien mithilfe entsprechender Steuerelemente im Formular festlegen 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

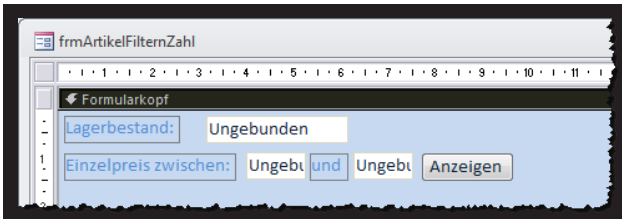



**Bild 3:** Filtern nach Mindest- und Höchstpreis

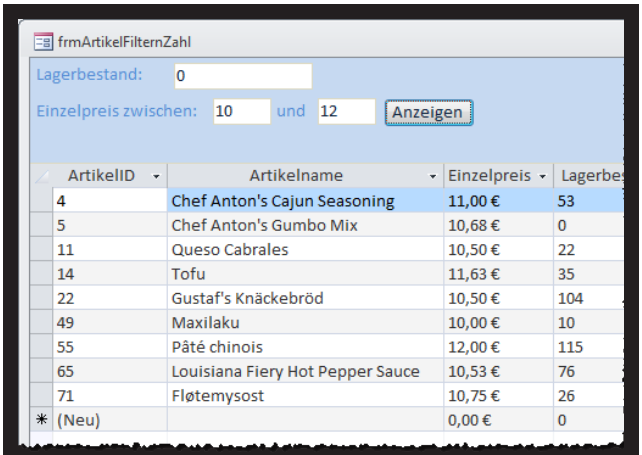



**Bild 4:** Preisfilter im Einsatz

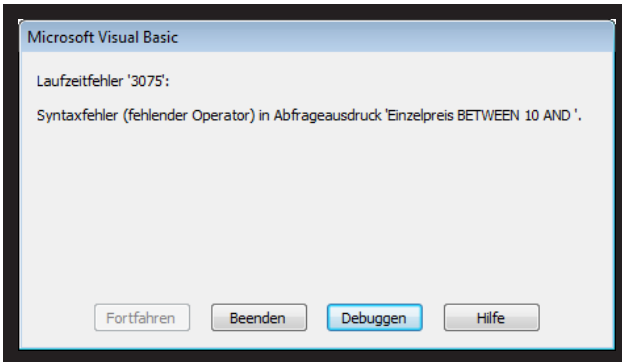



**Bild 5:** Fehler beim Auslassen eines Parameters

# 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-

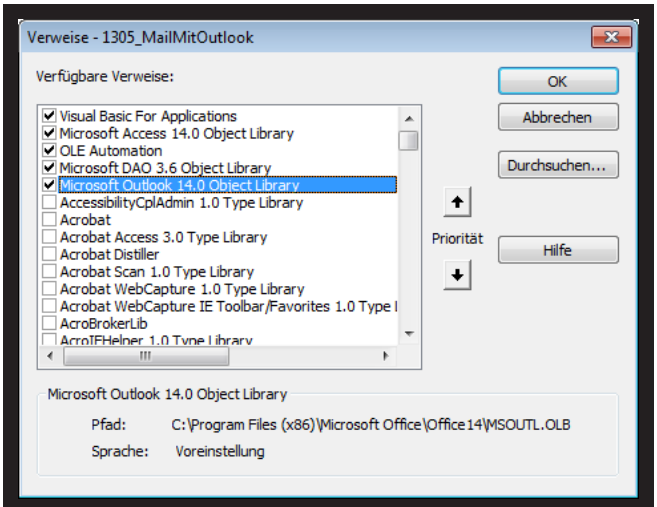



**Bild 1:** Verweise eines VBA-Projekts

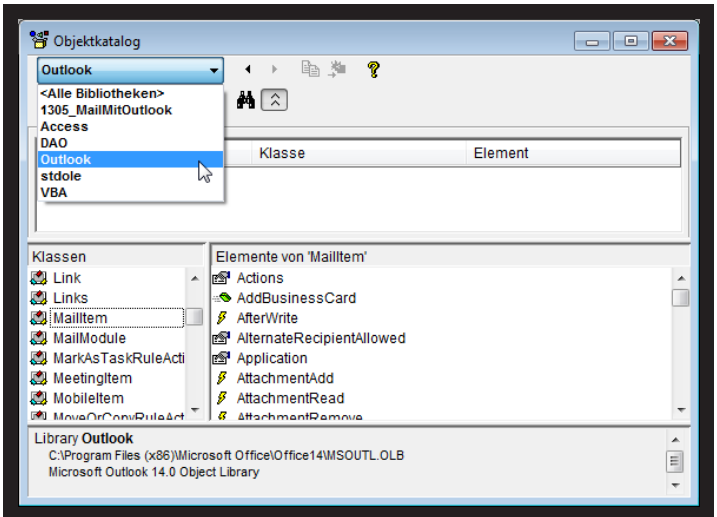



**Bild 2:** Die Outlook-Bibliothek im Objektkatalog

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

# 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.

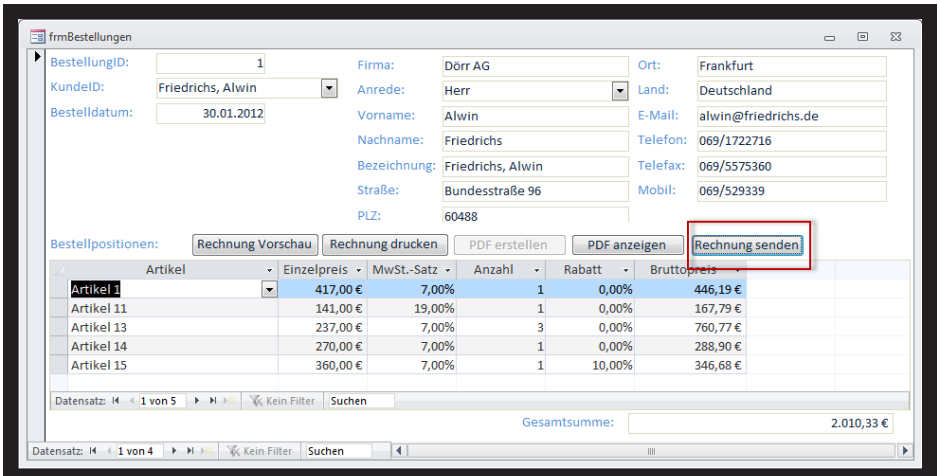



**Bild 1:** Hinzufügen einer Schaltfläche zum Versenden der Rechnung per E-Mail

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

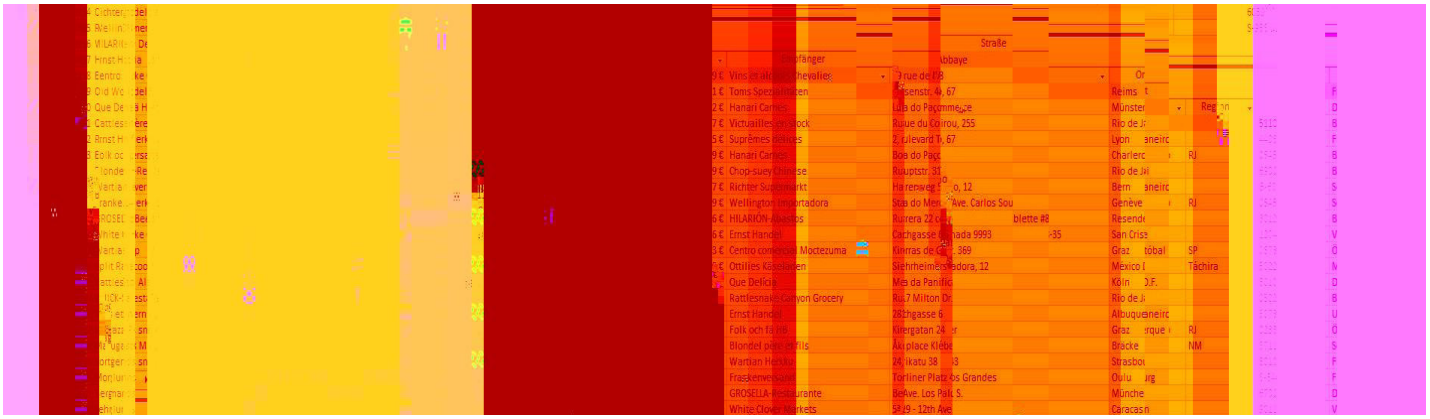
Den eigentlichen Rechnungsversand löst der Benutzer durch einen Klick auf die neu hinzugefügte Schaltfläche aus. Damit dies geschieht, stellen Sie die Eigenschaft **Beim Klicken** der Schaltfläche **cmdRechnungSenden** auf den Wert **[Ereignisprozedur]** ein, klicken auf die Schaltfläche mit den drei 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**.

Ausgabe 06/2013

# ACCESS BASICS

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



## In diesem Heft:

### E-MAILS SICHER VERSENDEN

ERSTELLEN SIE MAILS IN OUTLOOK UND ÜBERNEHMEN SIE NACHTRÄGLICH VORGENOMMENE ÄNDERUNGEN IN DIE DATENBANK.

SEITE 3

### EREIGNISSE ABGREIFEN

LERNEN SIE, WIE SIE EREIGNISSE ANDERER OBJEKTE WIE FORMULARE ODER OUTLOOK-E-MAILS ABGREIFEN UND IN ACCESS PER VBA MIT EIGENEM CODE FÜLLEN KÖNNEN

SEITE 8

### FILTERKRITERIEN FÜR FORMULARE, TEIL III: KOMBINATIONSFELDER

ERFAHREN SIE, WIE SIE DATENSÄTZE NACH DEM INHALT VON KOMBINATIONSFELDERN FILTERN

SEITE 12

# 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).

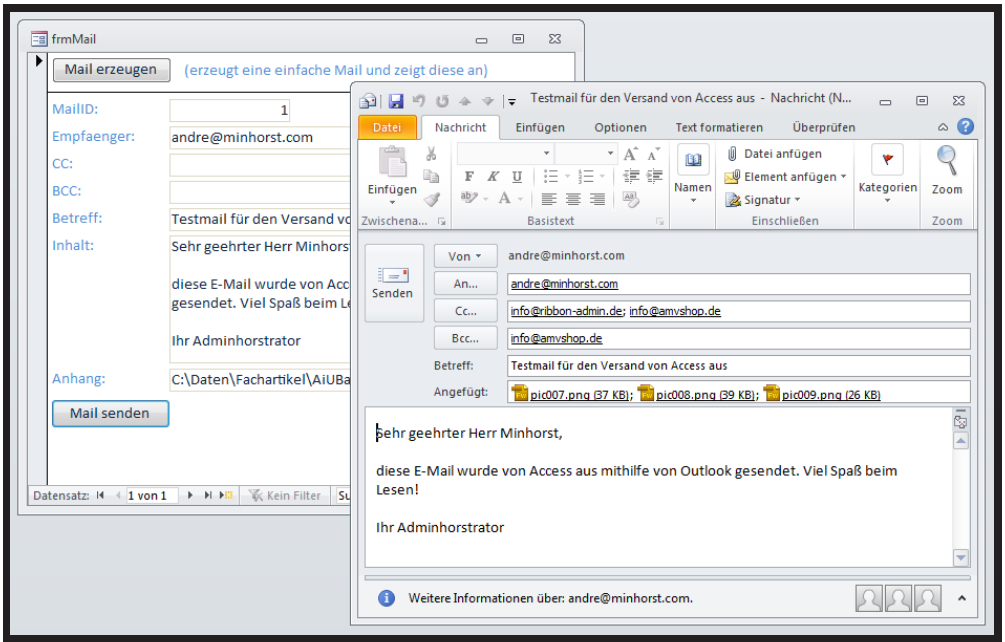



**Bild 1:** Erstellen einer E-Mail per VBA und Anzeige in der Outlook-Vorschau

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
```

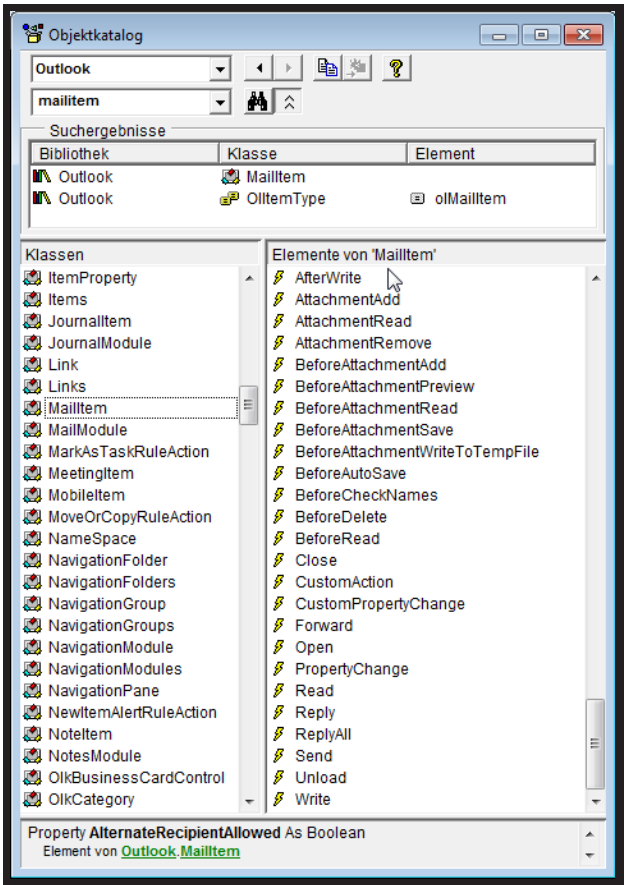



**Bild 2:** Ereignisse des **MailItem**-Objekts in Outlook

```
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.

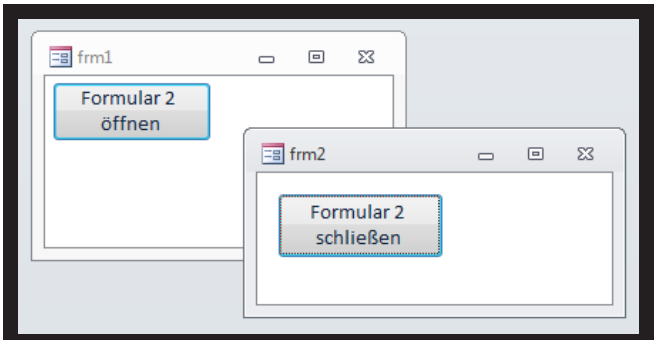



**Bild 1:** Formular 1 öffnet Formular 2, Formular 2 wird wieder geschlossen

## 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.

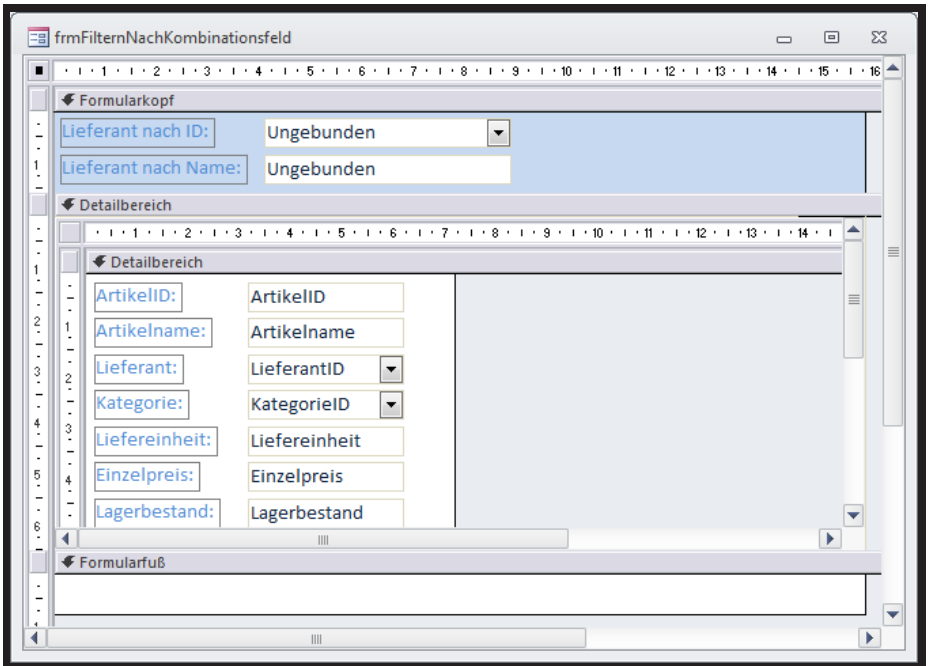



**Bild 1:** Formular zum Filtern nach Kombinationsfeldern und ihren Inhalten

## 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.

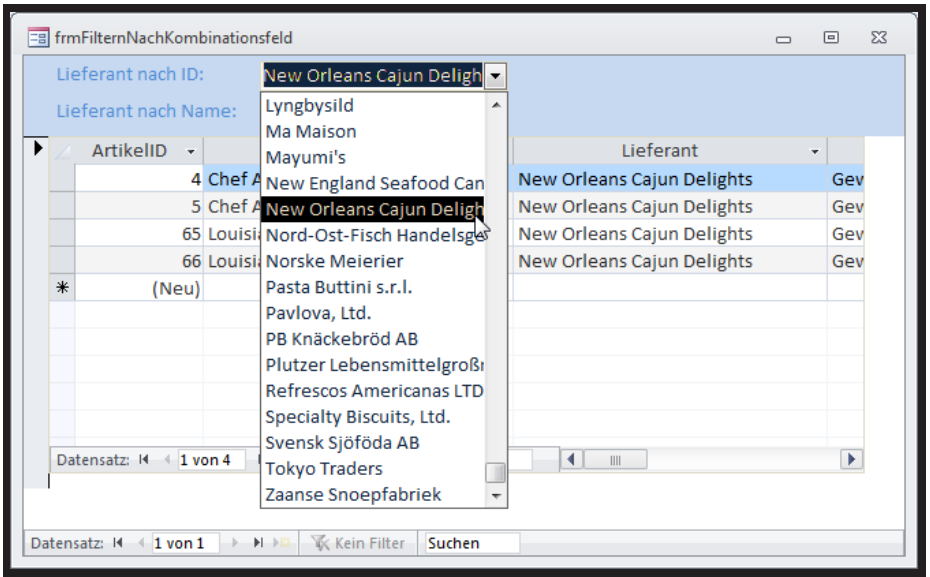



**Bild 2:** Filtern von Datensätzen nach der gebundenen Spalte des Nachschlagefeldes

```
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

# 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**

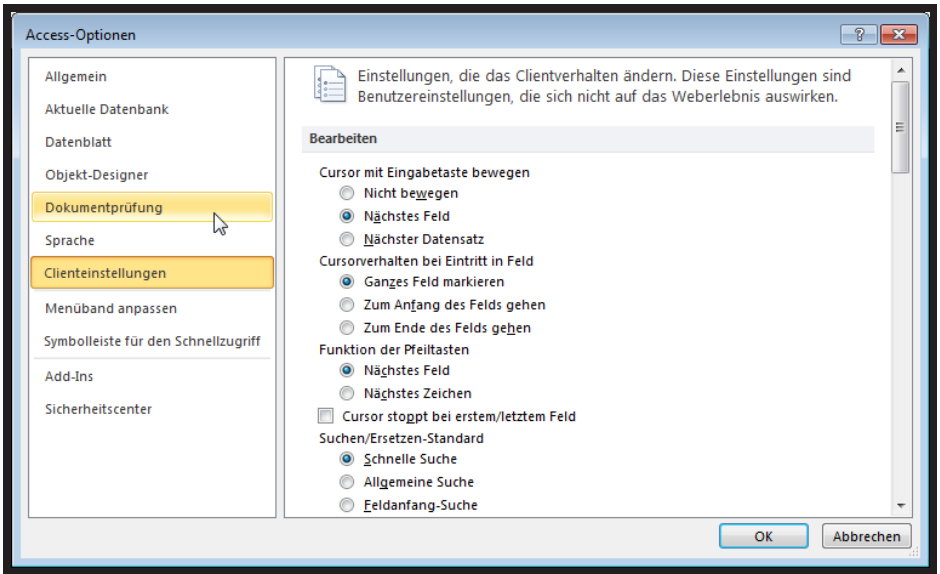



**Bild 1:** Einstellungen für das Verhalten beim Eintritt in ein Feld

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.

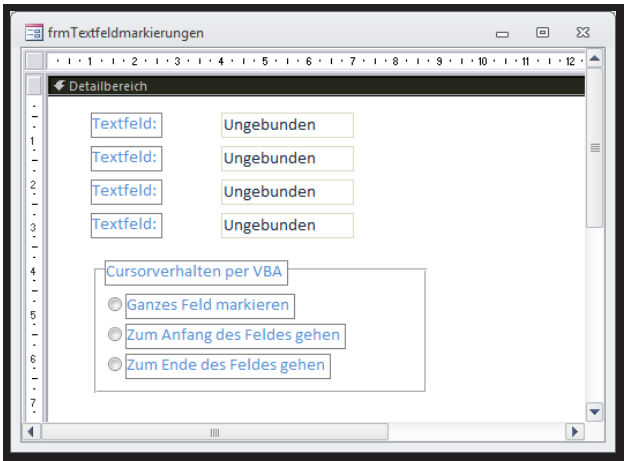



**Bild 3:** Einstellen des Cursorverhaltens per Formular/VBA

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

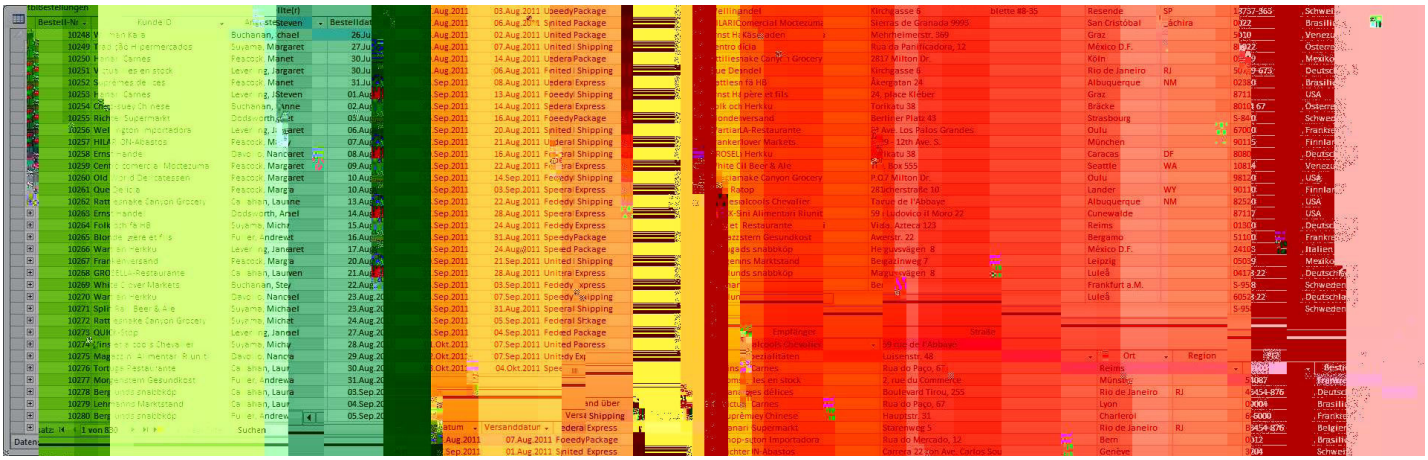
```
Private Sub ogrCursorverhalten_AfterUpdate()
    SetOption "Behavior Entering Field", Me!ogrCursorverhalten
End Sub
```

**Listing 2:** Cursorverhalten per VBA einstellen

Ausgabe 07/2013

# ACCESS BASICS

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



## In diesem Heft:


### REKURSION MIT VBA

LERNEN SIE, WIE SIE DIE DATEN IN HIERARCHISCH ODER REFLEXIV VERKNÜPFTEN TABELLEN DURCHLAUFEN, UM DIESE ETWA ALS BAUM AUSZUGEBEN.

SEITE 3

### PLATZHALTER ERSETZEN

STATTEN SIE STANDARDTEXTE FÜR ANSCHREIBEN ET CETERA MIT PLATZHALTERN AUS UND ERSETZEN SIE DIESE BEI BEDARF MIT ECHTEN DATEN WIE ETWA ANSCHRIFT ODER BESTELLDATEN.

SEITE 10

### TREEVIEW, TEIL I

NUTZEN SIE BEREITS DAS TREEVIEW-STEUERELEMENT? LERNEN SIE DIE GRUNDLAGEN DIESES ALLROUND-STEUERELEMENTS KENNEN UND SETZEN SIE ES IN IHREN EIGENEN FORMULAREN EIN.

SEITE 18

# 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:

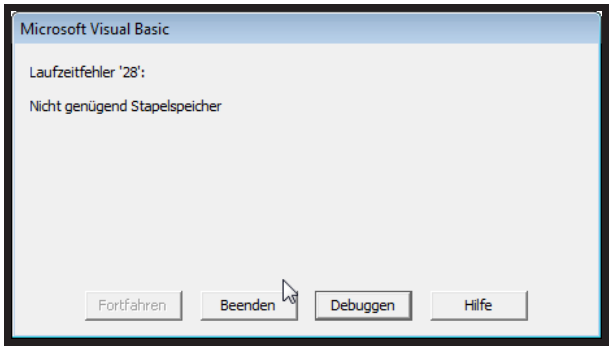



**Bild 1:** Fehler bei rekursiver Funktion ohne Abbruchbedingung

```
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

# 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.

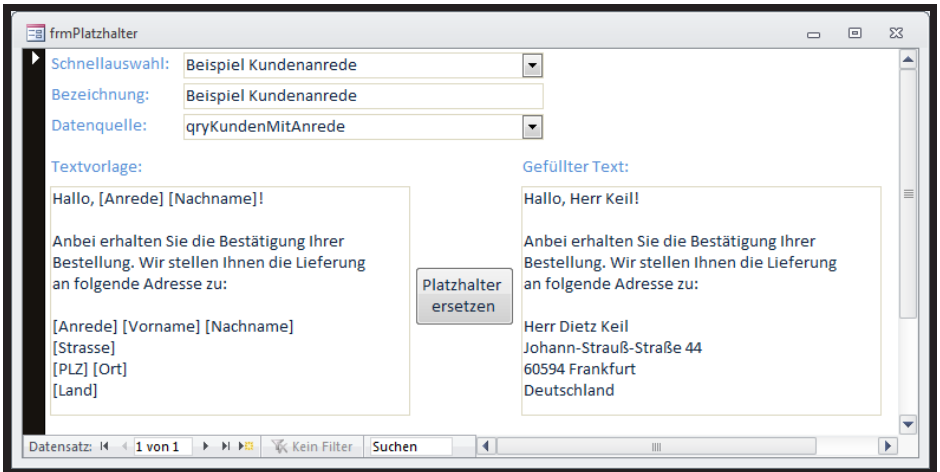



**Bild 1:** Beispielformular für das Ersetzen von Platzhaltern

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.

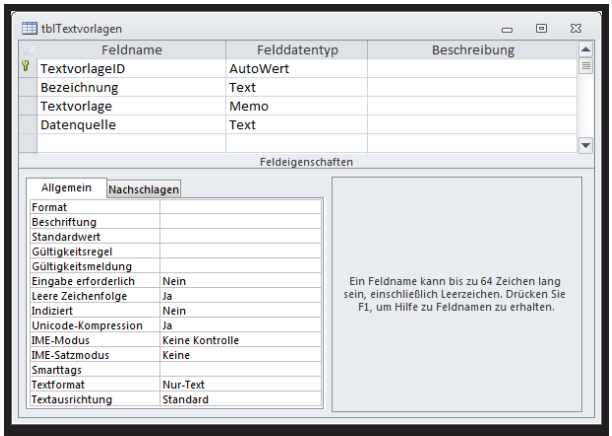



**Bild 2:** Entwurfsansicht der Tabelle **tblTextvorlagen**

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.

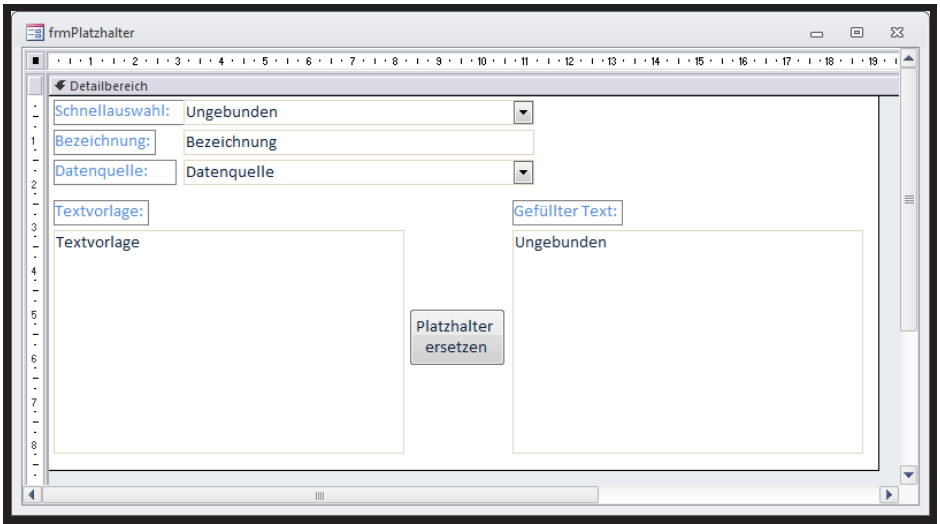



**Bild 3:** Entwurfsansicht des Formulars **frmPlatzhalter**

```
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

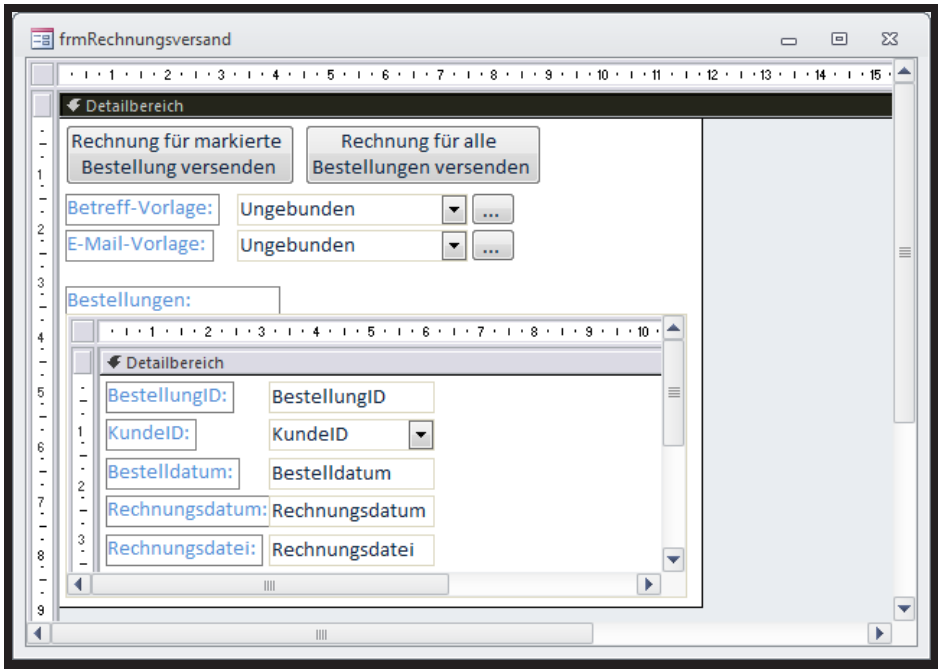



**Bild 1:** Kombinationsfelder zum Auswählen der Vorlagen für Betreff und E-Mail

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-

```
Private Sub cmdVorlagenBetreffAnzeigen_Click()
    If Not IsNull(Me!cboVorlageBetreff) Then
        DoCmd.OpenForm "frmPlatzhalter", WhereCondition:="TextvorlageID = " & Me!cboVorlageBetreff
    Else
        DoCmd.OpenForm "frmPlatzhalter"
    End If
End Sub
```

**Listing 1:** Schaltfläche zum Bearbeiten einer Vorlage, ausgehend von dem im Kombinationsfeld cboVorlageBetreff ausgewählten Eintrag

nisprozeduren, die durch die beiden Schaltflächen ausgelöst werden, unterscheiden sich nur unwesentlich voneinander. Daher finden Sie in Bild 1 nur die erste der beiden abgebildet. Die Prozedur prüft, ob der Benutzer bereits einen Eintrag mit dem Kombinationsfeld ausgewählt hat. Falls ja, öffnet er das Formular **frmPlatzhalter** und zeigt direkt die im Kombinationsfeld ausgewählte Textvorlage an. Anderenfalls öffnet die Prozedur das Formular so, dass dies den ersten Datensatz der Datenherkunft anzeigt (siehe Listing 1).

In Aktion sehen die beiden Formulare wie in Bild 2 aus.

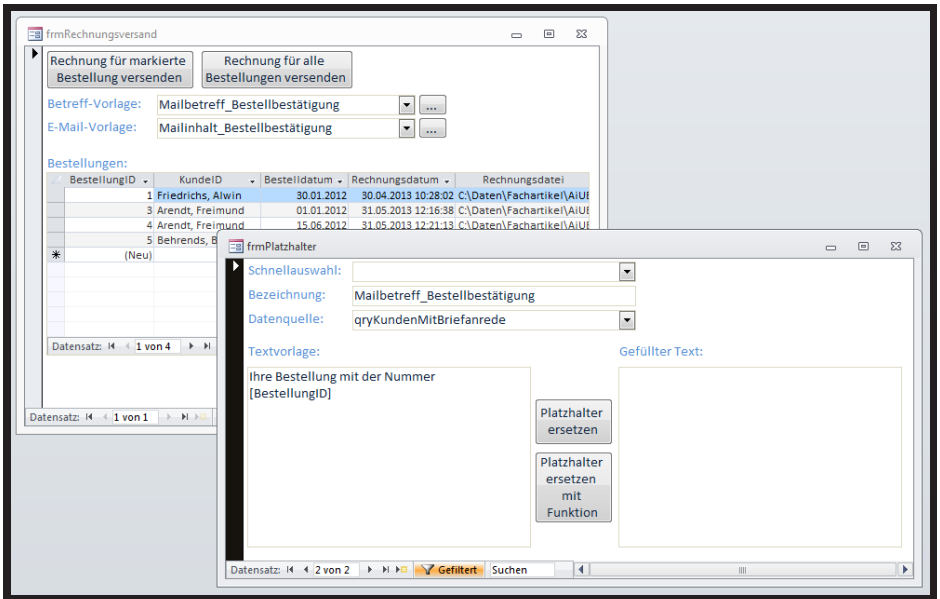



**Bild 2:** Öffnen des Formulars zum Anpassen der Textvorlagen

## Mails anpassen und versenden

Der Versand der E-Mails wurde ja, wie im vorherigen Teil der Artikelreihe beschrieben, durch die beiden Schaltflächen **cmdAlleRechnungen** oder **cmdMarkierteRechnung** eingeleitet.

Beide Prozeduren haben die Routine **RechnungErstellenUndVersenden** aufgerufen, welche – sofern nur eine Rechnung versendet werden soll – den Primärschlüsselwert für die entsprechende Rechnung erwartet.

Die Prozedur **RechnungErstellenUndVersenden** prüft also, ob **lngBestellungID** größer als **0** ist und stellt in diesem Fall einen entsprechenden Filterausdruck ein, der später bei Erstellen des Recordsets berücksichtigt wird (siehe Listing 2).

Den Betreff und den Inhalt, die vorher halbstatisch direkt im Code verankert wurden, bezieht die Prozedur direkt per **DLookup** aus der Tabelle **tblTextvorlagen**. Dabei ermittelt sie jeweils die Textvorlage, die der Benutzer zuvor mithilfe der beiden Kombinationsfelder **cboVorlageBetreff** und **cboVorlageInhalt** festgelegt hat.

Dann erstellt die Prozedur ein Recordset, das je nach Eingangsparameter entweder einen bestimmten oder alle Datensätze der Herkunftstabelle enthält. Anschließend durchläuft die Prozedur alle angegebenen Datensätze. Dabei erstellt sie erst die Rechnung im PDF-Format und erstellt dann eine neue E-Mail.

Diese füllt Sie wie in der Version aus dem vorherigen Teil der Artikelreihe. Der Unterschied ist, dass Betreff und Inhalt aus den zuvor gefüllten Variablen **strBetreff** und **strInhalt** stammen.

Vorher ersetzt die Prozedur **PlatzhalterErsetzen**, die im Artikel **Platzhalter in Texten ersetzen** vorgestellt wurde, noch die Platzhalter.

# 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

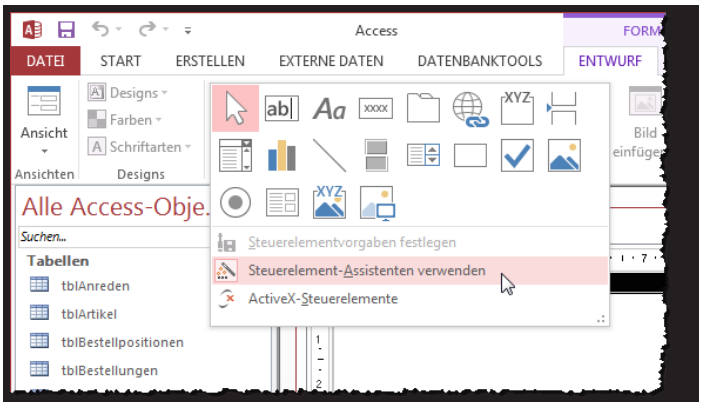



**Bild 1:** Aufrufen der Liste der ActiveX-Steuerelemente

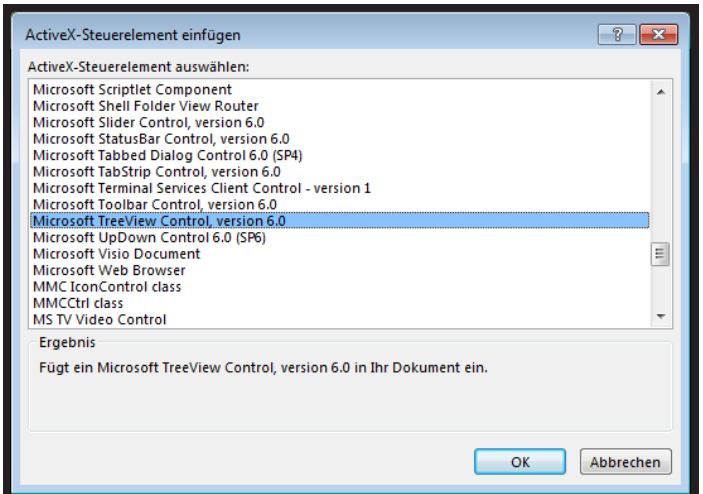



**Bild 2:** Auswahl des TreeView-Steuerelements

**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-**

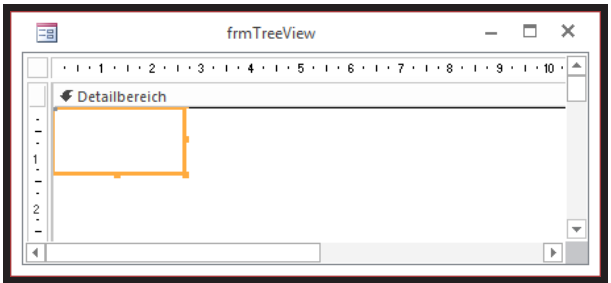



**Bild 3:** Ein frisches TreeView-Steuerelement im Formularentwurf

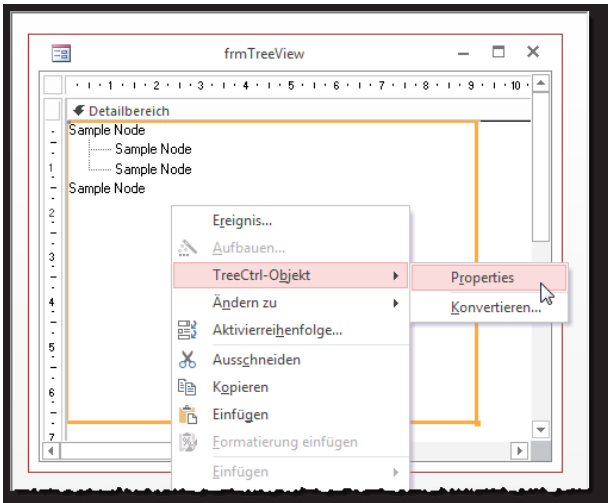



**Bild 4:** Aufruf des Eigenschaftsfensters eines ActiveX-Steuerelements

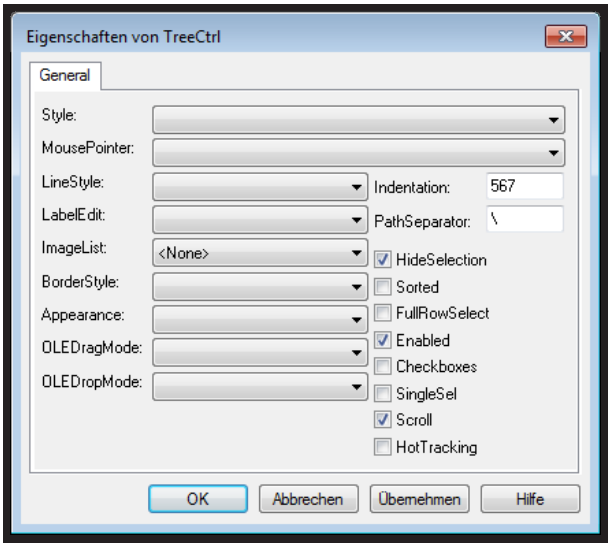



**Bild 5:** Die Eigenschaften des TreeView-Steuerelements – ohne Werte und Auswahlmöglichkeit

Ausgabe 08/2013

# ACCESS BASICS

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



## In diesem Heft:


### MAILS MIT OUTLOOK SCHICKEN, TEIL II

MICROSOFT OUTLOOK IST FÜR VIELE ACCESS-NUTZER DAS MITTEL DER WAHL, UM E-MAILS ZU VERSENDEN. LERNEN SIE DIE WICHTIGSTEN TECHNIKEN KENNEN.

SEITE 3

### BILDER IN ACCESS, TEIL I

BENUTZERDEFINIERTE BILDER KOMMEN IN ACCESS AN DEN VERSCHIEDENSTEN STELLEN VOR: IN DEN DATEN, AUF SCHALTFLÄCHEN, IM RIBBON ET CETERA. ERFAHREN SIE ALLES RUND U BILDER IN ACCESS.

SEITE 11

### DAS ANLAGEFELD

MIT ACCESS 2007 HAT MICROSOFT EINEN NEUEN FELDDATENTYP SAMT PASSENDEM STEUERELEMENT EINGEFÜHRT. LESEN SIE ALLES, WAS SIE ZUM ANLAGEFELD WISSEN MÜSSEN!

SEITE 17

# 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.

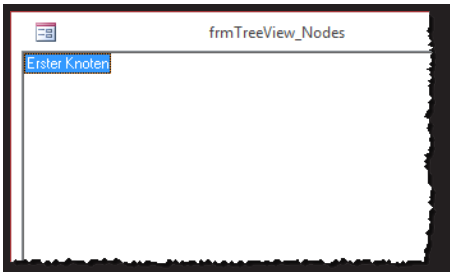



**Bild 1: Erstes Element im TreeView-Steuerelement**

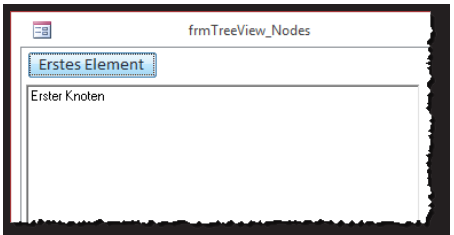



**Bild 2: Erstes Element per Schaltfläche**

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:

# 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

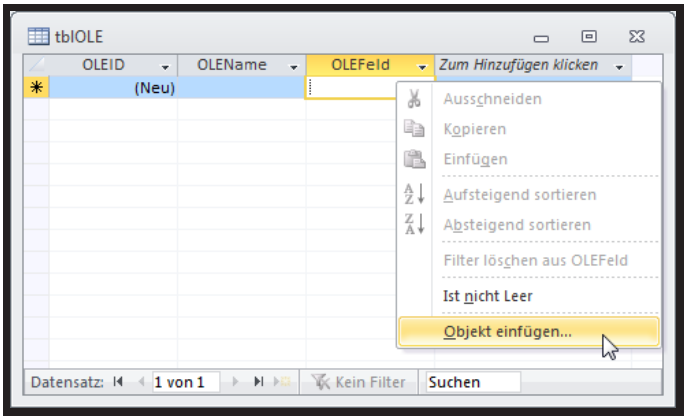



Bild 2: Füllen eines OLE-Feldes

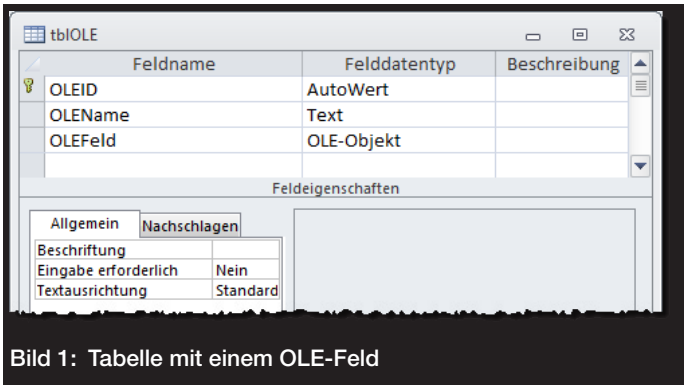



Bild 1: Tabelle mit einem OLE-Feld

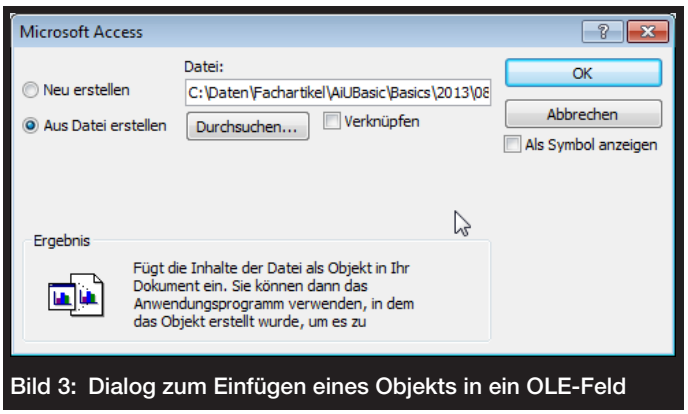



Bild 3: Dialog zum Einfügen eines Objekts in ein OLE-Feld

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

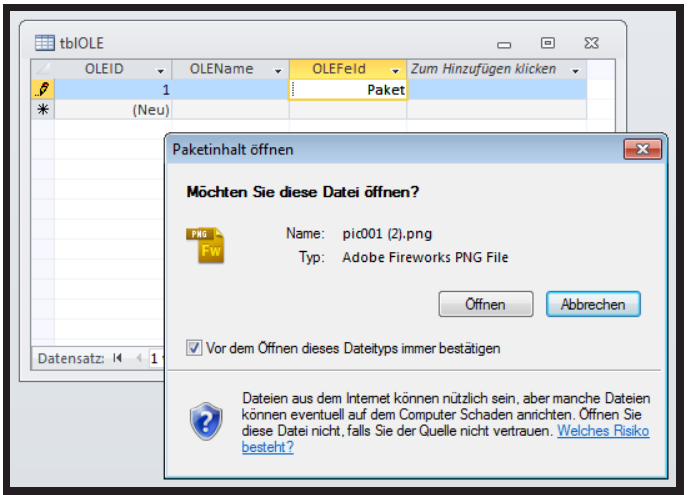


Bild 4: Öffnen einer in einem OLE-Feld gespeicherten Datei

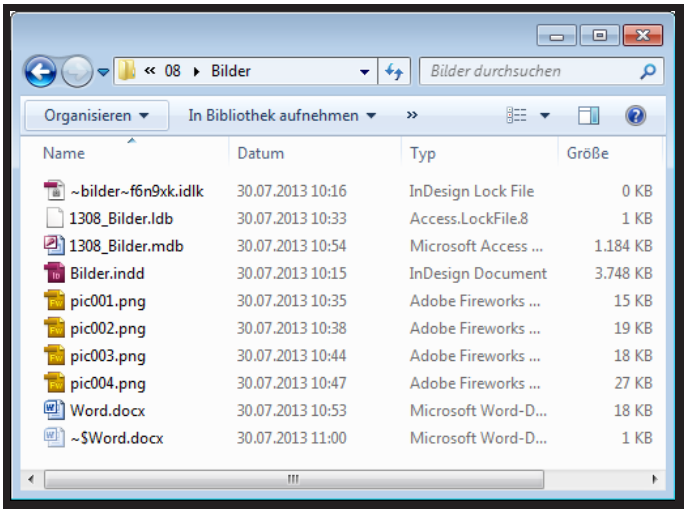


Bild 5: Temporäre Word-Datei

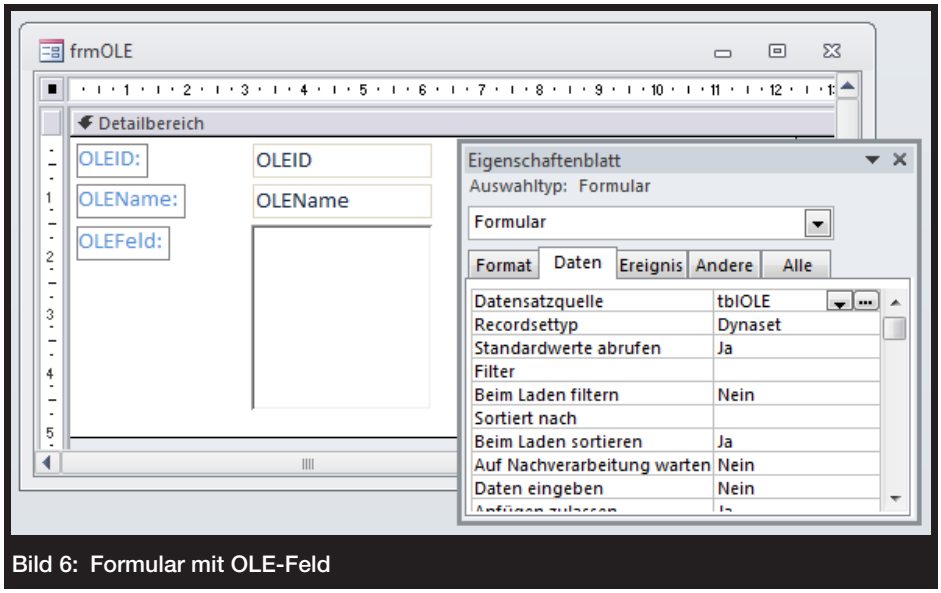


Bild 6: Formular mit OLE-Feld

# 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.

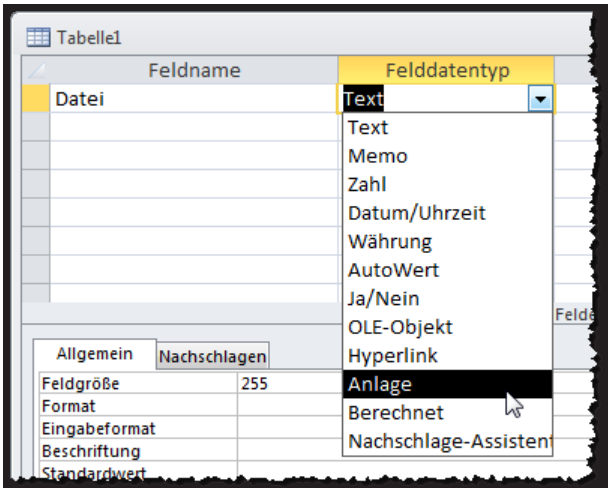


Bild 1: Anlegen eines neuen Anlage-Feldes

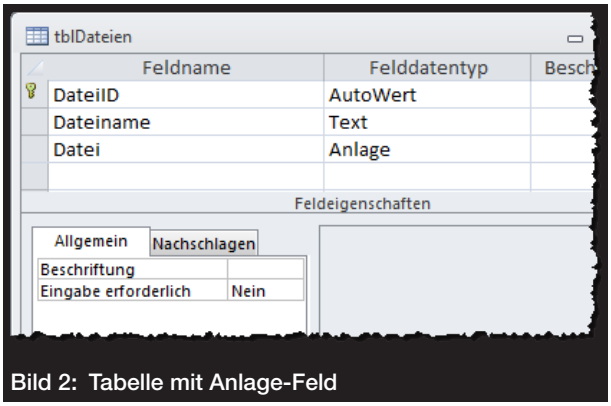


Bild 2: Tabelle mit Anlage-Feld

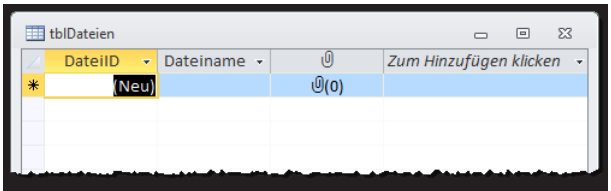


Bild 3: Das Anlage-Feld in der Datenblattansicht

## 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

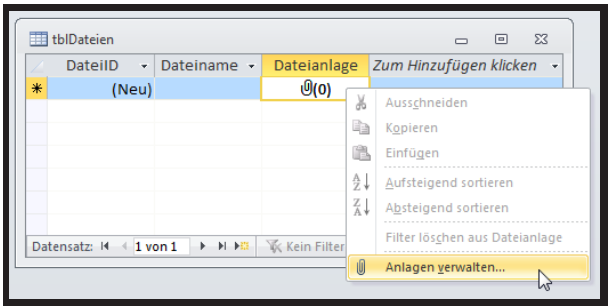



Bild 4: Kontextmenü-Eintrag zum Öffnen des Dialogs zum Verwalten der Anlagen

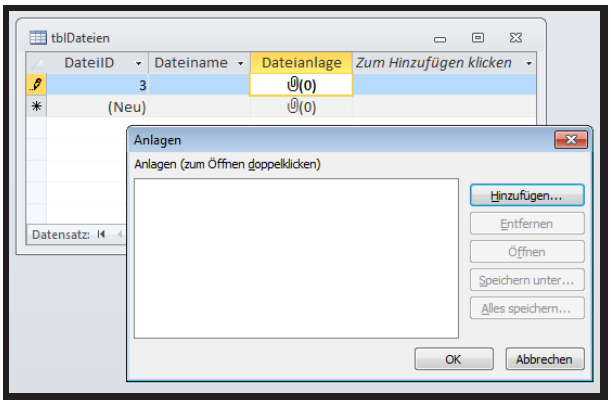



Bild 5: Dialog zum Verwalten von Anlagen

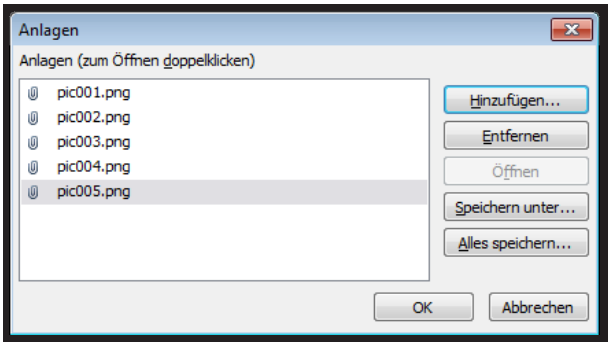



Bild 6: Anlagen-Dialog mit einigen Anlagen

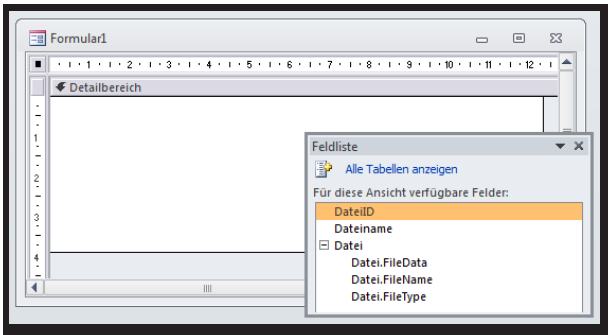



Bild 7: Hinzufügen eines Anlage-Felds zum Formularentwurf

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

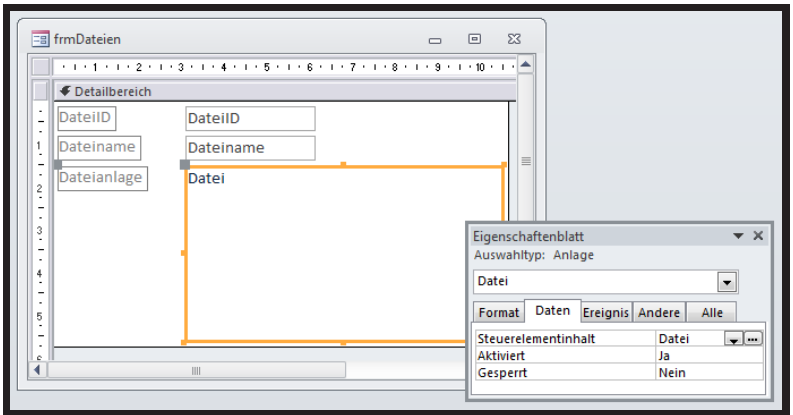
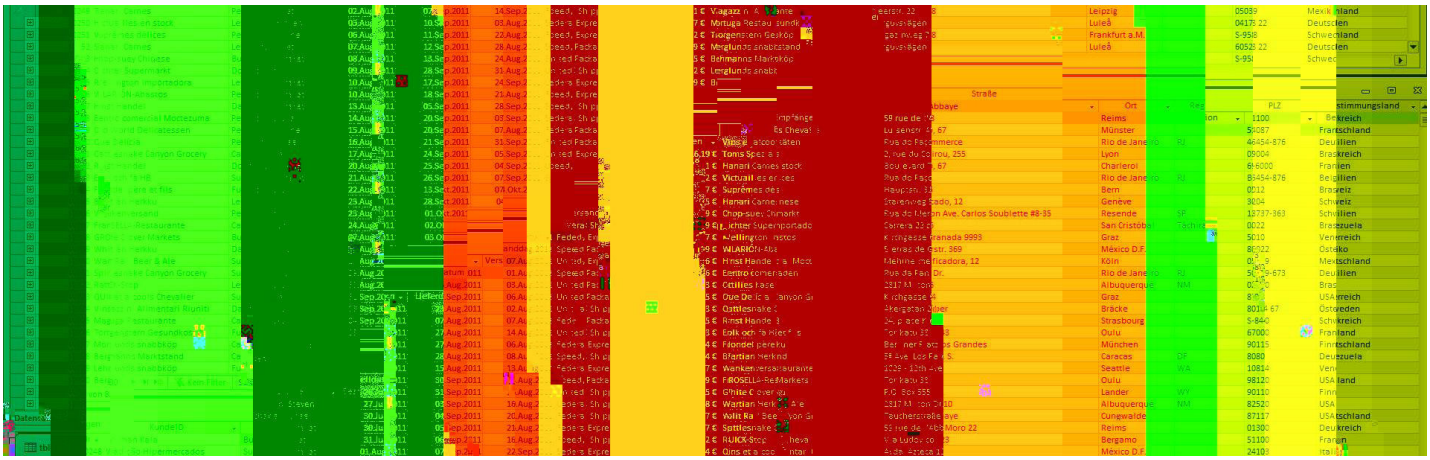



Bild 8: Das Steuerelement wird mit dem Typ Anlage ausgewiesen.

Ausgabe 09/2013

# ACCESS BASICS

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



## In diesem Heft:


### ANLAGEFELDER UND VBA

ERFAHREN SIE, WIE SIE BILDER UND ANDERE DATEIEN PER VBA IN ANLAGEFELDER EINFÜGEN UND DIESE AUCH WIEDER AUSLESEN – UND DAS AUCH NOCH MIT MEHREREN BILDERN GLEICHZEITIG.

SEITE 3

### TREEVIEW: VERKNÜPFTE DATEN, TEIL I

DAS TREEVIEW-STEUEELEMENT IST DAS MITTEL DER WAHL ZUR ANZEIGE VERKNÜPFTER DATEN. WIR ZEIGEN, WIE SIE DATEN AUS VERKNÜPFTEN TABELLEN IN DAS TREEVIEW LADEN.

SEITE 9

### BILDER IN ACCESS: BILDER IN OLE-FELDERN

WER BILDER IN TREEVIEWS, MENÜS UND RIBBONS ANZEIGEN WILL, MUSS DIESE ZUNÄCHST IN DER DATENBANK SPEICHERN. ERFAHREN SIE, WIE DIES MIT OLE-FELDERN GELINGT.

SEITE 19

# 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).

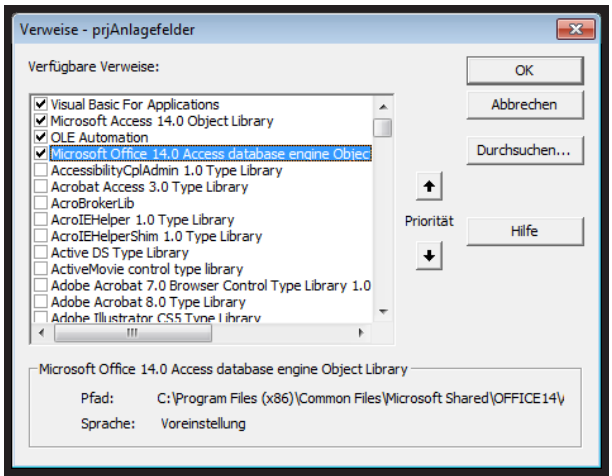


Bild 1: Verweis auf die neue DAO-Bibliothek

### 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

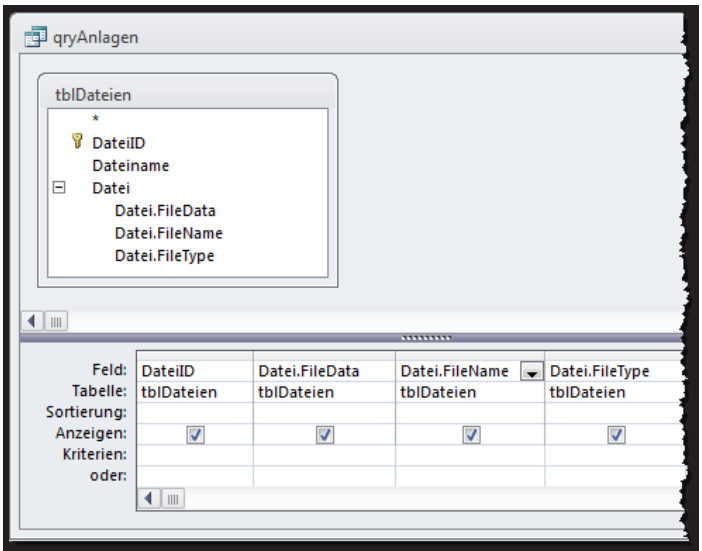


Bild 2: Das Anlage-Feld im Abfrageentwurf

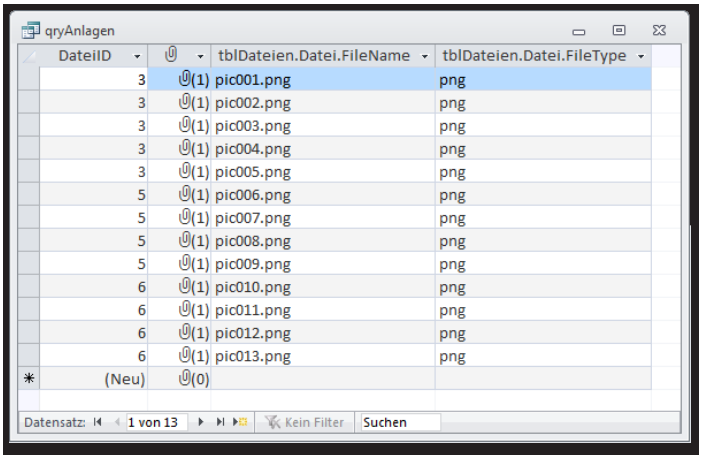


Bild 3: Abfrage, die jeden Datensatz entsprechend der Anzahl der enthaltenen Anlagen anzeigt.

```
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

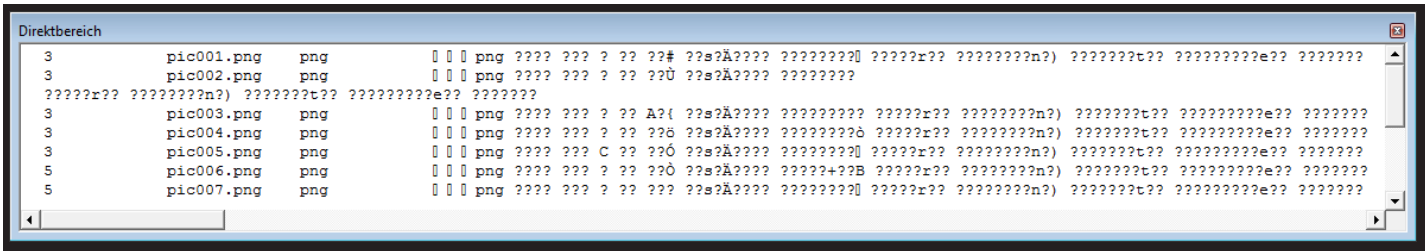



**Bild 4: Inhalt eines Anlagefeldes**

# 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.

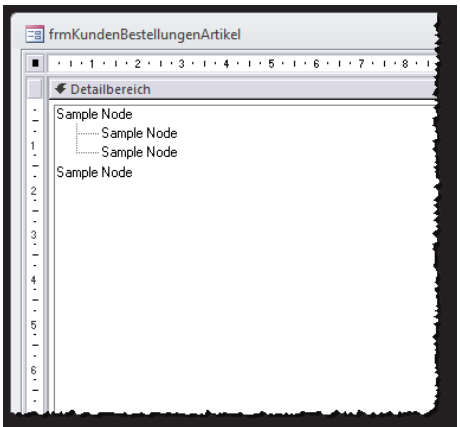



**Bild 2:** Formular mit dem frisch hinzugefügten TreeView-Steuerelement in der Entwurfsansicht

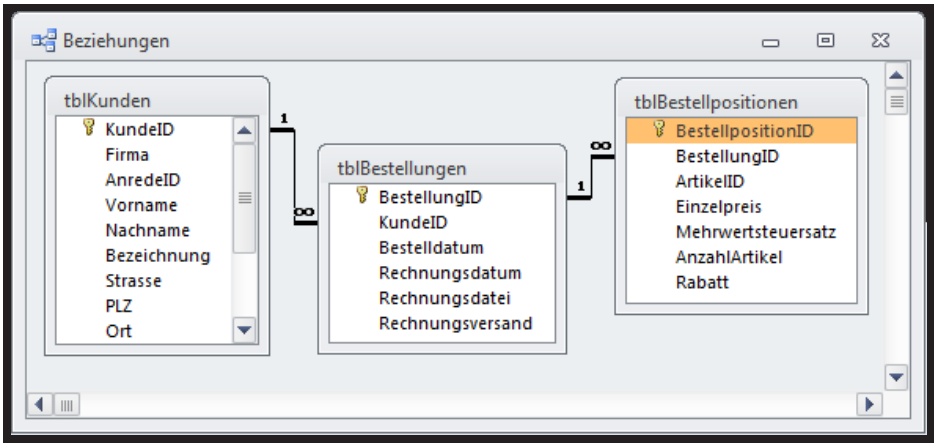



**Bild 1:** Datenherkunft für die Anzeige hierarchischer Daten im TreeView-Steuerelement

- 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.

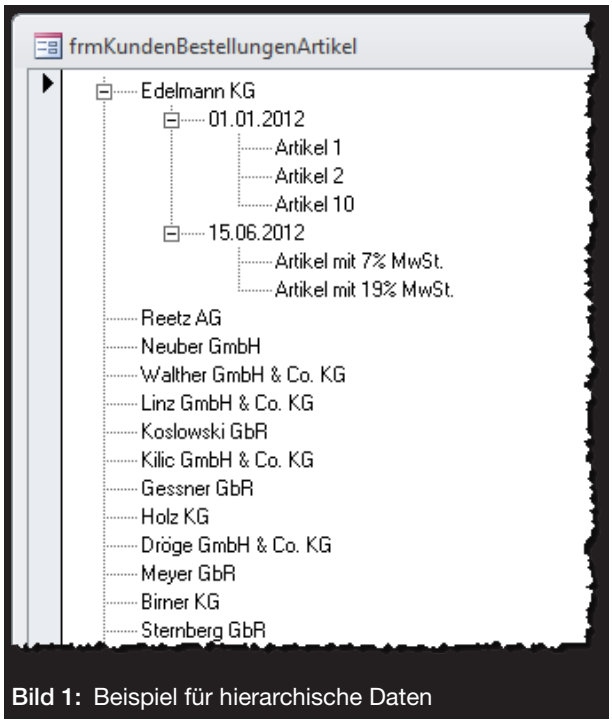



**Bild 1:** Beispiel für hierarchische Daten

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).

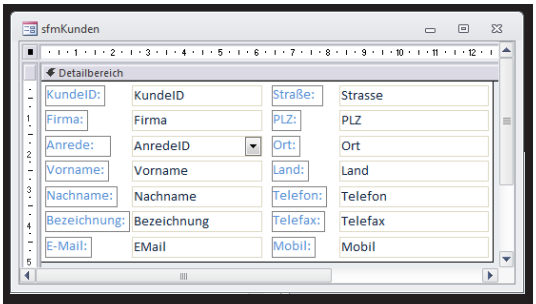



**Bild 2:** Das Unterformular **sfmKunden** in der Entwurfsansicht

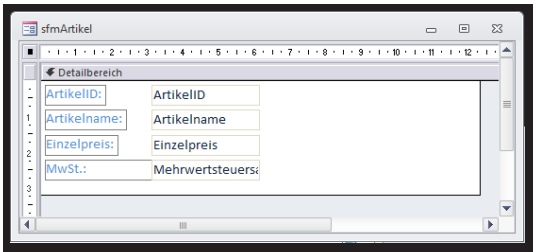



**Bild 3:** Unterformular zur Anzeige der Artikeldetails

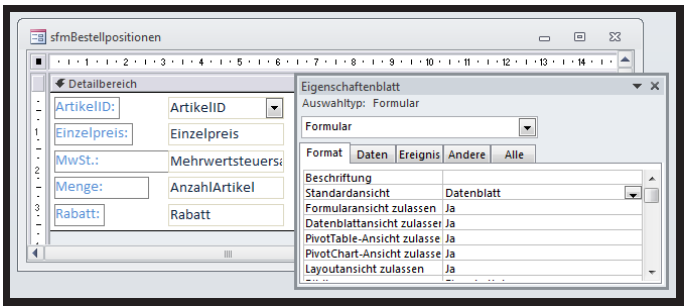



**Bild 4:** Das Unterformular **sfmBestelldetails** soll in der Datenblattansicht erscheinen.

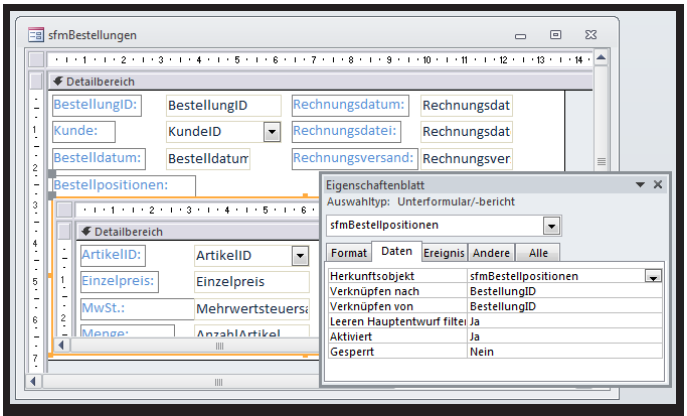



**Bild 5:** Verknüpfung zwischen Unterformular und übergeordnetem Hauptformular

## 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

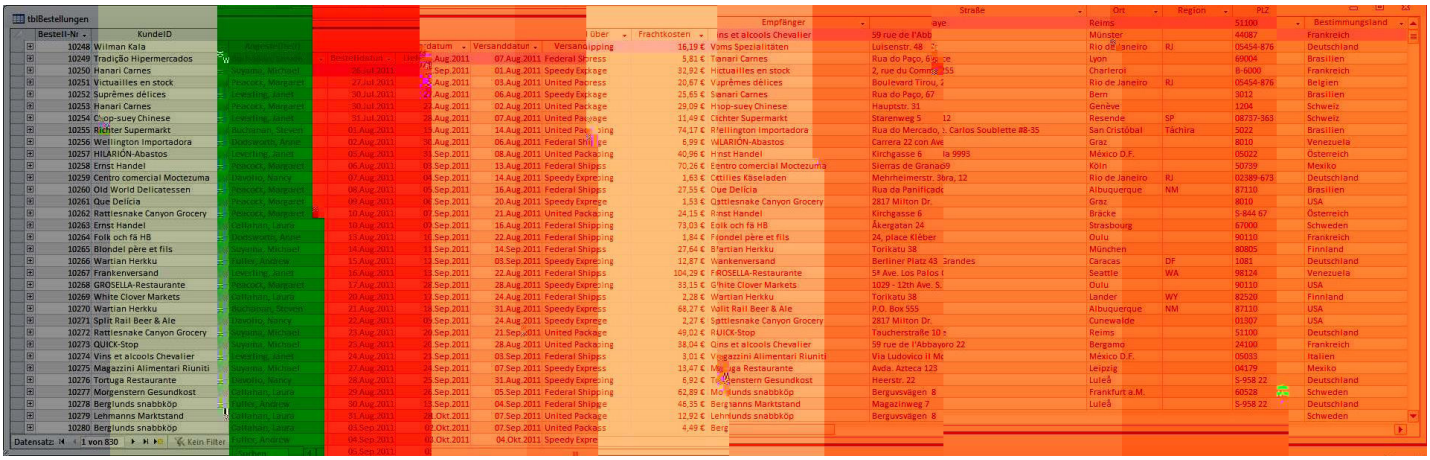
Nun haben wir ein TreeView-Steuerelement und ein leeres Unterformular-Steuerelement sowie eine Reihe von Unterformularen, die zu verschiedenen Zeitpunkten eingeblendet werden sollen. Damit dies geschieht, müssen wir für das TreeView-Steuerelement ein entsprechendes Ereignis anlegen.

Die Deklaration der Objektvariablen, welche das **TreeView**-Steuerelement

Ausgabe 10/2013

# ACCESS BASICS

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



## In diesem Heft:


### KONTEXTMENÜS

KONTEXTMENÜS SIND EINE EINFACHE MÖGLICHKEIT, BEFEHLE DIREKT IM KONTEXT EINES ELEMENTS AUFZURUFEN. LEIDER LASSEN SICH DIESE AB A2007 NUR NOCH PER VBA ERSTELLEN. WIR ZEIGEN, WIE ES FUNKTIONIERT.

SEITE 3

### TREEVIEW: VERKNÜPFTE DATEN, TEIL II

IM ZWEITEN TEIL DER ARTIKELREIHE ZEIGEN WIR, WIE SIE DIE DATEN AUS TABELLEN MIT REFLEXIVER BEZIEHUNG IM TREEVIEW-STEUERELEMENT ANZEIGEN.

SEITE 11

### DAS BILD-STEUERELEMENT

DAS BILD-STEUERELEMENT WURDE MIT ACCESS 2007 DEUTLICH AUFGEWERTET. WIR ZEIGEN, WIE SIE BILDER STATISCH UND DYNAMISCH IM BILD-STEUERELEMENT ANZEIGEN.

SEITE 17

# 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.

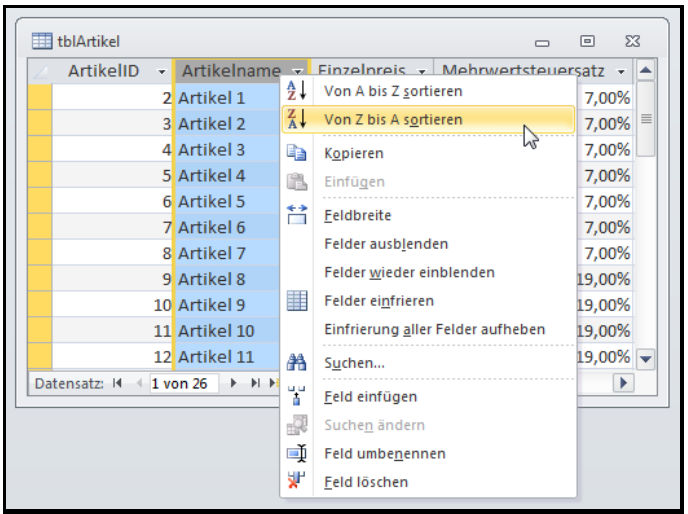


**Bild 1:** Beispiel für ein eingebautes Kontextmenü

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 _
            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-

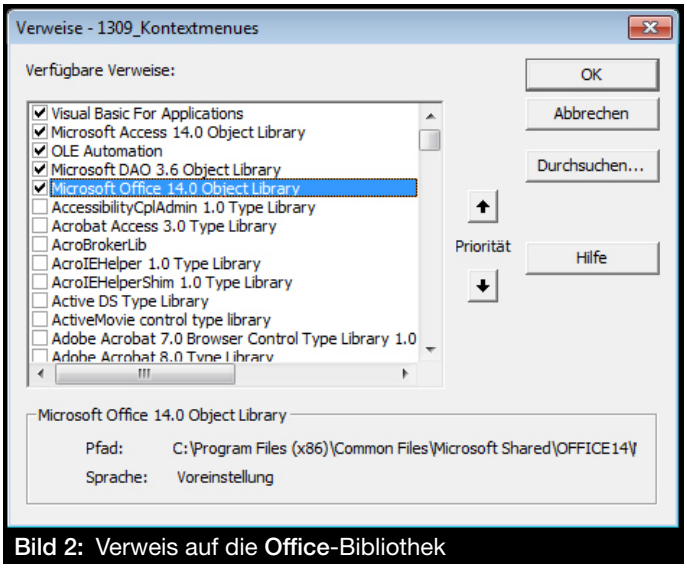



**Bild 2:** Verweis auf die **Office**-Bibliothek

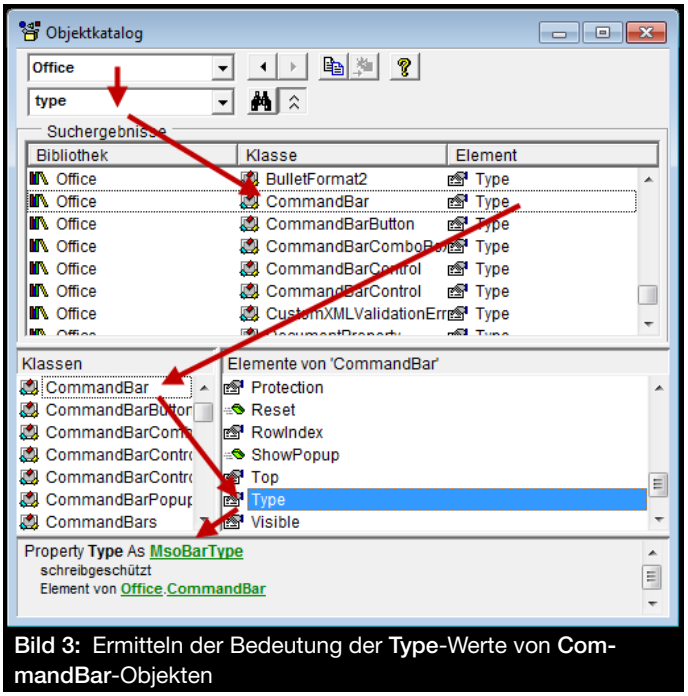



**Bild 3:** Ermitteln der Bedeutung der **Type**-Werte von **CommandBar**-Objekten

# 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.

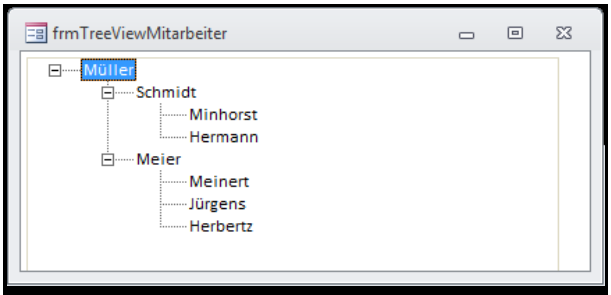



**Bild 1:** TreeView-Steuerelement zur Abbildung der Mitarbeiter-Hierarchie

## 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).

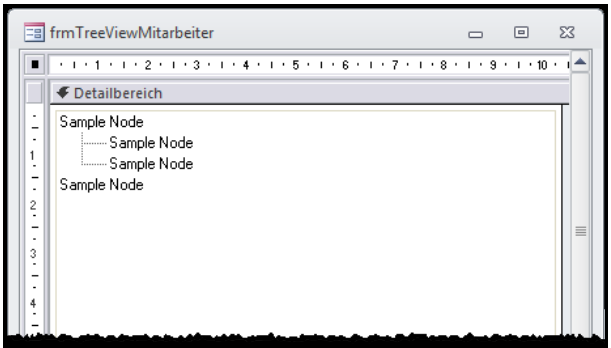



**Bild 2:** Basis für das zu füllende Formular mit **TreeView**-Steuerelement

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 MSComctlLib.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).

```
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.

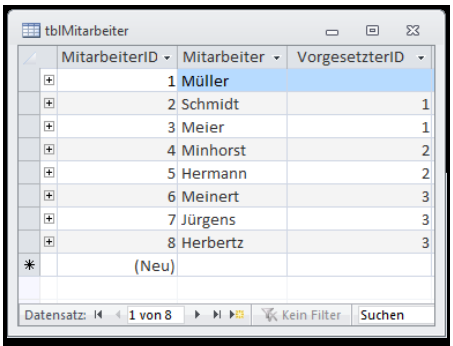



**Bild 3:** Die reflexiven Daten kommen aus einer einzigen Tabelle.

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**

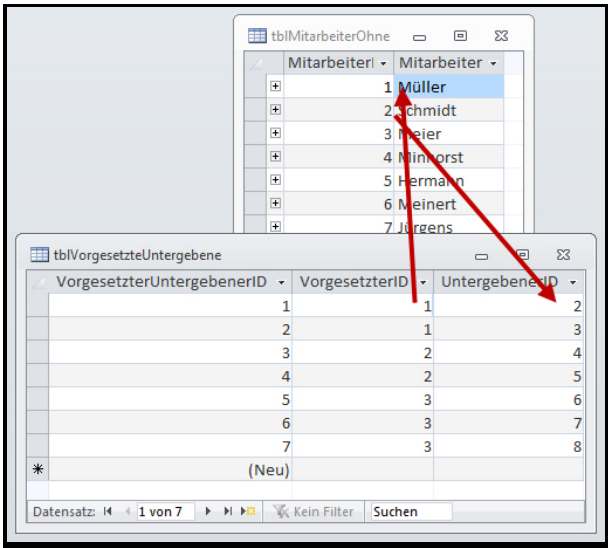



**Bild 4:** Die Beziehung kann aber auch über eine Verknüpfungstabelle hergestellt werden.

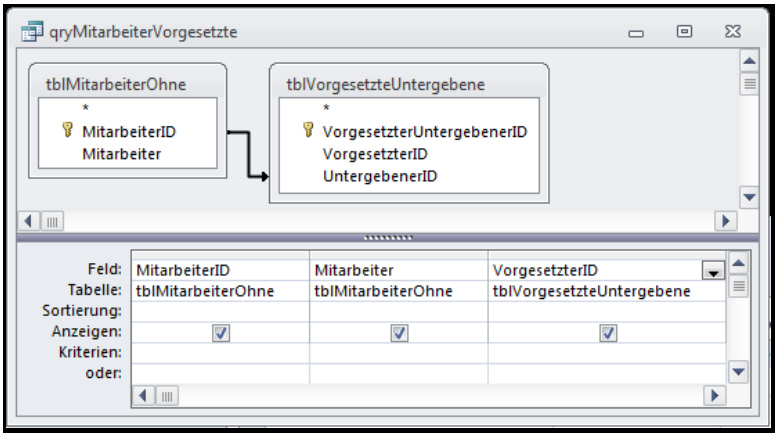



**Bild 5:** Im Falle der Verknüpfung führen wir die notwendigen Daten in einer Abfrage zusammen.

# 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).

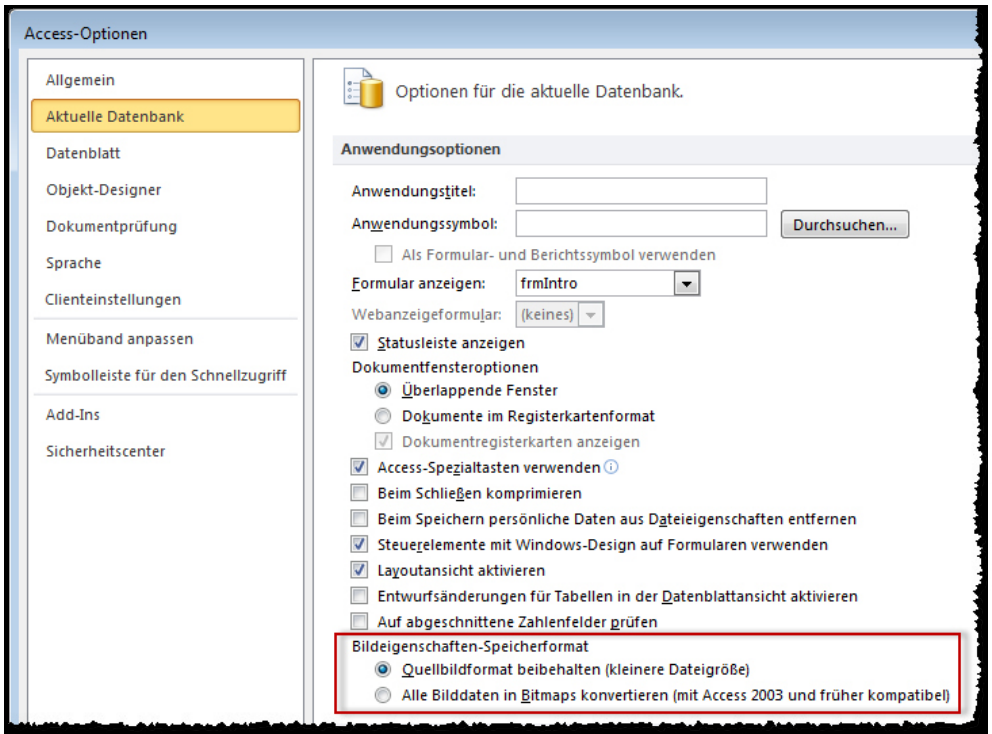



**Bild 1:** Einstellen des Speicherformats für Bilder

### 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**)

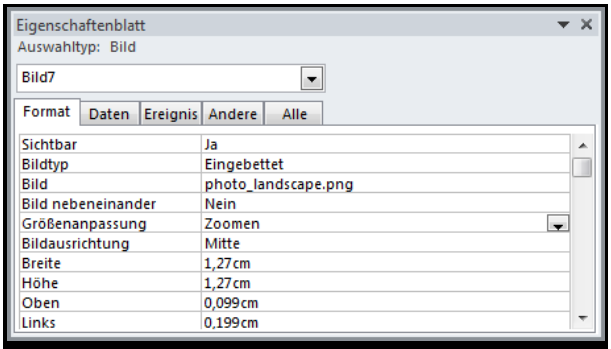


**Bild 3:** Eigenschaften des Bild-Steuerelements

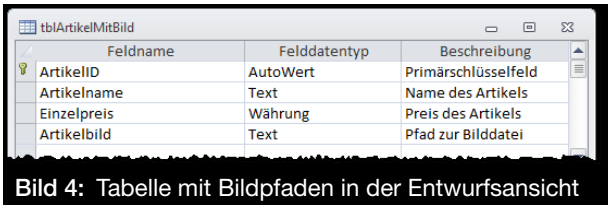


**Bild 4:** Tabelle mit Bildpfaden in der Entwurfsansicht

## 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

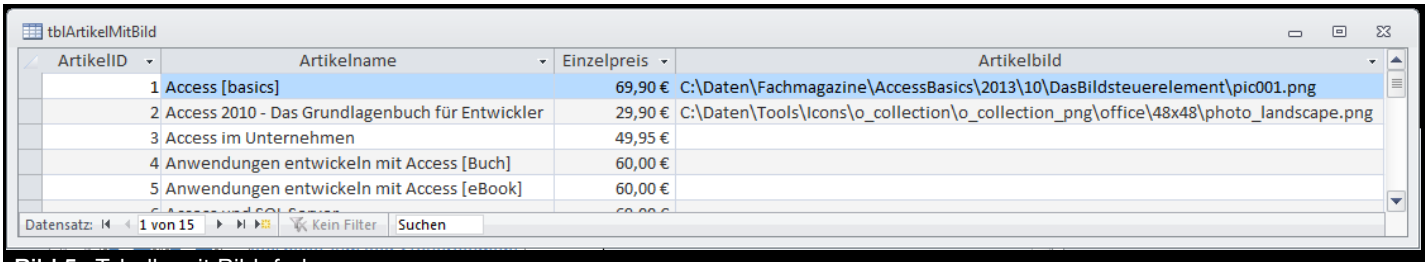
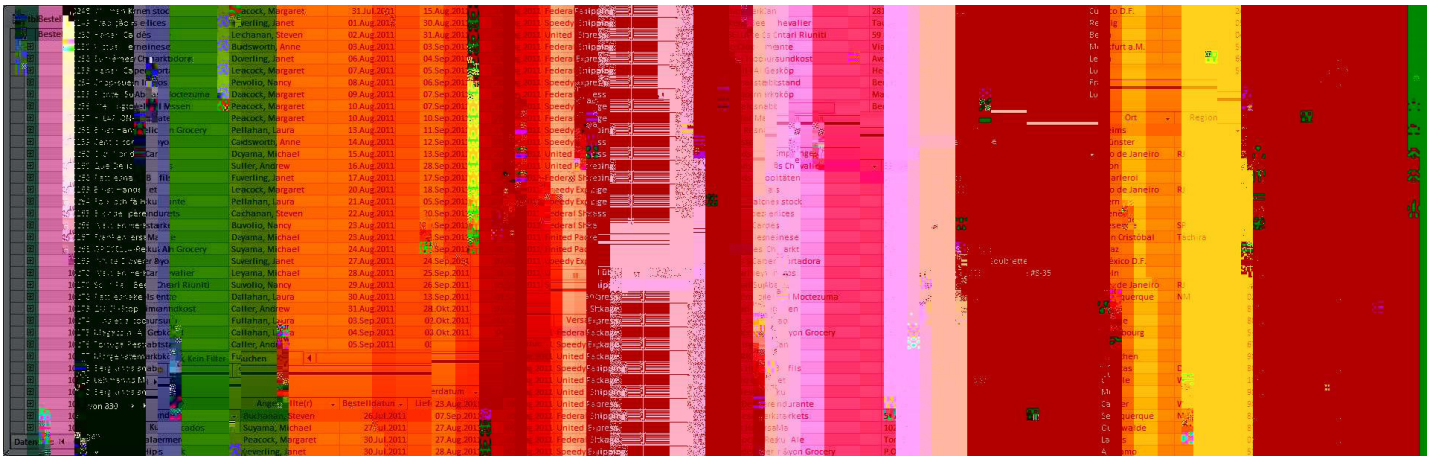


**Bild 5:** Tabelle mit Bildpfaden

Ausgabe 11/2013

# ACCESS BASICS

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



## In diesem Heft:


### BILDER IN ACCESS, TEIL III: BILDER AUS OLE-FELDERN ANZEIGEN

ERFAHREN SIE, WIE SIE BILDER AUS DEM OLE-FELD EINER TABELLE IM BILD-STEUERELEMENT VON ACCESS ANZEIGEN - AB ACCESS 2007

SEITE 3

### DAS IMAGELIST-STEUERELEMENT

WER TREEVIEW ODER LISTVIEW MIT ICONS VERSEHEN WILL, KOMMT UM DEN EINSATZ DES IMAGELIST-STEUERELEMENTS NICHT HERUM. DIESER ARTIKEL ZEIGT, WIE SIE DAS STEUERELEMENT MIT BILDERN FÜLLEN.

SEITE 6

### KONTEXTMENÜS IM TREEVIEW, TEIL I: ANLEGEN UND LÖSCHEN

KONTEXTMENÜS SIND EINE ERGONOMISCHE METHODE, UM MIT DEM TREEVIEW-STEUERELEMENT ZU ARBEITEN. LERNEN SIE DIE WICHTIGSTEN TECHNIKEN KENNEN!

SEITE 13

# 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.

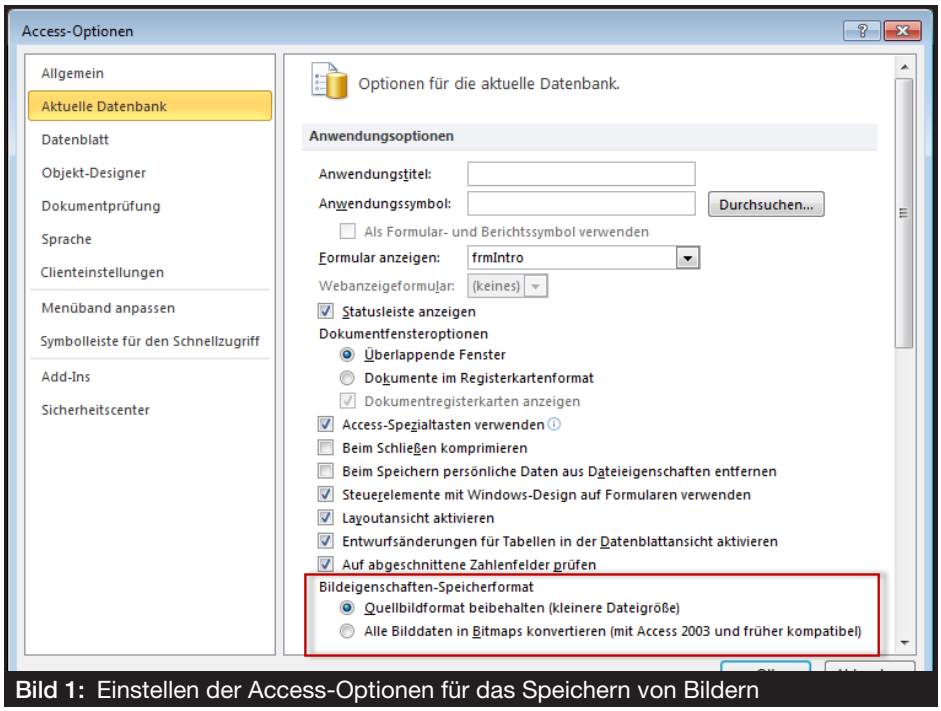



**Bild 1:** Einstellen der Access-Optionen für das Speichern von Bildern

## 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

- **mdlBLOBs0713**
- **mdlOLE**

Diese Module wurden von Sascha Trowitzsch programmiert und von mir an die Access-Version von 2013 angepasst.

## OLE-Bild in Bildsteuerelement anzeigen

Um ein Bild aus einem OLE-Feld in einem Bildsteuerelement anzuzeigen, laden Sie das Bild aus dem OLE-Feld zunächst in eine spezielle Variable und weisen diese dann dem Bildsteuerelement zu.

Für das folgende Beispiel wollen wir die Bilder aus dem OLE-Feld der Tabelle **tblOLEBilder** verwenden. Dazu binden Sie ein Formular an diese Tabelle und fügen diesem neben den beiden Feldern **OLEBildID** und **Bildname** noch ein Bildsteuerelement hinzu. Beim Hinzufügen zeigt Access automatisch einen **Grafik einfügen**-Dialog an, den Sie aber mit der **Escape**-Taste schließen können – das Bildsteuerelement wird dann zunächst ohne Inhalt angelegt. Legen Sie für das Bildsteuerelement den Namen **ctlOLE** fest.

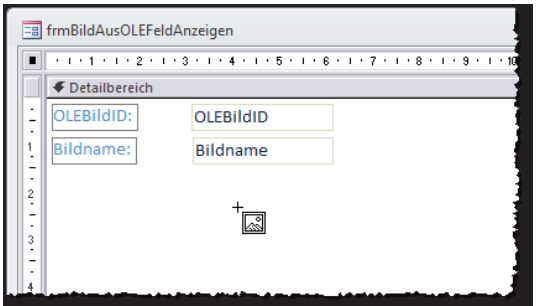



**Bild 2:** Hinzufügen eines Bildsteuerelements zu einem Formular

Damit das Formular beim Anzeigen eines Datensatzes auch gleich das im OLE-Feld gespeicherte Bild anzeigt, fügen Sie der Ereignisprozedur, die durch das Ereignis **Beim Anzeigen** ausgelöst wird, die Zeilen aus Listing 1 hinzu. Diese Prozedur deklariert und füllt ein **Database**-Objekt namens **db**. Die folgende Anweisung übernimmt gleich den Job, das Bild aus dem OLE-Feld im Bildsteuerelement anzuzeigen. Dazu ruft sie die Funktion **OLE2Binary** auf, die folgende Parameter erwartet:

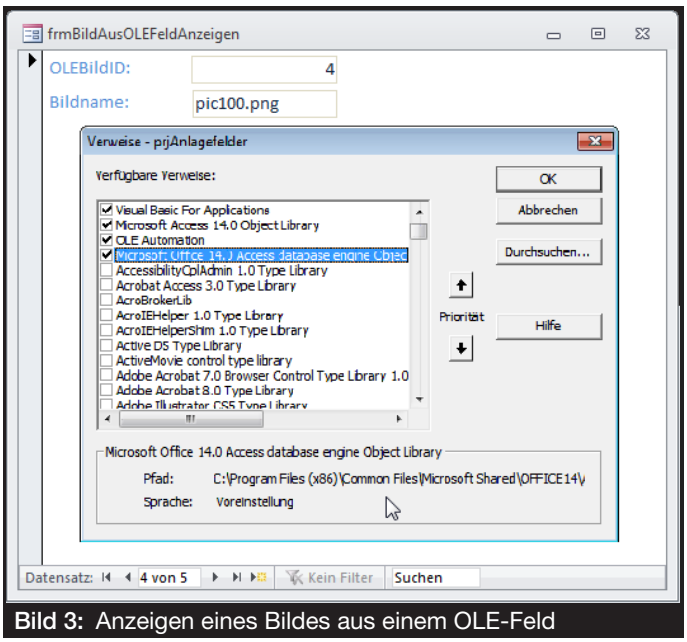



**Bild 3:** Anzeigen eines Bildes aus einem OLE-Feld

- **dbs**: Verweis auf die Datenbank
- **strTable**: Name der Tabelle mit dem OLE-Feld
- **strIDField**: Name des Primärschlüsselwertes
- **lngID**: Primärschlüsselwert des Datensatzes mit dem anzuzeigenden OLE-Feldinhalt
- **strOLEField**: Name des OLE-Feldes

Das Ergebnis der Funktion weisen Sie direkt der Eigenschaft **PictureData** des Bildsteuerelements zu. Dieses zeigt dann das gewünschte Bild an.

## Bilder in Schaltflächen

Wenn Sie Schaltflächen mit Bildern versehen wollen, werden Sie typischerweise die dazu vorgesehenen Bilder direkt mit den Mitteln der

```
Private Sub Form_Current()
    Dim db As DAO.Database
    Set db = CurrentDb
    Me!ctlOLE.PictureData = mdlOLE.OLE2Binary(db, "tblOLEBilder", "OLEBildID", Me!OLEBildID, "OLEBild")
End Sub
```

**Listing 1:** Anzeigen des Inhalts eines OLE-Feldes im Bildsteuerelement

# 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).

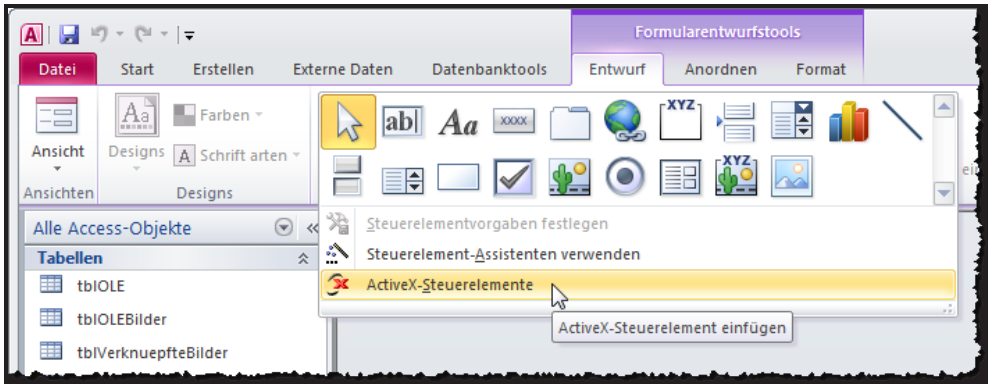



**Bild 1:** Öffnen des Dialogs zum Hinzufügen von ActiveX-Steuerelementen

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

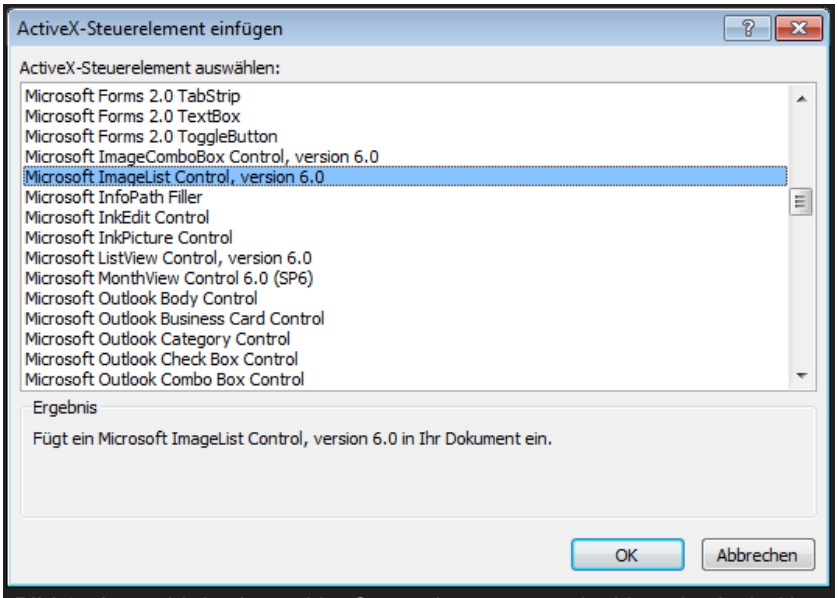



**Bild 2:** Auswahl des ImageList-Steuerelements aus der Liste der ActiveX-Steuerelemente

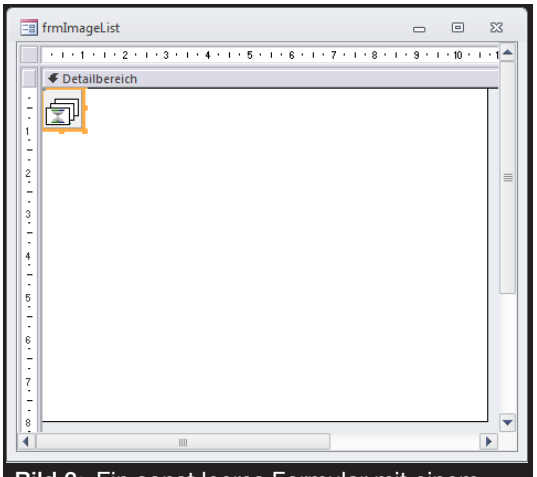



**Bild 3:** Ein sonst leeres Formular mit einem ImageList-Steuerelement

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**.

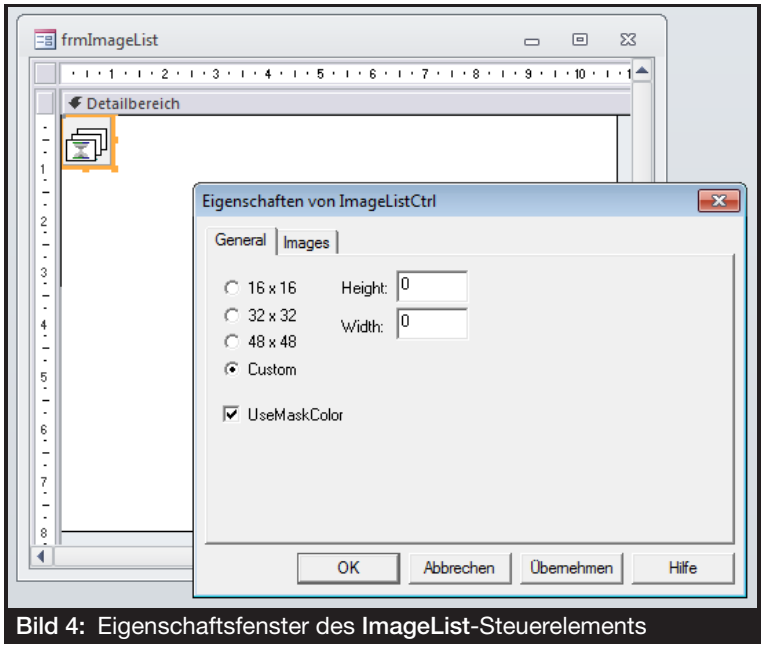



**Bild 4:** Eigenschaftsfenster des **ImageList**-Steuerelements

## 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.

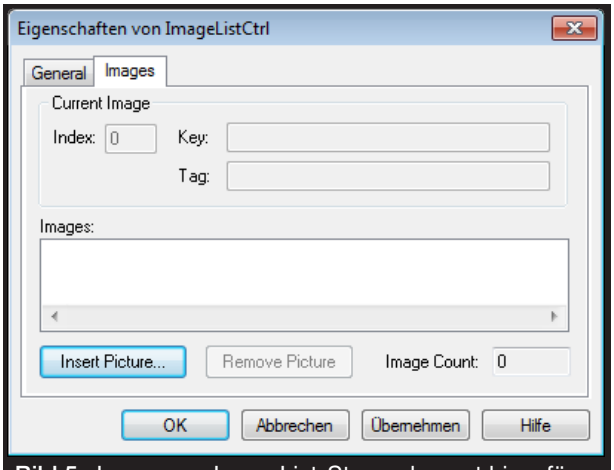



**Bild 5:** Icons zum ImageList-Steuerelement hinzufügen

# 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.

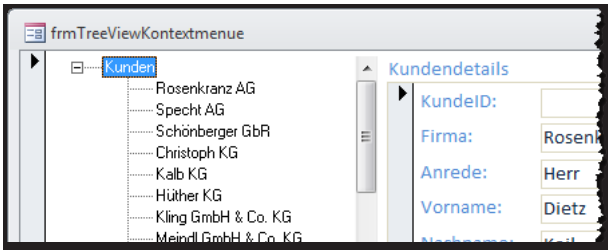



**Bild 1:** Hinzufügen eines Root-Knotens zum **TreeView**-Steuerelement

## 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).

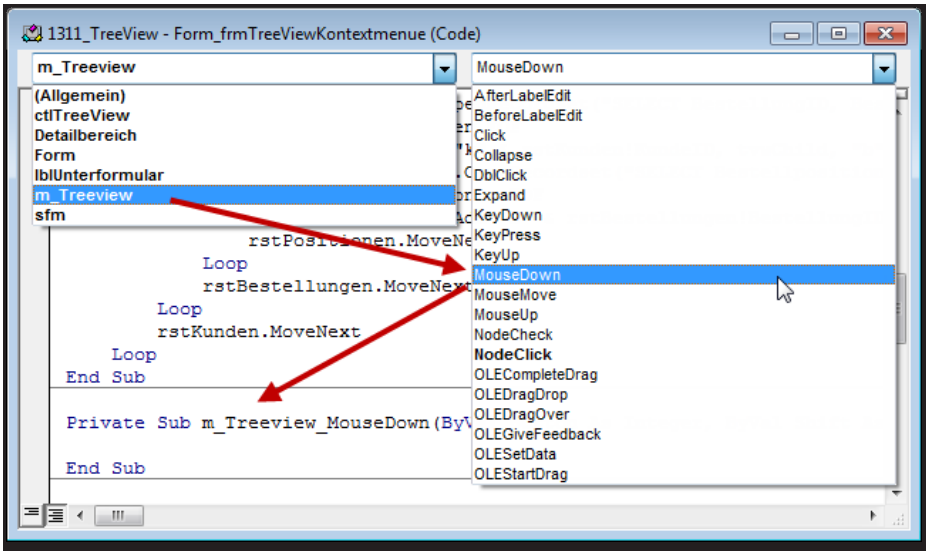


**Bild 2:** Anlegen der Ereignisprozedur MouseDown

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:

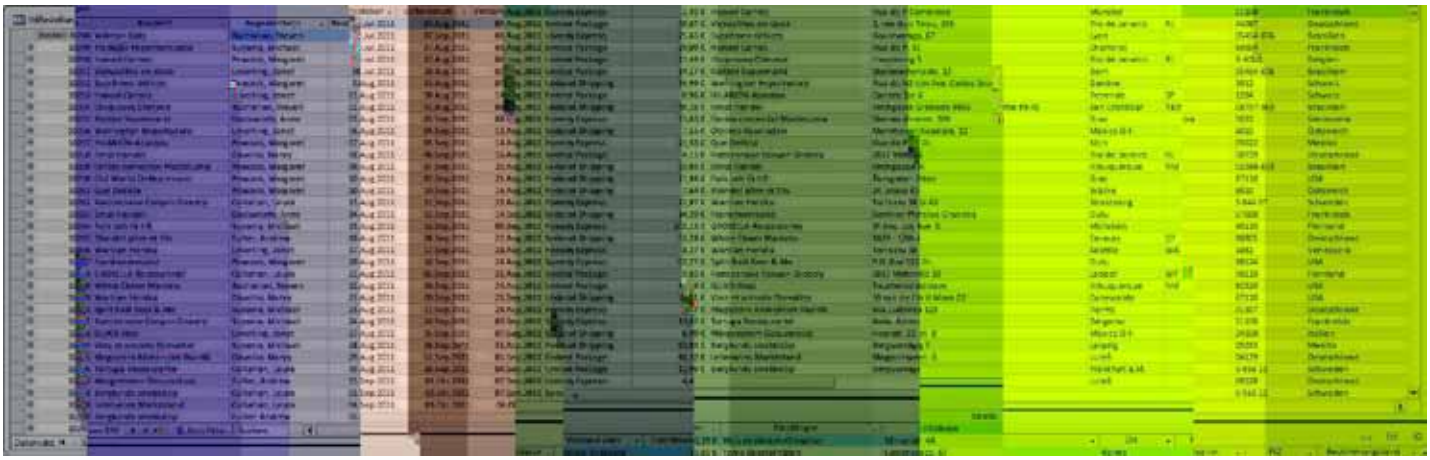
```
Set objNode = m_Treeview.HitTest(x, y)
```

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:

Ausgabe 12/2013

# ACCESS BASICS

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



## In diesem Heft:


### STEUERELEMENTE: DAS KONTROLLKÄSTCHEN

DAS KONTROLLKÄSTCHEN EIGNET SICH PERFEKT ZUR DARSTELLUNG VON JA/NEIN-FELDERN. DIESER ARTIKEL STELLT DIESES STEUERELEMENT AUSFÜHRLICH VOR.

SEITE 3

### ANWENDUNGSPARTS

WER TABELLEN, FORMULARE ODER MODULE IMMER WIEDER NUTZT, SOLLTE AB ACCESS 2010 ANWENDUNGSPARTS DARAUS ERSTELLEN. DIESE FÜGEN SIE AUF EINFACHSTE WEISE IN DIE AKTUELLE DATENBANK EIN.

SEITE 7

### AUTOWERTE

AUTOWERTE HELFEN BEIM EINDEUTIGEN KENNZEICHNEN VON DATENSÄTZEN. ERFAHREN SIE, WELCHE TÜCKEN SIE DENNOCH AUFWEISEN UND WIE SIE AUTOWERTE RICHTIG EINSETZEN.

SEITE 13

# 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

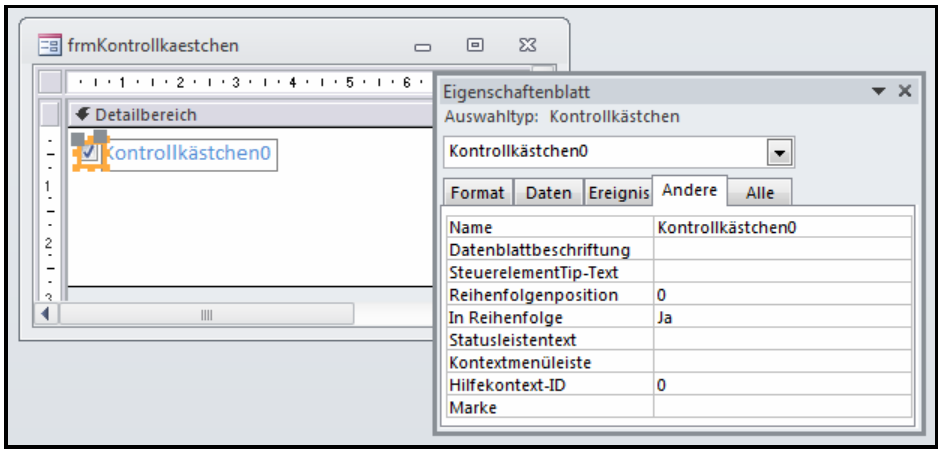



**Bild 1:** Hinzufügen eines ersten Kontrollkästchens

den meisten übrigen Steuerelementen platziert Access das Bezeichnungsfeld rechts neben dem eigentlichen Steuerelement und nicht links daneben.

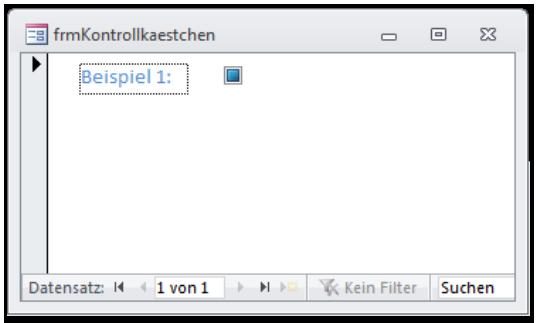



**Bild 2:** Kontrollkästchen ohne Wert

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**.

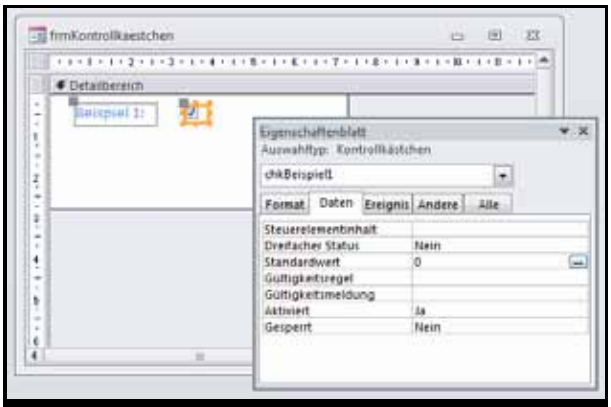


**Bild 4:** Einstellen des Standardwerts

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.

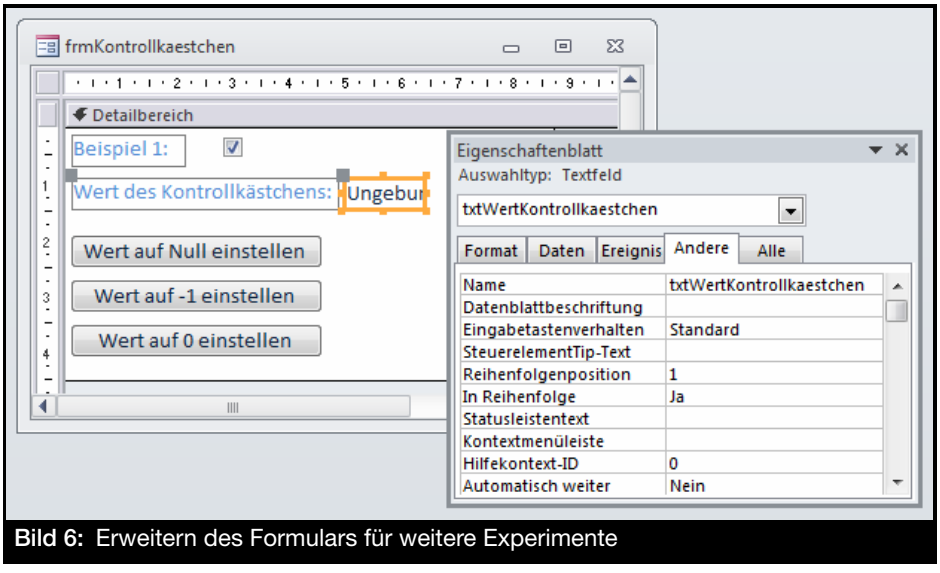


**Bild 6:** Erweitern des Formulars für weitere Experimente

### 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 enthalten bereits Beispiele für benutzerdefinierte, also selbst erstellte Anwendungsparts.

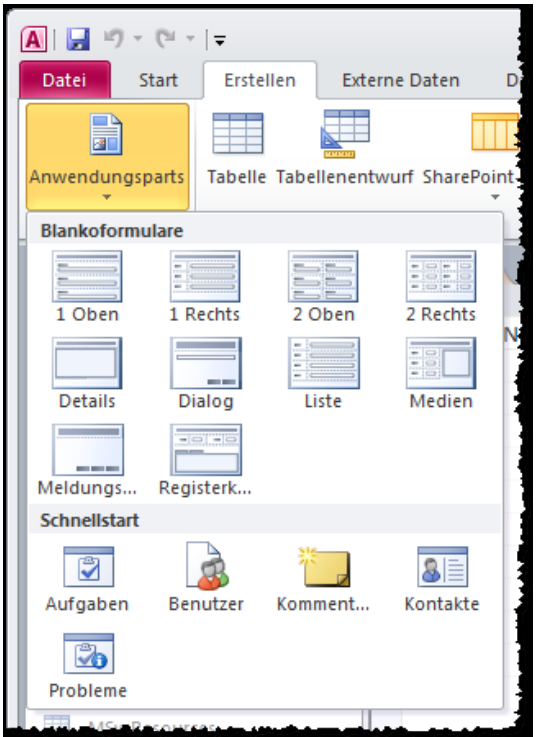



**Bild 1:** Auswahl vorgefertigter Anwendungsparts

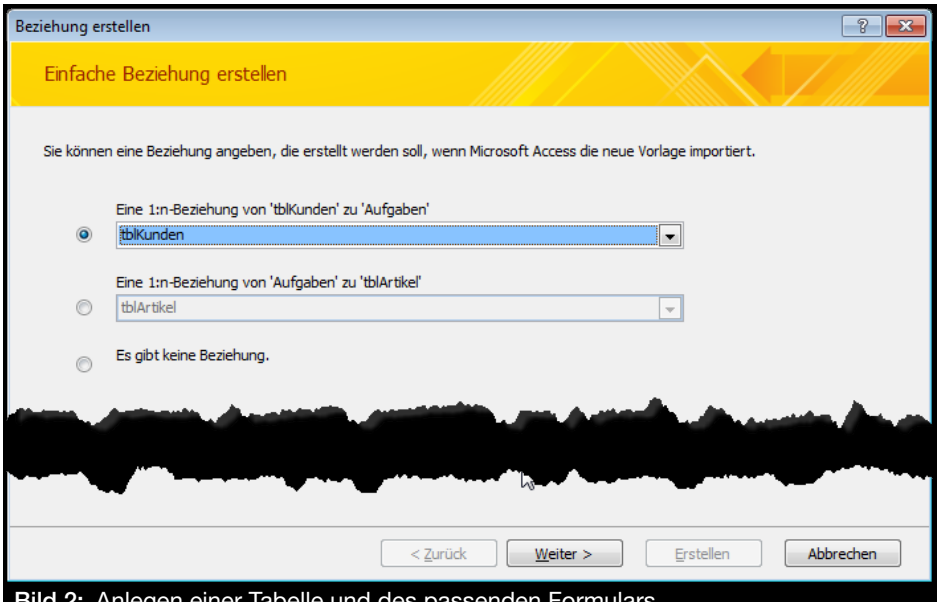



**Bild 2:** Anlegen einer Tabelle und des passenden Formulars

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

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:

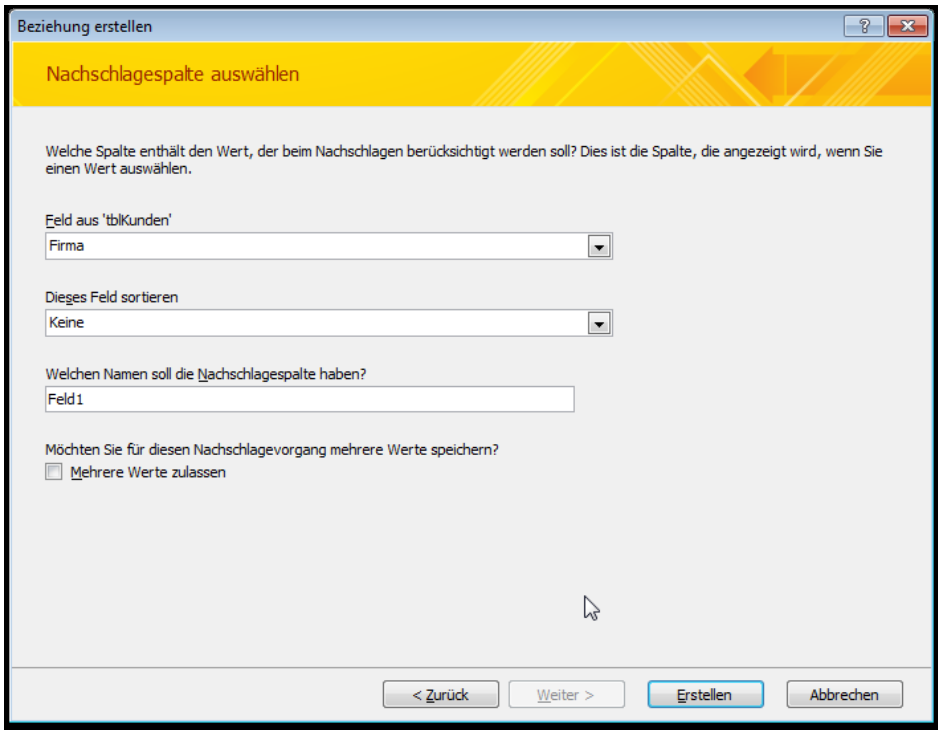



**Bild 3:** Erstellen einer Beziehung zwischen der neuen Tabelle der Tabelle **tblKunden**

- 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.

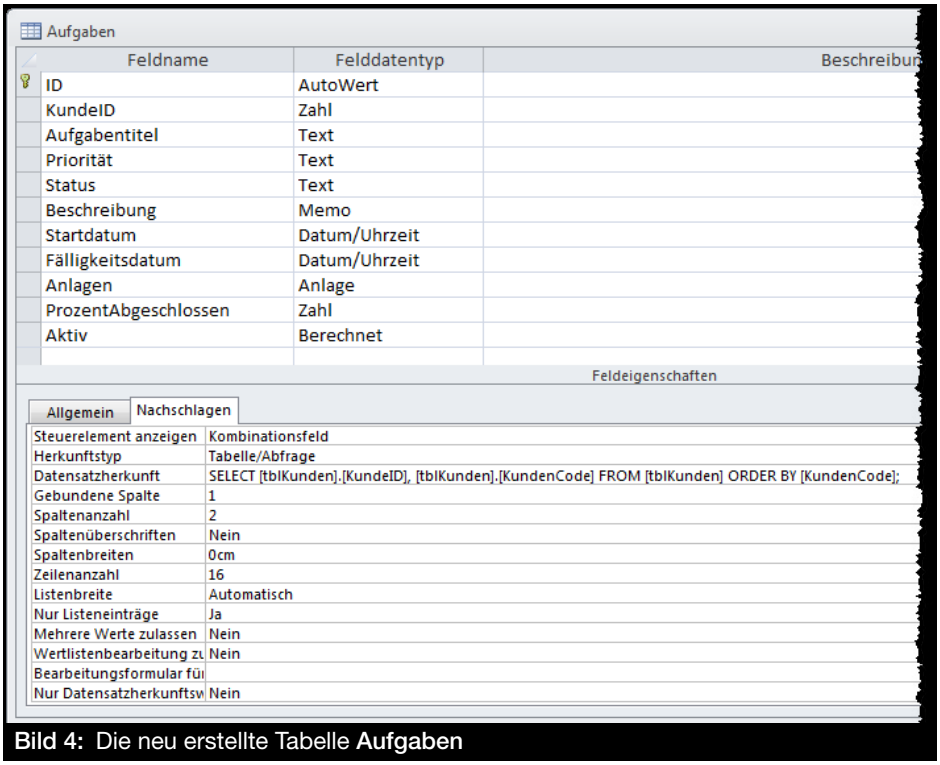



**Bild 4:** Die neu erstellte Tabelle **Aufgaben**

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,

# 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).

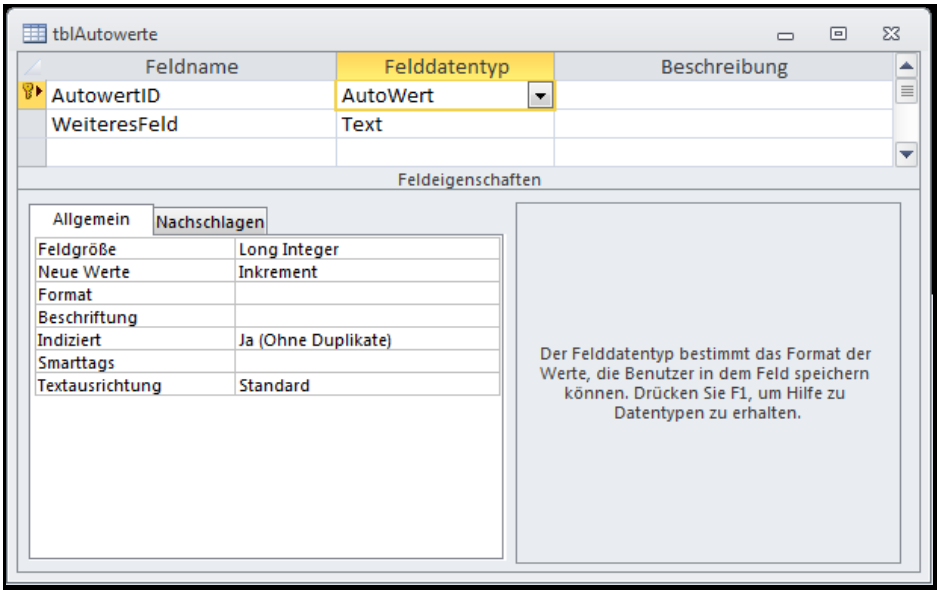



**Bild 1:** Anlegen einer Tabelle mit einem Autowert

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).

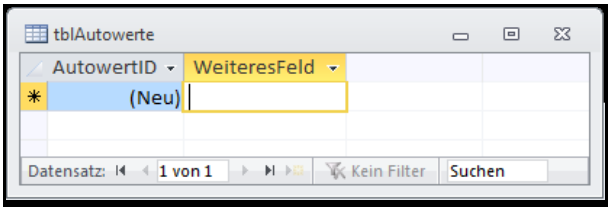



**Bild 2:** Das Autowert-Feld zeigt den Eintrag **(Neu)** an.

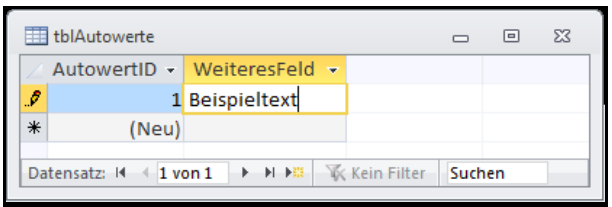



**Bild 3:** Er wird eingefügt, sobald der Datensatz geändert wurde.

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

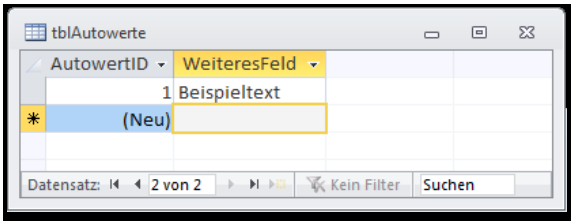



**Bild 4:** Weitere Datensätze erhalten den bisher größten Wert plus eins als Autowert.

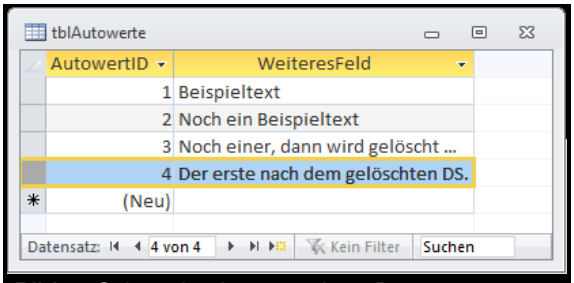



**Bild 5:** Schön durchnummerierte Datensätze

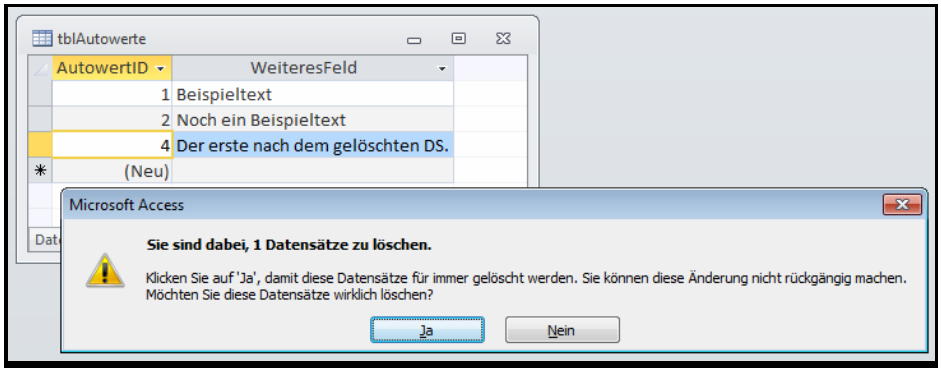



**Bild 6:** Aber wehe, ein Datensatz wird gelöscht!

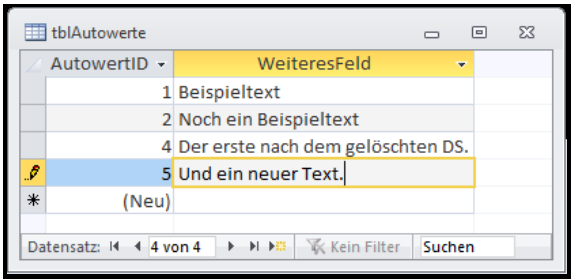



**Bild 7:** Die entstandene Lücke wird durch einen neuen Datensatz nicht geschlossen.

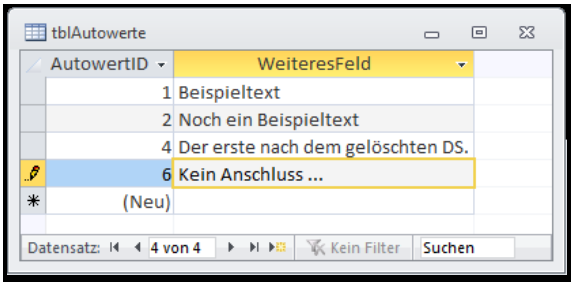



**Bild 8:** Nach dem Löschen des letzten Datensatzes wird dessen Autowert nicht neu vergeben

# 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, ...

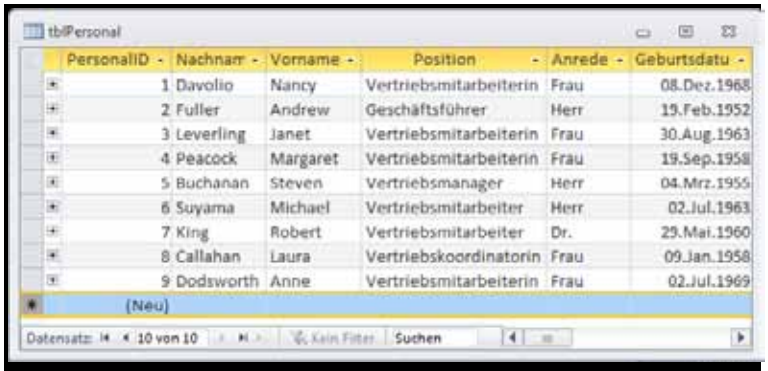


**Bild 2:** ... Markieren des Ziels, also eines leeren Datensatzes ...

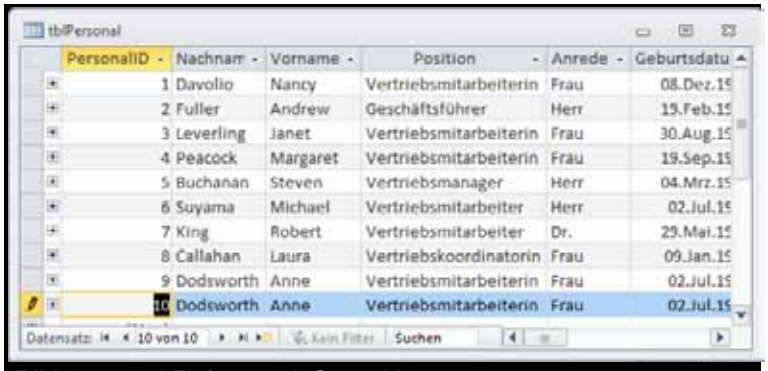


**Bild 3:** ... und Einfügen mit Strg + V.

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, ...

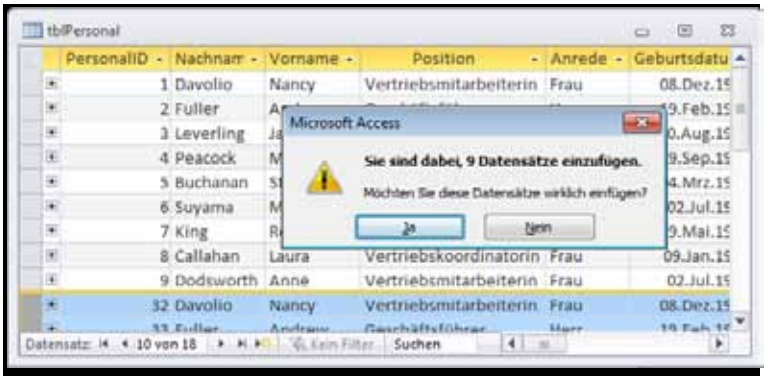


**Bild 5:** ... die allerdings nicht ohne vorherige Rückfrage kopiert werden.

## 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.

Ausgabe 1/2014

# ACCESS BASICS

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



## In diesem Heft:


### TABELLEN ENTWERFEN: GUIDS

NEBEN DEN NUMERISCHEN AUTOWERTEN KÖNNEN SIE AUCH GUIDS VERWENDEN. WAS DAS IST UND WIE SIE DAMIT ARBEITEN, ERFAHREN SIE IN DIESER AUSGABE!

SEITE 6

### DATENSÄTZE KOPIEREN PER VBA

LERNEN SIE, WIE SIE DATENSÄTZE PER VBA KOPIEREN. DIE TECHNIKEN LASSEN SICH AUF DATENSÄTZE IN FORMULAREN, ABER AUCH INNERHALB VON PROZEDUREN ANWENDEN.

SEITE 10

### VERKNÜPFTE DATENSÄTZE KOPIEREN

EINE NEUE RECHNUNG MIT INDIVIDUELLEN POSITIONEN ERSTELLEN, DIE ES IN ÄHNLICHER FORM SCHON EINMAL GAB? SCHNELLER ALS MIT KOPIEREN UND ANPASSEN GELINGT DIES NICHT!

SEITE 15

# 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.

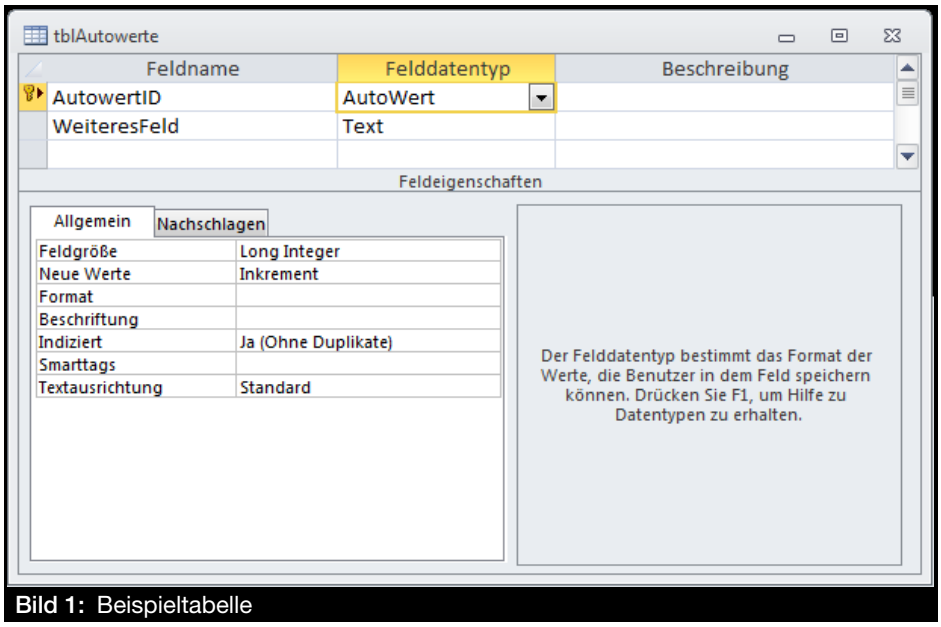



**Bild 1:** Beispieltabelle

## 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

um den Autowert des neuen Datensatzes auszugeben. Nach dem Update können Sie nicht mehr auf den Autowert zugreifen, da dies den Datensatzzeiger direkt auf die folgende Position verschiebt. Genaugenommen liefert dies je nach der gewählten Datenherkunft unterschiedliche Werte.

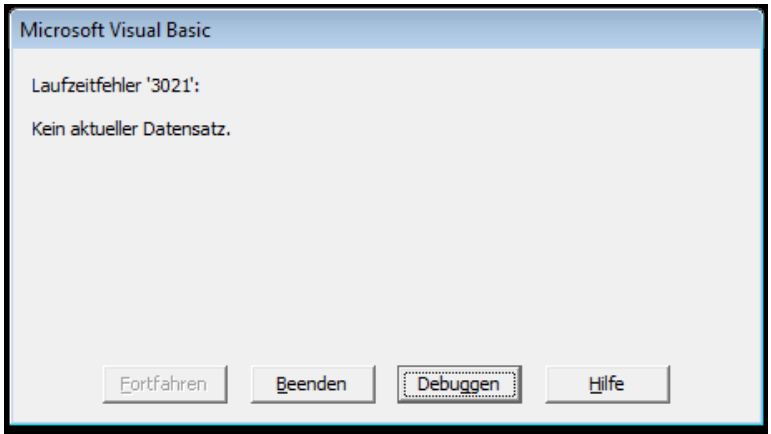



**Bild 2:** Fehler, der beim Abfragen eines Autowertes auftreten kann

Wir gehen also zunächst davon aus, dass wir den Autowert hinter dem Aufruf der **Update**-Methode abfragen – also etwa so:

```
rst.AddNew
rst!WeiteresFeld = "Test"
rst.Update
MsgBox "Die neue ID lautet: " _
    & rst!AutowertId
```

Bei der ersten Variante verwenden wir ein Recordset, das alle Datensätze der zugrunde liegenden Tabelle enthält – also das aus dem bereits referenzierten Listing:

```
Set rst = db.OpenRecordset(_
        "tblAutowerte", dbOpenDynaset)
```

Das Ergebnis im Meldungsfenster lautet in diesem Fall **-3**! Dies ist auf keinen Fall der korrekte Wert – so viel wissen wir. Aber warum liefert Access hier den Wert **-3**? Diese Frage lassen wir guten Gewissens offen, denn wir wissen ja, dass wir damit auf jeden Fall falsch liegen und den neuen Autowert vor dem Auslösen der **Update**-Methode ermitteln müssen.

Wenn wir als zweite Variante das Recordset so einschränken wollen, dass es keine Datensätze enthält, definieren wir die Datenherkunft in **OpenRecordset** so:

```
Set rst = db.OpenRecordset(_
         "SELECT * FROM tblAutowerte _
          WHERE 1 = 2", dbOpenDynaset)
```

Wenn wir nun die Update-Methode aufrufen und erst danach auf den neuen Autowert zugreifen wollen, liefert Access eine handfeste Fehlermeldung (siehe Bild 2). Diesen Fehler muss der Einsteiger allerdings erst noch interpretieren lernen: Dadurch, dass wir eine leere Datensatzgruppe mit einem Datensatz füllen und dann durch den Aufruf der **Update**-Methode zum folgenden Datensatz wechseln (der ja nicht vorhanden ist), landet der Datensatzzeiger hinter dem letzten Datensatz. Der Zugriff auf einen nicht vorhandenen Datensatz führt schließlich zum Fehler.

### Datenbankunabhängigkeit

Diese Methode ist grundsätzlich unabhängig von eventuell in anderen Datenbankfrontends ausgeführten Anfügevorgängen von Datensätzen in die betroffene Tabelle. Der Datensatzzeiger verbleibt hier auf dem neuen Datensatz, während dessen Autowert ermittelt wird.

### ID bei INSERT INTO ermitteln

Bei der Verwendung einer **INSERT INTO**-Abfrage verwenden Sie die

```
Public Sub AutowertMitExecuteInsert()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim strSQL As String
    Set db = CurrentDb
    'Fügt einen neuen Datensatz zu einer Tabelle hinzu
    strSQL = "INSERT INTO tblAutowerte(WeiteresFeld) VALUES('Test')"
    db.Execute strSQL, dbFailOnError
    If db.RecordsAffected > 0 Then
        'Öffnet eine Datensatzgruppe, die den Wert der Funktion @@IDENTITY zurückliefert
        Set rst = db.OpenRecordset("SELECT @@IDENTITY")
        'Gibt den Wert von @@IDENTITY aus
        Debug.Print rst.Fields(0)
        rst.Close
        Set rst = Nothing
    End If
    Set db = Nothing
End Sub
```

**Listing 2:** Autowert per @@IDENTITY 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).

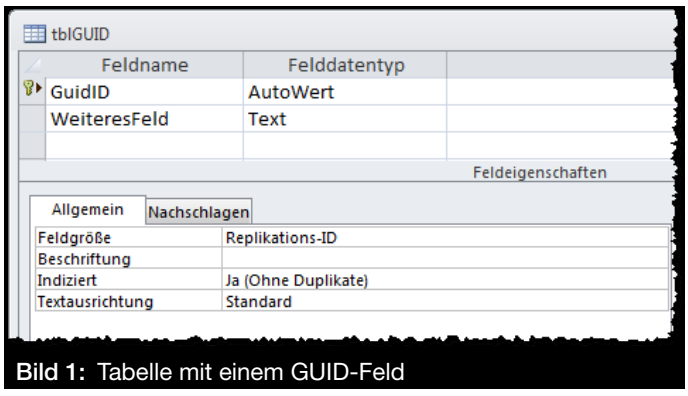



**Bild 1:** Tabelle mit einem GUID-Feld

Wenn Sie nun in die Datenblattansicht wechseln und einige neue Datensätze anlegen, zeigt das Feld **GuidID** frisch erzeugte GUID-Werte an (siehe Bild 2).

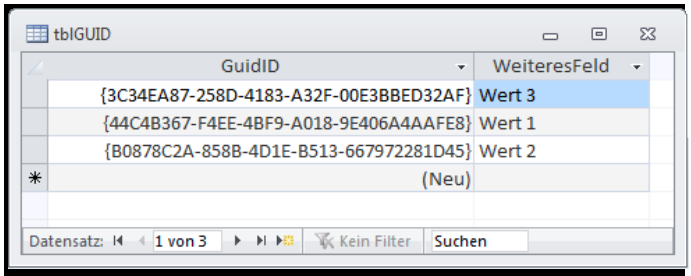



**Bild 2:** GUID-Werte einer Tabelle

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.

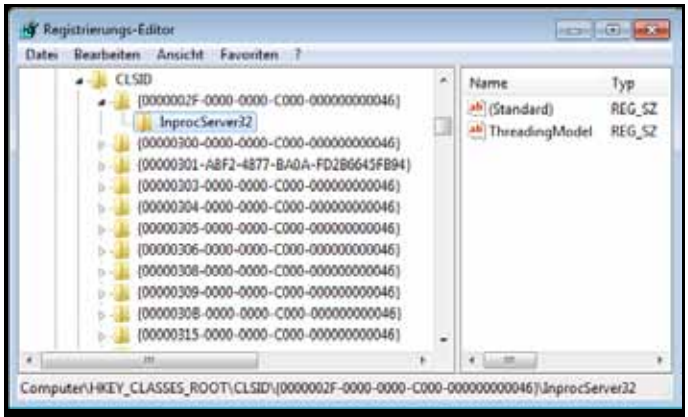



**Bild 3:** GUIDs in der Registry von Windows

## 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.

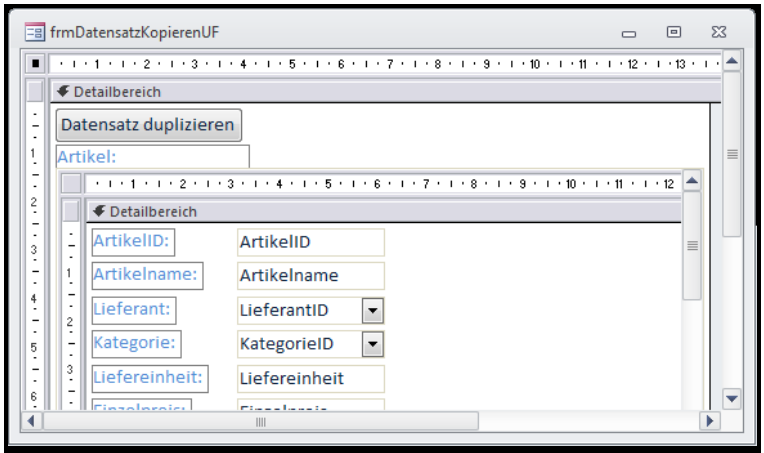



**Bild 1:** Haupt- und Unterformular mit der Schaltfläche zum Kopieren im Hauptformular und den Daten im Unterformular

```
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-**

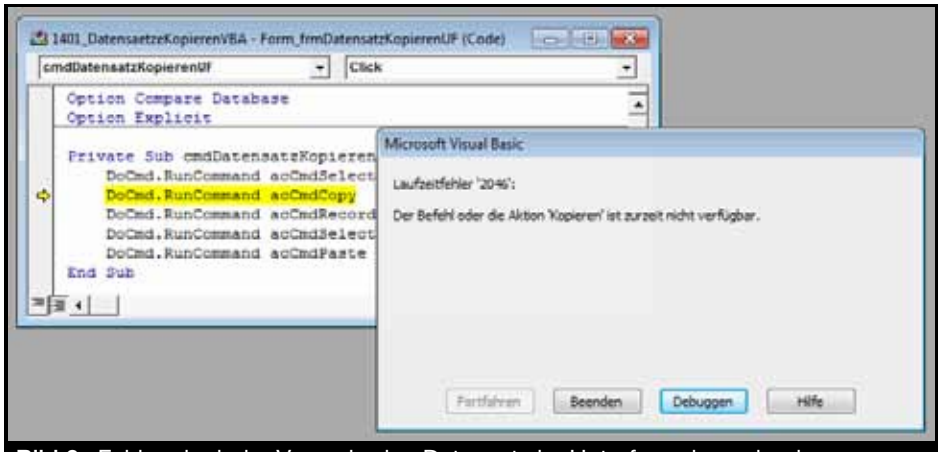



**Bild 2:** Fehler, der beim Versuch, den Datensatz im Unterformular zu kopieren, ausgelöst wird.

**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:

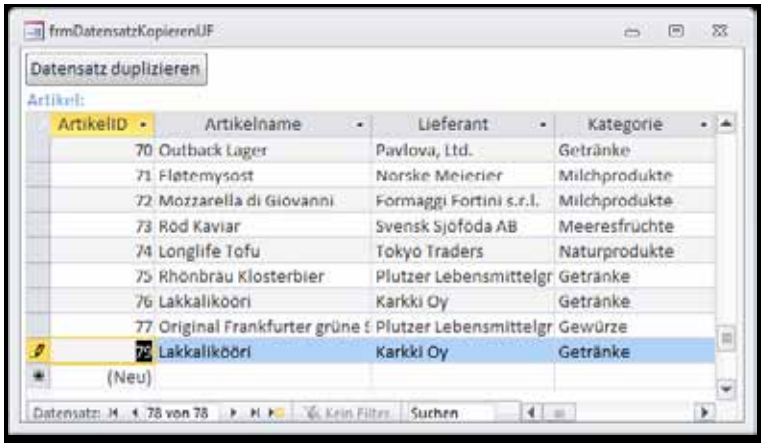


**Bild 3:** Kopieren und Einfügen eines Datensatzes im Unterformular per Schaltfläche

```
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).

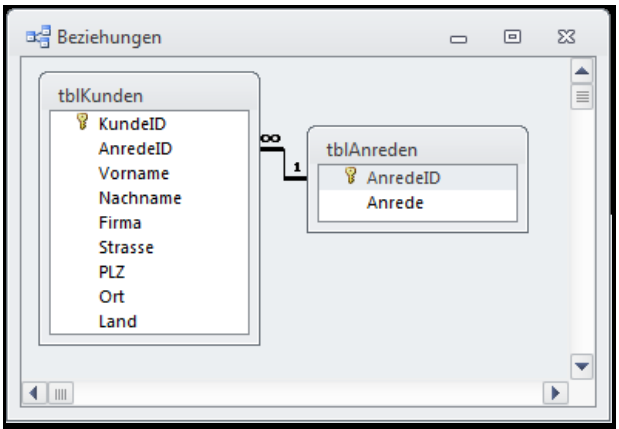



**Bild 1:** Beziehung zwischen Kunden und Anreden

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.

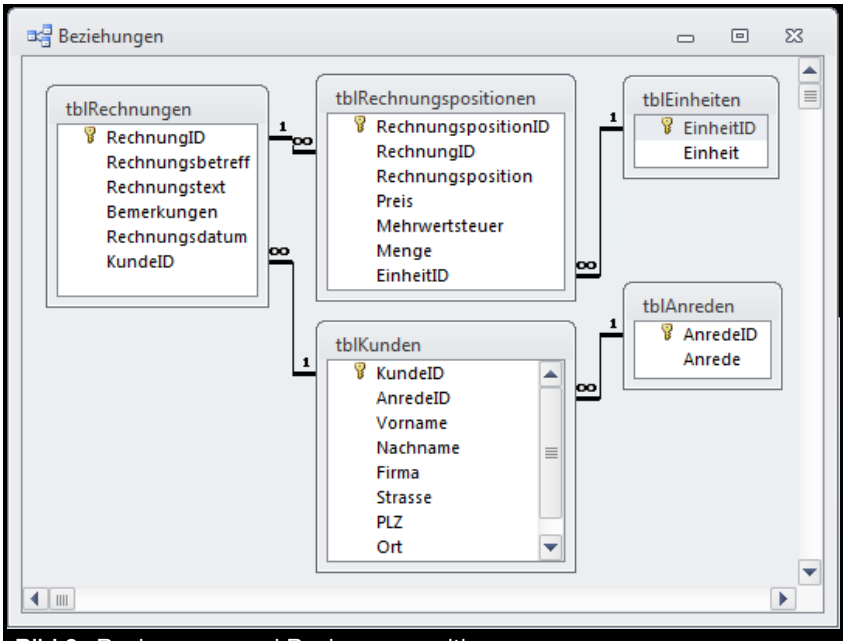



**Bild 2:** Rechnungen und Rechnungspositionen

## 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).

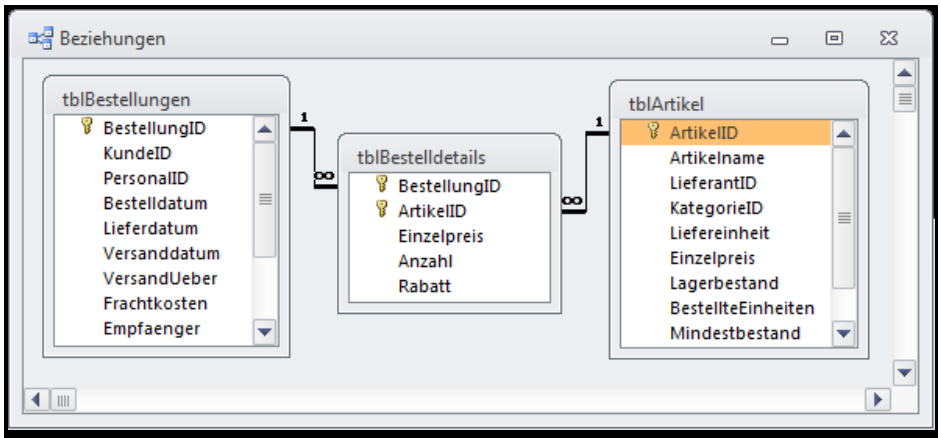



**Bild 3:** Bestellungen, Bestellpositionen und Artikel

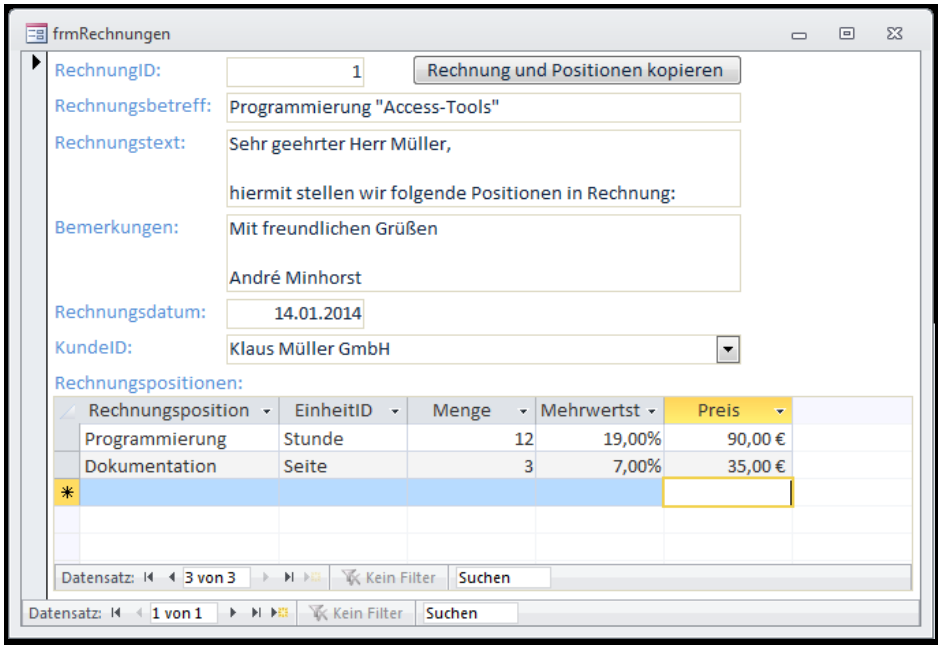



**Bild 4:** Formular zum Erstellen individueller Rechnungen

## Daten aus 1:n-Beziehungen kopieren

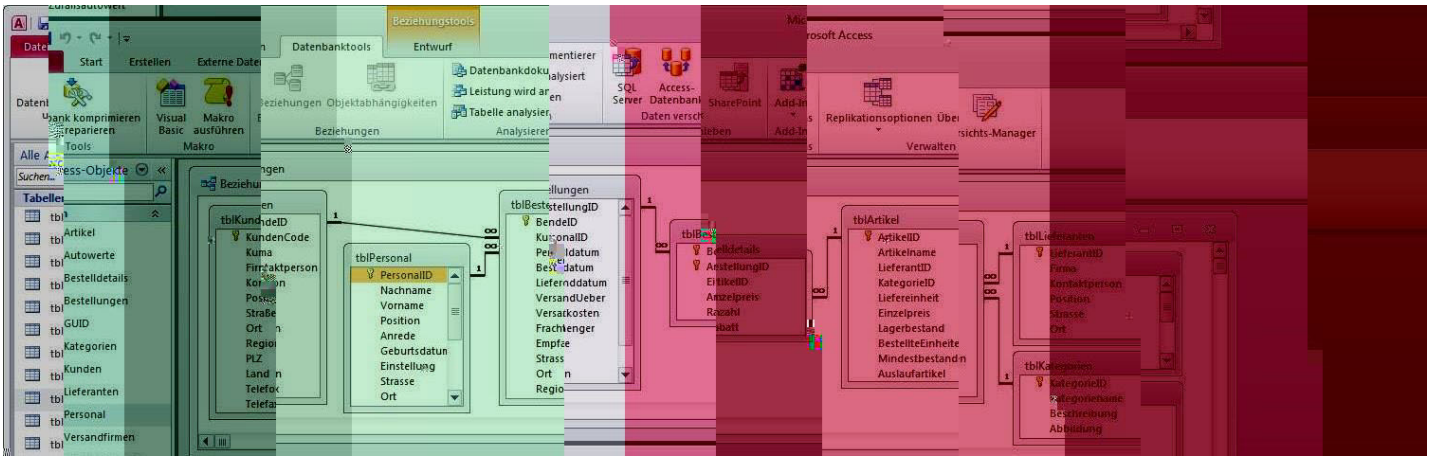
Nun wollen wir uns das erste Beispiel der individuellen Rechnungen ansehen. Den relevanten Teil des Datenmodells haben Sie ja bereits weiter oben kennengelernt. Zusätzlich haben wir ein Formular erstellt, dass die Daten der Tabelle tblRechnungen anzeigt und die Daten der per 1:n-Beziehung verknüpften 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

Ausgabe 02/2014

# ACCESS BASICS

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



## In diesem Heft:

### 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.

SEITE 3

# 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).

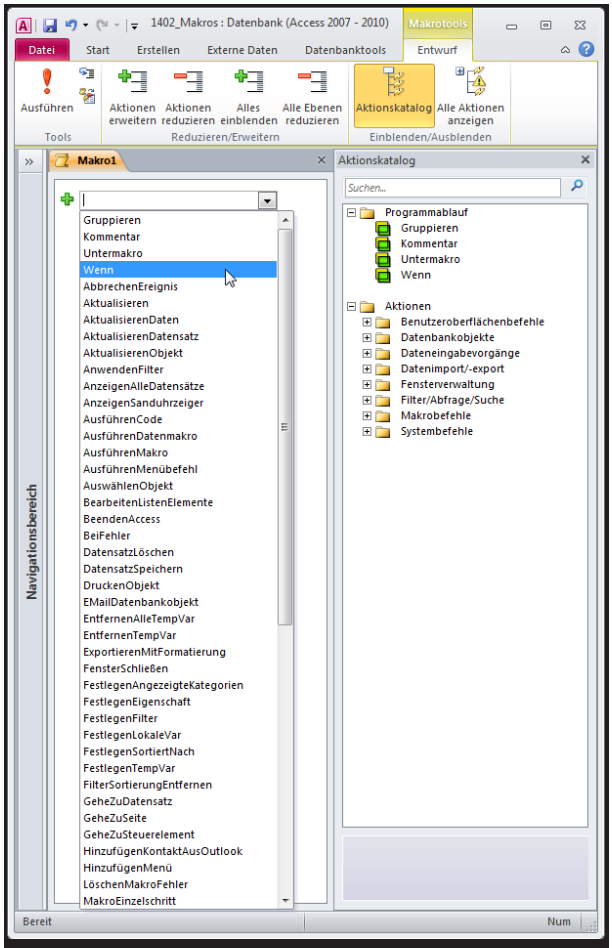



**Bild 1:** Auswählen einer Makroaktion direkt im Makro-Editor ...

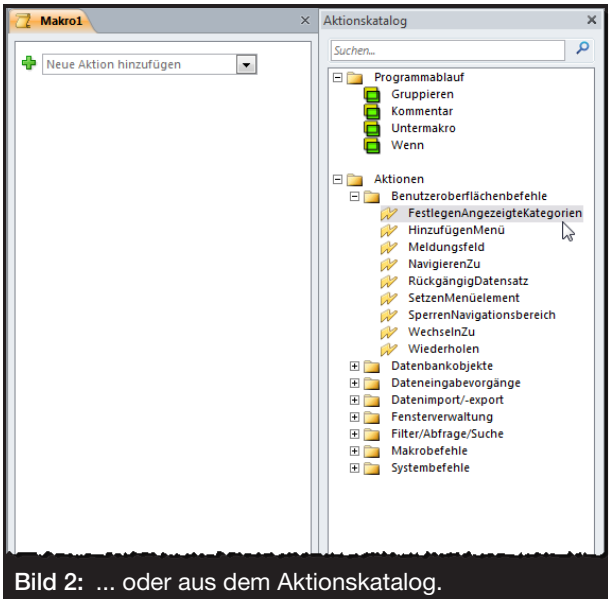



**Bild 2:** ... oder aus dem Aktionskatalog.

- 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.

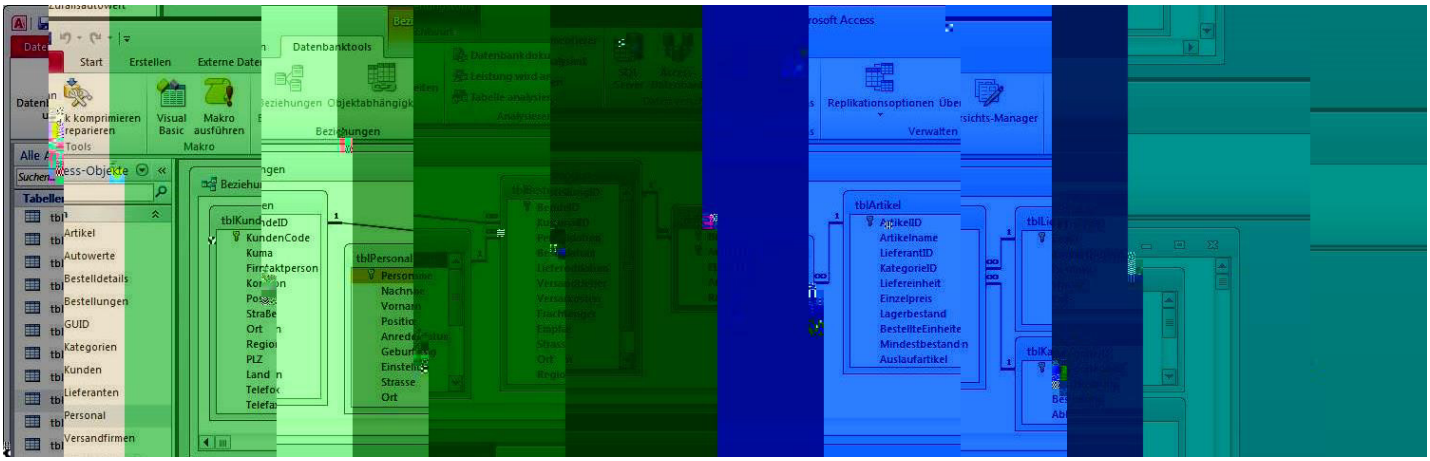
- Sie können bereits erstellte Makros ganz einfach als Untermakros zu weiteren Makros hinzufügen. Diese werden im Aktionskatalog in der Kategorie **In diesen Datenbank|Makros** eingeblendet.

- Sie können Teile des Makrocodes kopieren und an anderer Stelle wieder einfügen. Das dabei verwendete Format ist XML. Sie können ein Makro also auch

Ausgabe 03/2014

# ACCESS BASICS

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



## In diesem Heft:


### PROGRAMMIEREN MIT MAKROS, TEIL II

IM ZWEITEN TEIL DIESER REIHE ERFAHREN SIE MEHR ÜBER DIE ALTERNATIVE ZUR PROGRAMMIERUNG VON DATENBANKANWENDUNGEN PER VBA.

SEITE 3

### FEHLERBEHANDLUNG IN MAKROS

AUCH MAKROS BIETEN IN DEN NEUEREN ACCESS-VERSIONEN DIE MÖGLICHKEIT EINER FEHLERBEHANDLUNG. DIESER ARTIKEL ZEIGT, WIE ES FUNKTIONIERT!

SEITE 11

### DATENMAKROS I: GRUNDLAGEN

MAKROS UNTERSTÜTZEN AUCH DATENOPERATIONEN, WIE SIE MIT DAO UNTER VBA MÖGLICH SIND. DIESER ARTIKEL ZEIGT, WAS GEHT UND WAS NICHT.

SEITE 16

# 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

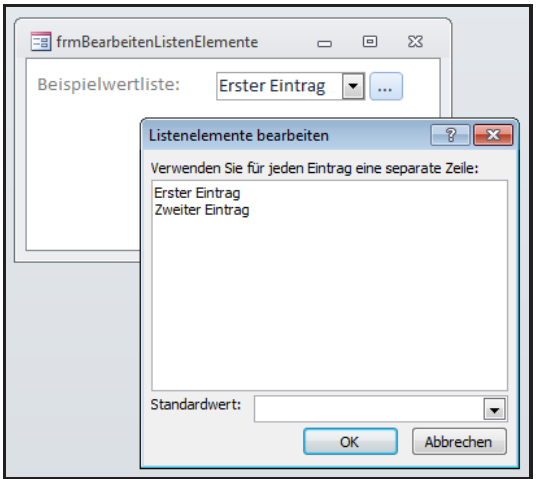



**Bild 1:** Öffnen des Dialogs zum Anzeigen einer Wertliste zur Bearbeitung

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.

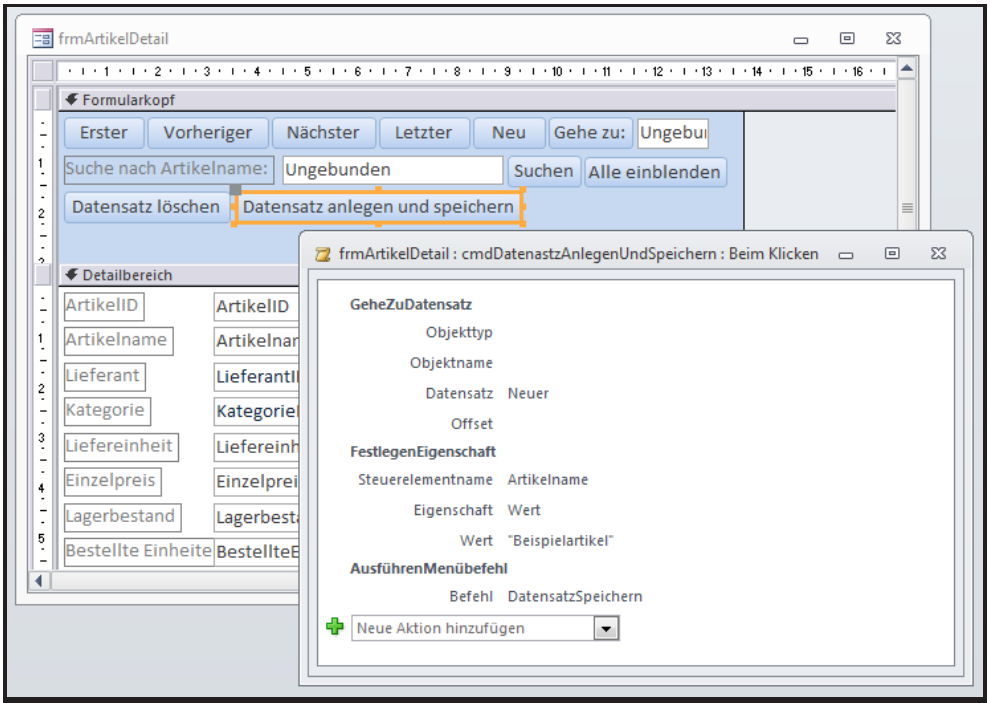


**Bild 2:** Anlegen und speichern eines neuen Datensatzes

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

# 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):

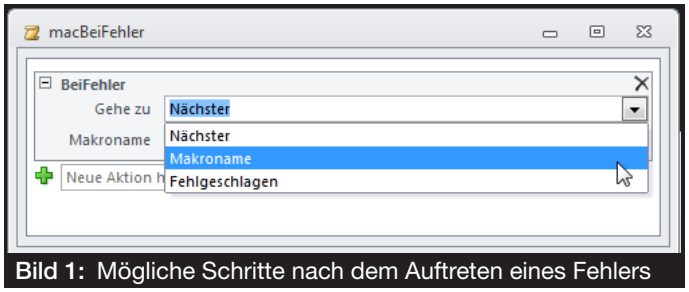



**Bild 1:** Mögliche Schritte nach dem Auftreten eines Fehlers

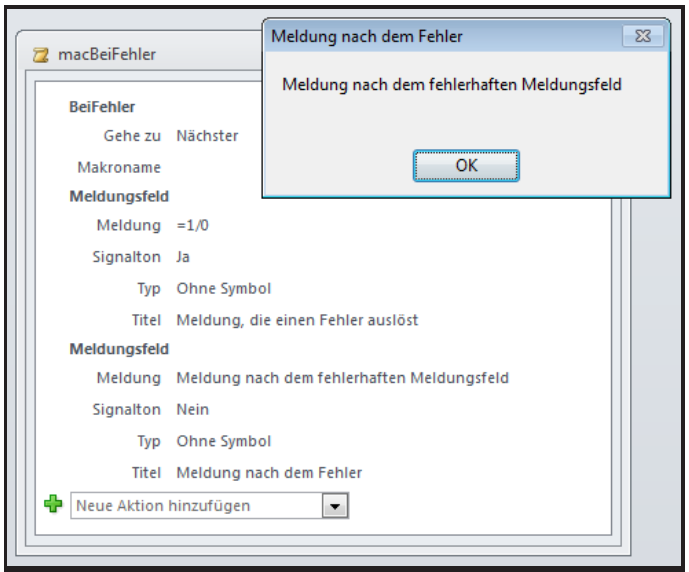



**Bild 2: Gehe zu: Nächster** fährt nach dem Fehler mit der folgenden Makroaktion fort.

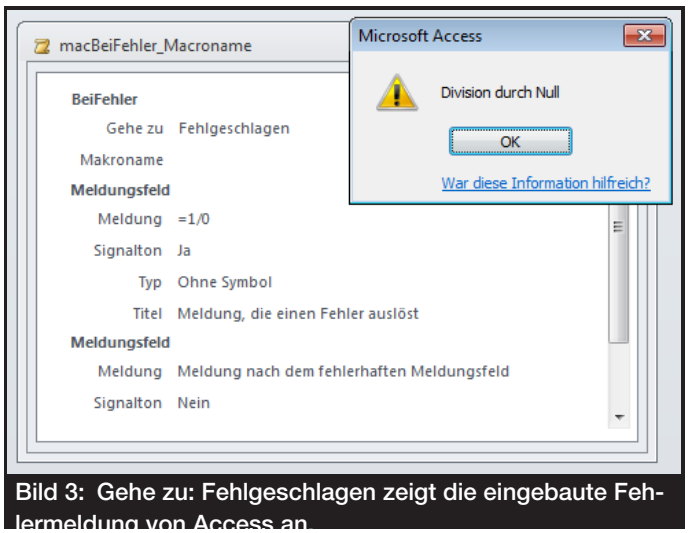



**Bild 3: Gehe zu: Fehlgeschlagen** zeigt die eingebaute Fehlermeldung von Access an.

- **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).

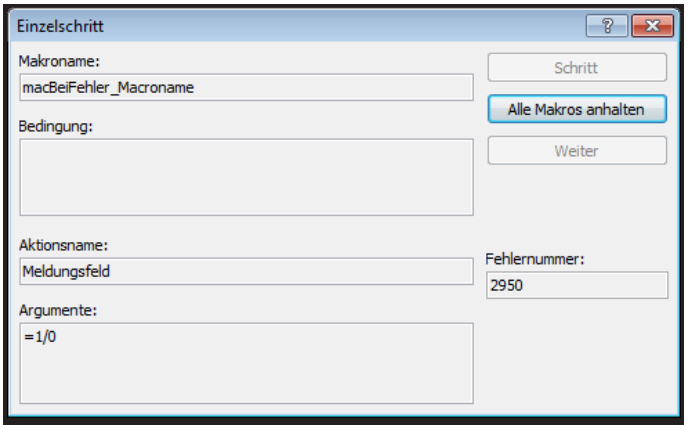


Bild 4: Fehlermeldung eines Makros

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].[Beschrei-
bung]
& "' 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-**

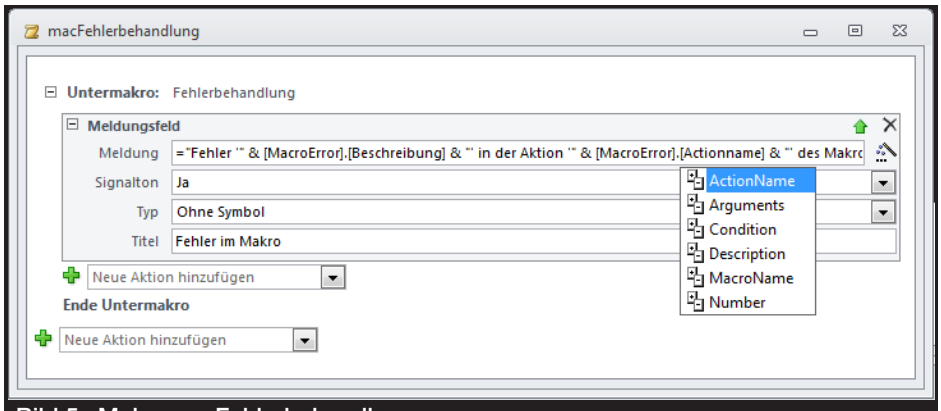


Bild 5: Makro zur Fehlerbehandlung

# 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

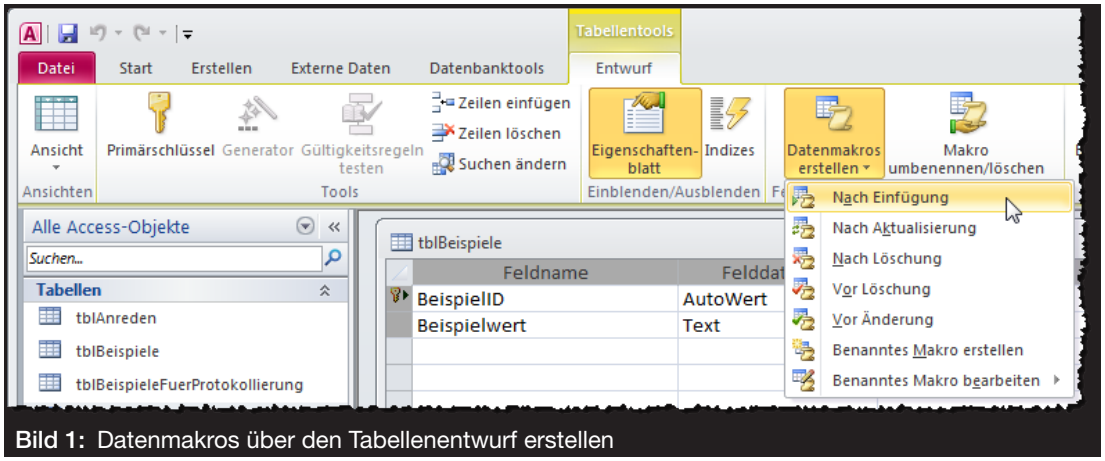


**Bild 1:** Datenmakros über den Tabellenentwurf erstellen

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):

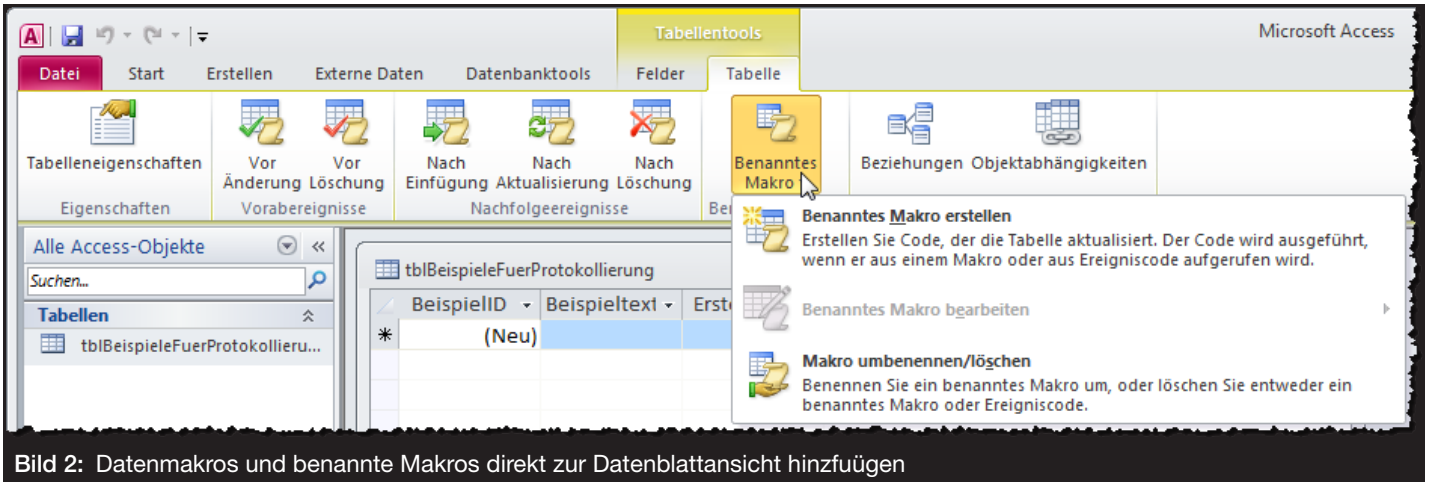
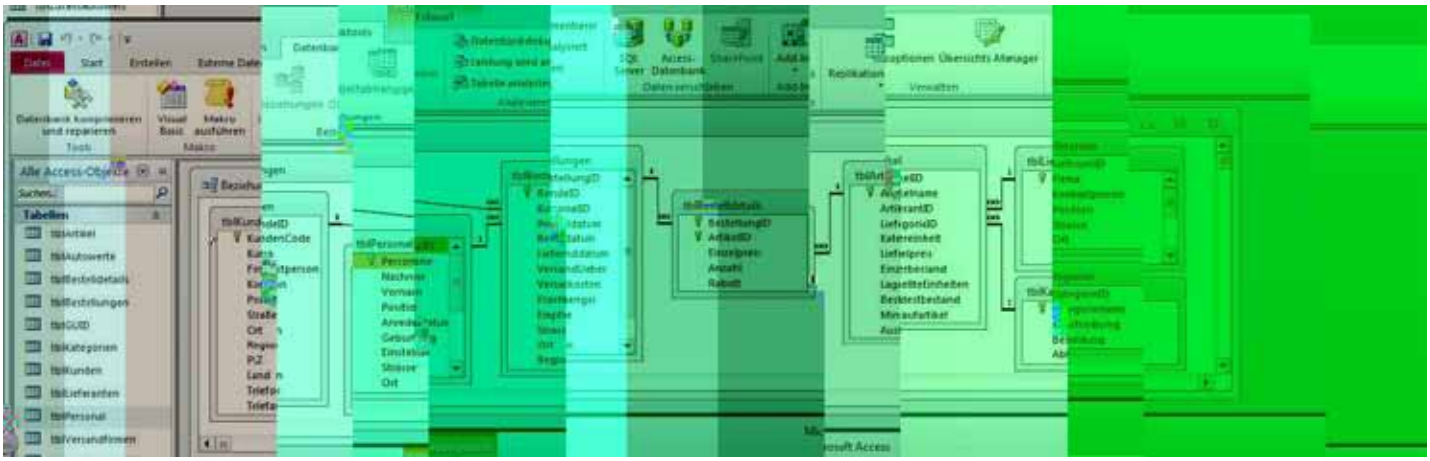


**Bild 2:** Datenmakros und benannte Makros direkt zur Datenblattansicht hinzufügen

Ausgabe 04/2014

# ACCESS BASICS

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



## In diesem Heft:


### VERTRAUENSWÜRDIGE DATENBANKEN

ALLES RUND UM DIE MIT ACCESS 2007 EINGEFÜHRTEN MECHANISMEN FÜR DIE SICHERE AUSFÜHRUNG VON DATENBANKEN AUS VERSCHIEDENEN QUELLEN

SEITE 3

### VERTRAUENSWÜRDIGKEIT PER MAKRO PRÜFEN

CODE IN NICHT VERTRAUENSWÜRDIGEN DATENBANKEN WIRD MITUNTER EINFACH NICHT AUSGEFÜHRT. PER MAKRO WEISEN SIE DEN BENUTZER DARAUF HIN.

SEITE 9

### DATENMAKROS II: DATENOPERATIONEN

MAKROS ERLAUBEN SEIT ACCESS 2010 DEN GEZIELTEN ZUGRIFF AUF DATEN UND DIE MANIPULATION VON DATENSÄTZEN. ERFAHREN SIE, WIE ES GEHT!

SEITE 11

# 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

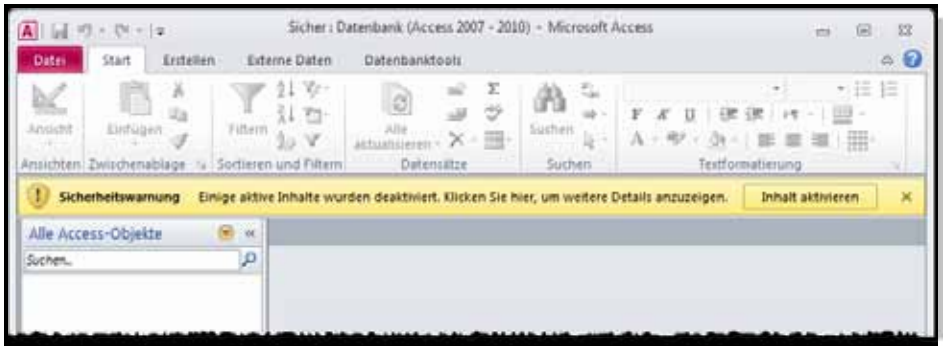



**Bild 1:** Hinweis auf eine nicht als vertrauenswürdig eingestufte Datenbankdatei

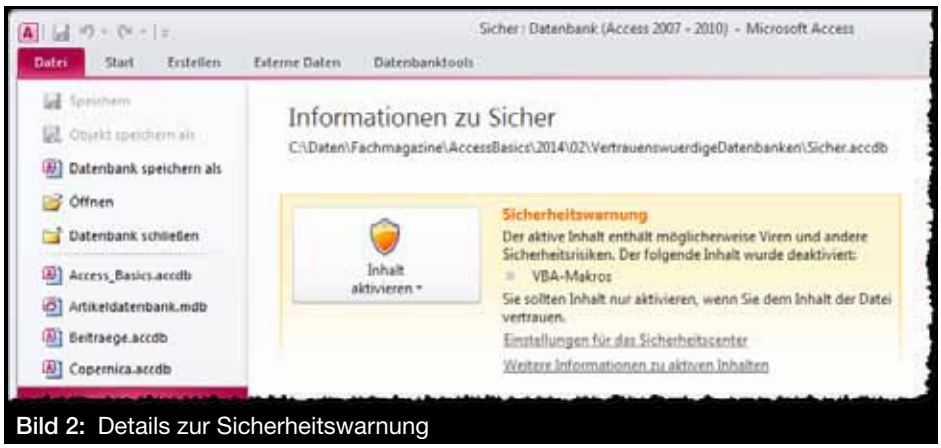



**Bild 2:** Details zur Sicherheitswarnung

Sie die Anwendung für eine Sitzung „entsichern". Beim nächsten Öffnen erscheint erneut der gelbe Balken mit der Sicherheitswarnung.

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.

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.

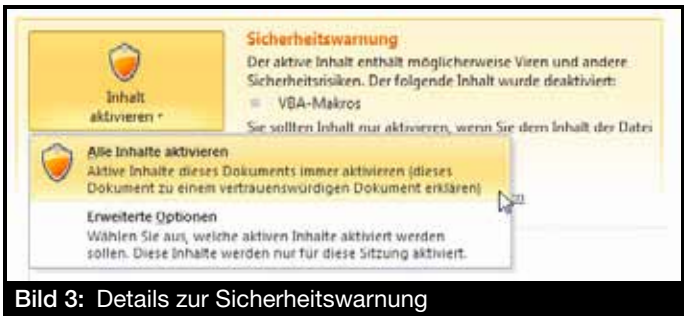



**Bild 3:** Details zur Sicherheitswarnung

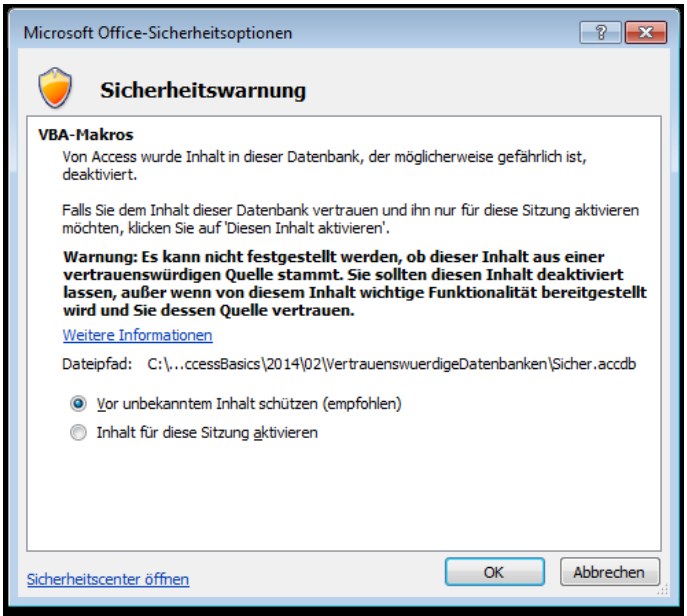



**Bild 4:** Aktivieren der Datenbank für eine Sitzung

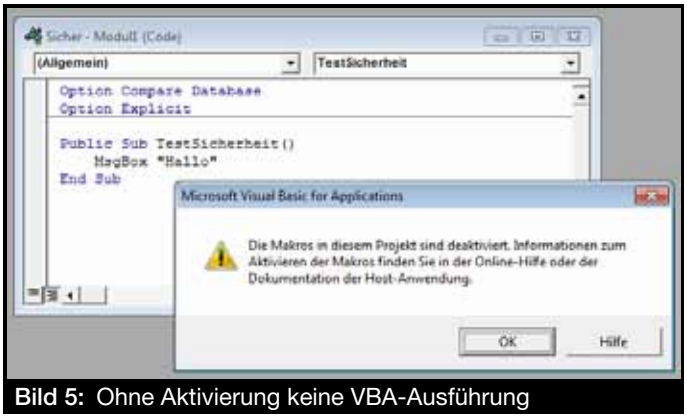



**Bild 5:** Ohne Aktivierung keine VBA-Ausführung

## 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

# 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.

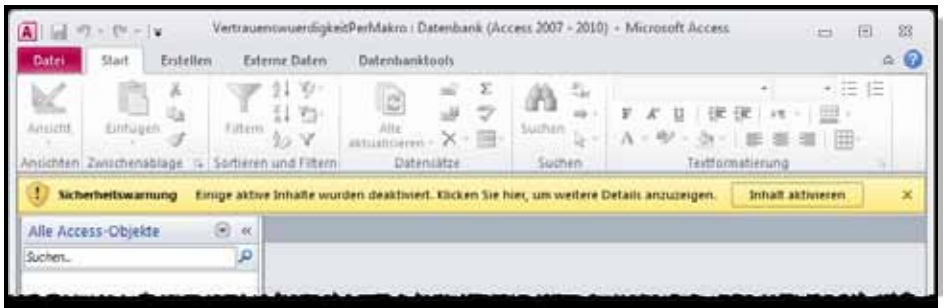



**Bild 1:** Diese Sicherheitswarnung wird gern übersehen ... oder erscheint erst gar nicht.

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.

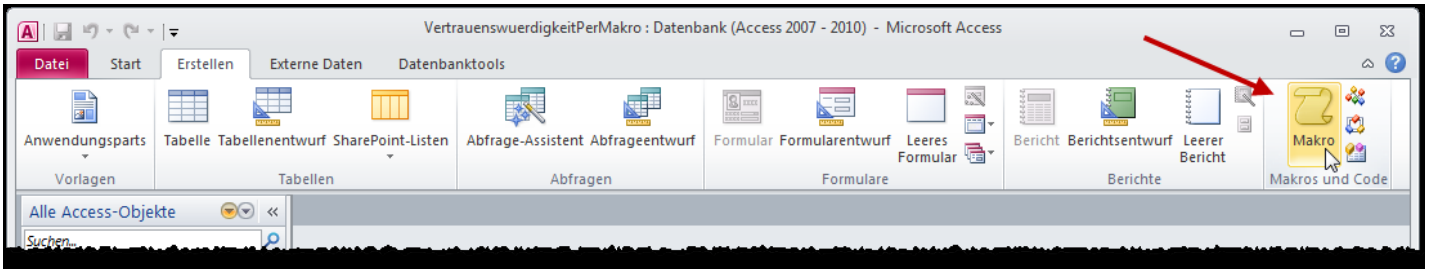



**Bild 2:** Anlegen eines neuen Makros

### Makro anlegen

Um das benötigte Makro zu erstellen, legen Sie etwa unter Access 2010 mit dem Ribbon-Befehl **Erstellen|Makros und Code|Makro** ein neues Makro an.

Diesem fügen Sie dann zunächst eine **Wenn**-Bedingung hinzu. Der Bedingungsausdruck lautet:

```
Nicht [CurrentProject].[IsTrusted]
```

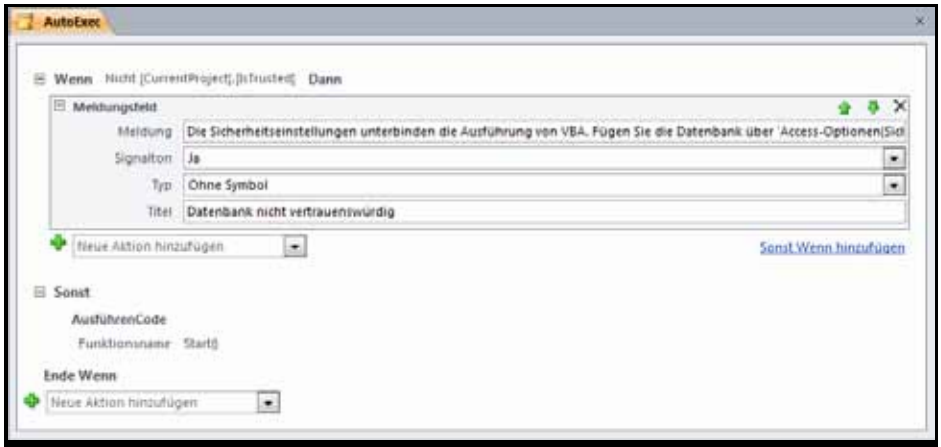


**Bild 3:** Das neue **AutoExec**-Makro

Achtung: Sie müssen unbedingt **CurrentProject** und **IsTrusted** in eckige Klammern einfassen.

Diese Bedingung ist erfüllt, wenn die Datenbank nicht als vertrauenswürdig eingestuft wird – also wenn sie nicht in einem vertrauenswürdigen Verzeichnis liegt oder zur Liste der vertrauenswürdigen Dokumente hinzugefügt wurde.

Wenn diese Bedingung wahr ist, soll das Makro eine Meldung mit dem Text aus Bild 4 anzeigen. Diesen fügen Sie daher für den innerhalb der Wenn-Bedingung befindlichen Meldungsfenster-Makrobefehl für die Eigenschaft **Meldung** ein. Außerdem legen Sie einen Text für die Eigenschaft **Titel** fest.

Wir fügen noch eine zweite Bedingung hinzu, die prüft, ob die Anwendung vertrauenswürdig ist und in diesem Fall gleich die für den Start der Anwendung vorgesehenen VBA-Code ausführt – hier könnten Sie dann beispielsweise das Startformular öffnen, die Datenbank mit einem Backend verbinden et cetera.

Wir haben für die **Start**-Funktion lediglich eine einfache Meldung hinterlegt:

```
Public Function Start()
    MsgBox "Die Ausführung von VBA 
                          funktioniert."
End Function
```

Speichern Sie das Makro unter dem Namen **AutoExec**, damit Access es gleich beim Öffnen der Datenbank automatisch startet.

Auf diese Weise geschieht Folgendes: Wenn die Datenbank nicht vertrauenswürdig ist, erscheint in jedem Fall die entsprechende Meldung der ersten Wenn-Bedingung des AutoExec-Makros.

Wenn die Sicherheitseinstellungen die Anzeige der gelben Statusleiste erlauben, kann der Benutzer anschließend direkt mit einem Klick auf die Schaltfläche **Inhalt aktivieren** aktivieren.

Das **AutoExec**-Makro wird dann erneut ausgeführt, wobei die erste Bedingung falsch und die zweite wahr wird. Dies ruft dann die für den Start vorgesehene Funktion **Start()** auf.

### Zusammenfassung

Diese Makrofunktion sollte eigentlich in jeder Access-Anwendung enthalten sein, die unter Access 2007 und jünger verwendet und an unerfahrene Benutzer weitergegeben wird.

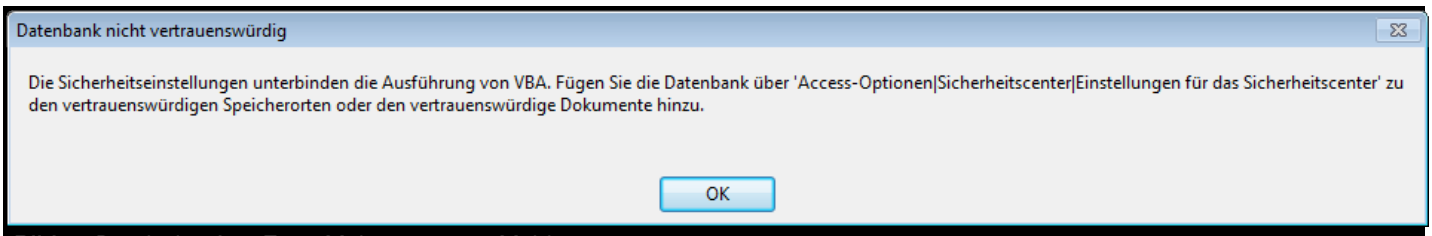


**Bild 4:** Durch das AutoExec-Makro erzeugte Meldung

# 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).

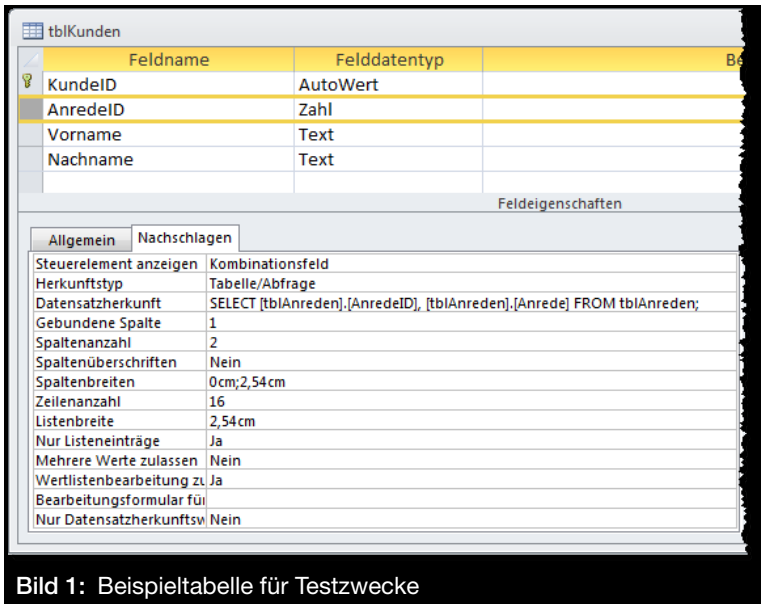



**Bild 1:** Beispieltabelle für Testzwecke

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.

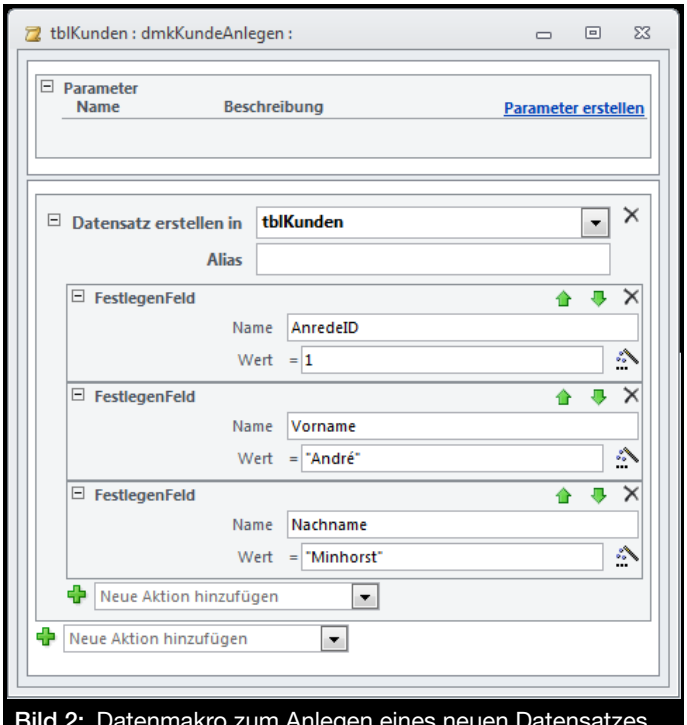



**Bild 2:** Datenmakro zum Anlegen eines neuen Datensatzes in der Tabelle tblKunden

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

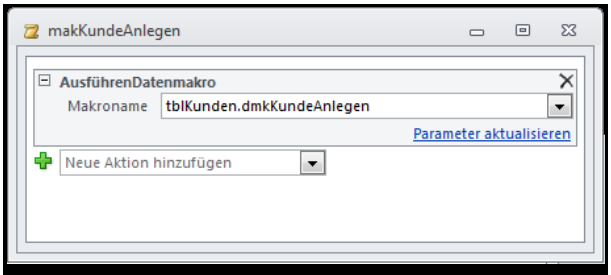



**Bild 3:** Dieses Makro ruft das Datenmakro zum Anlegen eines Kundendatensatzes auf.

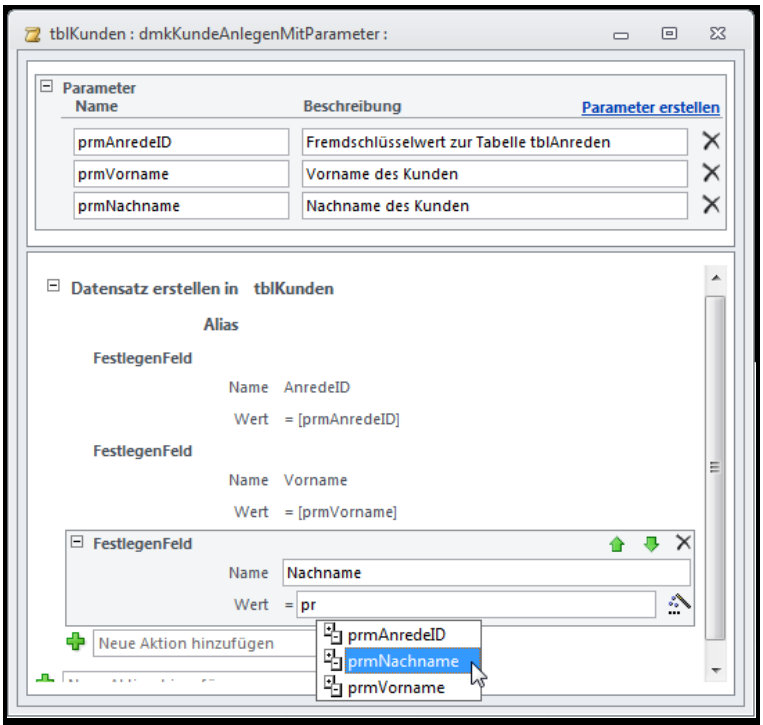



**Bild 4:** Benanntes Makro mit Parametern

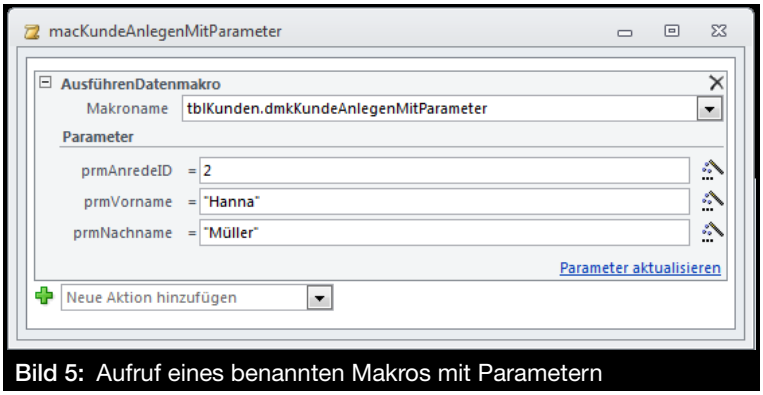



**Bild 5:** Aufruf eines benannten Makros mit Parametern

im Makro-Entwurf und erzeugen so drei Zeilen zur Eingabe von Parametern. Diese nennen wir **prmAnredeID**, **prmVorname** und **prmNachname**.

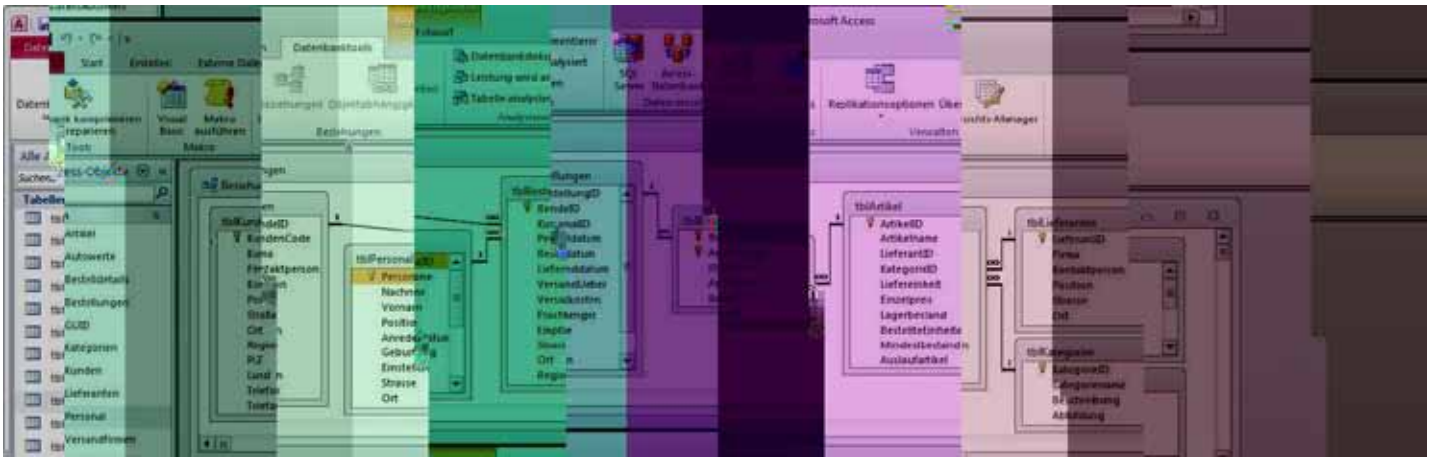
Nun fügen Sie wieder die Makroaktion **Datensatz erstellen in** hinzu, legen die Tabelle **tblKunden** als Ziel fest und tragen unterhalb dieser Makroaktion drei Mal den Befehl **FestlegenFeld** ein. Die Feldnamen lauten wieder wie im vorherigen Beispiel. Allerdings geben wir diesmal nicht direkt die 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

Ausgabe 05/2014

# ACCESS BASICS

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



## In diesem Heft:


### TEMPVARS UNTER VBA

SPEICHERN SIE WICHTIGE INFORMATIONEN IN TEMPVARS ZWISCHEN UND RUFEN SIE DIESE WIEDER AB. TEMPVARS SIND IM GEGENSATZ ZU GLOBALEN VARIABLEN FEHLERRESISTENT!

SEITE 3

### PROGRAMMIEREN MIT COLLECTIONS

IN COLLECTIONS SPEICHERN SIE VARIALBEN ODER OBJEKTE UNTER ANGABE DES GEWÜNSCHTEN SCHLÜSSELS AB. ERFAHREN SIE ALLES ÜBER DIE PROGRAMMIERUNG DIESER OBJEKTE!

SEITE 8

### MEHRERE FORMULARINSTANZEN ANZEIGEN

ERFAHREN SIE, WIE SIE ZWEI ODER MEHR DATENSÄTZE DER SELBEN TABELLE IN VERSCHIEDENEN INSTANZEN DES GLEICHEN FORMULARS ANZEIGEN!

SEITE 12

# 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.

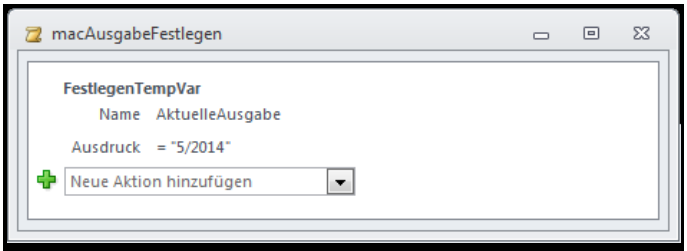



**Bild 1:** Festlegen einer temporären Variablen per Makro

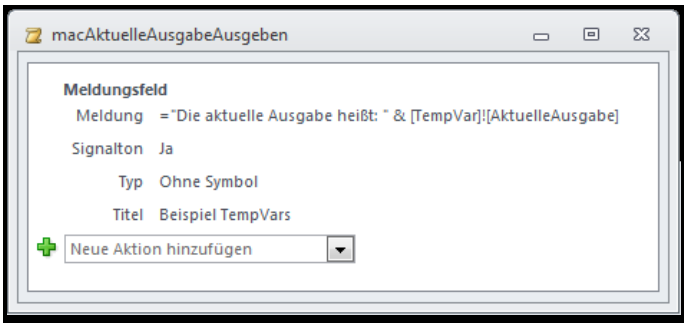



**Bild 2:** Ausgabe des Inhalts der temporären Variablen

Temporäre Variablen nehmen Variablen des Typs **Variant** auf, also auch Strings, Zahlen et cetera, aber keine Objekte.

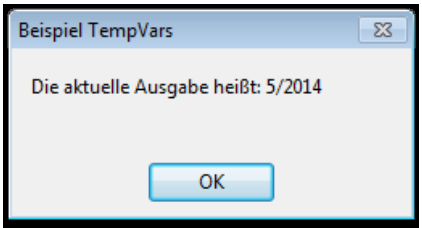



**Bild 3:** Per Makro angezeigtes Meldungsfenster mit dem Inhalt einer temporären Variablen

### 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.

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.

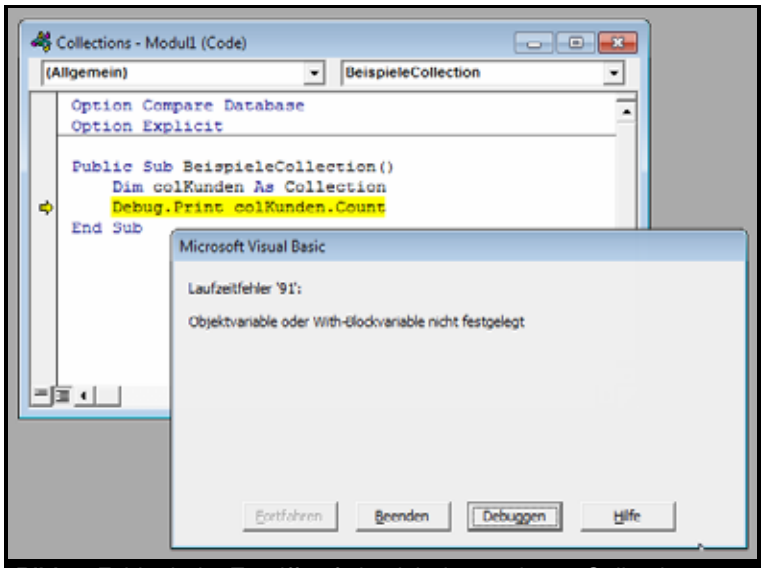



**Bild 1:** Fehler beim Zugriff auf ein nicht instanziertes Collection-Objekt

Dies können Sie in der gleichen Zeile erledigen, welche auch das Objekt deklariert:

```
Dim colKunden As New Collection
```

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.

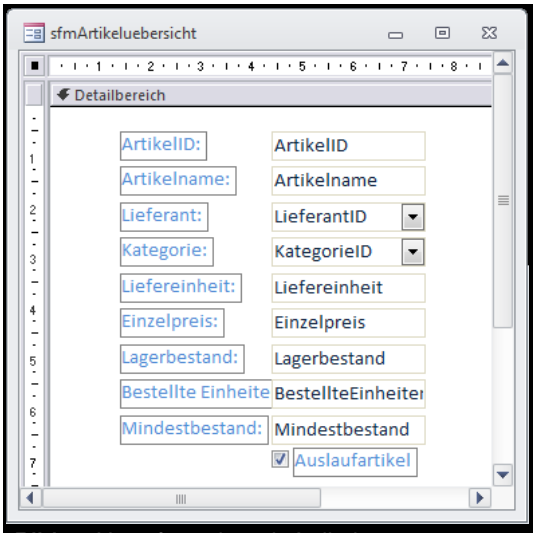


**Bild 1:** Unterformular mit Artikeln

## 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.

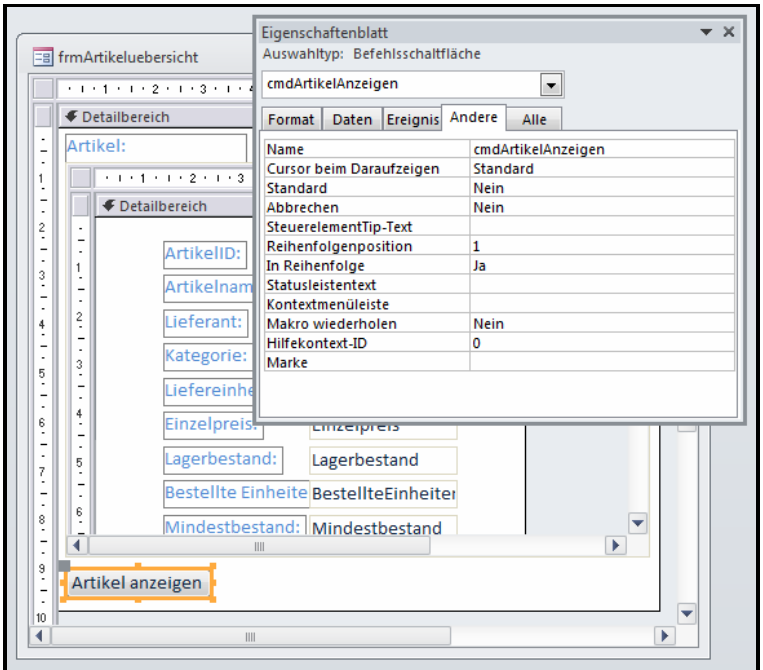


**Bild 2:** Haupt- und Unterformular zur Auswahl der Detailansichten

## 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.

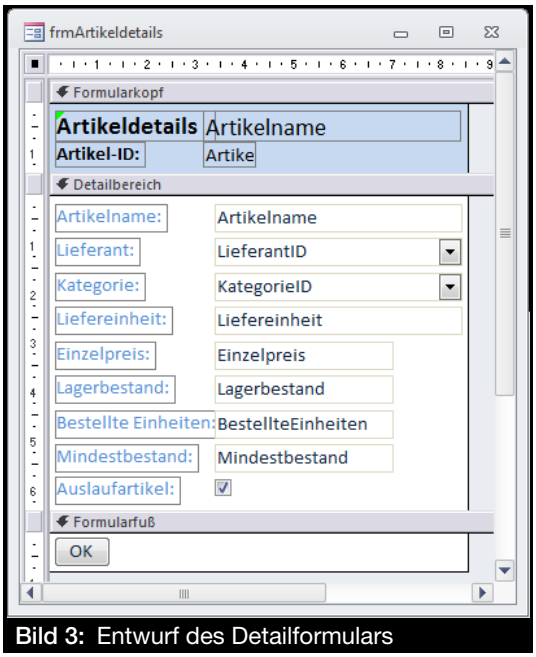


**Bild 3:** Entwurf des Detailformulars

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.

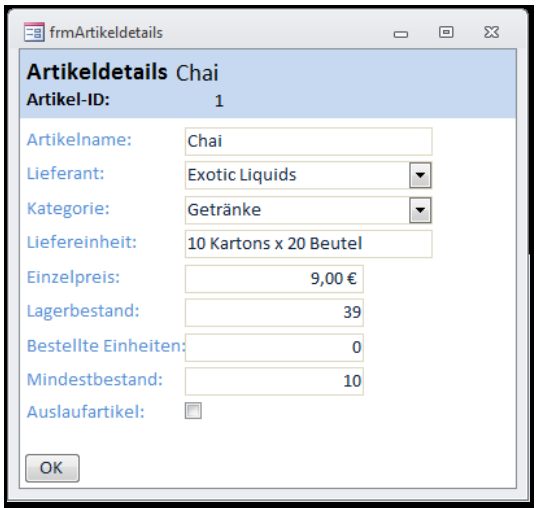


**Bild 4:** Das Detailformular in der Formularansicht

## 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

# 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?", _
                vbYesNo) = vbNo Then
        Me.TimerInterval = 0
    End If
End Sub
```

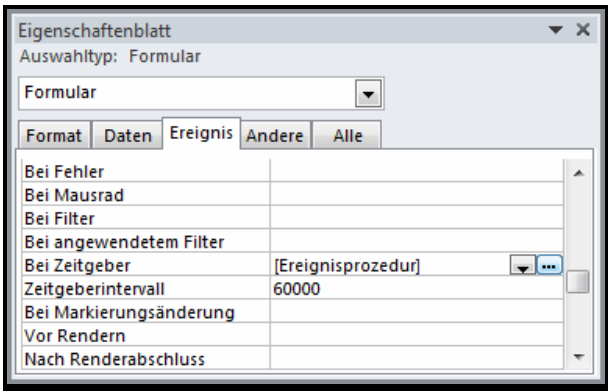



**Bild 1:** Anlegen eines Zeitgebers

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]**.

Ausgabe 06-07/2014

# ACCESS BASICS

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




## In diesem Heft:

### DETAILFORMULARE AUFRUFEN

ERFAHREN SIE, WIE SIE VON EINEM ÜBERSICHTSFORMULAR AUS DEN GEWÜNSCHTEN DATENSATZ IN EINEM WEITEREN FORMULAR IM DETAIL ANZEIGEN.

SEITE 3

### DATENSÄTZE FILTERN PER KOMBINATIONSFELD

FILTERN SIE DIE DATEN IN EINEM UNTERFORMULAR DURCH DIE AUSWAHL EINES KRITERIUMS PER NACHSCHLAGEFELD IM HAUPTFORMULAR.

SEITE 24

### DATENMAKROS III: EREIGNISSE

LERNEN SIE DIE NEUEN TABELLENEREIGNISSE KENNEN, DIE BEIM ANLEGEN, ÄNDERN ODER LÖSCHEN VON DATENSÄTZEN AUSGELÖST WERDEN!

SEITE 12

# 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.

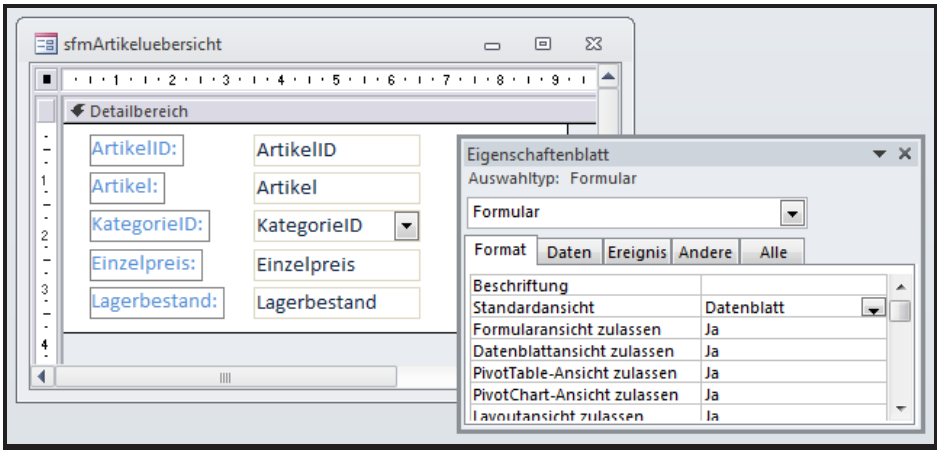


**Bild 1:** Unterformular mit der Liste der Artikel

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).

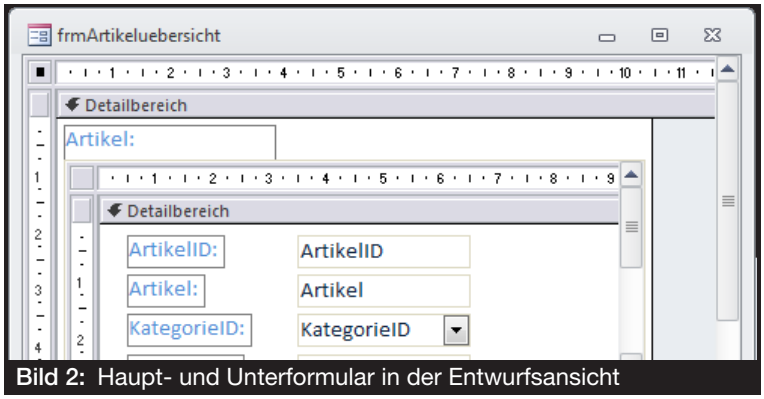


**Bild 2:** Haupt- und Unterformular in der Entwurfsansicht

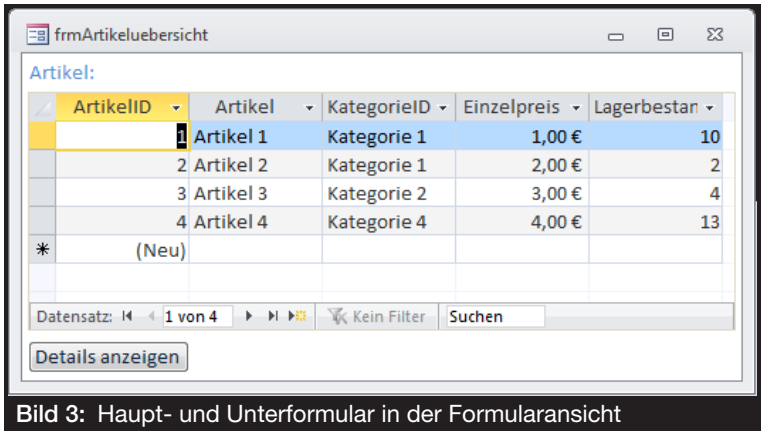


**Bild 3:** Haupt- und Unterformular in der Formularansicht

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).

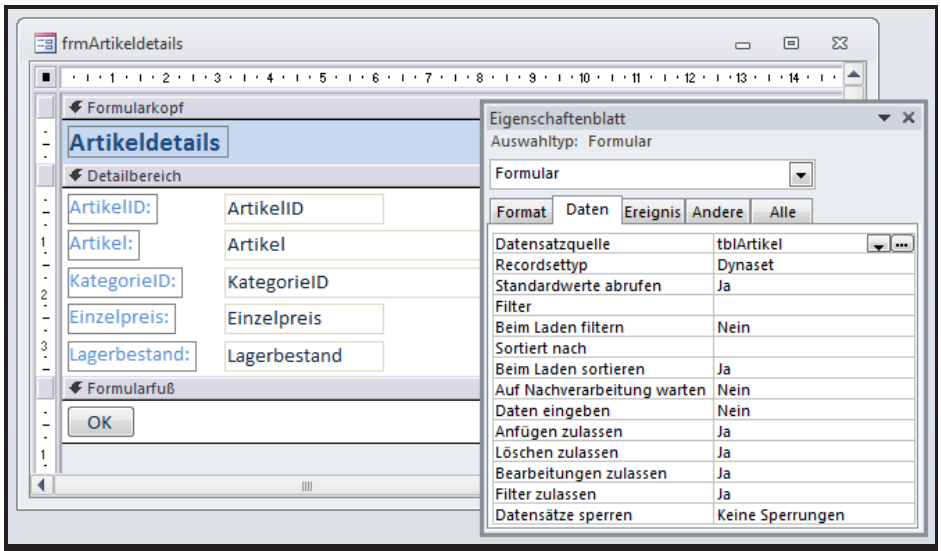


**Bild 4:** Entwurf des Detailformulars

## 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).

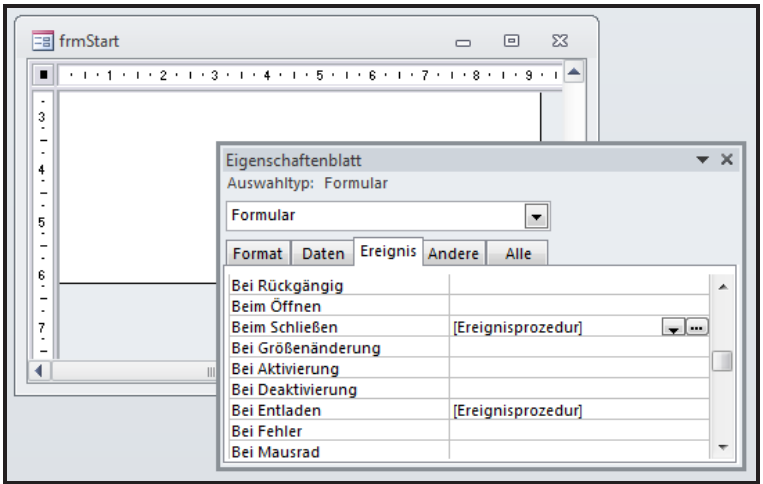



**Bild 1:** Ereignisse anlegen, die beim Schließen eines Formulars ausgelöst werden

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).

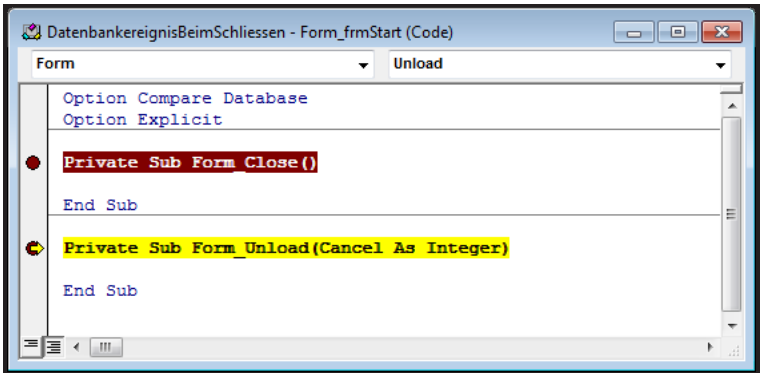



**Bild 2:** Haltepunkte der beiden beim Schließen ausgelösten Prozeduren im Test

```
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).

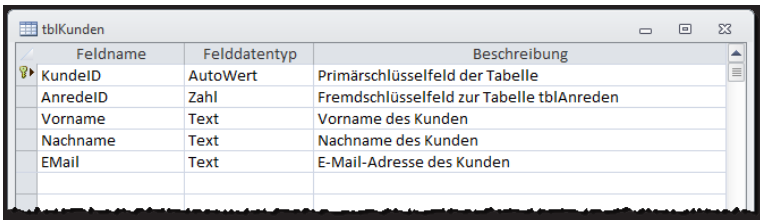



**Bild 1:** Tabelle zum Darstellen der Möglichkeiten von Tabellenereignissen

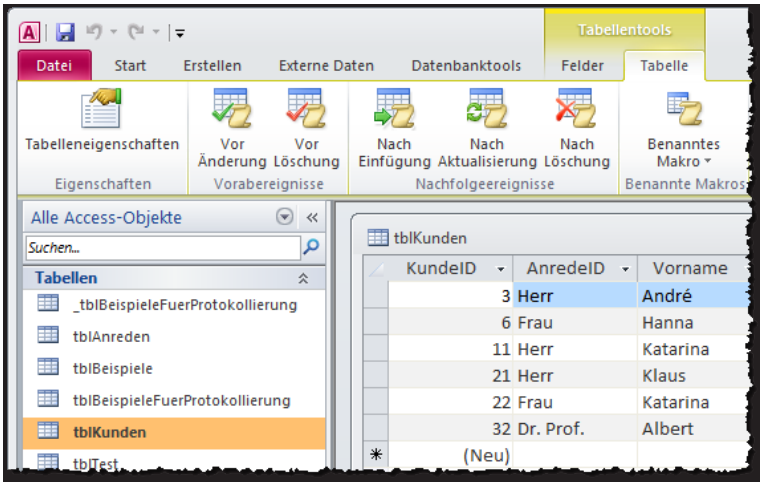



**Bild 2:** Schaltflächen zum Hinzufügen von Tabellenereignissen in der Datenblattansicht

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.

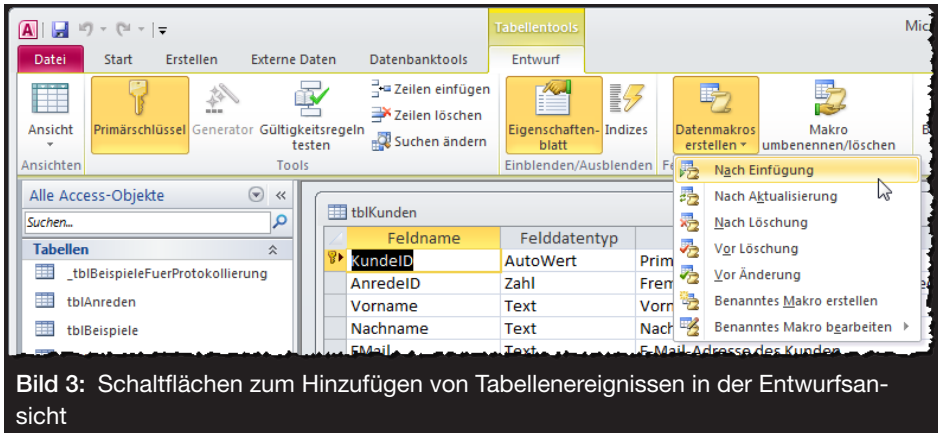



**Bild 3:** Schaltflächen zum Hinzufügen von Tabellenereignissen in der Entwurfsansicht

- **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.

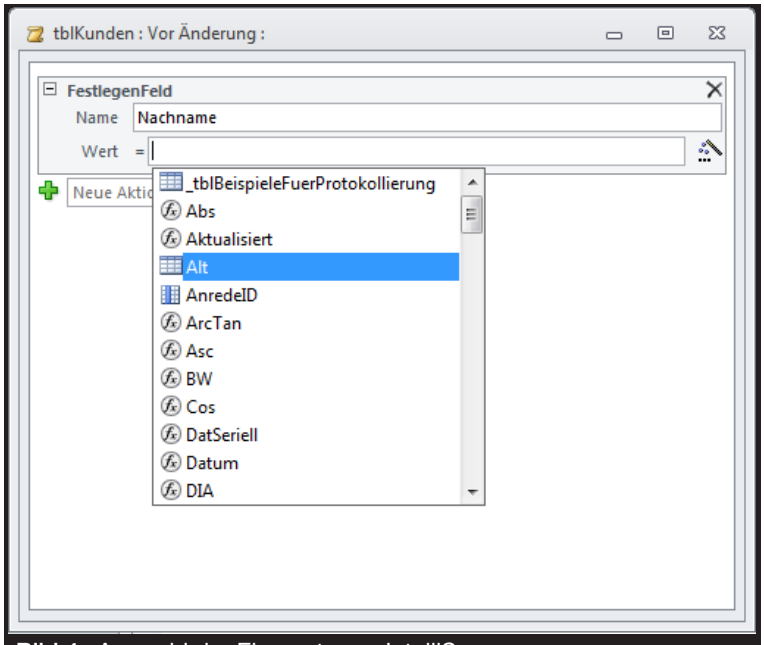



**Bild 4:** Auswahl der Elemente per IntelliSense

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-**

# 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**).

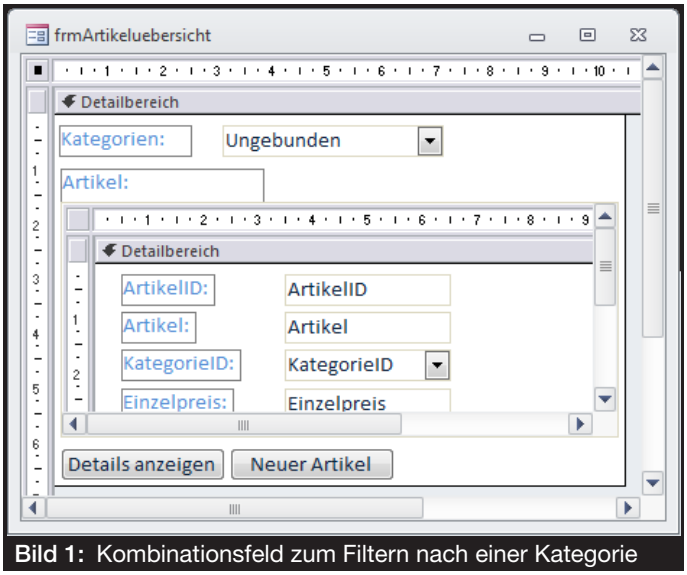


**Bild 1:** Kombinationsfeld zum Filtern nach einer Kategorie

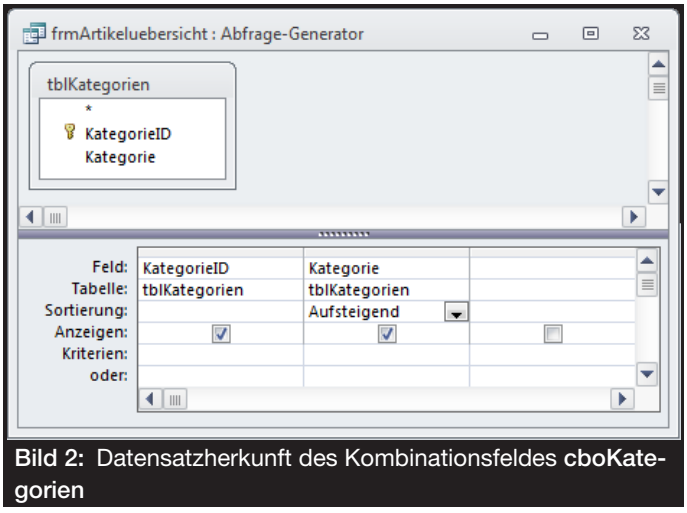


**Bild 2:** Datensatzherkunft des Kombinationsfeldes **cboKategorien**

## 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
```

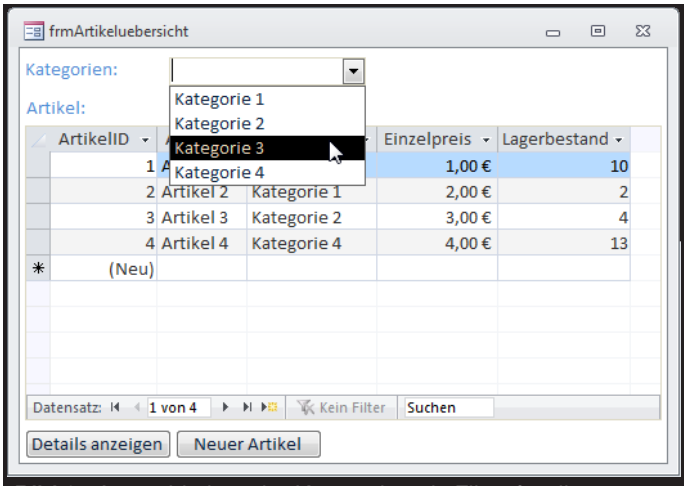



**Bild 3:** Auswahl einer der Kategorien als Filter für die angezeigten Datensätze im Unterformular

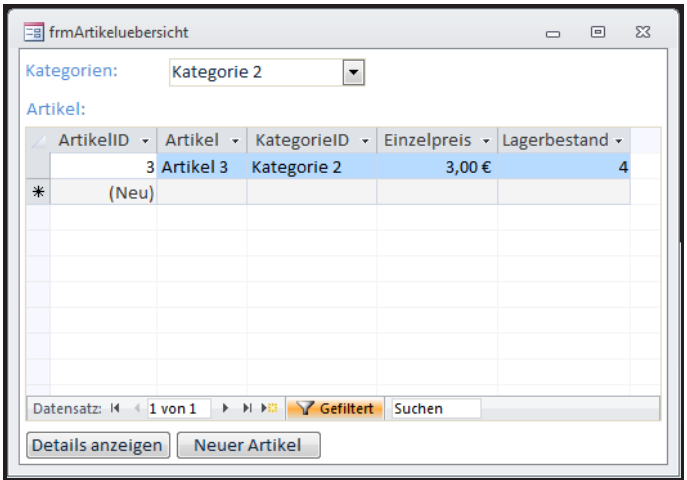



**Bild 4:** Filtern nach einer der Kategorien

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

# 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.

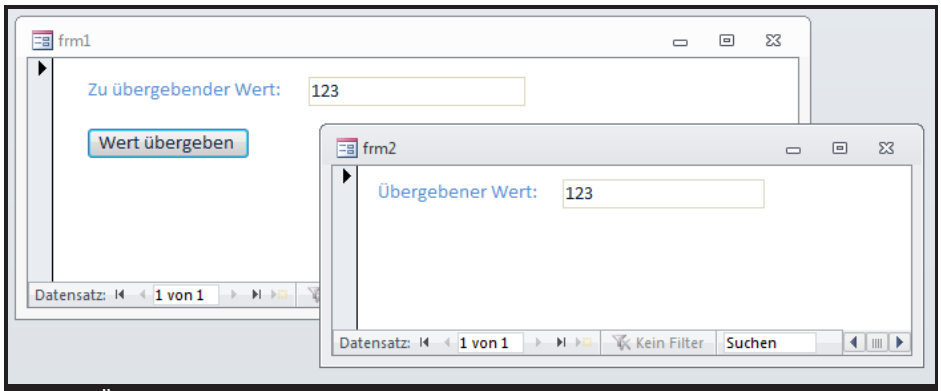



**Bild 1:** Übergeben eines Wertes von einem Formular zum nächsten

```
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

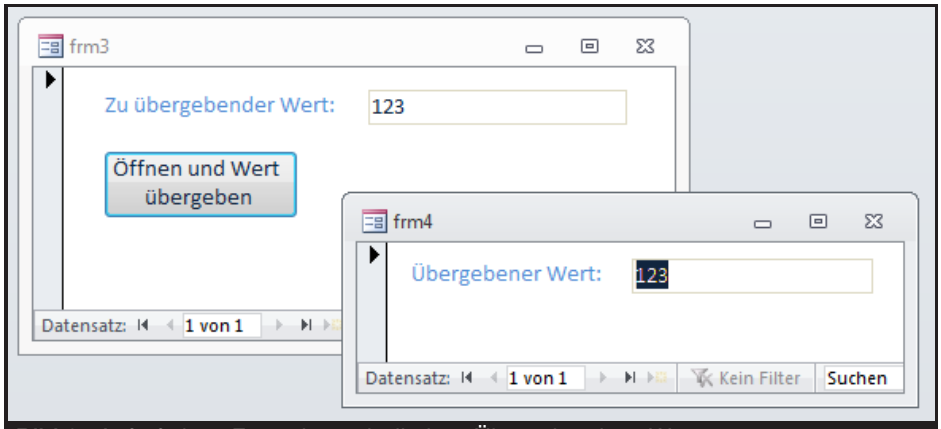



**Bild 2:** Aufruf eines Formulars mit direkter Übergabe eines Wertes

```
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

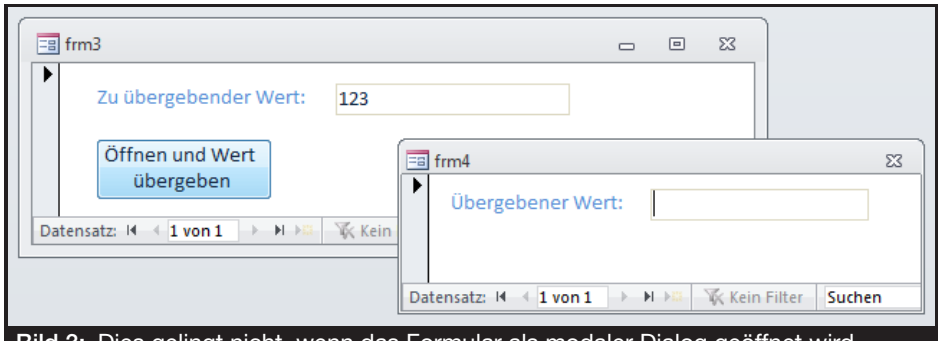



**Bild 3:** Dies gelingt nicht, wenn das Formular als modaler Dialog geöffnet wird.

# 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




**Bild 1:** Ausgabe der mit **Join** zusammengestellten Artikelliste

die Vorarbeit erledigt und wir müssen nur noch mit der **Join**-Funktion eine Zeichenkette aus den im Array **strArtikel**() gespeicherten Artikelnamen erstellen. Als Trennzeichen verwenden wir dabei das Komma, gefolgt von einem Leerzeichen. Die Ausgabe im Meldungsfenster sieht wie in Bild 1 aus.

## Die Split-Funktion

Die **Split**-Funktion arbeitet im Prinzip genau umgekehrt wir die **Join**-Funktion: Sie erwartet als ersten Parameter eine aufzuteilende Zeichenkette und als zweiten eine weitere Zeichenkette, welche das Trennzeichen enthält, nach dem die Zeichenkette aus dem ersten Parameter aufgeteilt werden soll. Wenn Sie also beispielsweise die obige Zeichenkette **Artikel1, Artikel2, Artikel3, Artikel4** aufteilen und das die einzelnen Elemente in ein Array einfügen wollen, verwenden Sie den folgenden Aufruf:

```
strArtikel() = Split("Artikel1, Artikel2, Artikel3, Artikel4", ", ")
```

Das Ergebnis lässt sich hier nicht so einfach abbilden, also schauen wir uns auch hier eine kleine Prozedur an, welche den Vorgang veranschaulicht – siehe Listing 2. Diese Prozedur deklariert eine Variable namens **strArtikelliste**, welche die Zeichenkette **Artikel1, Artikel2, Artikel3, Artikel4** aufnimmt. Die Variable **strArtikel** ist als Array mit dem Datentyp **String** deklariert. Mit dem Aufruf der **Split**-Funktion teilt die Prozedur die Zeichenkette jeweils an den Stellen auf, die ein Komma gefolgt von einem Leerzeichen enthalten. Die nun folgende **For...Next**-Schleife durchläuft alle Elemente des Arrays vom ersten bis zum letzten Element und gibt jeweils den Indexwert und das enthaltene Element im Direktbereich des VBA-Editors aus. Die oberste und die unterste Array-Grenze ermittelt die **For...Next**-Schleife dabei mit den Funktionen **UBound** und **LBound**. Das Ergebnis im Direktbereich sieht schließlich wie folgt aus:

```
Public Sub Beispiel_Split()
    Dim strArtikel() As String
    Dim strArtikelliste As String
    Dim i As Integer
    strArtikelliste = "Artikel1, Artikel2, Artikel3, Artikel4"
    strArtikel() = Split(strArtikelliste, ", ")
    For i = LBound(strArtikel) To UBound(strArtikel)
        Debug.Print i, strArtikel(i)
    Next i
End Sub
```

**Listing 2:** Aufteilen einer Zeichenkette in die einzelnen Elemente eines Arrays

```
Public Function AnzahlWoerterErmitteln(strZeichenkette As String) As Long
    Dim strArray() As String
    strArray() = Split(strZeichenkette, " ")
    AnzahlWoerterErmitteln = UBound(strArray) - LBound(strArray) + 1
End Function
```

**Listing 3:** Zählen der Wörter einer Zeichenkette

```
0   Artikel1
1   Artikel2
2   Artikel3
3   Artikel4
```

Ein Anwendungsbereich, den Sie vielleicht bereits im Artikel **Von Formular zu Formular mit OpenArgs** kennengelernt haben ist die Übergabe mehrerer Werte mit dem **OpenArgs**-Parameter. Dort fügen wir im aufrufenden Formular ein oder mehrere Texte mit einem entsprechenden Zeichen zu einer Zeichenkette zusammen, etwa **Text1|Text2|Text3**. Im aufgerufenen Formular nehmen wir diese Zeichenkette dann mit der **Split**-Anweisung auseinander und greifen dann über den Index des Array auf die einzelnen Elemente zu.

## Anzahl der Wörter bestimmen

Sie können die **Split**-Funktion auch dazu nutzen, die Anzahl der Wörter einer Zeichenkette zu ermitteln. Dies erledigen Sie mit der Funktion aus Listing 3. Diese erwartet eine Zeichenkette aus einem oder mehreren Wörtern als Parameter. Die Zeichenkette wird mit der **Split**-Funktion überall dort, wo der Text ein Leerzeichen enthält, in die einzelnen Elemente eines Arrays aufgeteilt. Der Wert des oberen Index minus dem Wert des unteren Index plus **1** entspricht der Anzahl der durch ein Leerzeichen voneinander getrennten Wörter. Dies gelingt auch mit einer leeren Zeichenkette, da **UBound** dann den Wert **-1** liefert.

# 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

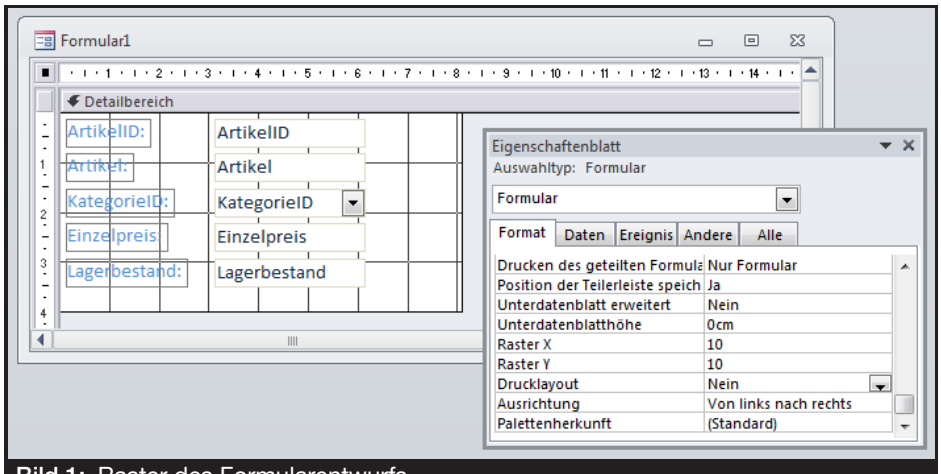



**Bild 1:** Raster des Formularentwurfs

diesen werden die Steuerelemente dann bei Auswahl von **Ausrichten|Am Raster** ausgerichtet. Manchmal zeigt Access das feinere Raster durch Punkte an – siehe Bild 2.

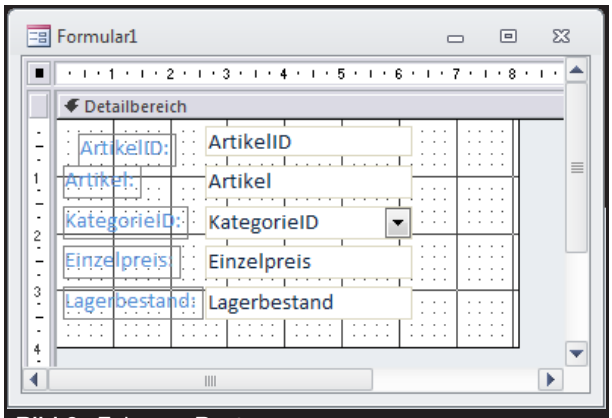


**Bild 2:** Feineres Raster

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!

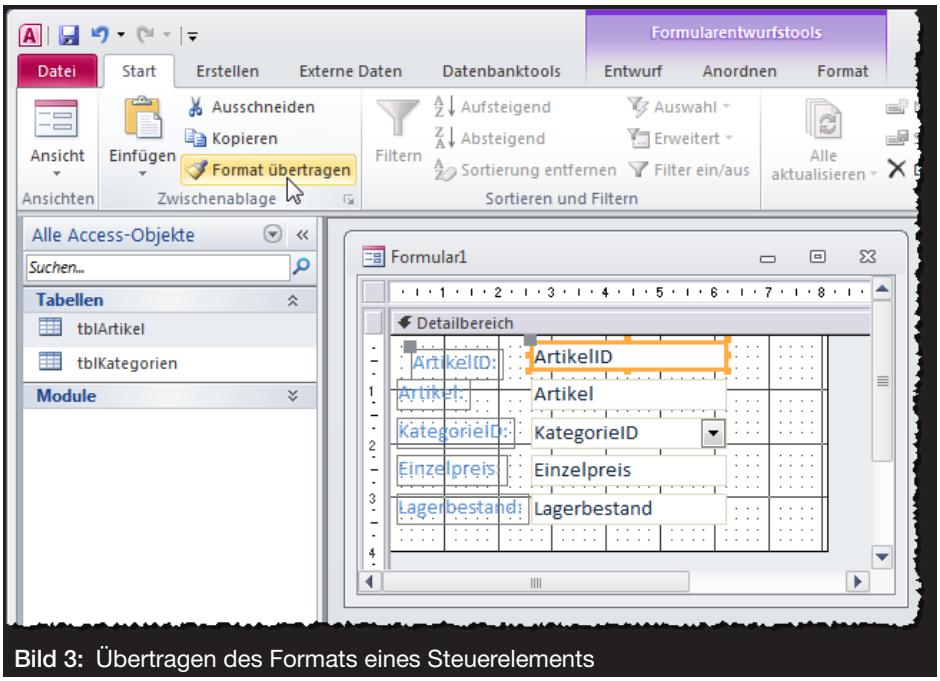


**Bild 3:** Übertragen des Formats eines Steuerelements

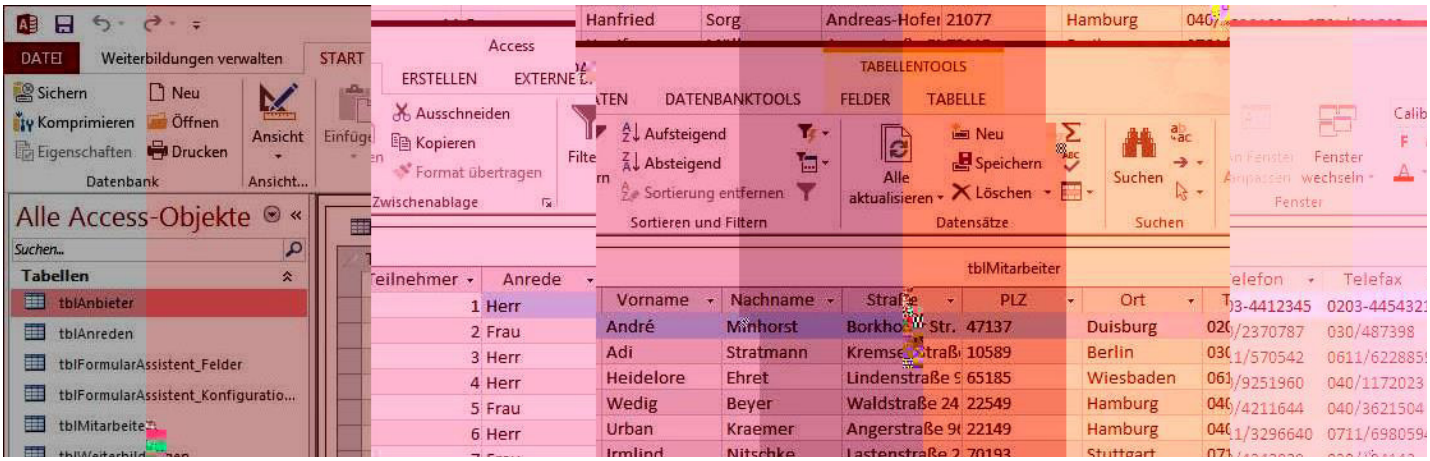
Wenn Sie gleich mehrere Steuerelemente mit dieser Formatierung aussstatten wollen, klicken Sie im zweiten Schritt doppelt auf **Format übertragen**. Der Eintrag im Ribbon bleibt dann hervorgehoben, bis Sie ihn erneut anklicken oder die **Escape**-Taste betätigen.

In der Zwischenzeit können Sie die zu formatierenden Steuerelemente nacheinander anklicken.

Ausgabe 08/2014

# ACCESS BASICS

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




## In diesem Heft:

### ACCESS-BERICHTE DRUCKEN PER VBA

LERNEN SIE, WELCHE FALLSTRICKE ES BEIM AUSDRUCKEN VON BERICHTEN PER VBA GIBT UND VERHINDERN SIE GÄNGIGE FEHLER!

SEITE 3

### ANWENDUNGSEINSTELLUNGEN ABSPEICHERN

SPEICHERN SIE EINSTELLUNGEN ETWA ZUM AUSSEHEN DER BENUTZEROBERFLÄCHE IHRER ANWENDUNG UND STELLEN SIE DIESE PER VBA WIEDER HER!

SEITE 10

### REGISTRY-ZUGRIFF PER VBA

SPEICHERN SIE DIE EINSTELLUNGEN EINER DATENBANK AUSSERHALB DIESER DATENBANK - UND ZWAR IN DER REGISTRY!

SEITE 16

# 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**.

### Drucken über Menüelemente

Öffnen Sie einen Bericht in der Seitenansicht, so präsentiert Ihnen Access ab Version 2007 ein Ribbon wie in Bild 1.

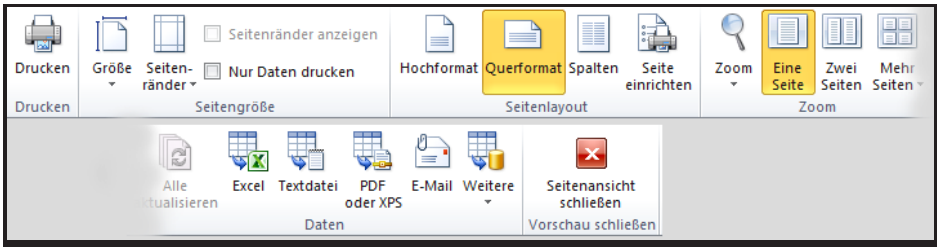



**Bild 1:** Ribbon bei Seitenansicht eines Berichts in Access 2010

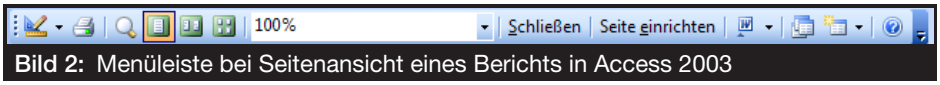



**Bild 2:** Menüleiste bei Seitenansicht eines Berichts in Access 2003

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.

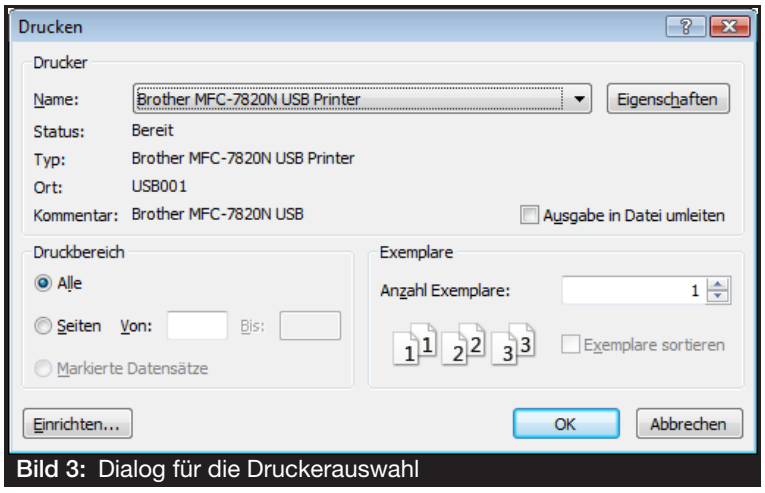



**Bild 3:** Dialog für die Druckerauswahl

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.

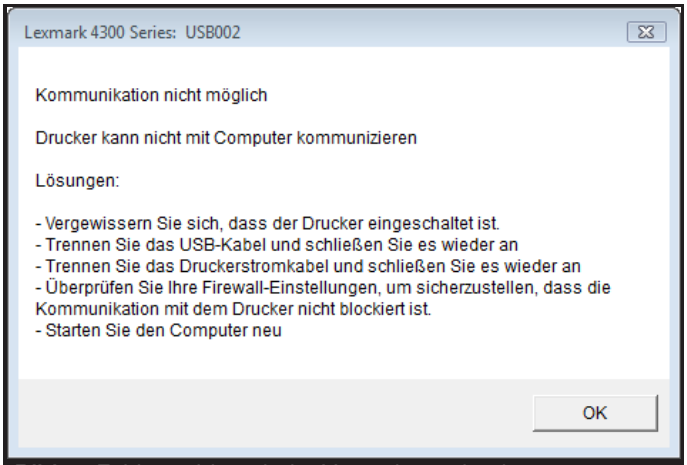


**Bild 4:** Fehlermeldung beim Versuch, zu drucken

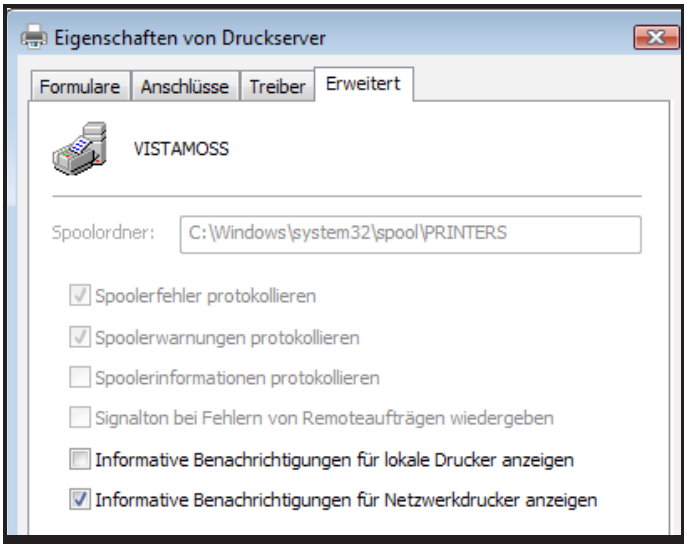


**Bild 5:** Systemeinstellungen für den Druckauftrag-Spooler von Windows

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.

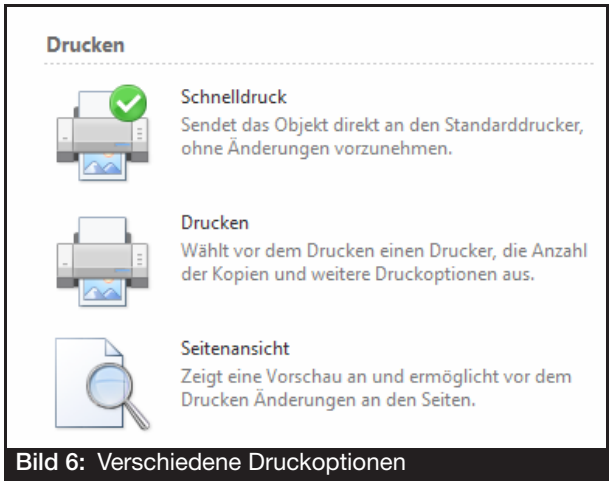


**Bild 6:** Verschiedene Druckoptionen

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-

# 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**

# 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

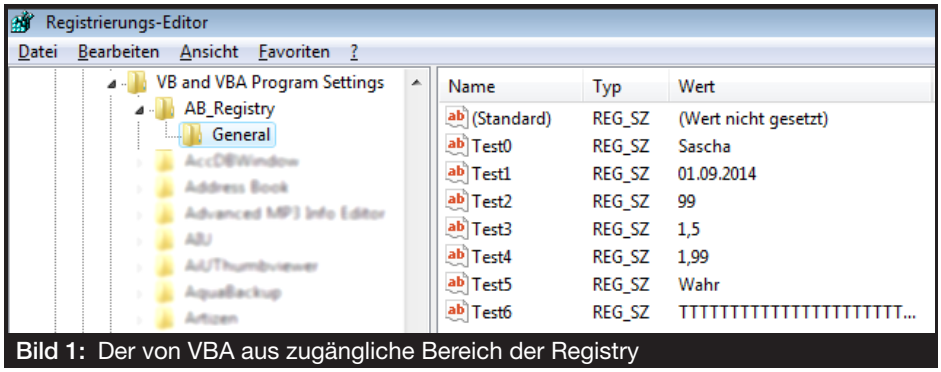


**Bild 1:** Der von VBA aus zugängliche Bereich der Registry

```
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**

Ausgabe 09-10/2014

# ACCESS BASICS

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



## In diesem Heft:

### ZUFALLSZAHLEN ERZEUGEN UND NUTZEN

ERFAHREN SIE, WIE SIE UNTER VBA ZUFALLSZAHLEN ERZEUGEN UND DIESE IN DER PRAXIS EINSETZEN!

SEITE 11

### FILTERN UND SUCHEN

LERNEN SIE DIE EINGEBAUTEN FUNKTIONEN ZUM FILTERN UND SUCHEN UNTER ACCESS KENNEN – ES GIBT MEHR MÖGLICHKEITEN, ALS MAN AUF DEN ERSTEN BLICK ERKENNT!

SEITE 3

### EX- UND IMPORT ZU MICROSOFT WORD

ERFAHREN SIE, WIE SIE DATEN AUS ACCESS NACH WORT EXPORTIEREN UND UMGEKEHRT – INKLUSIVE SERIENDRUCK VON ACCESS AUS!

SEITE 33

# 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.

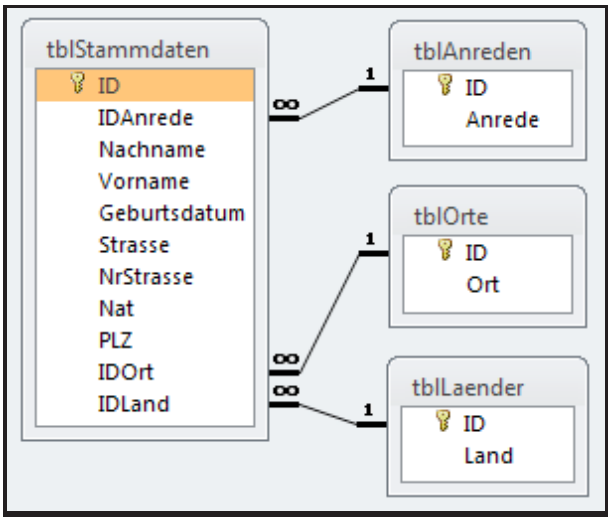



**Bild 2:** Beziehungslayout der Beispieltdatenbank

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

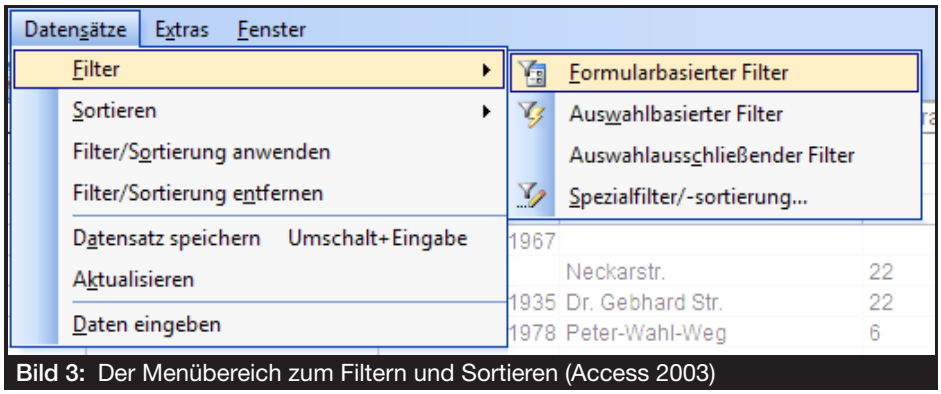


**Bild 3:** Der Menübereich zum Filtern und Sortieren (Access 2003)

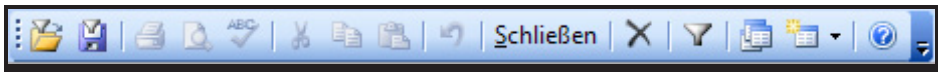


**Bild 4:** Die optionale Symbolleiste Filtern/Sortieren (Access 2003)

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.

# 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

# 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.

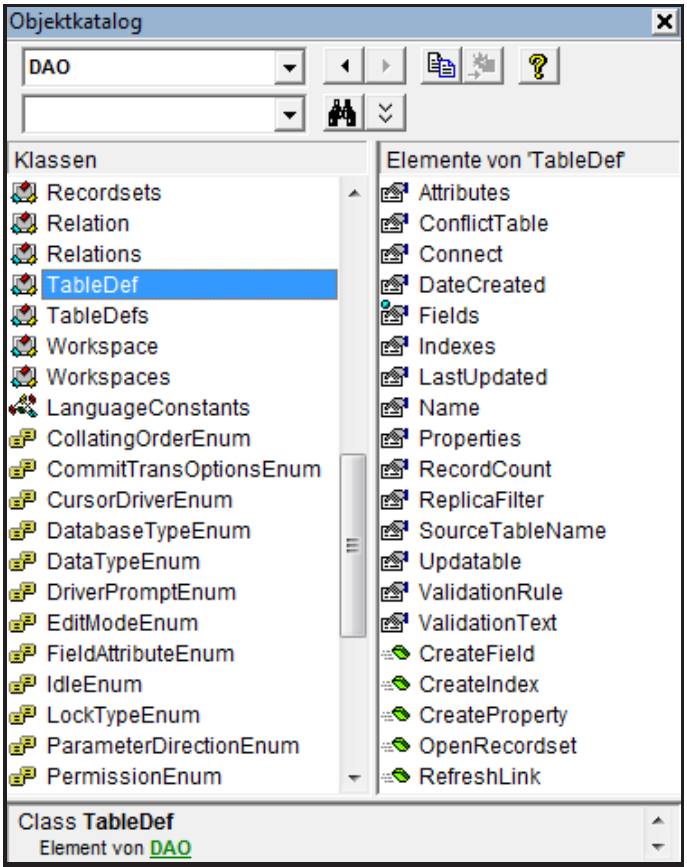



**Bild 1:** Die Klasse DAO.TableDef im VBA-Objektkatalog

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
```

# 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**

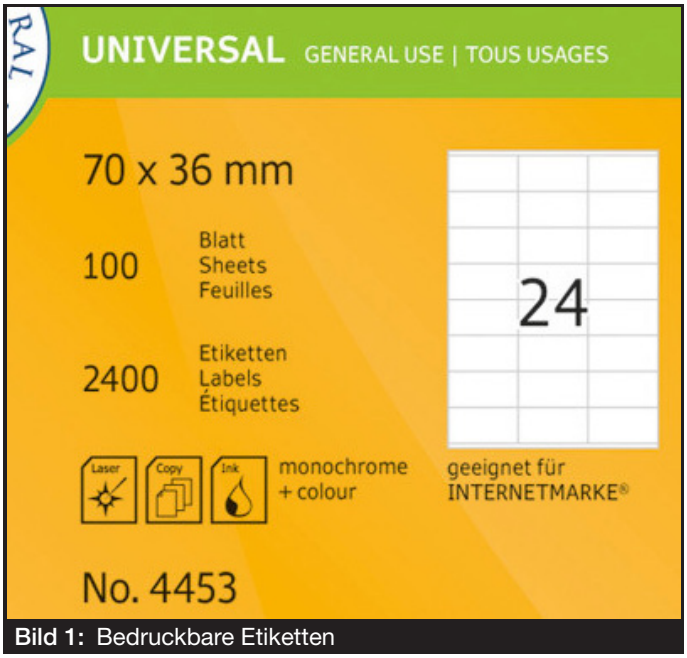


**Bild 1:** Bedruckbare Etiketten

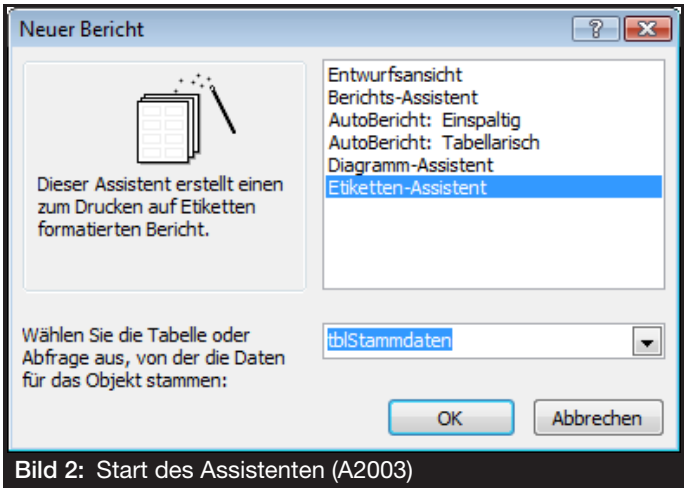


**Bild 2:** Start des Assistenten (A2003)

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.

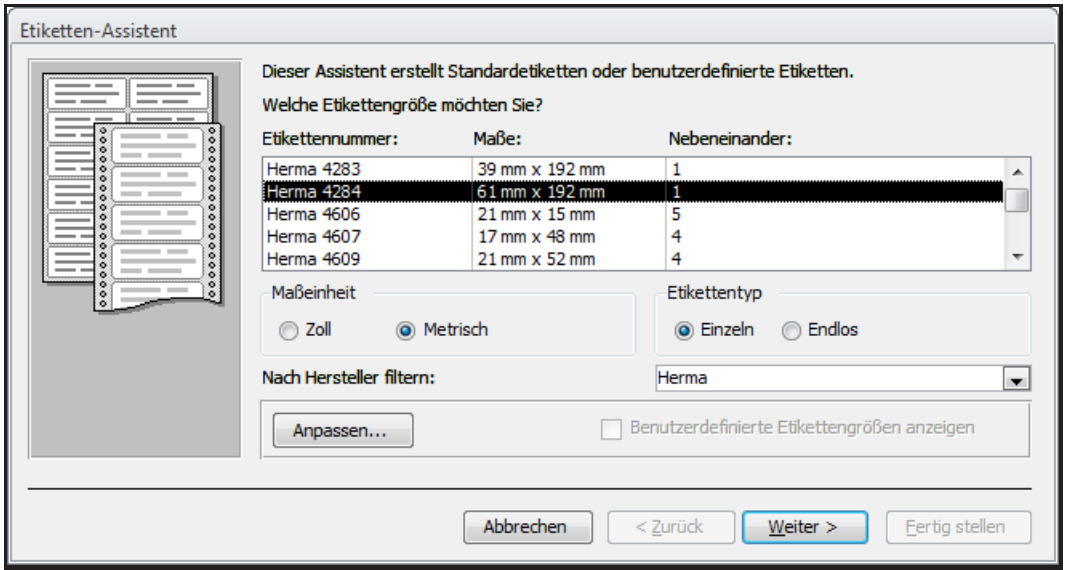


**Bild 3:** Auswahl eines Etikettenprodukts im Assistenten

## 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

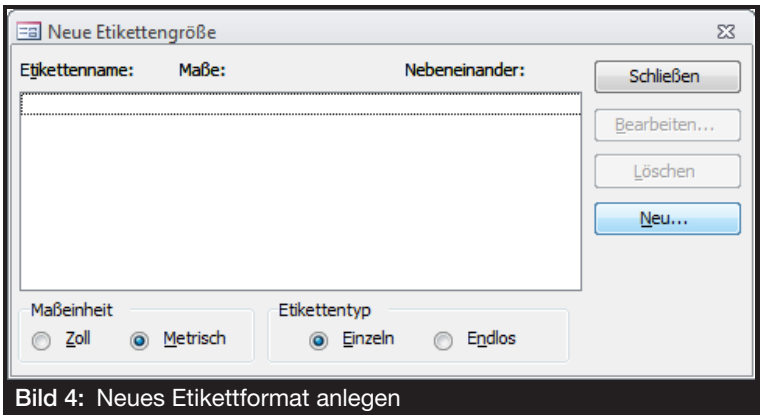


**Bild 4:** Neues Etikettformat anlegen

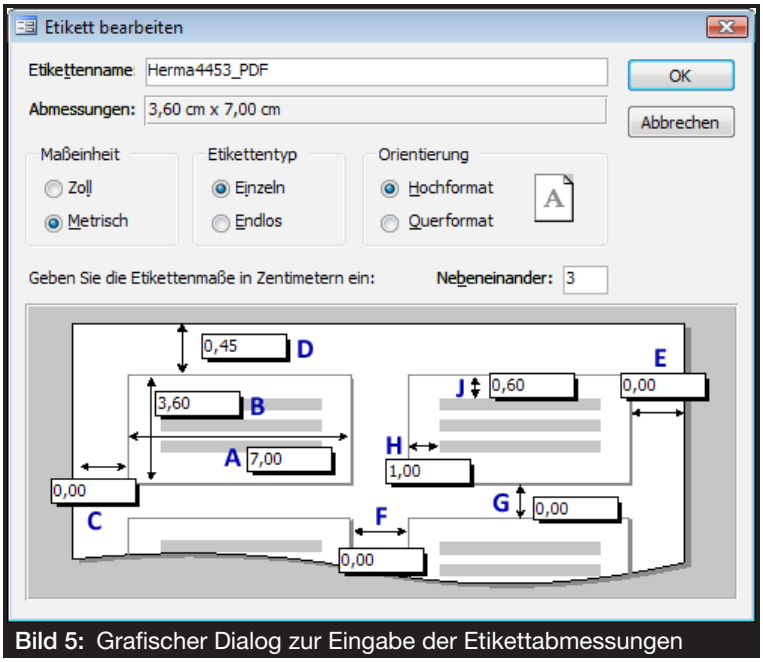


**Bild 5:** Grafischer Dialog zur Eingabe der Etikettabmessungen

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

# 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.

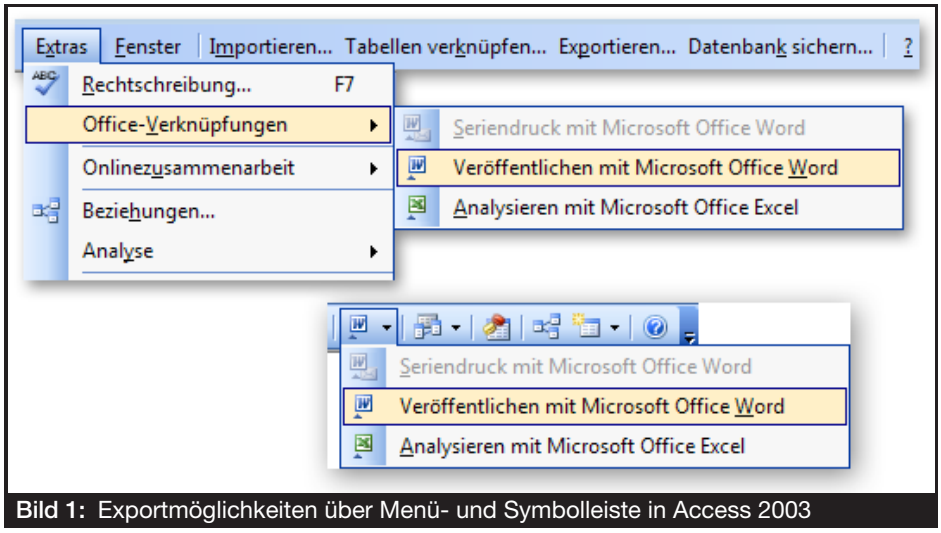



**Bild 1:** Exportmöglichkeiten über Menü- und Symbolleiste in Access 2003

### 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-

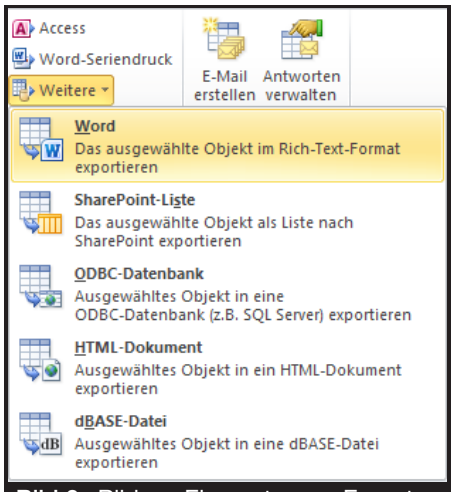



**Bild 2:** Ribbon-Elemente zum Export nach Word

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.

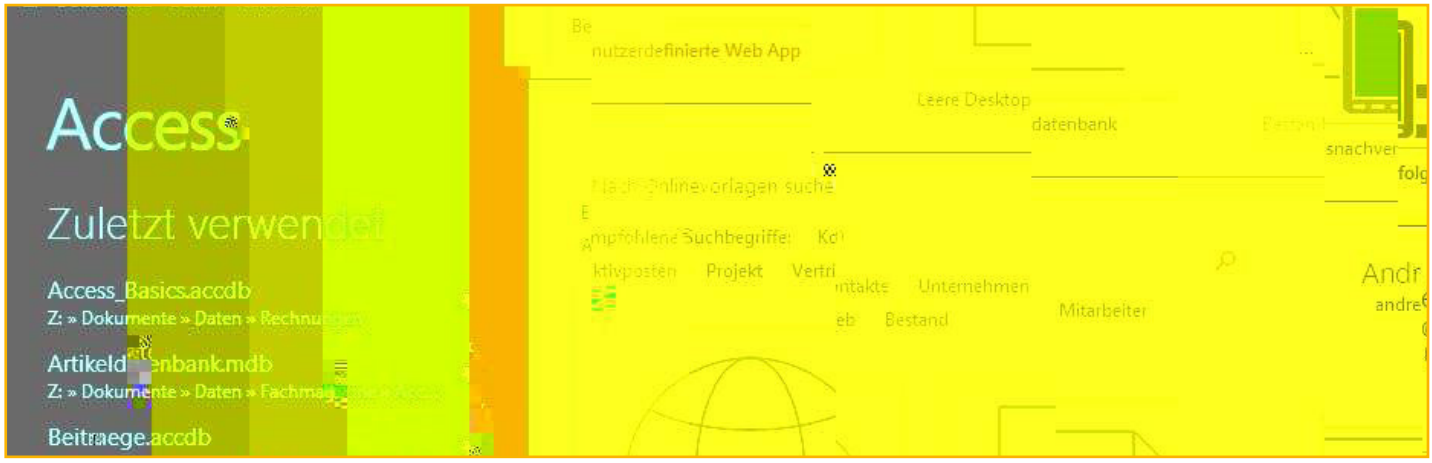
## Export per VBA

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:

Ausgabe 11-12/2014

# ACCESS BASICS

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




## In diesem Heft:


### MIT DEM VBA-EDITOR ARBEITEN

DIE ENTWICKLUNGSUMGEBUNG VON ACCESS IST ALT, ABER GUT! LERNEN SIE DIE VERSCHIEDENEN ELEMENTE KENNEN UND ERFAHREN SIE, WIE SIE DEN VBA-EDITOR ERWEITERN KÖNNEN.

SEITE 3

### ADRESSEN KONSOLIDIEREN

FAST JEDE DATENBANK VERWALTET ADRESSEN IN IRGENDEINER FORM. ERFAHREN SIE, WIE SIE DAS DATENMODELL FÜR VERSCHIEDENE ZWECKE OPTIMAL AUFBAUEN.

SEITE 9

### DIAGRAMME IN FORMULAREN

MANCHE DATEN LASSEN SICH EINFACH AM BESTEN GRAFISCH DARSTELLEN. WIR BELEUCHTEN, WIE DIES MIT ACCESS-BORDMITTELN AM BESTEN GELINGT.

SEITE 31

# 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.

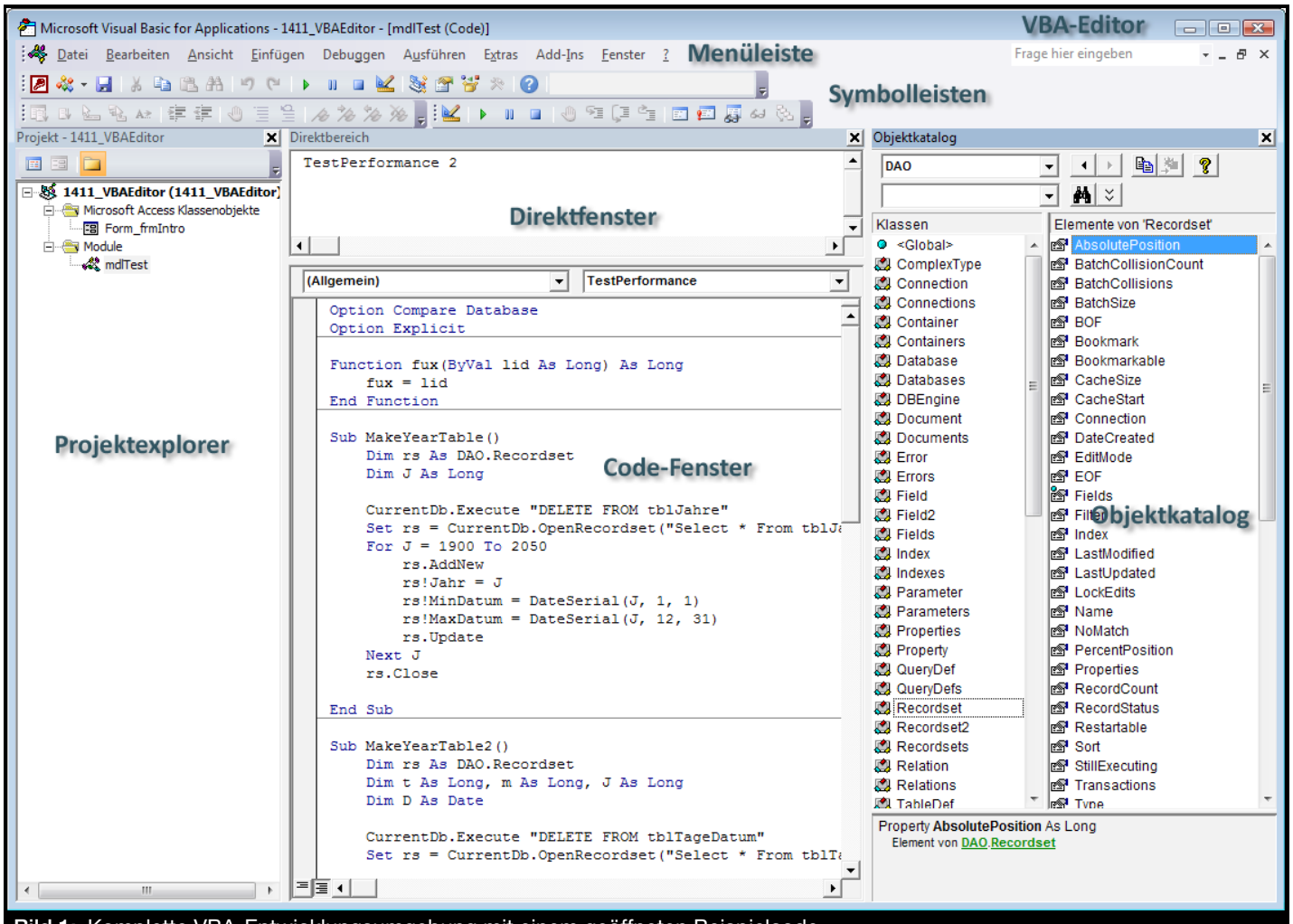



**Bild 1:** Komplette VBA-Entwicklungsumgebung mit einem geöffneten Beispielcode

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.

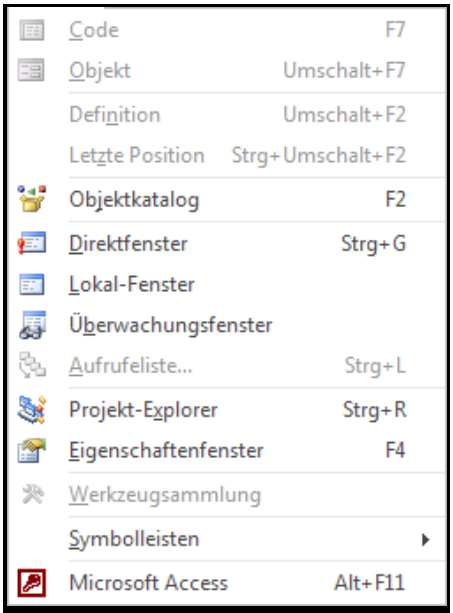


**Bild 2:** Menü Ansicht des VBA-Editors

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).

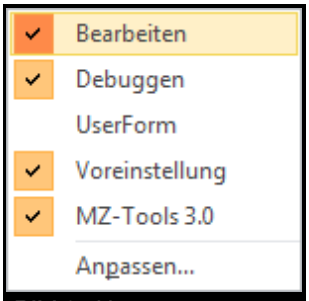


**Bild 3:** Untermenü zu Symbolleisten

## 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-

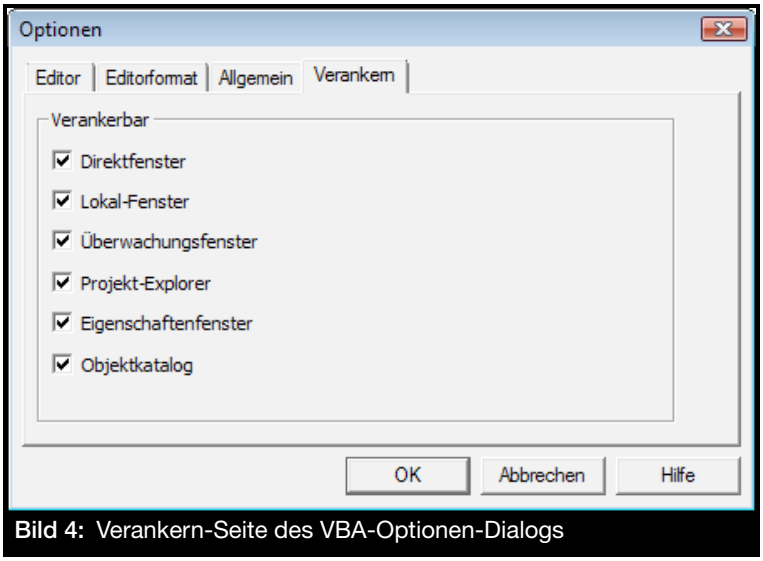


**Bild 4:** Verankern-Seite des VBA-Optionen-Dialogs

# 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).

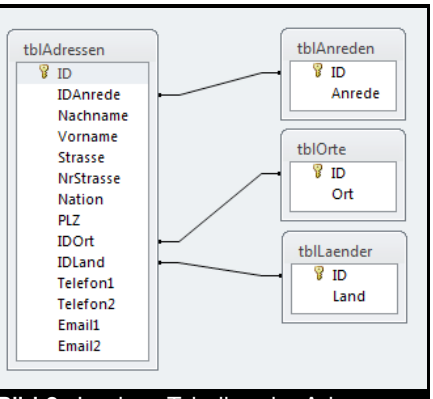
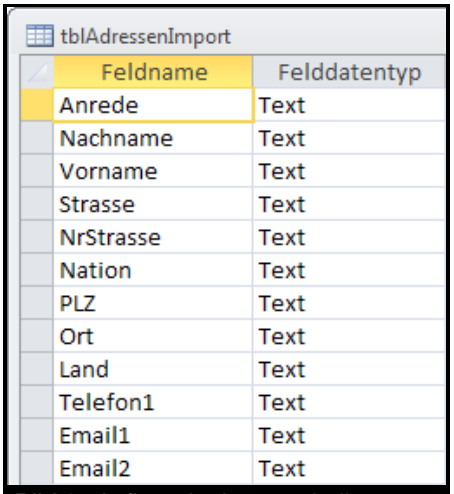



**Bild 2:** Lookup-Tabellen der Adressentabelle


tblAdressenImport

| Feldname | Felddatentyp |
|---|---|
| Anrede | Text |
| Nachname | Text |
| Vorname | Text |
| Strasse | Text |
| NrStrasse | Text |
| Nation | Text |
| PLZ | Text |
| Ort | Text |
| Land | Text |
| Telefon1 | Text |
| Email1 | Text |
| Email2 | Text |

**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.

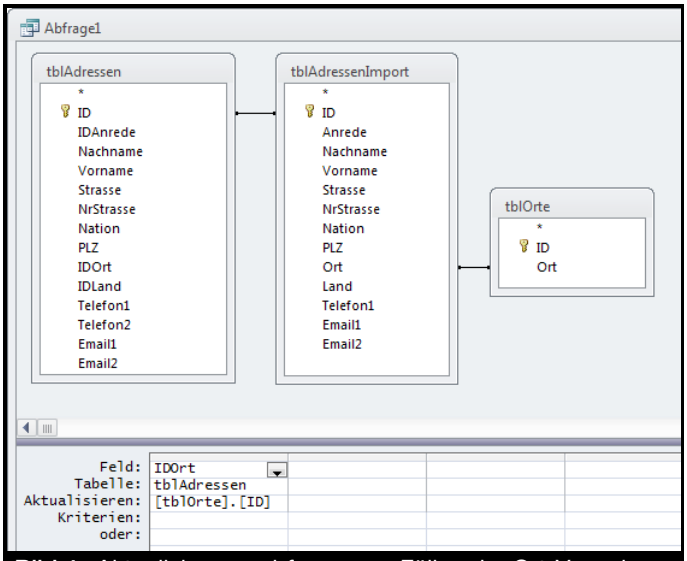



**Bild 4:** Aktualisierungsabfrage zum Füllen der Ort-Verweise

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

# 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.

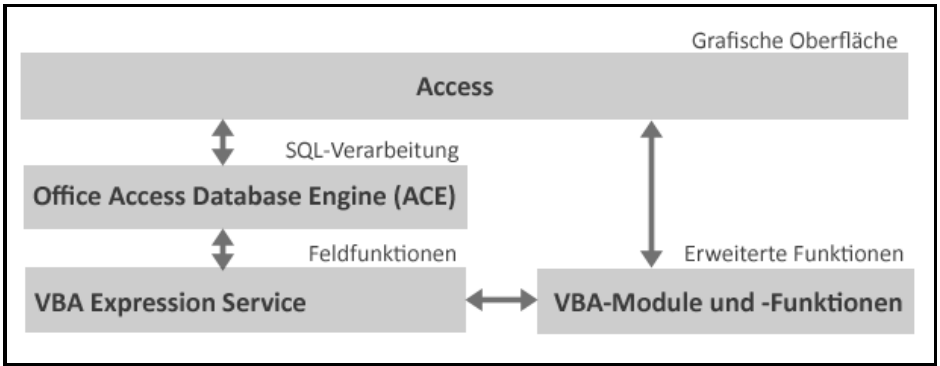



**Bild 2:** An einer Access-Abfrage beteiligte Komponenten (Schema)

## 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:

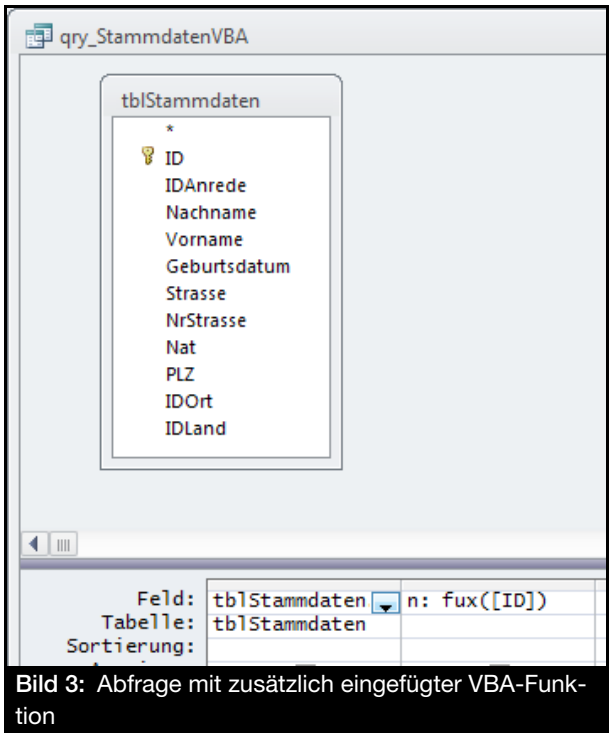



**Bild 3:** Abfrage mit zusätzlich eingefügter VBA-Funktion

```
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

# 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.

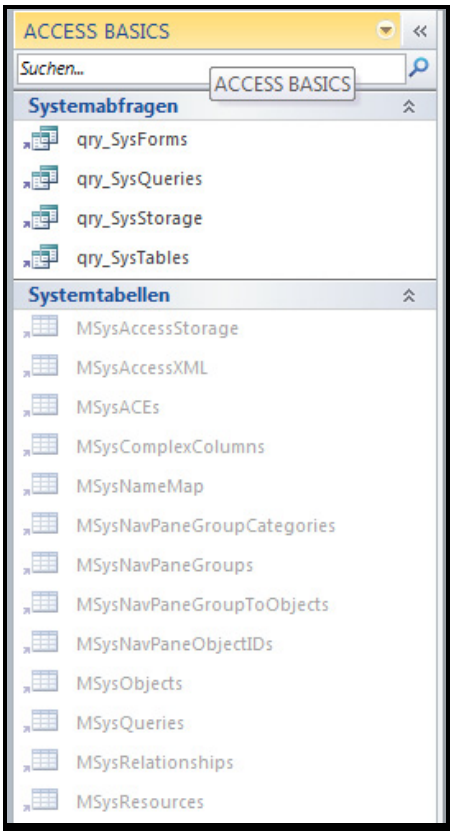


**Bild 1:** MSys-Systemtabellen

## 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

# 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.

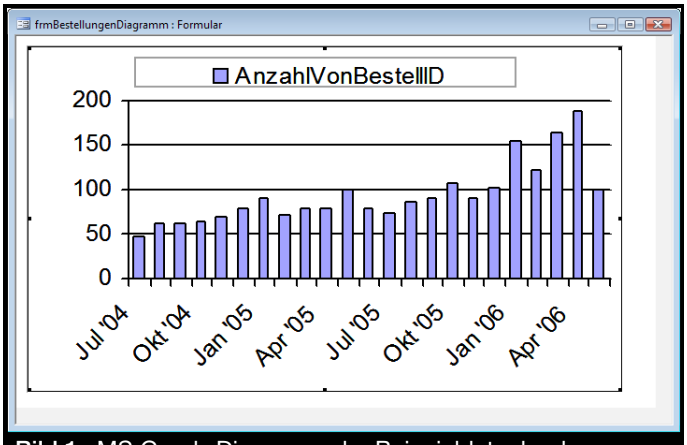



**Bild 1:** MS Graph-Diagramm der Beispieldatenbank

## 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.

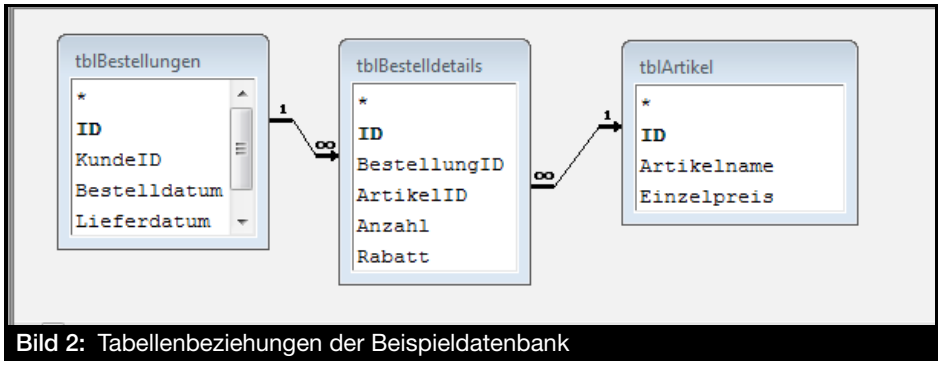



**Bild 2:** Tabellenbeziehungen der Beispieldatenbank

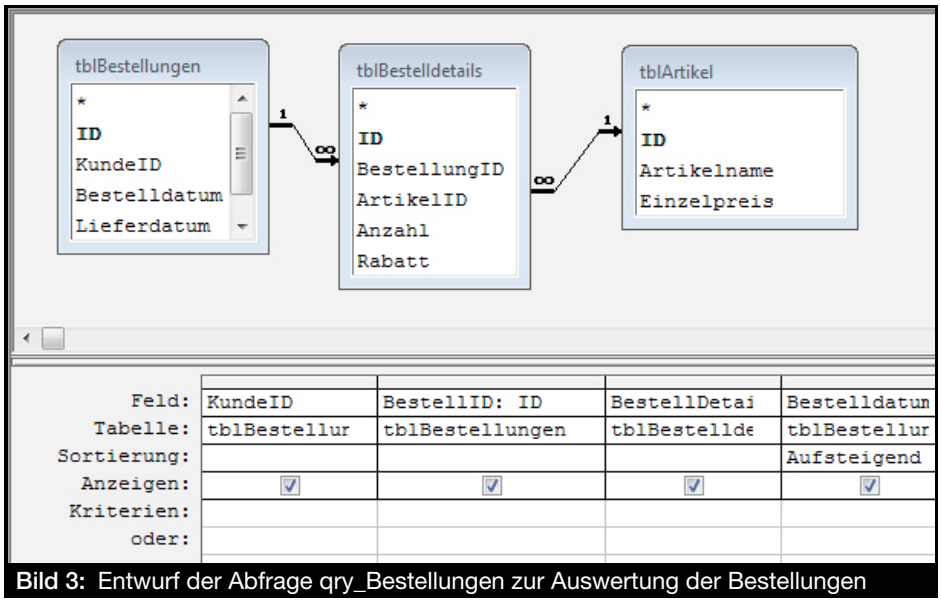



**Bild 3:** Entwurf der Abfrage qry_Bestellungen zur Auswertung der Bestellungen

## 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.

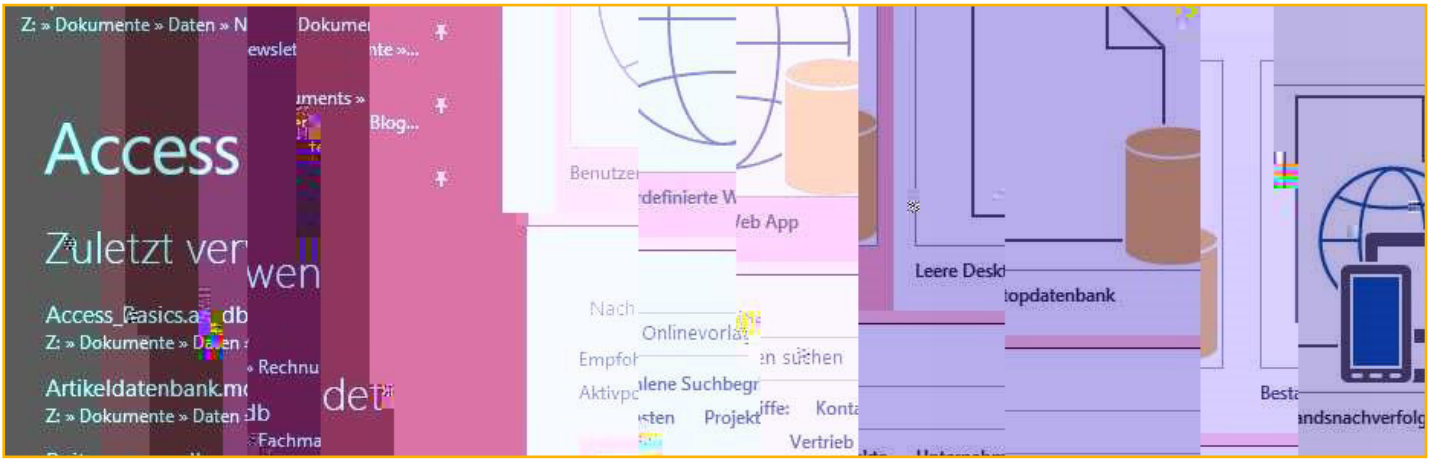
## Fehlanzeige unter Access 2013

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

Ausgabe 01/2015

# ACCESS BASICS

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




## In diesem Heft:


### DATENBANKEN UND OBJEKTE SICHERN

ERFAHREN SIE, WIE SIE DATENBANKEN UND DIE ENTHALTENEN OBJEKTE REGELMÄSSIG SICHERN, UM DATENVERLUSTEN VORZUBEUGEN..

SEITE 3

### TABELLEN UND ABFRAGEN PER DAO

TABELLEN UND ABFRAGEN VERWALTEN SIE NICHT NUR ÜBER DIE BENUTZEROBERFLÄCHE, SONDERN AUCH PER VBA. WIR ZEIGEN, WIE ES FUNKTIONIERT.

SEITE 9

### DIE WITH-ANWEISUNG VON VBA

WER IM CODE WIEDERHOLT AUF DAS GLEICHE OBJEKT ZUGREIFT, BRAUCHT DEN OBJEKTBEZUG PER WITH-SCHLÜSSELWORT NUR EINMALIG ZU DEFINIEREN.

SEITE 19

# 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

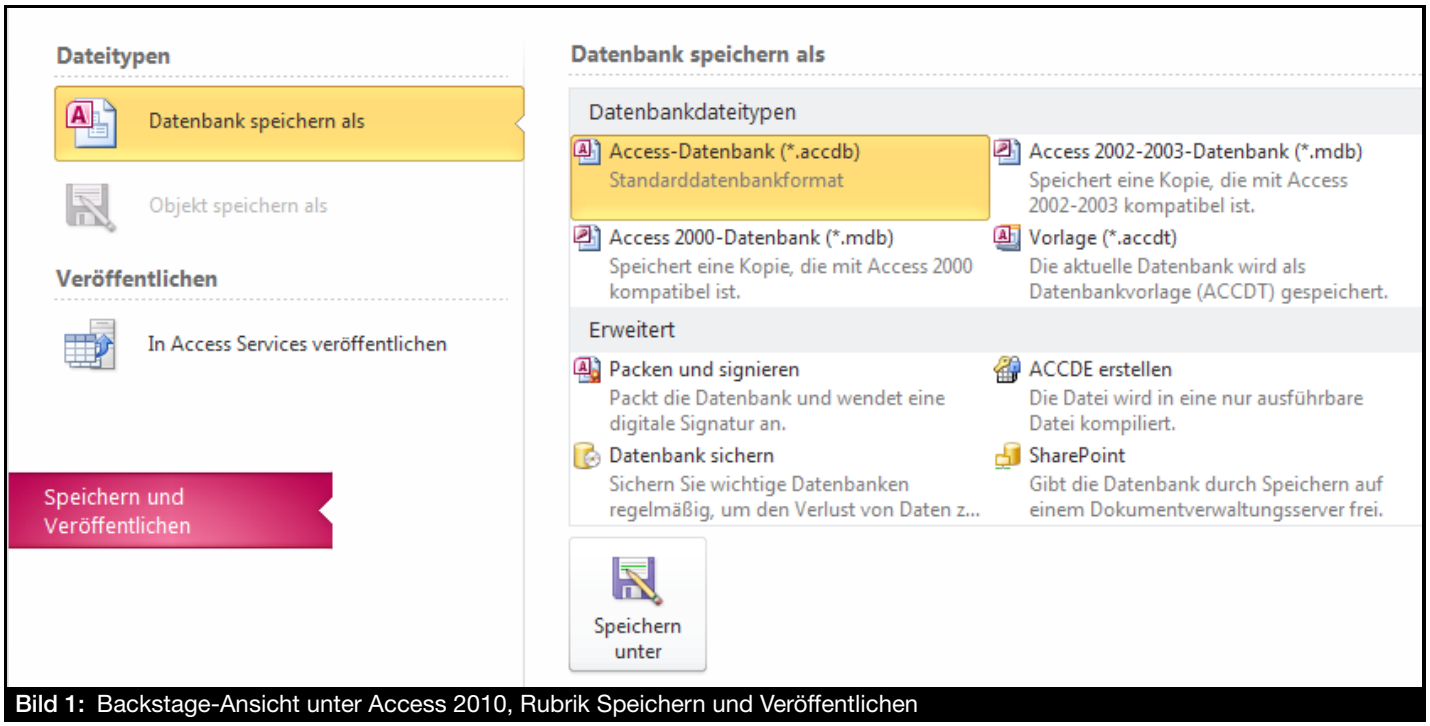



**Bild 1:** Backstage-Ansicht unter Access 2010, Rubrik Speichern und Veröffentlichen

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

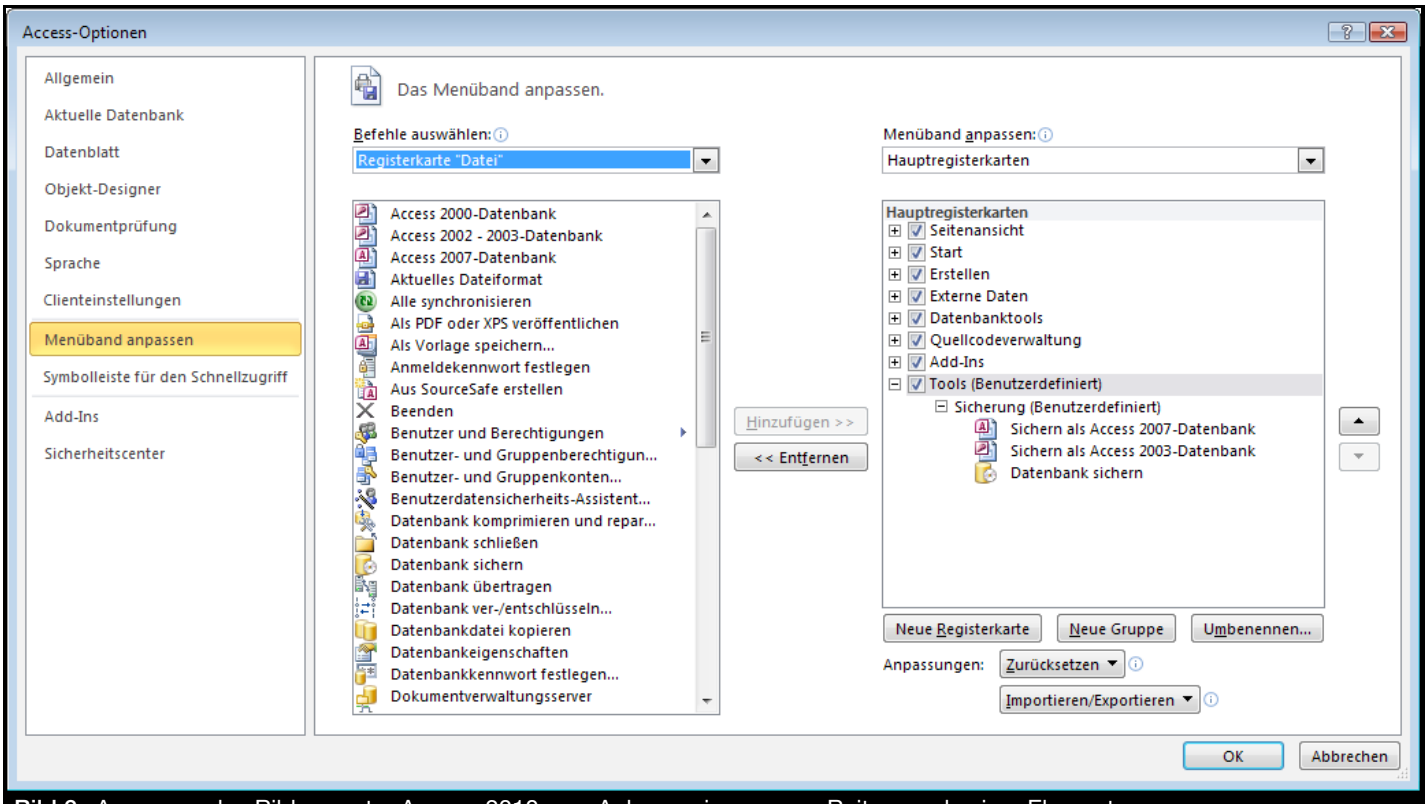



**Bild 2:** Anpassen des Ribbon unter Access 2010 zum Anlegen eines neuen Reiters und seiner Elemente

# 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.

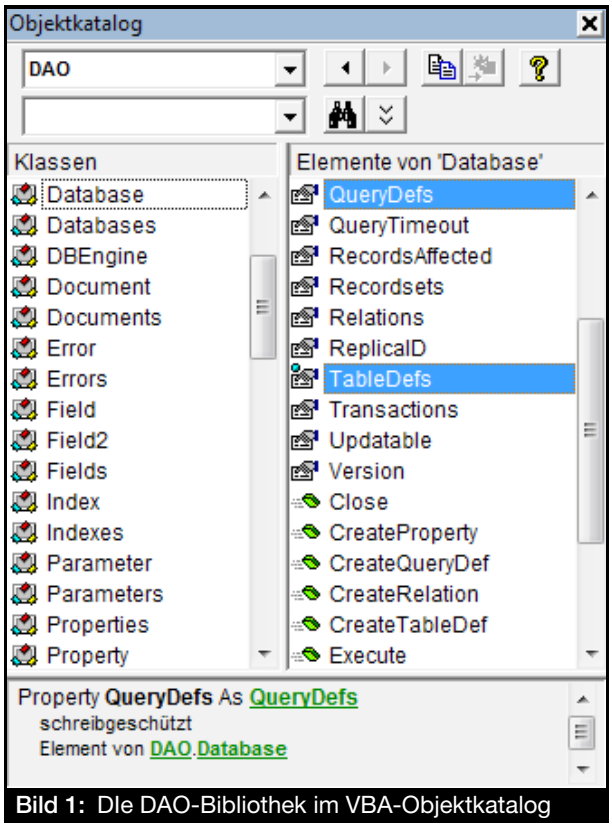



**Bild 1:** Die DAO-Bibliothek im VBA-Objektkatalog

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
```

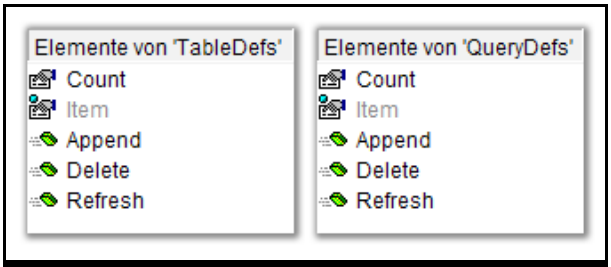



**Bild 2:** Methoden der TableDefs- und QueryDefs-Auflistungsklassen

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.

# 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**

Ausgabe 02/2015

# ACCESS BASICS

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




## In diesem Heft:


### ZIPPEN MIT BORDMITTELN

ERFAHREN SIE, WIE SIE DATEIEN MIT BORDMITTELN VON WINDOWS PER VBA PACKEN UND ENTPACKEN KÖNNEN.

SEITE 3

### NAVIGATIONSBEREICHE GESTALTEN

DEN NAVIGATIONSBEREICH AB ACCESS 2007 KÖNNEN SIE NACH EIGENEN WÜNSCHEN GESTALTEN. ERFAHREN SIE IN DIESEM ARTIKEL, WIE ES GEHT.

SEITE 11

### ACCESS ALS MP3-PLAYER

WER BEI DER ARBEIT GERN ETWAS MUSIK HÖREN MÖCHTE, KANN DIE FÄHIGKEITEN VON ACCESS MIT DEM WINDOWS MEDIA PLAYER KOMBINIEREN.

SEITE 15

# 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.

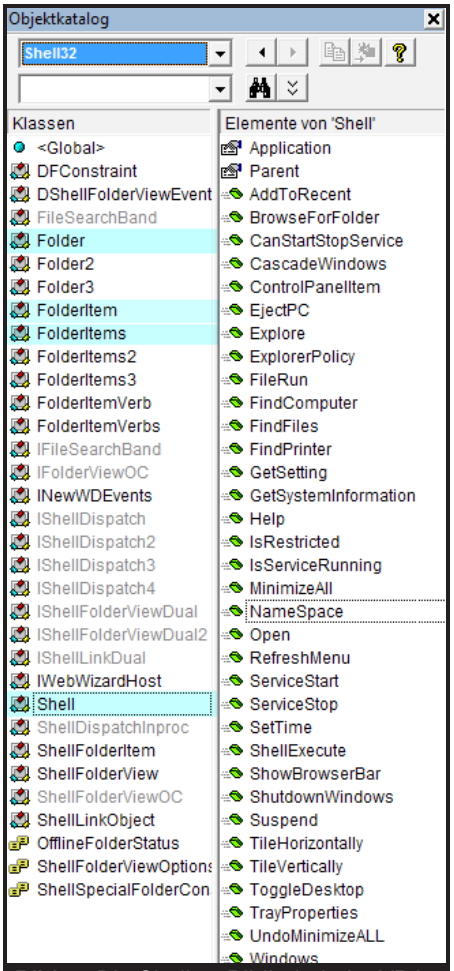



**Bild 1:** Die Shell32-Bibliothek, im VBA-Objektkatalog dargestellt

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:

# 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.

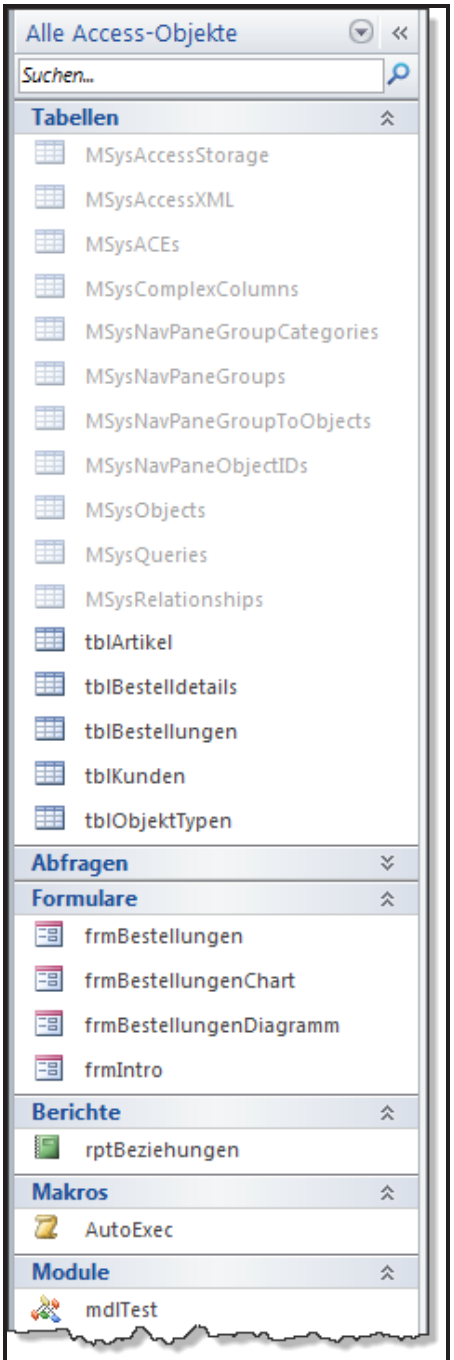



**Bild 1:** Datenbanknavigationsfenster

**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.

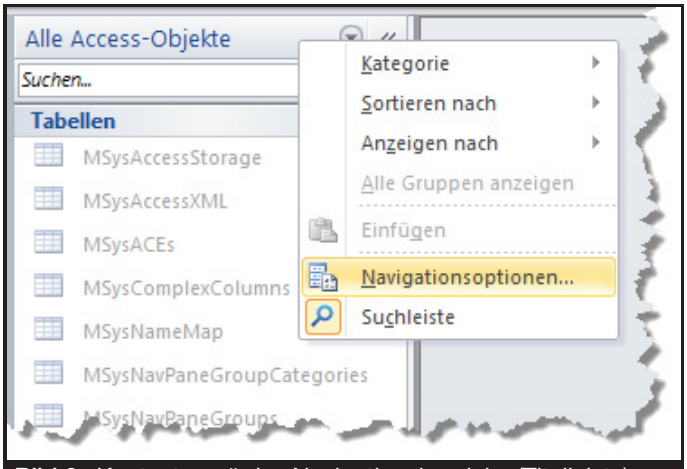



**Bild 2:** Kontextmenü des Navigationsbereichs (Titelleiste)

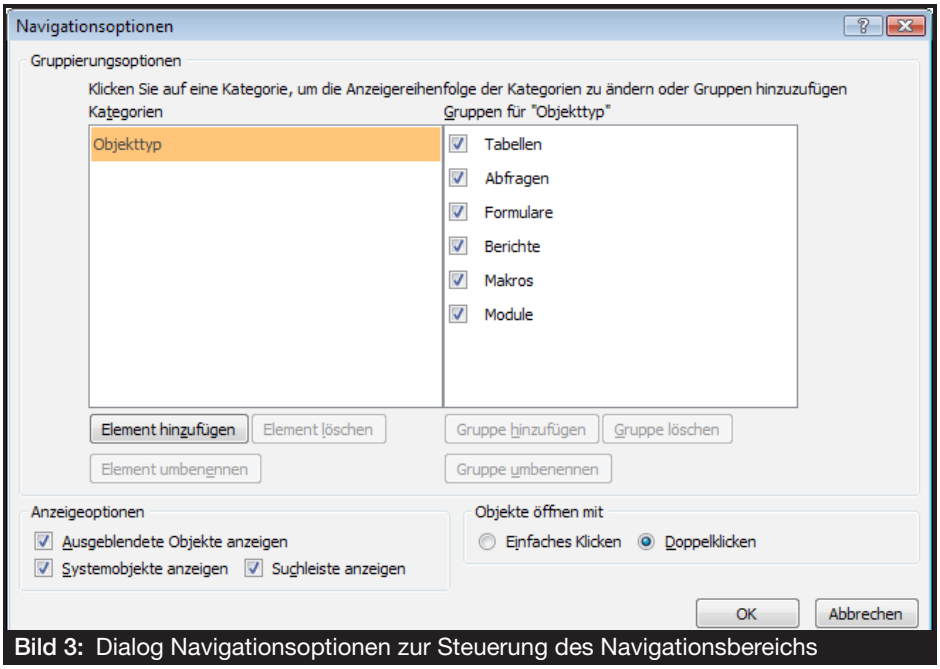



**Bild 3:** Dialog Navigationsoptionen zur Steuerung des Navigationsbereichs

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

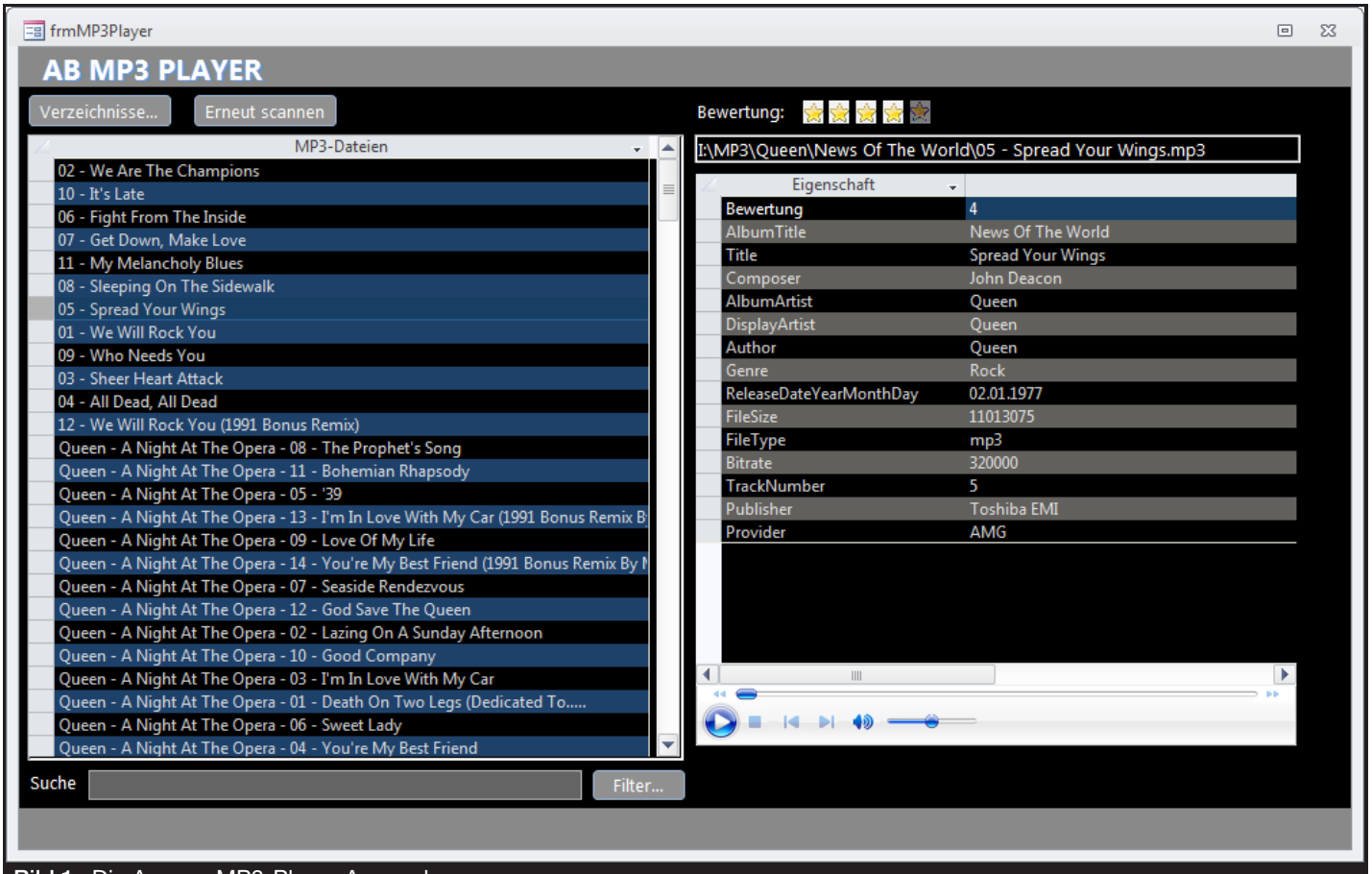


**Bild 1:** Die Access-MP3-Player-Anwendung

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-

Ausgabe 03/2015

# ACCESS BASICS

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




## In diesem Heft:


### TEXTDATEIEN EXPORTIEREN UND EINLESEN

DATEN ZWISCHEN ACCESS UND ANDEREN ANWENDUNGEN TAUSCHT MAN OFT ÜBER TEXTDATEIEN AUS. 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 ...

SEITE 3

### KODIERTE TEXTE MIT VBA UMWANDELN

DIE ASSISTENTEN VON ACCESS BIETEN VIELE EINSTELLUNGEN ZUM IMPORT UND EXPORT VON TEXTDATEIEN. DENNOCH 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. WIR ZEIGEN, WIE SIE AN SOLCHE PROBLEME HERANGEHEN!

SEITE 9

# 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.

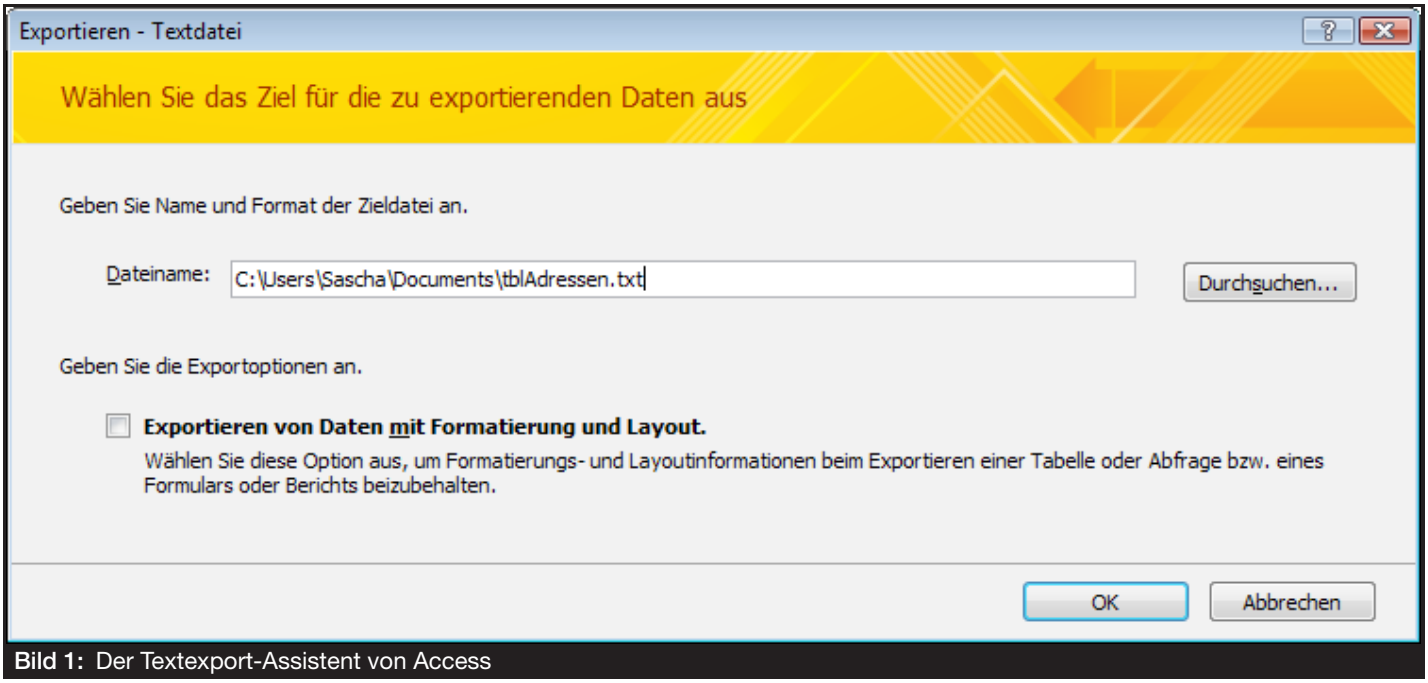


**Bild 1:** Der Textexport-Assistent von Access

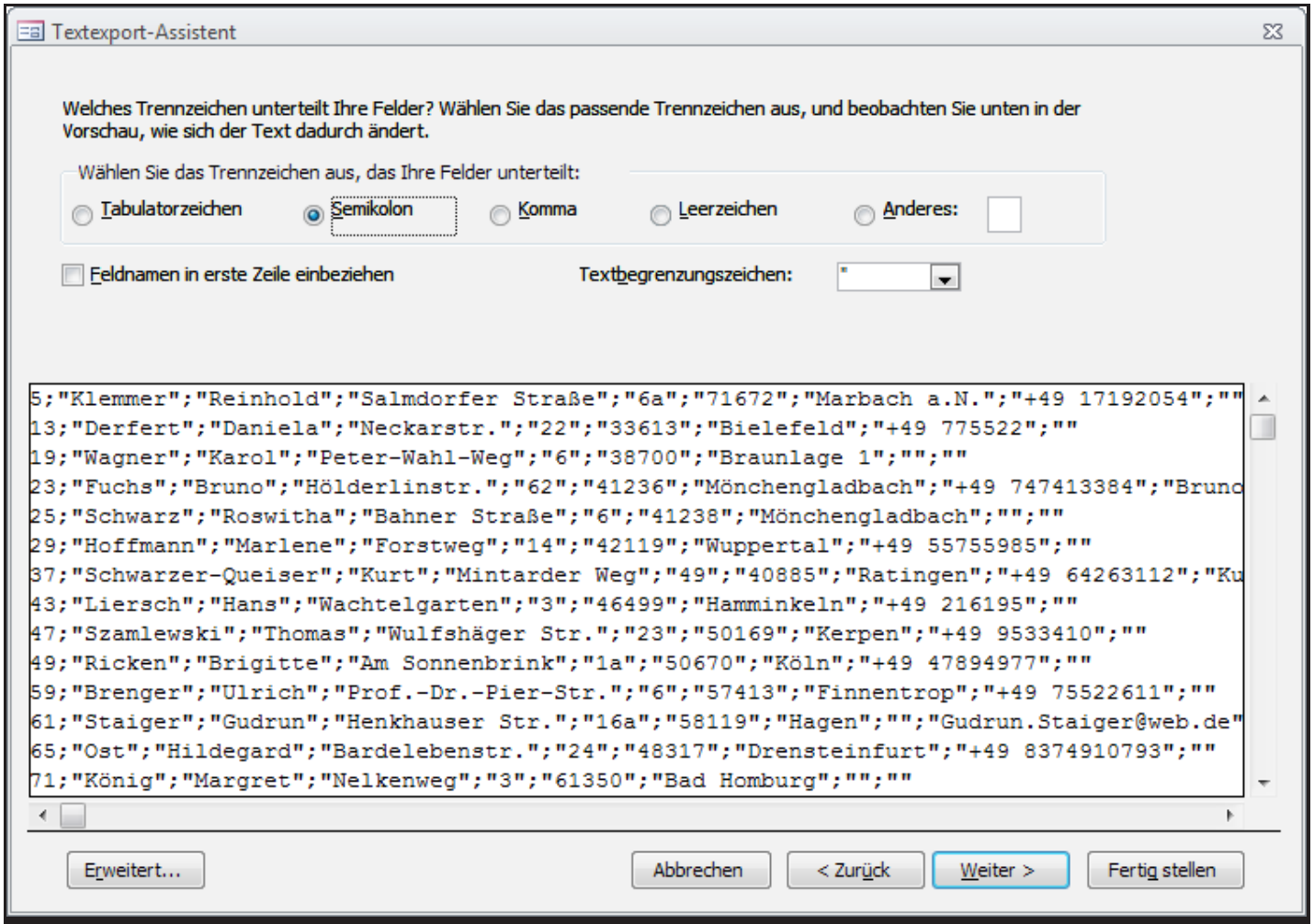



**Bild 2:** Einstellungenseite des Assistenten für den Textexport

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.

# 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.

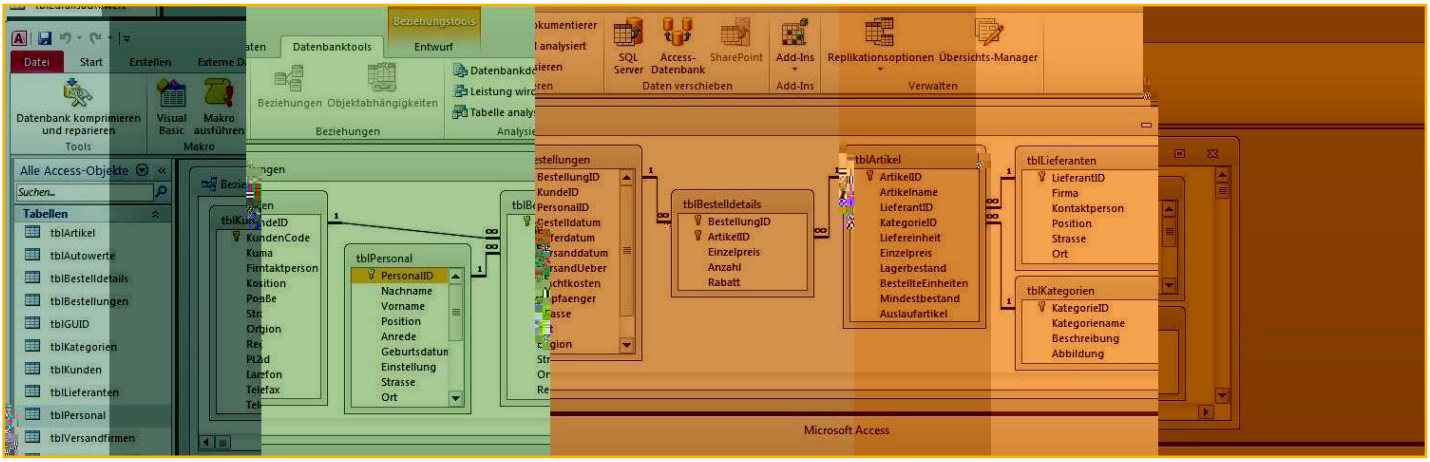
Es ist demnach ökonomischer, mehreren Sprachen eine Codepage zuzuweisen. So steht in der **CP**-Spalte der Sprachtabelle für Deutsch, Italienisch, Französisch, Spanisch, Englisch überall die **1252**. Filtern Sie die Tabelle nach diesem Wert: Erstaunlicherweise gibt es über Hundert Sprachen, die sich 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-**

Ausgabe 04/2015

# ACCESS BASICS

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




## In diesem Heft:

### 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.

SEITE 3

### 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. MEHR DAZU IN DIESEM BEITRAG!

SEITE 13

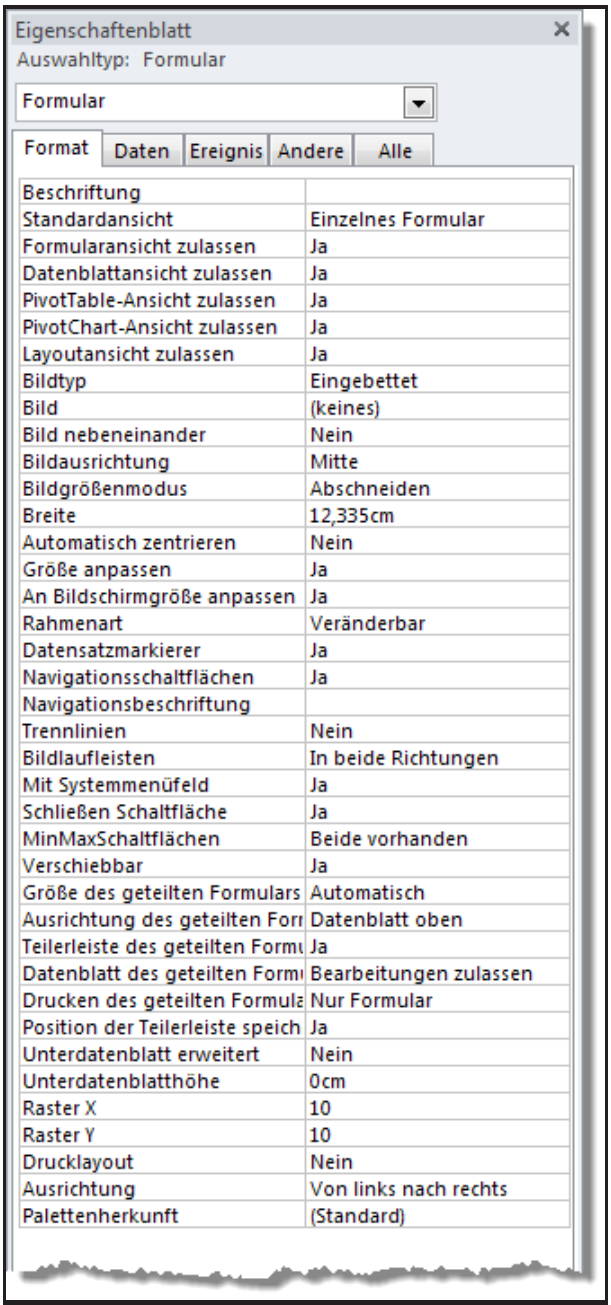
# Navigieren über Formulartricks

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.

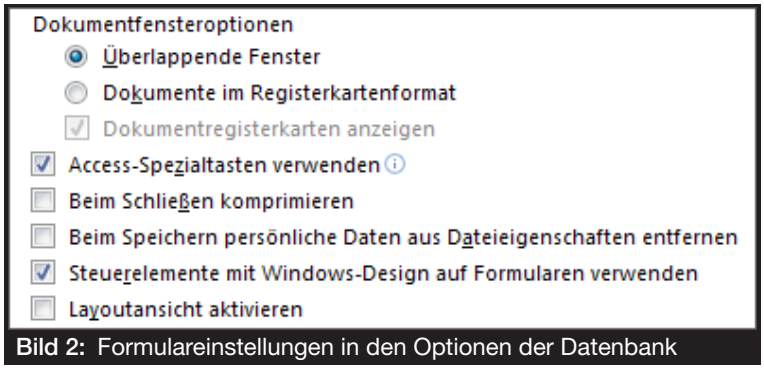


**Bild 2:** Formulareinstellungen in den Optionen der Datenbank

### 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

# 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
```

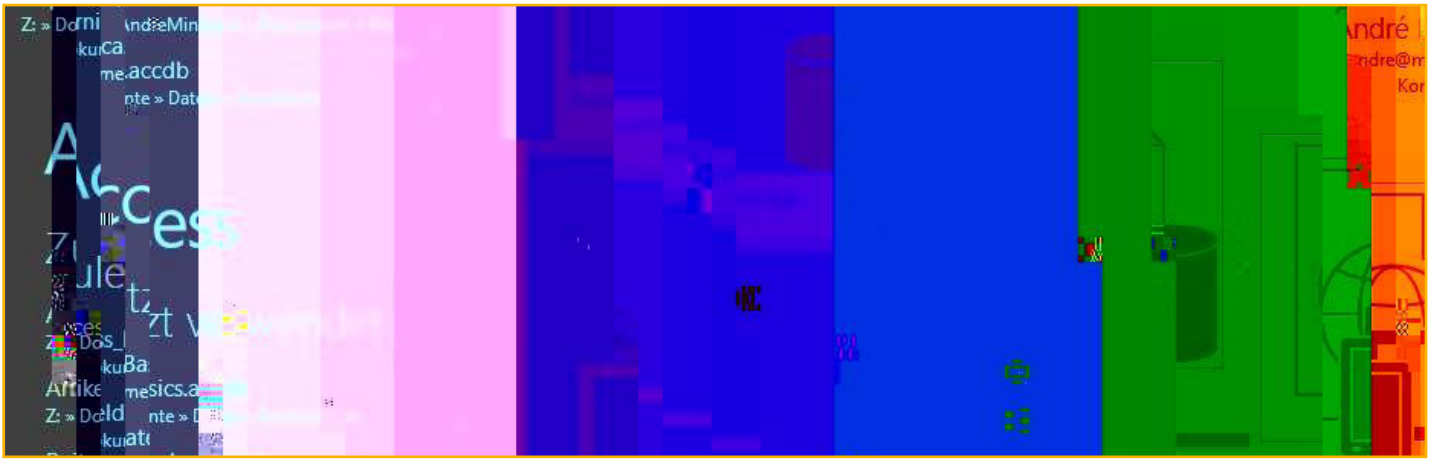
Darüber hinaus unterstützt **VBScript** nicht die Deklaration von **API**-Funktionen. Windows- oder andere DLLs können damit nicht angesprochen werden.

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**

Ausgabe 05/2015

# ACCESS BASICS

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



## In diesem Heft:

### DATEN INS WEB EXPORTIEREN

HABEN SIE SCHON EINMAL DARÜBER NACHGEDACHT, DIE DATEN IHRER DATENBANK IM HTML-FORMAT IN EINEM BROWSER DARZUSTELLEN? DIESER ARTIKEL LIEFERT DIE GRUNDLAGEN FÜR DEN EXPORT IHRER TABELLEN IN DIESES FORMAT.

SEITE 3

### REIMEN LEICHT GEMACHT

KENNEN SIE DAS? ZUR FEIER DES 70. GEBURTSTAGS IHRER MUTTER SOLLEN SIE EIN GEDICHT VERFASSEN. 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...

SEITE 14

# 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

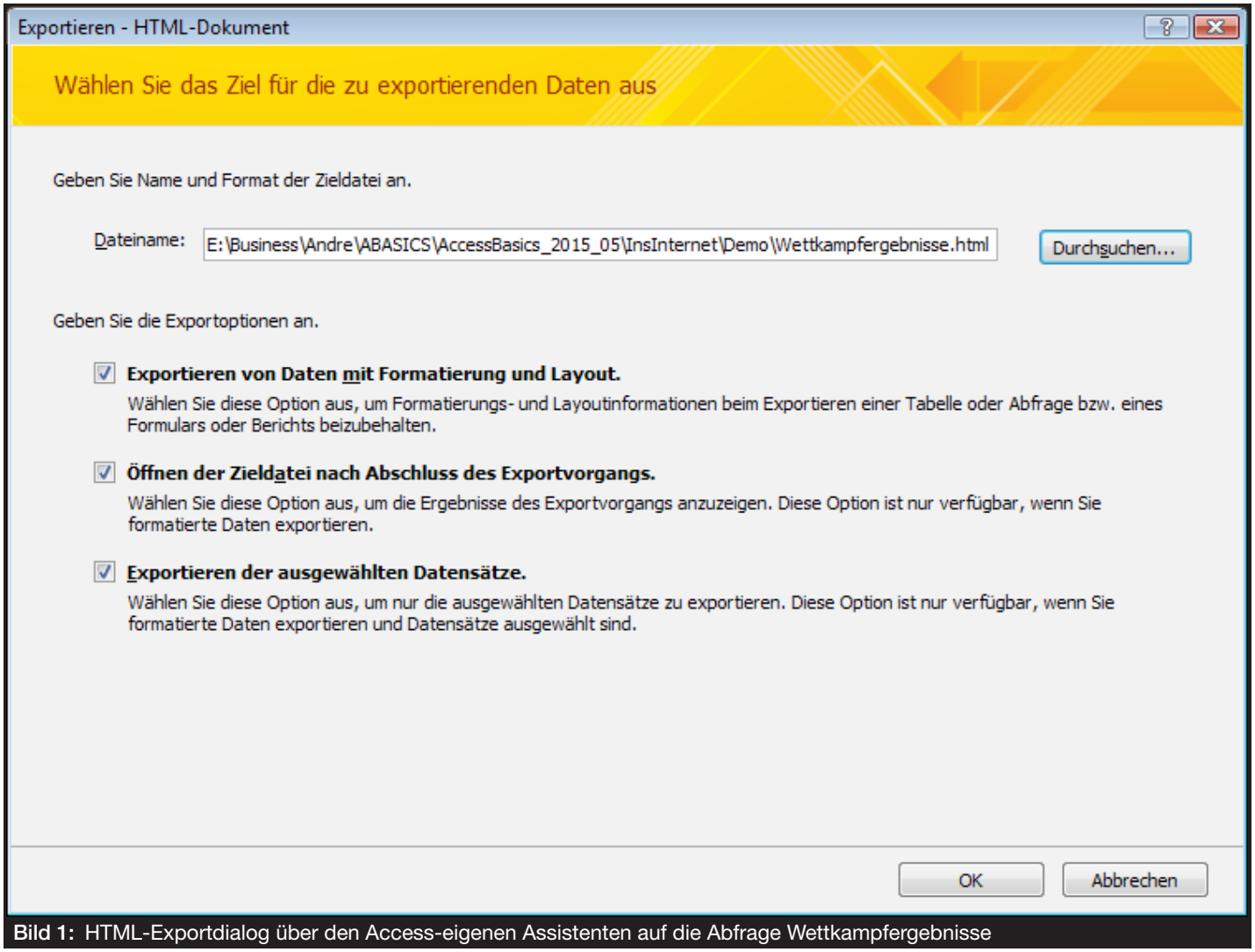



**Bild 1:** HTML-Exportdialog über den Access-eigenen Assistenten auf die Abfrage Wettkampfergebnisse

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:

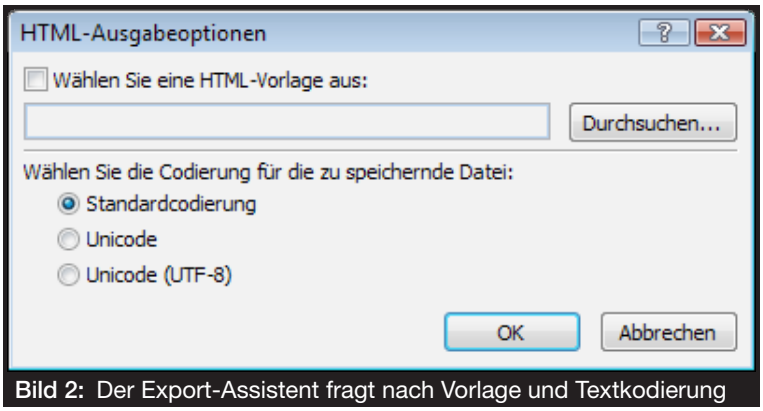



**Bild 2:** Der Export-Assistent fragt nach Vorlage und Textkodierung

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

# 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

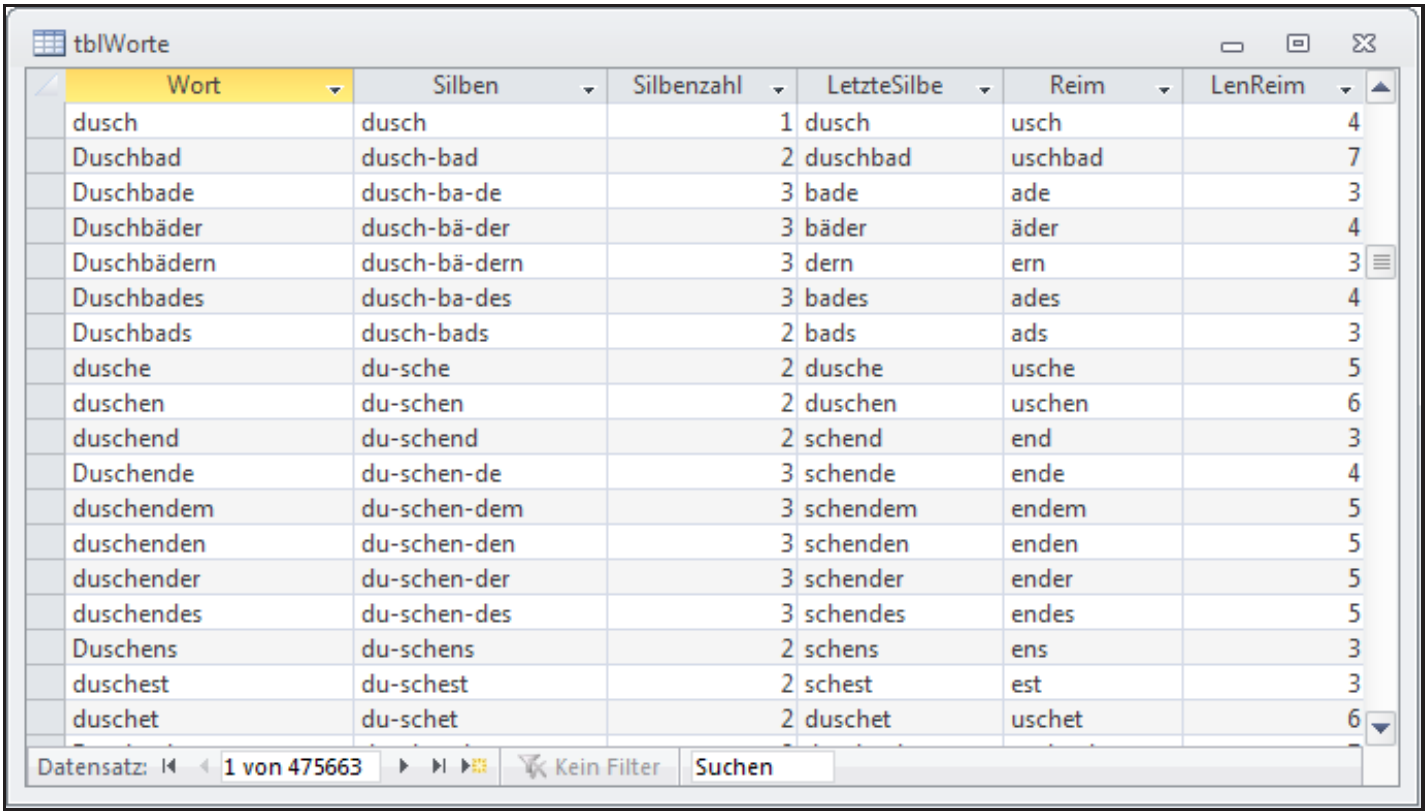
Auf der Suche nach Dateien, die alle deutschen Begriffe in allen Formen, wie Flexionen, et cetera, enthalten, stößt man im Netz auf verschiedene Quellen. Diese sind leider allesamt weder komplett, noch fehlerfrei.

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.

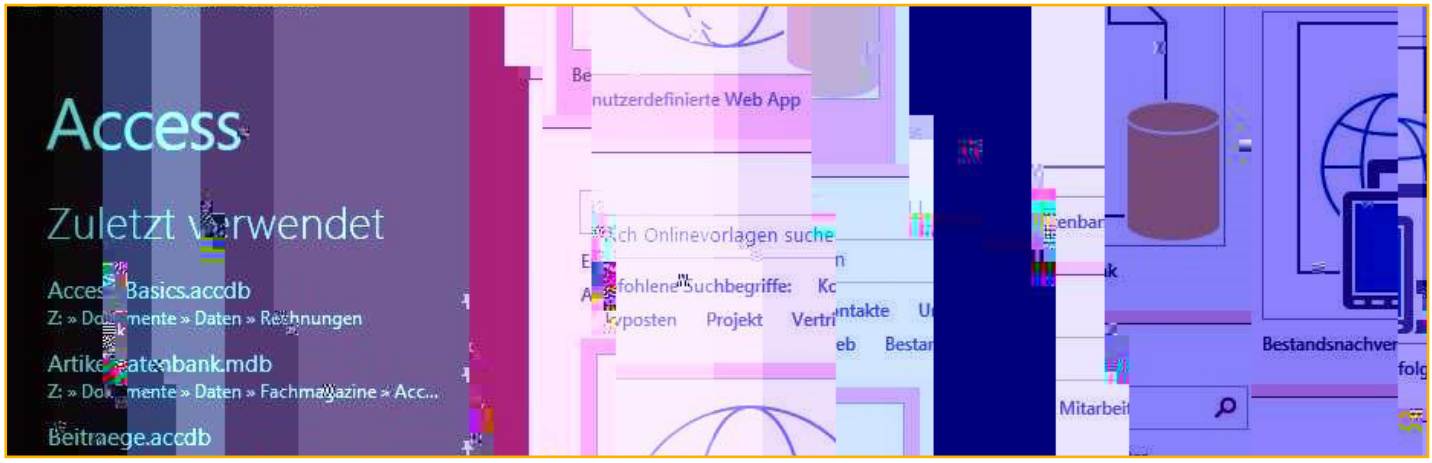
Nun werden Sie nicht unbedingt nur **Paarreime** erfinden wollen, also solche, in denen sich aufeinanderfolgende Zeilen reimen (**A-A-B-B**). Vielleicht sollen es **Kreuzreime** sein (**A-B-A-B**), oder **Blockreime** (**A-B-B-A**). Deshalb aktualisiert die **Listbox** ihren Inhalt nicht nur nach Beenden 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

Ausgabe 06/2015

# ACCESS BASICS

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




## In diesem Heft:


### DATENBANKVORLAGEN UND DESIGNS

WIR ZEIGEN, WIE SIE IHREN WORKFLOW MIT DATENBANK-VORLAGEN, ANWENDUNGS-PARTS UND DESIGNS BE-SCHLEUNIGEN KÖNNEN.

SEITE 3

### DATENTYPEN KONVERTIE-REN UNTER VBA

OHNE VARIABLEN KOMMT FAST KEINE PROGRAMMIER-SPRACHE AUS. LERNEN SIE, WIE SIE ZWISCHEN DEN VERSCHIEDENEN DATENTY-PEN HIN- UND HERKONVER-TIEREN.

SEITE 12

### MATHEMATISCHE VBA-FUNKTIONEN

LERNEN SIE MATHEMATI-SCHEN FUNKTIONEN DER VBA-BIBLIOTHEK KENNEN – UND WENN DIESE NICHTREI-CHEN, NEHMEN SIE EINFACH DIE VON EXCEL.

SEITE 18

# 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

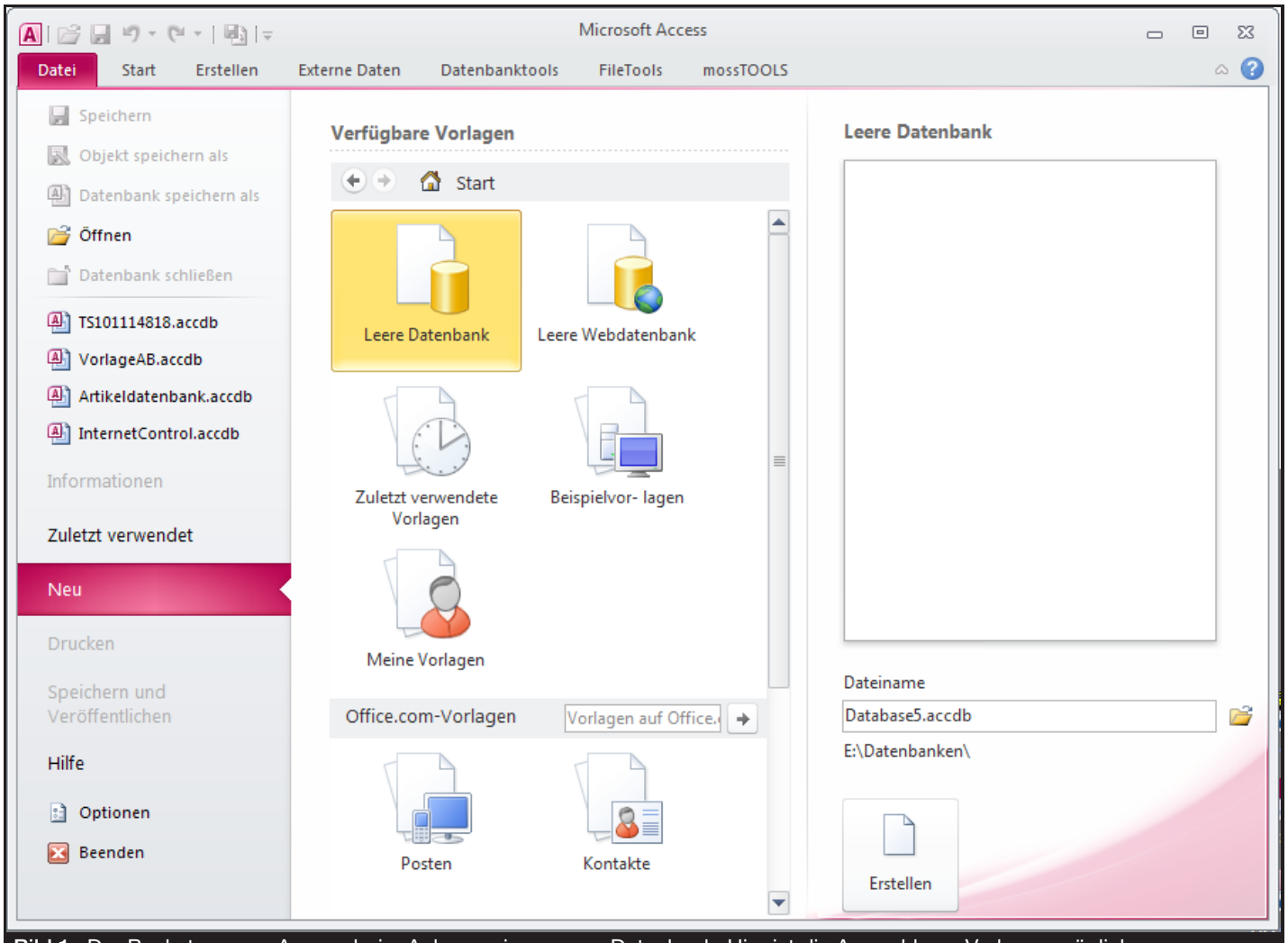



**Bild 1:** Der Backstage von Access beim Anlegen einer neuen Datenbank. Hier ist die Auswahl von Vorlagen möglich.

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.

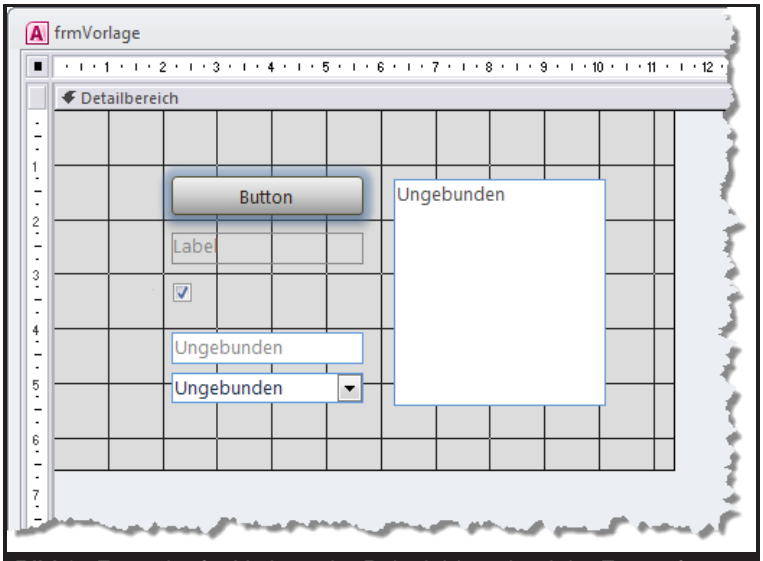



**Bild 2:** Formular frmVorlage der Beispieldatenbank im Entwurf

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

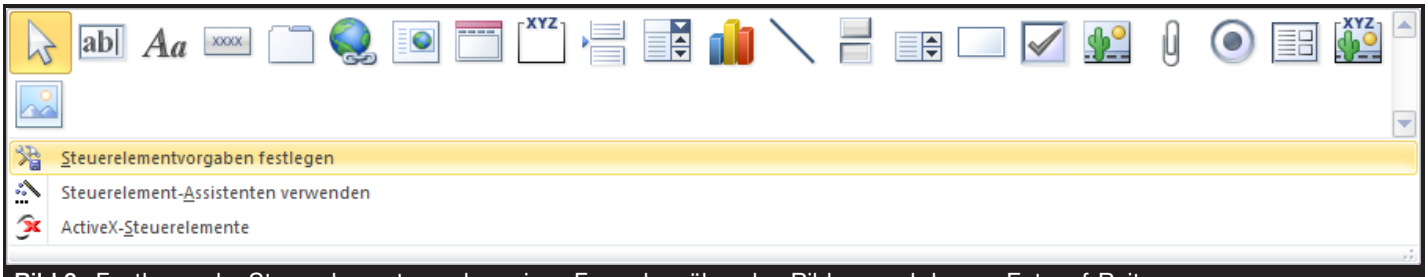



**Bild 3:** Festlegen der Steuerelementvorgaben eines Formulars über den Ribbon und dessen Entwurf-Reiters

# 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)
```

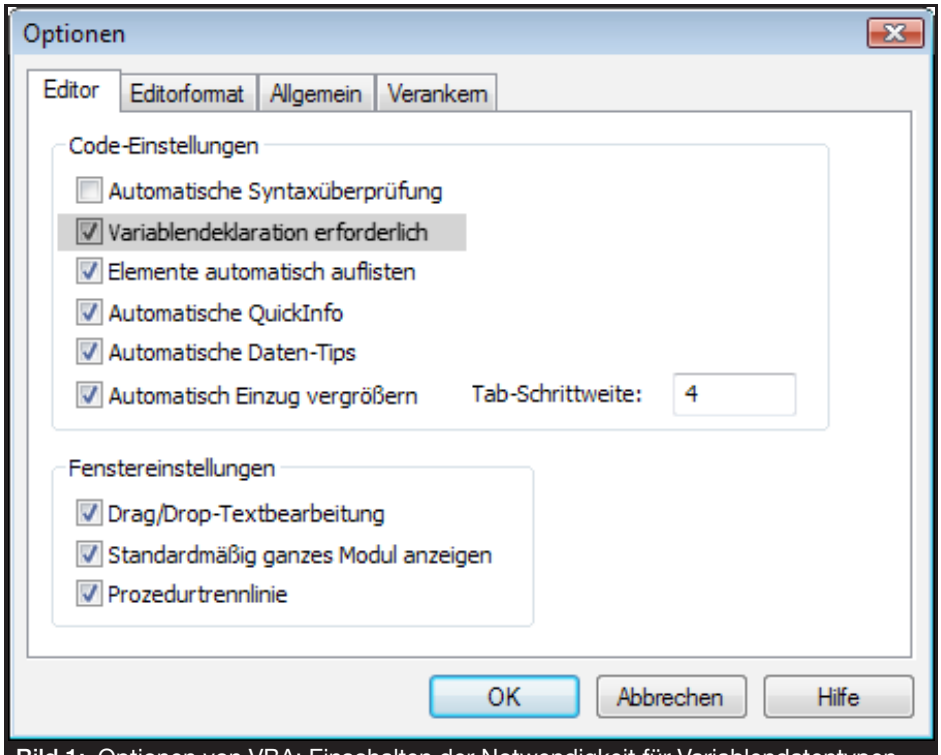



**Bild 1:** Optionen von VBA: Einschalten der Notwendigkeit für Variablendatentypen

# 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.

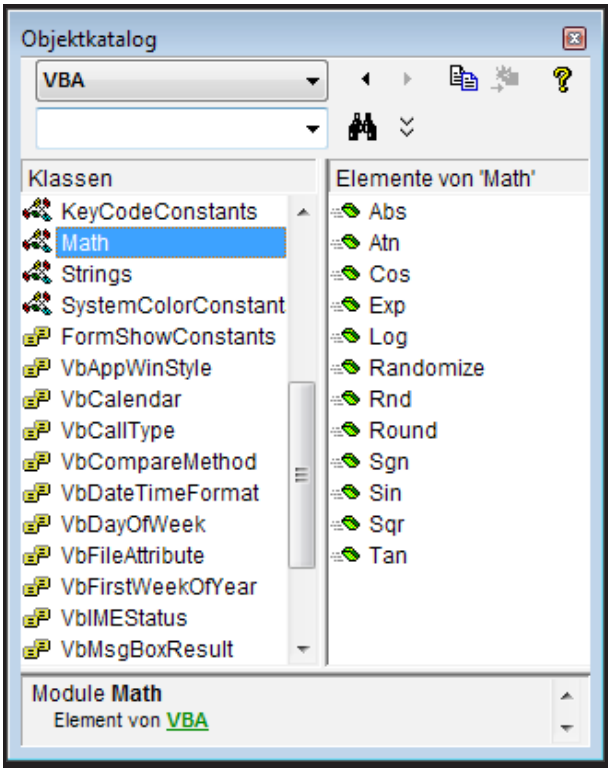



**Bild 1:** Die VBA-Klasse Math im Objektkatalog

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.

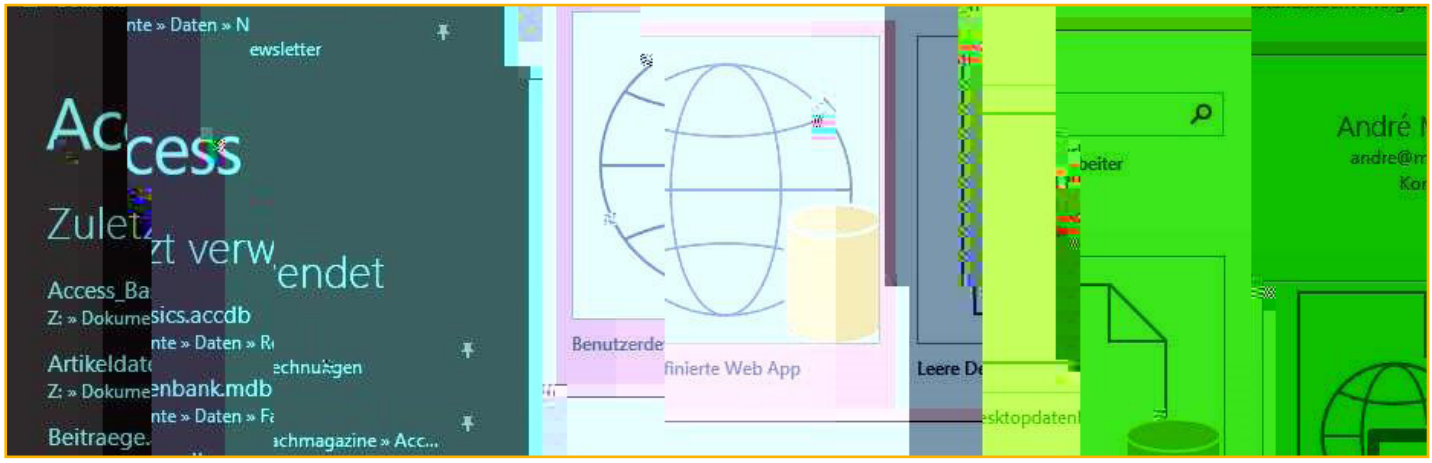
## Excel als Mathe-Helfer

Da **Excel** in der Regel auch da installiert ist, wo sich Access befindet, wäre es töricht, ausgerechnet den Spezialisten in Sachen Mathematik-

Ausgabe 07/2015

# ACCESS BASICS

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




## In diesem Heft:


### ARBEITEN MIT EXTERNEN BIBLIOTHEKEN

SICHER KENNEN SIE BEGRIFFE WIE VERWEISE, VBA-BIBLIOTHEKEN ODER TYPE LIBRARIES. ALLE BEZEICHNEN DABEI DIESELBE SACHE. KLÄREN WIR AUF, WAS ES DAMIT AUF SICH HAT!

SEITE 3

### DAS WEB IM FORMULAR

ACCESS BIETET EINEN EINGEBAUTEN WEBBROWSER. DIESER ARTIKEL ZEIGT, WIE SIE DIESEN ZUR ANZEIGE VON WEBSEITEN NUTZEN UND WAS ES DABEI ZU BEACHTEN GIBT.

SEITE 7

### BESSERES CODE-LAYOUT

DIE LESBARKEIT VON PROGRAMMCODE BESCHLEUNIGT ENTWICKLUNG UND WARTBARKEIT. GIBT ES IN IHREM CODE VERBESSERUNGSPOTENTIAL BEZÜGLICH STRUKTUR UND GESTALT?

SEITE 15

# 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.

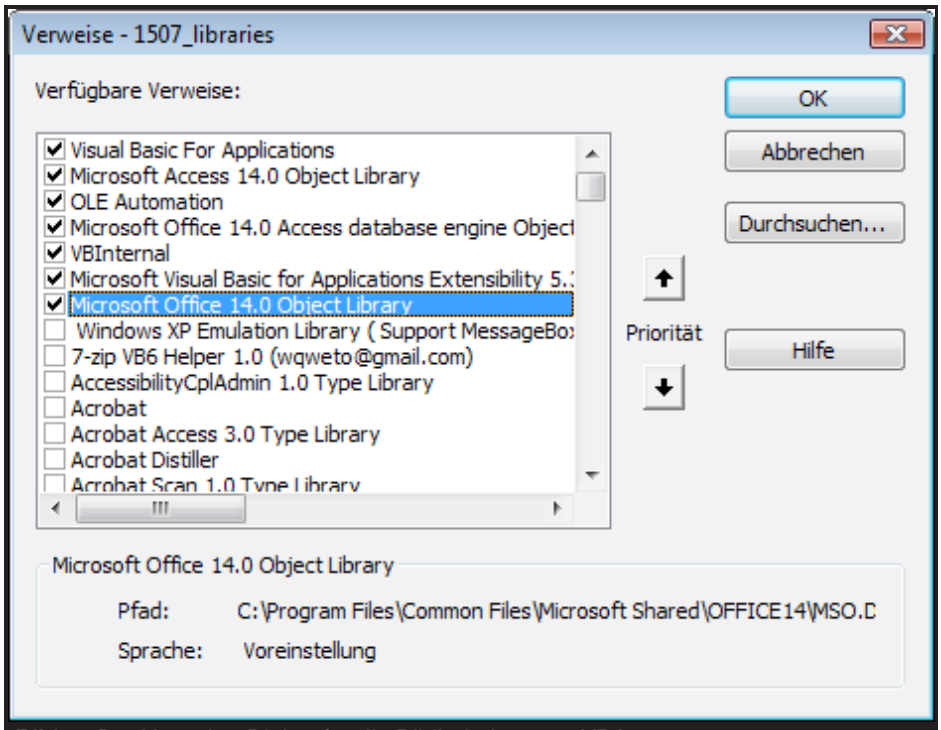


**Bild 1:** Der Verweise-Dialog für die Bibliotheken von VBA

### 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-

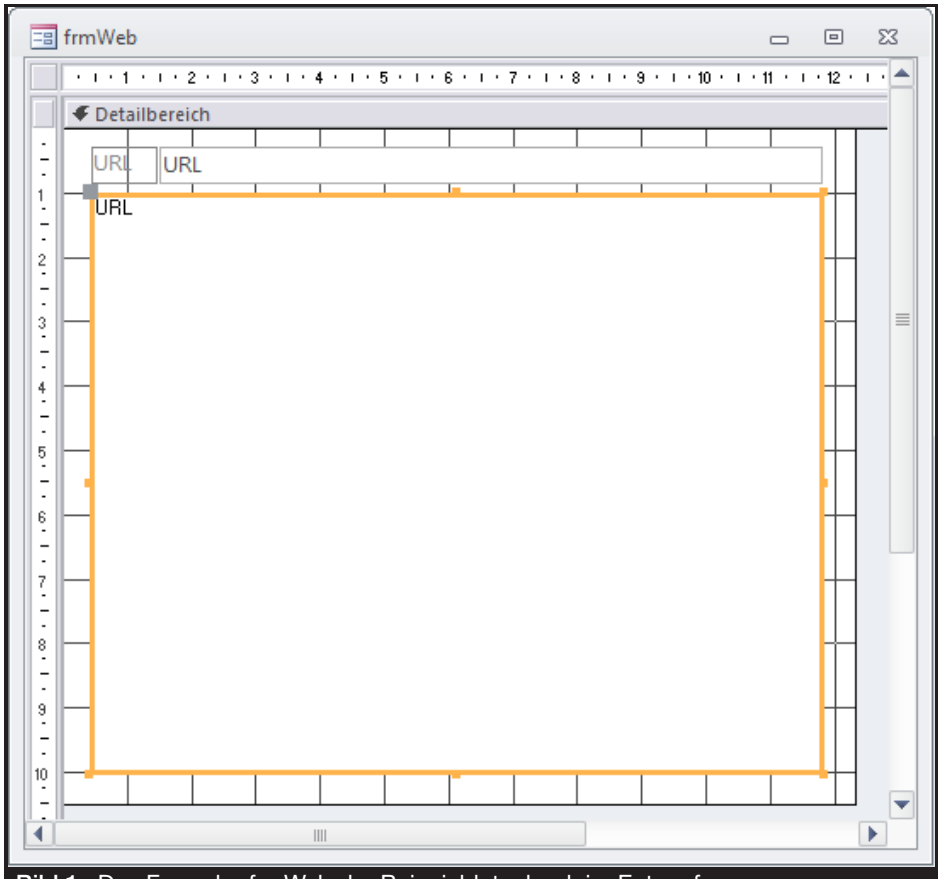



**Bild 1:** Das Formular frmWeb der Beispieldatenbank im Entwurf

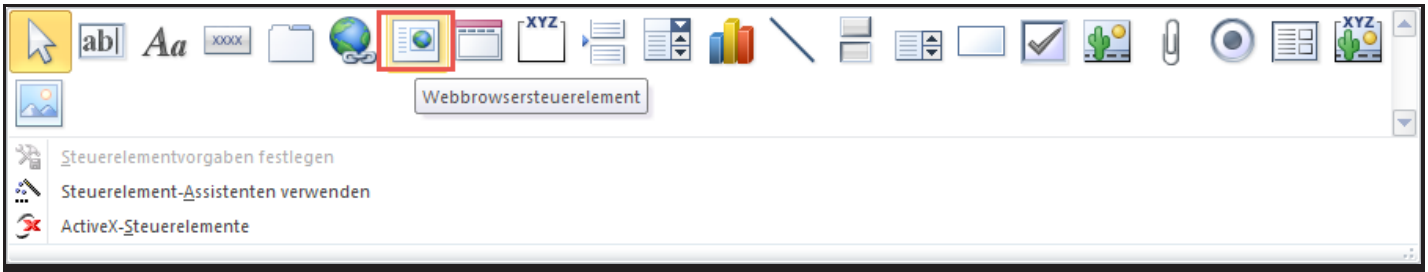



**Bild 2:** Das Webbrowsersteuerelement fügen Sie über den Ribbon-Reiter Entwurf in ein Formular ein

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.

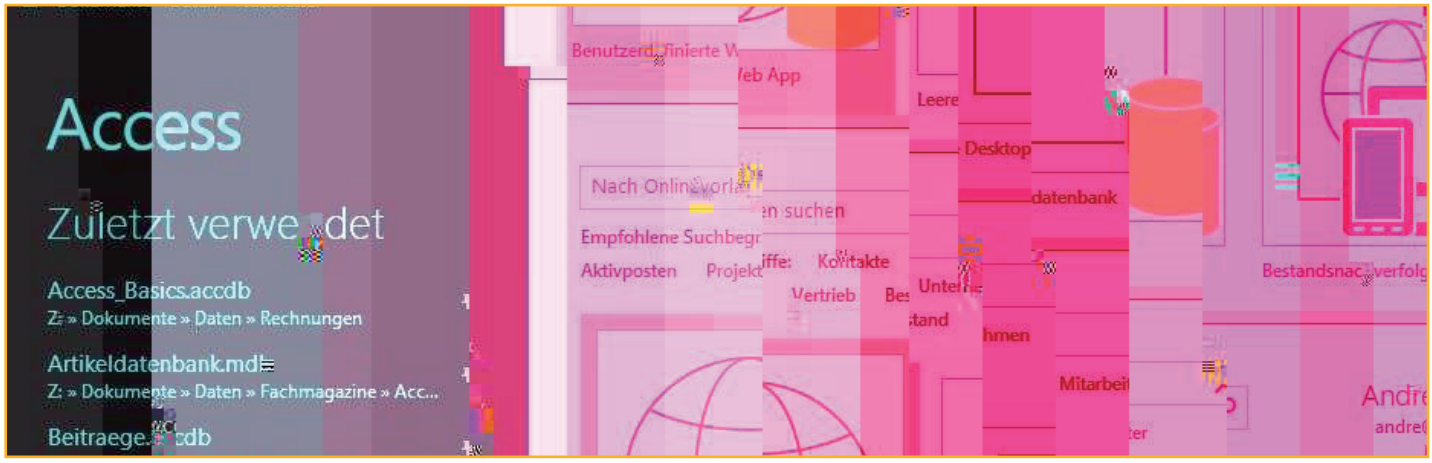



**Bild 3:** Der zweite Datensatz navigiert zur Webseite von Access im Unternehmen

Ausgabe 08/2015

# ACCESS BASICS

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




## In diesem Heft:


### ADODB ALS ALTERNATIVE ZU DAO

NEBEN DER BIBLIOTHEK DAO KÖNNEN SIE FÜR DEN ZUGRIFF AUF DATEN PER VBA AUCH MIT DER BIBLIOTHEK ADODB ARBEITEN. DIESER BEITRAG STELLT DIESE ALTERNATIVE VOR.

SEITE 3

### FRONTENDS AKTUALISIEREN

DIE NUTZUNG VON ACCESS-DATENBANKEN IM NETZWERK IST AUFWENDIGER ALS EINE EINZELPLATZANWENDUNG. DIESER BEITRAG FASST ZUSAMMEN, WAS DA AUF SIE ZUKOMMEN KANN.

SEITE 14

### ARRAYS ALS FUNKTIONSERGEBNIS

WENN EINE FUNKTION GENAU EINEN RÜCKGABEWERT HAT, IST DIE WELT IN ORDNUNG. INTERESSANT WIRD ES, WENN DIESE MEHRERE WERTE ZURÜCKLIEFERN SOLL. HIER KOMMEN ARRAYS INS SPIEL!

SEITE 19

# 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.

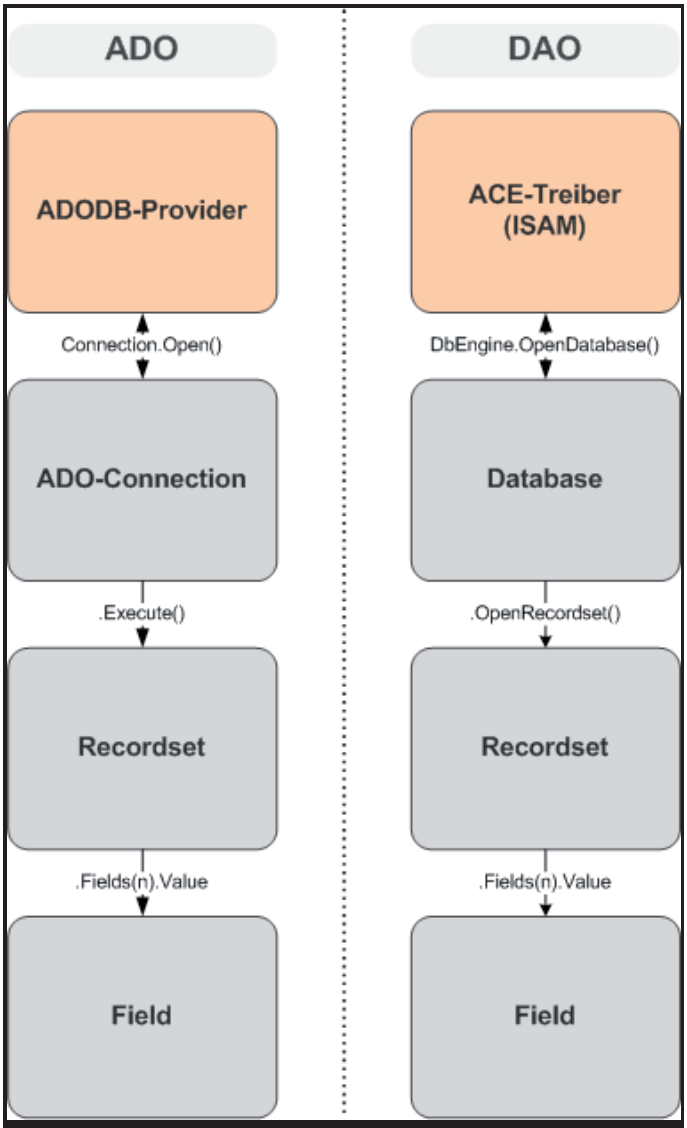



**Bild 1:** Die Datenzugriffsbibliotheken ADO und DAO in der logischen Gegenüberstellung

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** 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** 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**,

# 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.

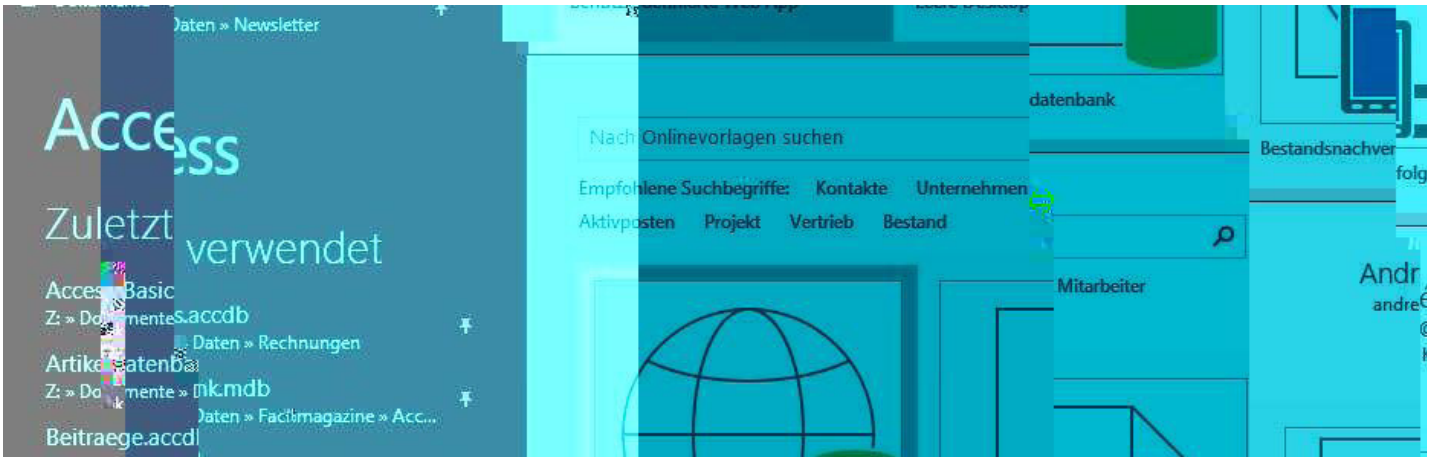
Das Skript in Listing 2 speichern Sie unter dem Namen **startdb_compressed.vbs** ebenfalls im Datenbankordner. Ähnlich, wie im vorigen Skript, ermittelt die Routine zunächst den Pfad des Datenbankordners über **ScriptFullName**, um darüber den Namen der **Frontend-Datei** in **sFile** zu generieren.

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.

Ausgabe 09/2015

# ACCESS BASICS

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




## In diesem Heft:


### REKURSIVE TABELLEN

IHR KUMPEL, KOMMT AUF DIE IDEE, FÜR ALLE BEWOHNER EINEN STAMMBAUM ANZULEGEN, UND BETREUT SIE ALS PROGRAMMIERER MIT DER AUFGABE, DIESE IN EINER DATENBANK ANZULEGEN, UM DAS ANSCHLIESSEND ZU PAPIER BRINGEN ZU KÖNNEN. WELCHE HÜRDEN DIES MIT SICH BRINGT UND WIE SIE DAS PROBLEM LÖSEN, ERFAHREN SIE HIER!

SEITE 3

### XML-IMPORT UND -EXPORT

DIE VON ACCESS UNTERSTÜTZTEN FORMATE FÜR EXPORT UND IMPORT NIMMT MIT JEDER VERSION STETIG AB. SO STRICH MAN ZULETZT 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...

SEITE 9

# 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,

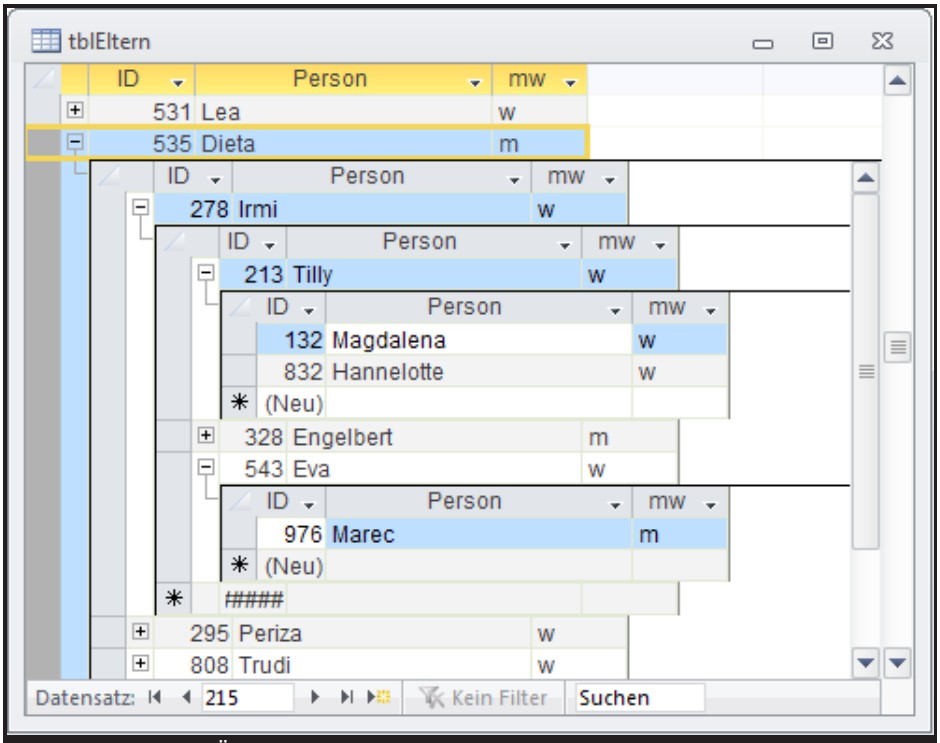



**Bild 2:** Nach dem Öffnen der Tabelle tblEltern lassen sich die automatisch generierten Unterdatenblätter mit den Kindern und Enkeln ausklappen

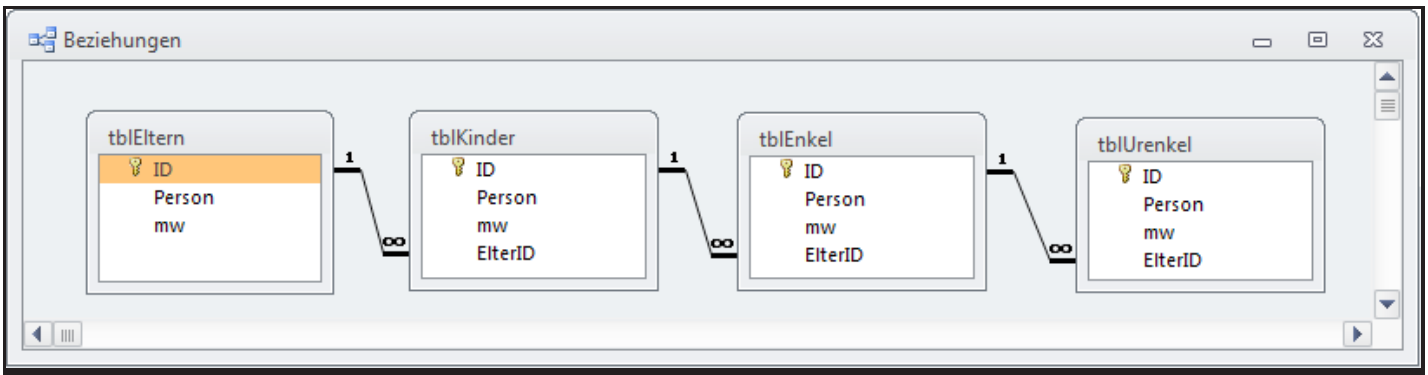



**Bild 1:** Der erste Entwurf des Datenmodells für die Stammbaumdatenbank mit allen Tabellen und deren Beziehungen

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...

# 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.

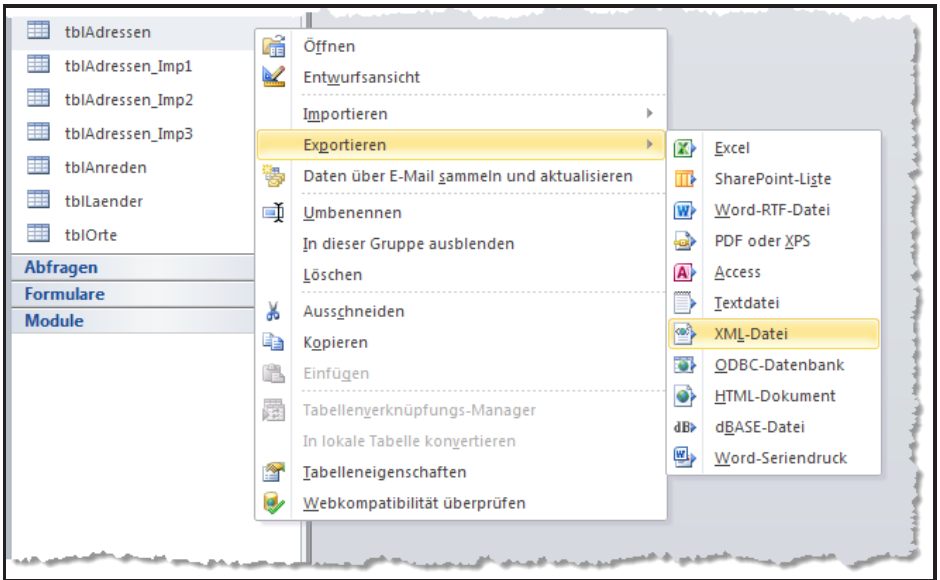



**Bild 1:** Exportieren einer Tabelle über das Kontextmenü im Navigationsbereich

## 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.

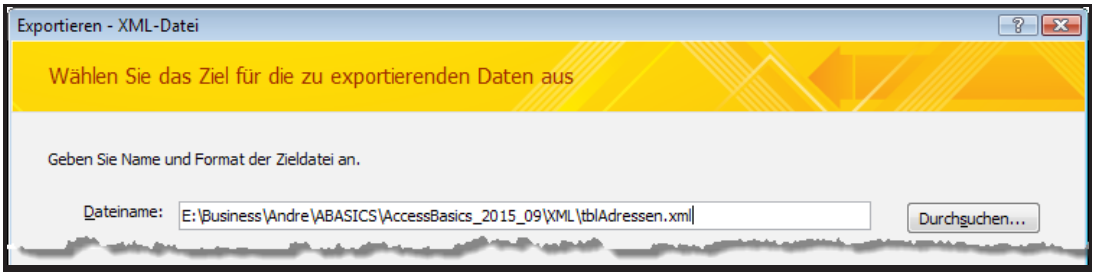


**Bild 2:** Ausschnitt des Access-Dialogs zum Export eines Objekts in eine XML-Datei

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.



**Bild 3:** Abfrage der grundlegenden Export-Optionen

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?

Für den letzten Export-Test klicken Sie im Optionen-Dialog auf die Schaltfläche **Weitere...**, die zusätzliche Exporteinstellungen im nun erscheinenden

Ausgabe 10/2015

# ACCESS BASICS

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




## In diesem Heft:


### RECORDSET-TYPEN

WENN SIE AUF DIE DATEN VON TABELLEN UND ABFRAGEN UNTER VBA ZUGREIFEN, NUTZEN SIE FAST IMMER EIN DAO-RECORDSET. DIESER ARTIKEL BELEUCHTET DIE VERSCHIEDENEN RECORDSET-TYPEN.

SEITE 3

### DAO-OBJEKTE UND -AUFLISTUNGEN

NEBEN DATABASE, RECORDSETS UND FIELDS GIBT ES NOCH EINE REIHE WEITERER DAO-OBJEKTE. DIESER ARTIKEL ZEIGT, WAS SIE DAMIT ANFANGEN KÖNNEN.

SEITE 9

### DATENBANKEN ÜBER VERKNÜPFUNG STARTEN

DATENBANKEN STARTEN SIE PER DOPPELKLICK. MIT EINER DESKTOPVERKNÜPFUNG HABEN SIE ABER NOCH VIEL MEHR MÖGLICHKEITEN. LESEN, SIE, WELCHE DIES SIND!

SEITE 15

# 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-

# 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**

# 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.

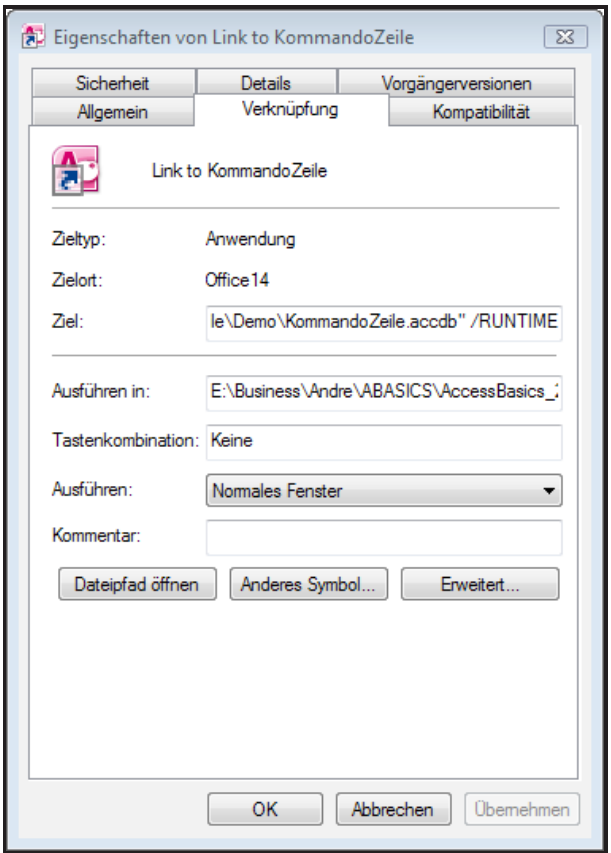



**Bild 1:** Verknüpfungseigenschaften für einen Link zur Datenbank Kommandozeile.accdb mit Startparameter

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

Ausgabe 11-12/2015

# ACCESS BASICS

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




## In diesem Heft:

**DOCMD- UND RUNCOMMAND-ANWEISUNGEN** SEITE 3

**NAVIGATION IN UND ZWISCHEN FORMULAREN** SEITE 13

**DAO-OBJEKTE UND -AUFLISTUNGEN, TEIL II** SEITE 20

**KOMMUNIKATIONSDATEN ALS BERICHT** SEITE 27

**MEHRSPALTIGER GRUPPIERTER BERICHT** SEITE 33

**BERICHT ALS UNTERFORMULAR** SEITE 39

# 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-

# 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-**

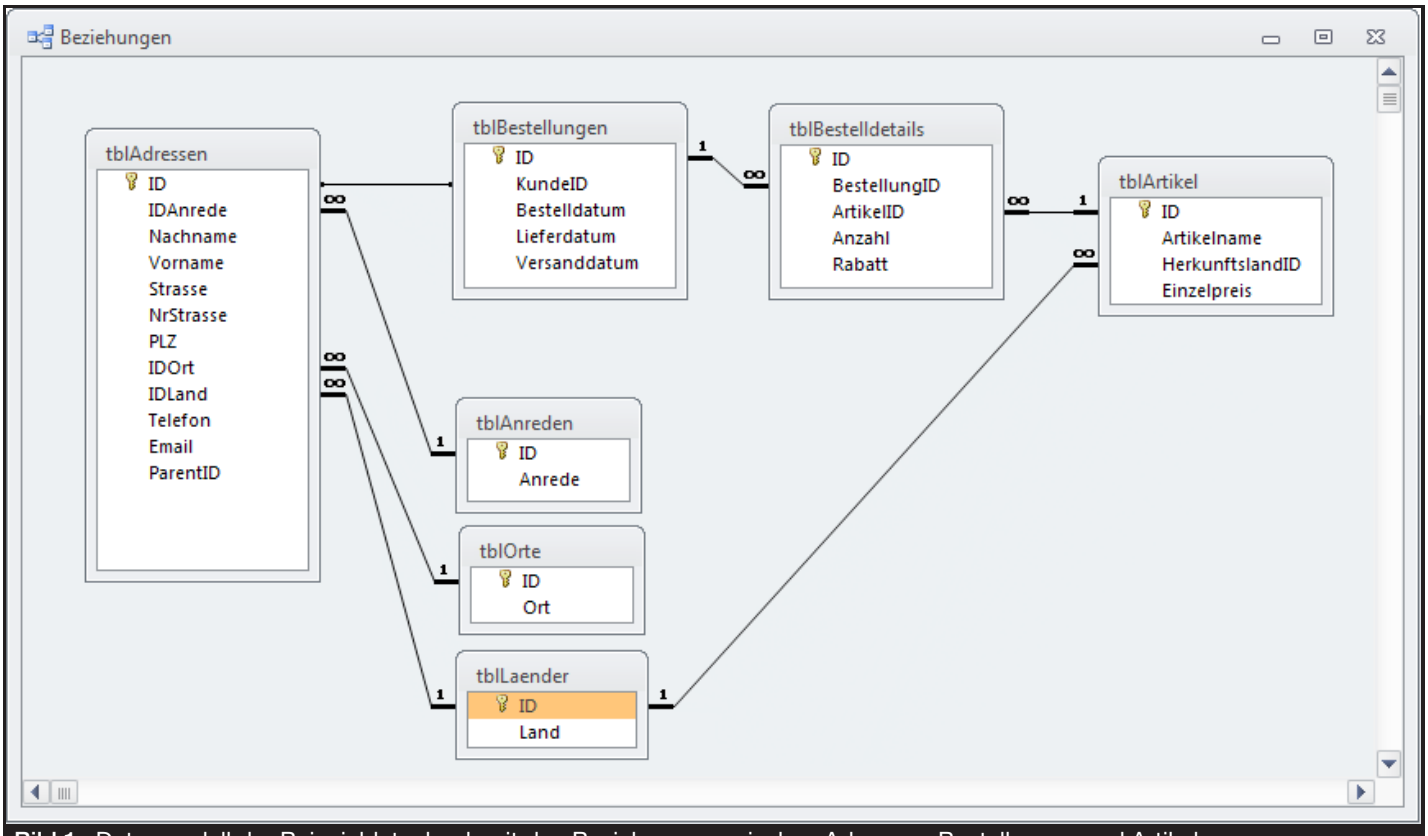



**Bild 1:** Datenmodell der Beispieldatenbank mit den Beziehungen zwischen Adressen, Bestellungen und Artikeln

**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.

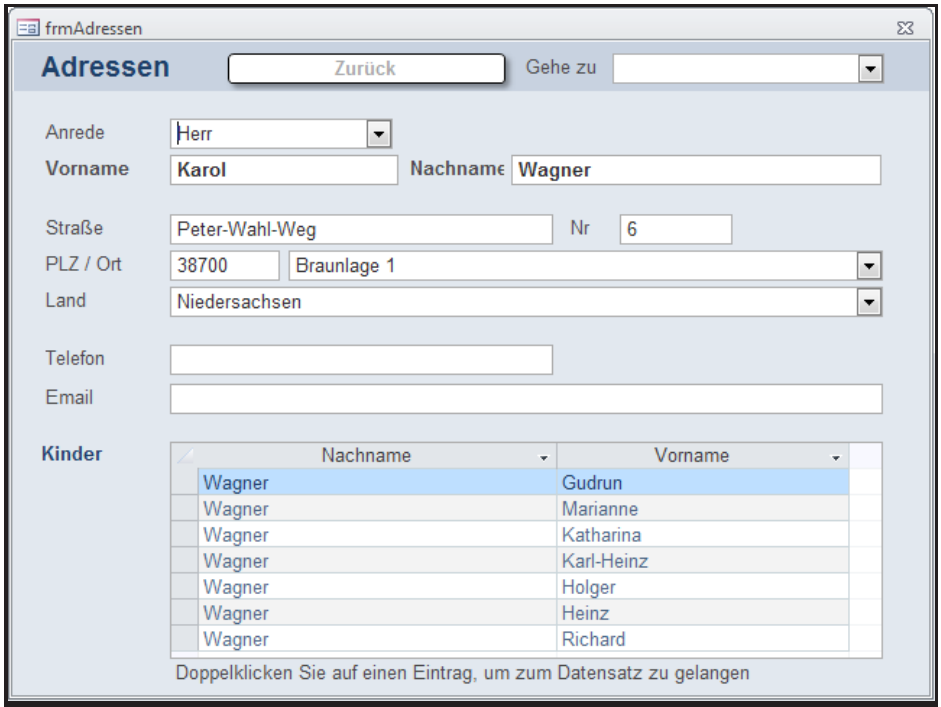



**Bild 2:** Das Adressformular frmAdressen der Beispieldatenbank

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.

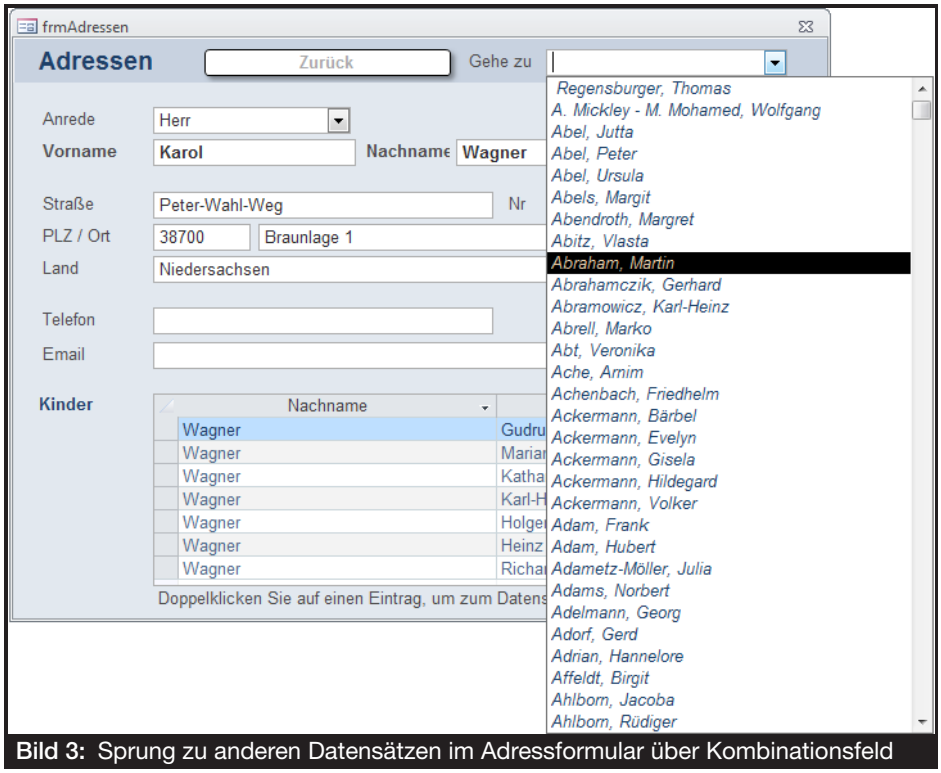



**Bild 3:** Sprung zu anderen Datensätzen im Adressformular über Kombinationsfeld

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

# 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.

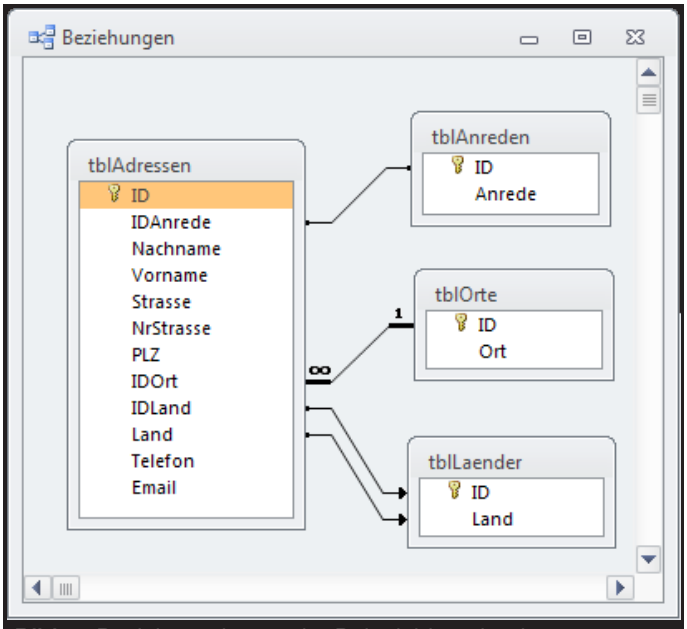



**Bild 1:** Beziehungslayout der Beispieldatenbank

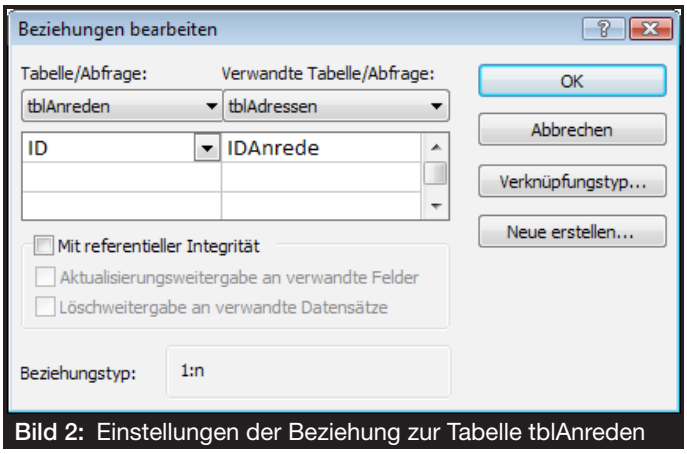



**Bild 2:** Einstellungen der Beziehung zur Tabelle tblAnreden

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

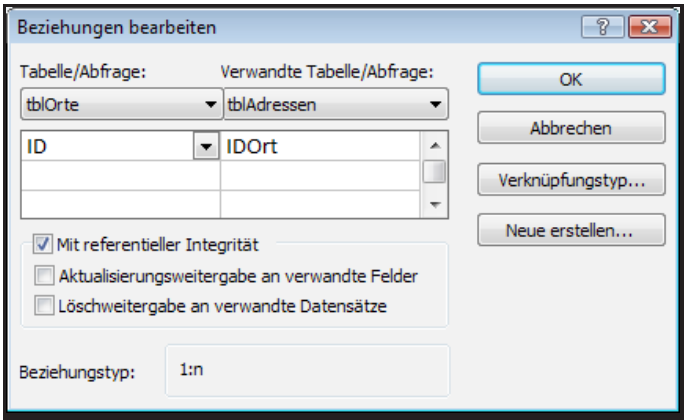


**Bild 3:** Einstellungen für die Beziehung zur Tabelle tblOrte

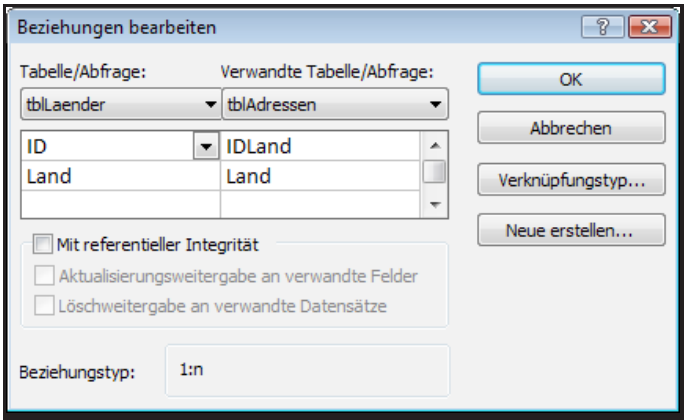


**Bild 5:** Einstellung für die Beziehung zur Tabelle tblLaender

Beziehung ausgegeben. Als Ergebnis erhalten Sie dies:

```
MSysNavPaneGroupCategoriesM-
SysNavPaneGroups
MSysNavPaneGroupsMSysNavPa-
neGroupToObjects
tblAnredentblAdressen
tblLaendertblAdressen
tblOrtetblAdressen
```

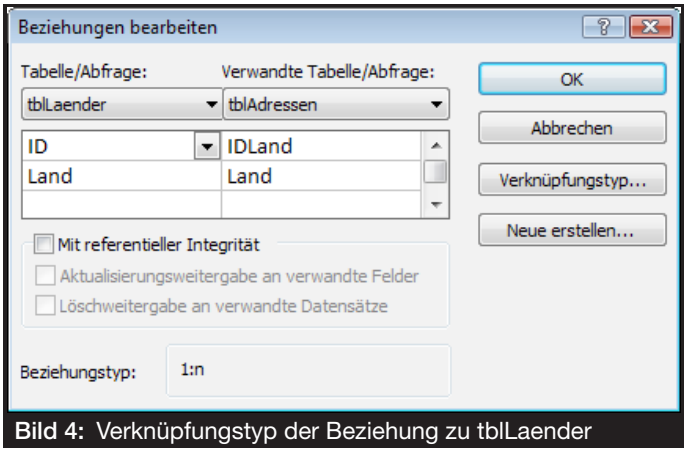


**Bild 4:** Verknüpfungstyp der Beziehung zu tblLaender

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.

# 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.

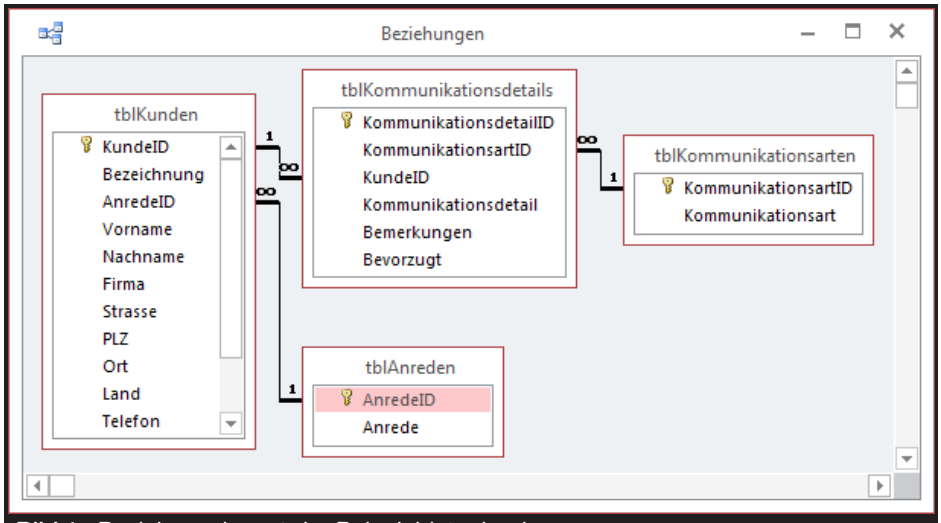



**Bild 1:** Beziehungslayout der Beispieldatenbank

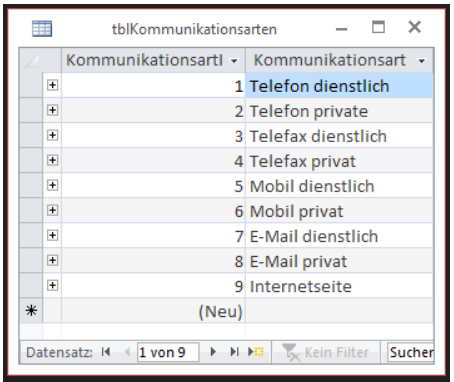
Die erste heißt **tblKommunikationsarten** und enthält alle möglichen Kommunikationsarten wie Telefon dienstlich, Telefax dienstlich, Mobil privat und so weiter (siehe Bild 2).

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).

### Bericht erstellen

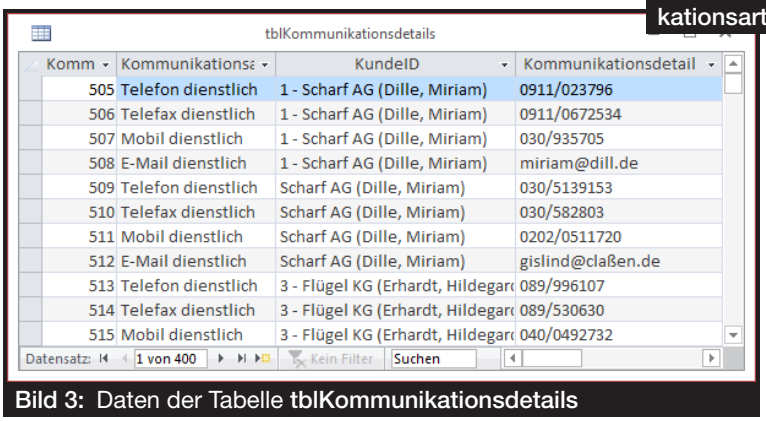
Der Bericht zur Darstellung der Kommunikationsdaten soll Daten aus allen im Datenmodell enthaltenen Tabellen außer der Tabelle **tblAnreden** anzeigen. Dazu gehören **Vorname**, **Nachname**, **Kommunikationsdetail** und **Kommunikationsart**. Also stellen wir zunächst eine entsprechende Datenherkunft in Form einer Abfrage


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**

zusammen, die wie in Bild 4 aussieht und die wie unter dem Namen **qryKommunikationsdaten** speichern.

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).

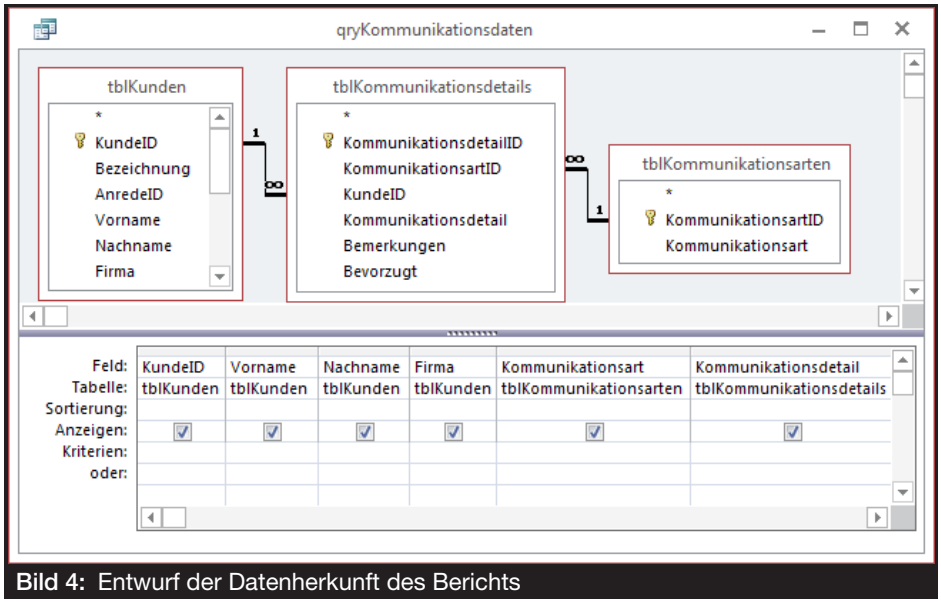



**Bild 4:** Entwurf der Datenherkunft des Berichts

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.

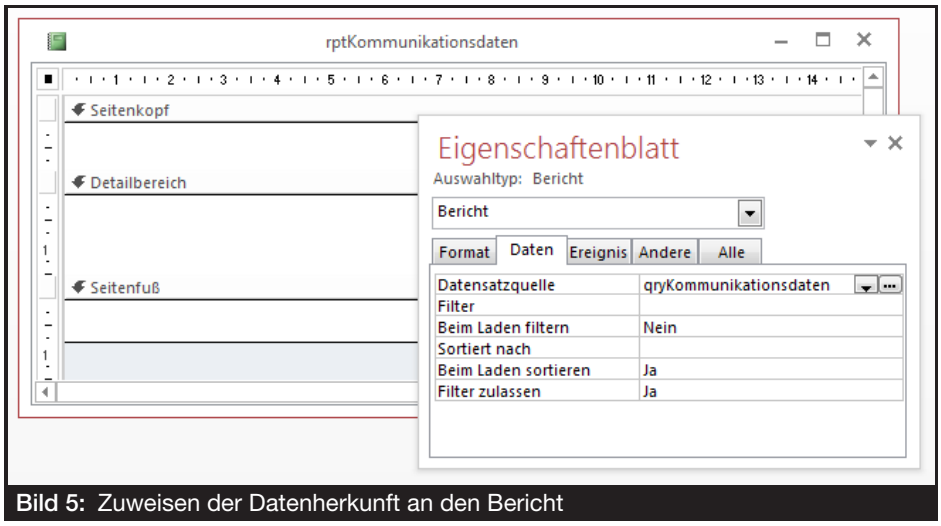



**Bild 5:** Zuweisen der Datenherkunft an den Bericht

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

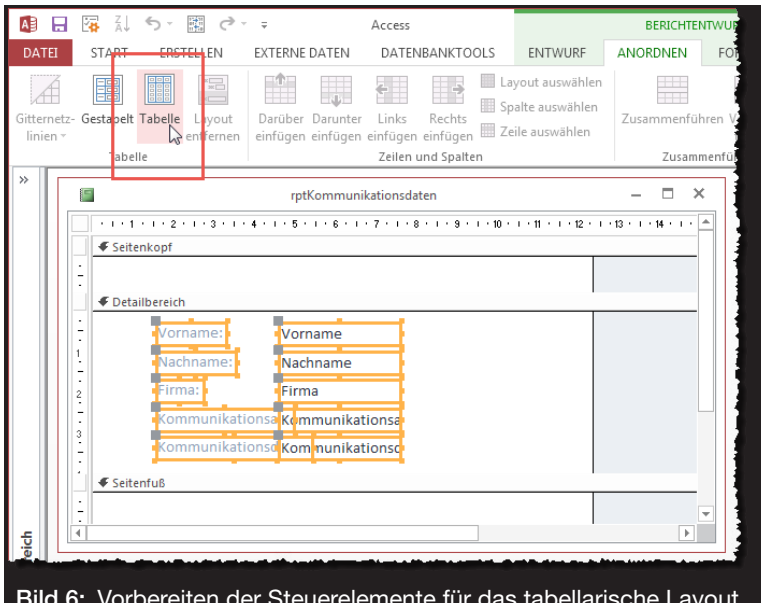



**Bild 6:** Vorbereiten der Steuerelemente für das tabellarische Layout

# 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.

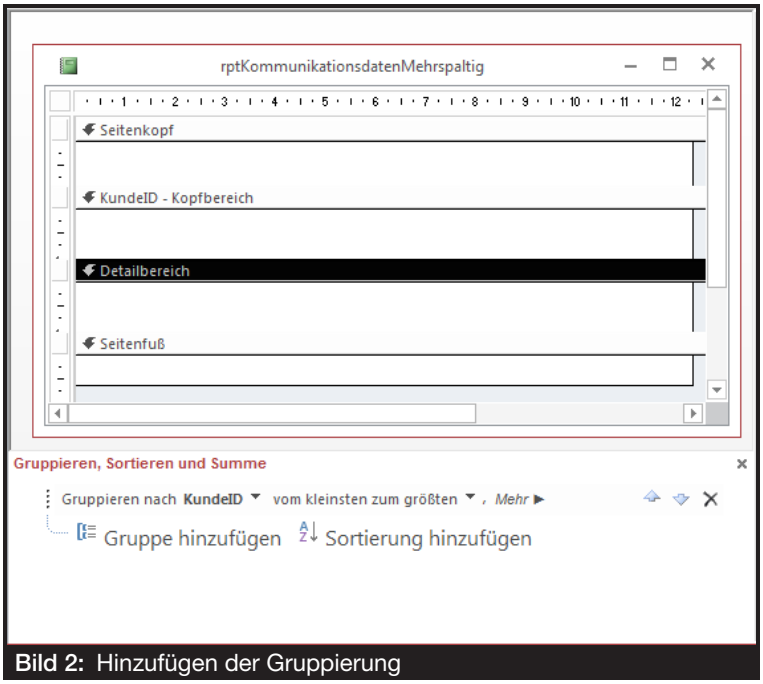



**Bild 2:** Hinzufügen der Gruppierung

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:

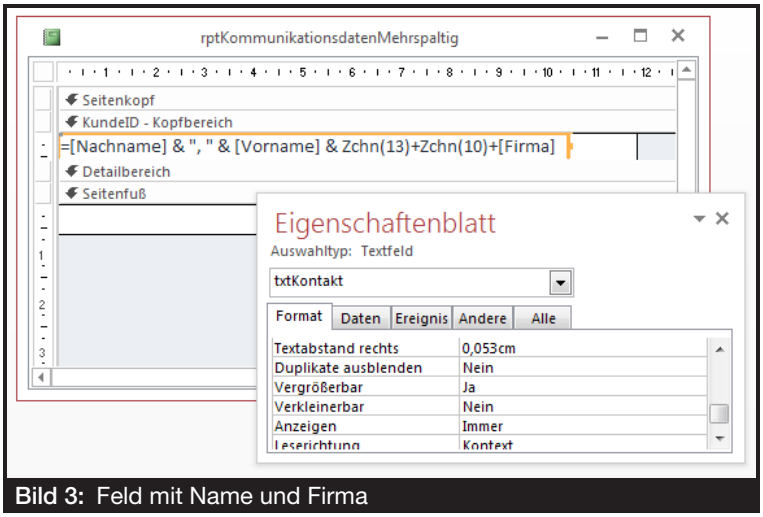



**Bild 3:** Feld mit Name und Firma

```
=[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.

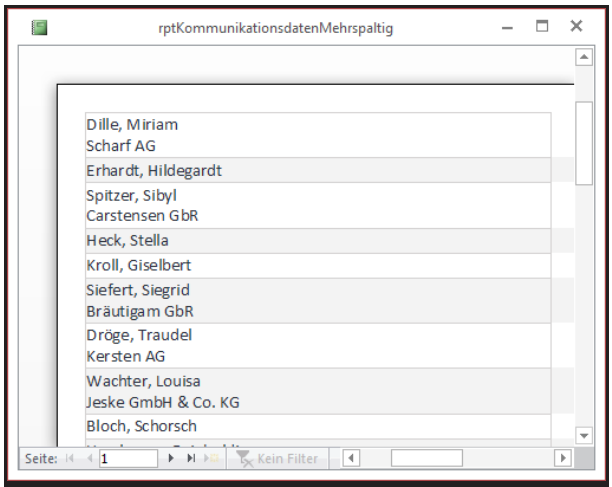



**Bild 4:** Nachname, Vorname und Firma erscheinen fast wie gewünscht

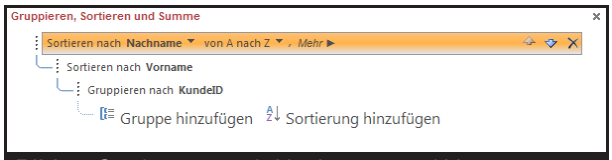



**Bild 5:** Sortierung nach Nachname und Vorname hinzufügen

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.

# 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.

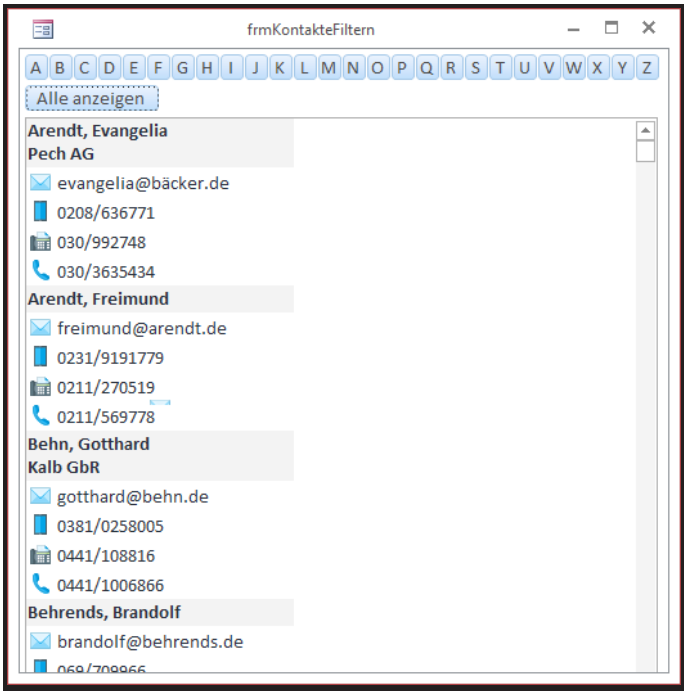


**Bild 1:** Herkunftsdaten für den Bericht

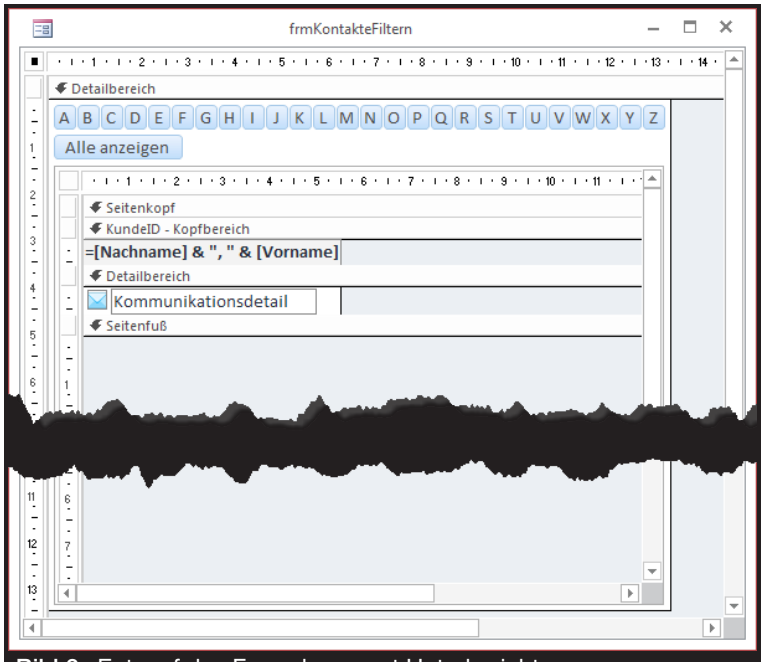


**Bild 2:** Entwurf des Formulars samt Unterbericht

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.

Ausgabe 01-02/2016

# ACCESS BASICS

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




## In diesem Heft:

**ZUSÄTZLICHE STEUERELEMENTE MIT MSFORMS**
SEITE 3

**KLASSENMODULE UND BENUTZERDEFINIERTE TYPEN**
SEITE 13

**BUBBLE SORT**
SEITE 11

**HOROSKOPGENERATOR**
SEITE 23

**ALLES PLAUSIBEL?**
SEITE 37

# 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.

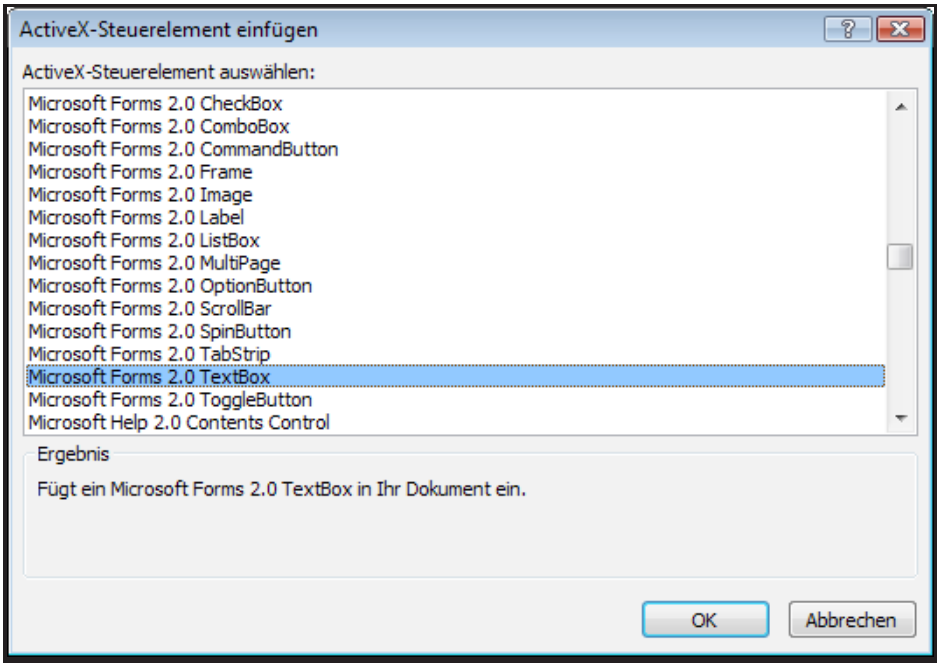



**Bild 2:** Die Steuerelemente der **MSForms**-Bibliothek Im ActiveX-Auswahldialog

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-**

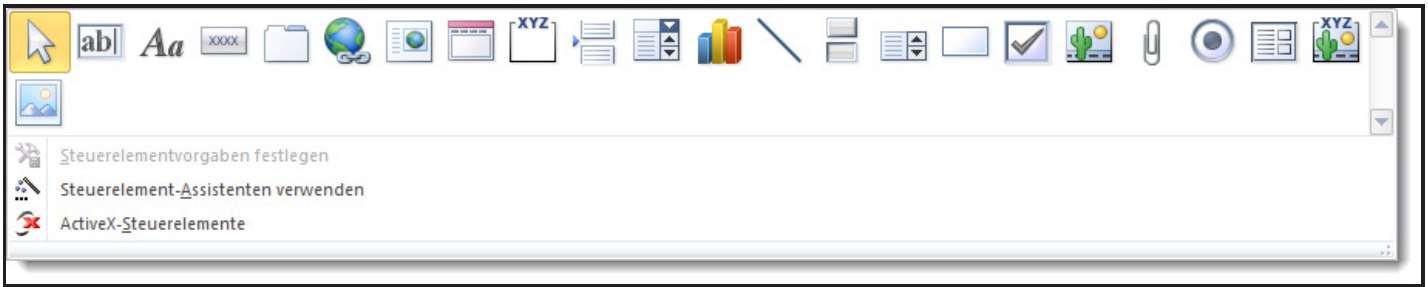



**Bild 1:** Hinzufügen von ActiveX-Steuerelementen zu einem Formular über den Ribboneintrag **Entwurf|Steuerelemente**

**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

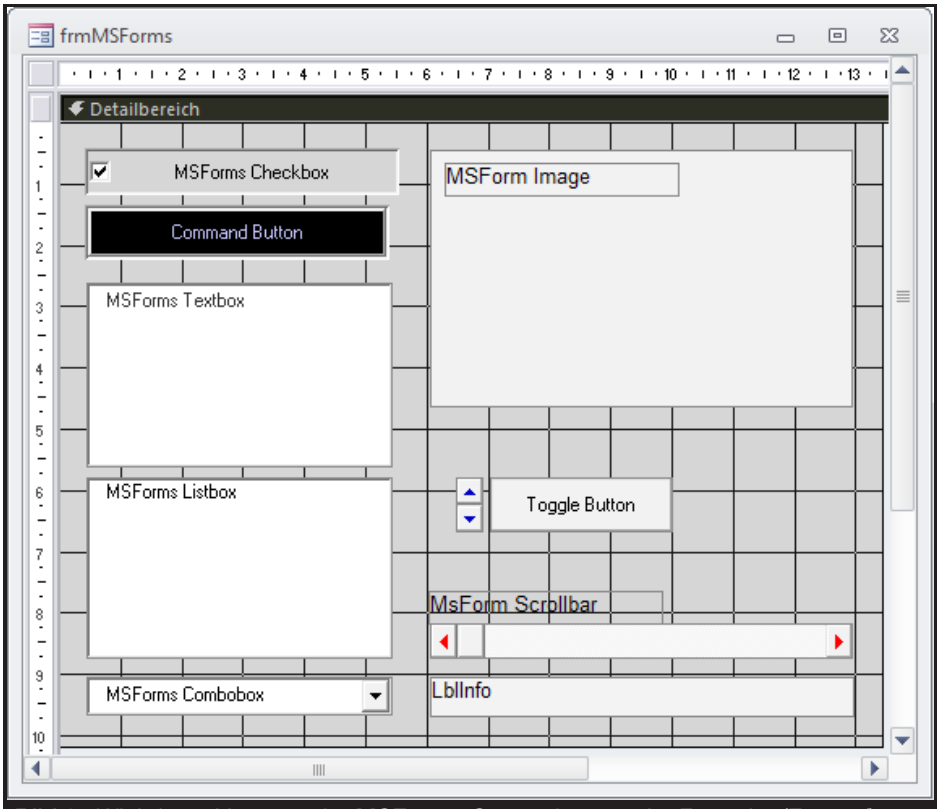



**Bild 3:** Wichtigste Vertreter der **MSForms**-Steuerelemente im Formular (Entwurf)

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
```

# 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**

vom Typ **String** sein müssen. Das Ganze funktioniert auch mit Zahlenwerten beliebiger Art, oder auch mit Datumswerten, da als Parameter **MyArray** ein Variant angegeben ist.

Mit dem optionalen Parameter **Order** können Sie weiterhin spezifizieren, ob die Elemente auf- oder Absteigend sortiert werden sollen. Nach dem Durchlaufen der Routine ist das übergebene Array dann sortiert.

Damit Sie beim Aufruf der Prozedur mit IntelliSense beglückt werden, ist der **Order**-Parameter vom Enumerationstyp **eSortArray**. Diese Enumeration ist im Kopf des Moduls deklariert. In der Folge wird Ihnen beim Aufrufen für den zweiten Funktionsparameter gleich eine der beiden Möglichkeiten **eSortAscending** (aufsteigend) oder **eSortDescending** (absteigend) angeboten.

Die erste Zeile der Prozedur testet zunächst über die **IsEmpty**-Funktion, ob **MyArray** überhaupt einen Inhalt hat. Falls nicht, so wird die Prozedur schlicht verlassen. Mit **UBound** wird nun ermittelt, wie viele Elemente das Array enthält. Die Hilfsvariable **n** nimmt diesen Wert auf.

Und nun geht es in die zwei verschachtelten **For-Next**-Schleifen mit ihren Iteratoren **i** und **j**. Die Variable **i** zählt vom ersten bis zum vorletzten Element des Arrays, die Variable **j** hingegen beginnt mit eins größer, als **i**, und endet beim letzten Element. Auf diese Weise kommen alle denkbaren Kombinationen von Elementen zusammen. Die Probe auf's Exempel bei vier Elementen:

```
i - j:
1 - 2
1 - 3
1 - 4
2 - 3
2 - 4
3 - 4
```

Je nach der über den -Parameter angegebenen Sortierreihenfolge werden nun die **i**- und **j**-Elemente des Arrays miteinander auf **Größer-** oder **Kleiner-als** verglichen und der Ergebnis-Wert in der **Boolean**-Variablen **ret** gespeichert. Enthält sie den Wert 0, also **False**, so kommt es zum Vertauschen der Elemente. Hier ist eine Hilfsvariable **tmp** vom Typ Variant nötig, die den Inhalt des einen Elements (**MyArray(i)**) zwischenspeichert.

Denn dieses Element wird ja mit dem Wert des j-Elements überschrieben. Der Zwischenspeicher wird anschließend **MyArray(j)** zugewiesen.

Auf diese Weise wandern mit fortschreitender Iteration die größeren Elemente immer weiter nach rechts, wenn die aufsteigende Reihenfolge gewählt wurde. Daher auch der Name **BubbleSort**, sinngemäß wie "aufsteigende Blasen".

Ein Test der Prozedur erfolgt in der Routine **TestSortNumberArray** des Listing 2. Das Array **arr** wird zunächst mit 100 Zufallszahlen des Typs **Double** im Bereich von Null bis 1000 bestückt, und dann wird die Sortierroutine **SortArray** mit absteigender Reihenfolge aufgerufen. In der abschließenden Schleife werden alle Elemente im VBA-Direktfenster ausgegeben.

```
Sub TestSortNumberArray()
    Dim arr() As Double
    Dim i As Long

    ReDim arr(99)
    For i = 0 To 99
        arr(i) = 1000 * Rnd
    Next i
    SortArray arr, eSortDescending
    For i = 0 To 99
        Debug.Print arr(i)
    Next i
End Sub
```

**Listing 2:** Testroutine für Zahlen-Array

```
Sub TestSortStringArray()
    Dim arr() As String
    Dim i As Long

    ReDim arr(99)
    For i = 0 To 99
        arr(i) = RandomString
    Next i
    SortArray arr, eSortAscending
    For i = 0 To 99
        Debug.Print arr(i)
    Next i
End Sub
```

**Listing 3:** Testroutine für String-Array

**TestSortStringArray** in Listing 3 macht dasselbe für Zeichenfolgen. Das Array **arr** wird über die hier nicht abgebildete Hilfsfunktion **RandomString** mit Werten versehen. Ergebnis wäre dann etwa derlei:

```
Amüla
Arelal
Bäbüsgäs
Bamrosü
Bedavübi
Bekavöp
Bimöt
Blüzenanes
Bönusete
Buatewipo
Dänulanäm
Depuga
Desib
```

# 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

# 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

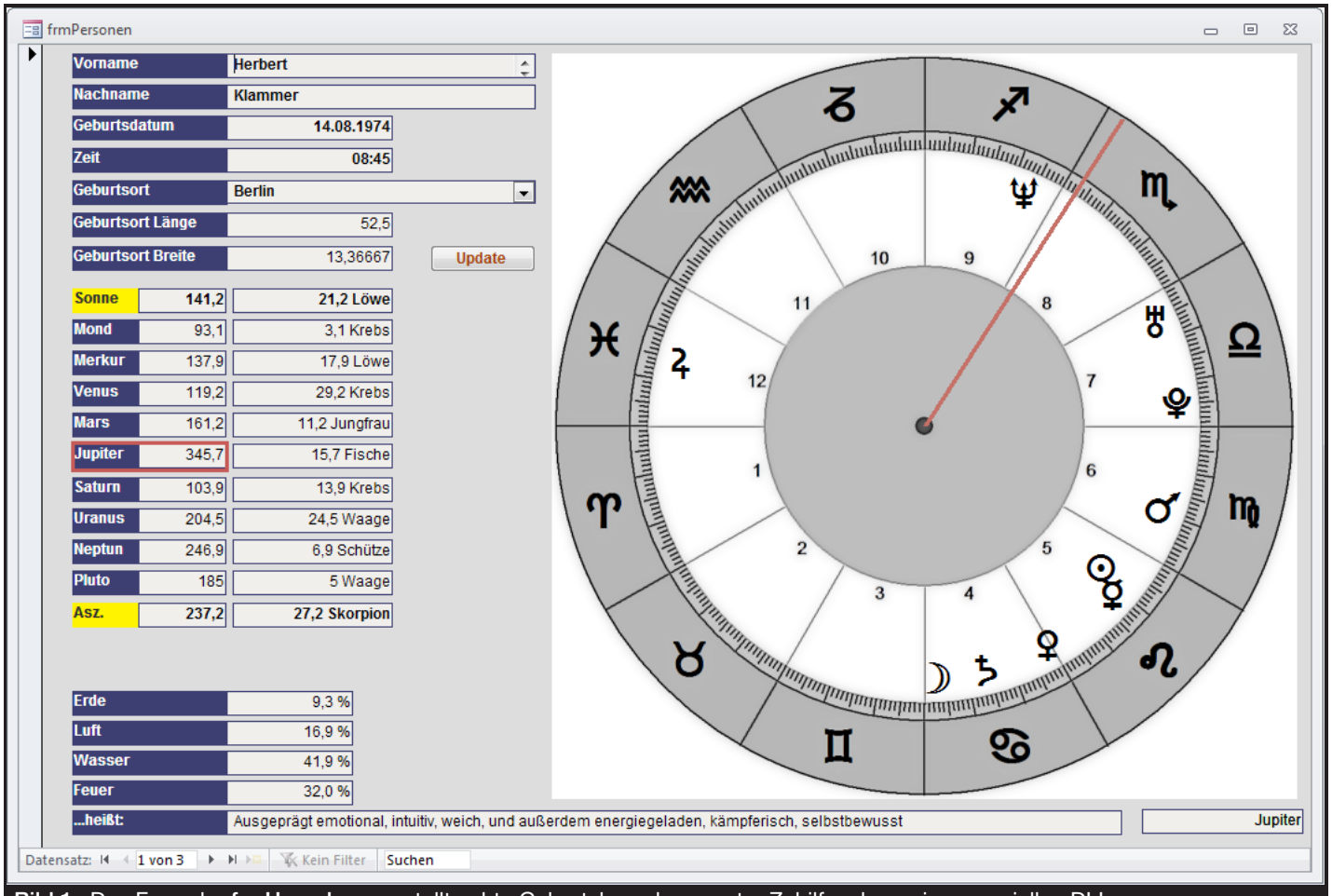



**Bild 1:** Das Formular **frmHoroskope** erstellt echte Geburtshoroskope unter Zuhilfenahme einer speziellen DLL

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

# 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.

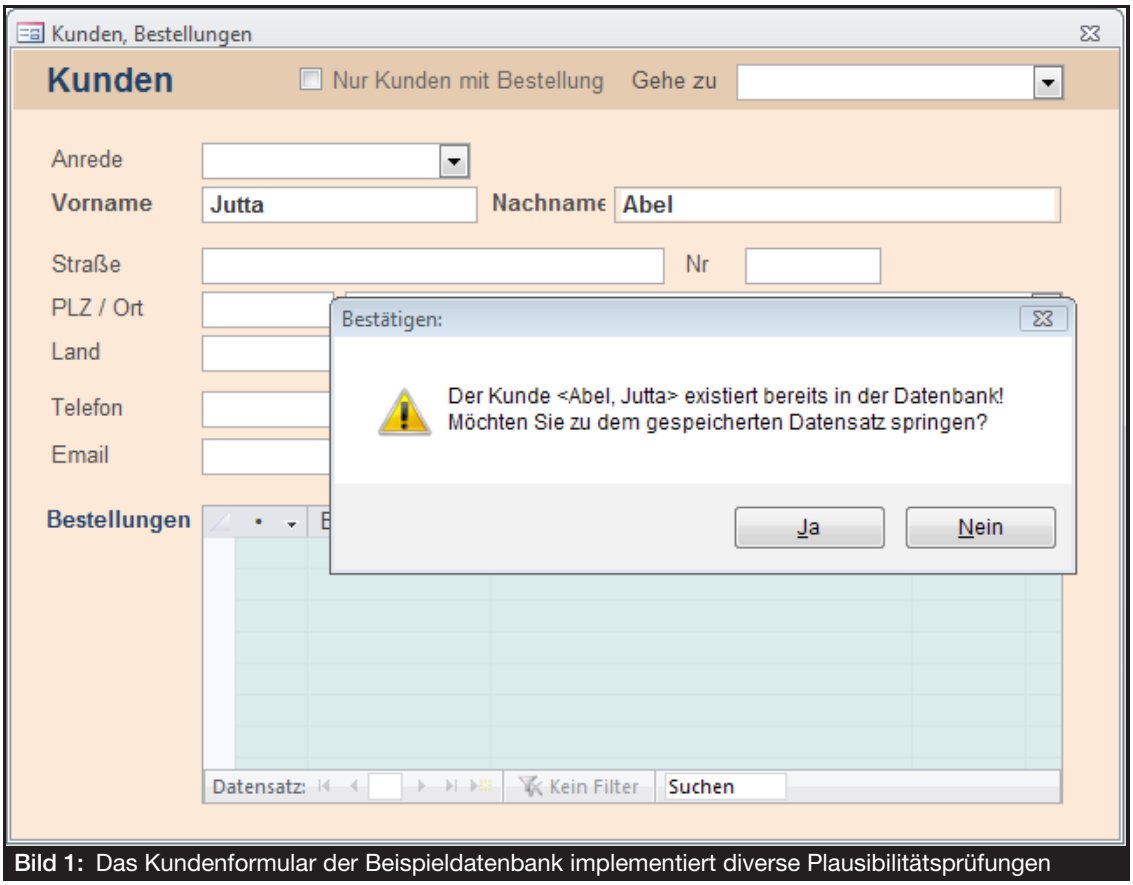


**Bild 1:** Das Kundenformular der Beispieldatenbank implementiert diverse Plausibilitätsprüfungen

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;
```

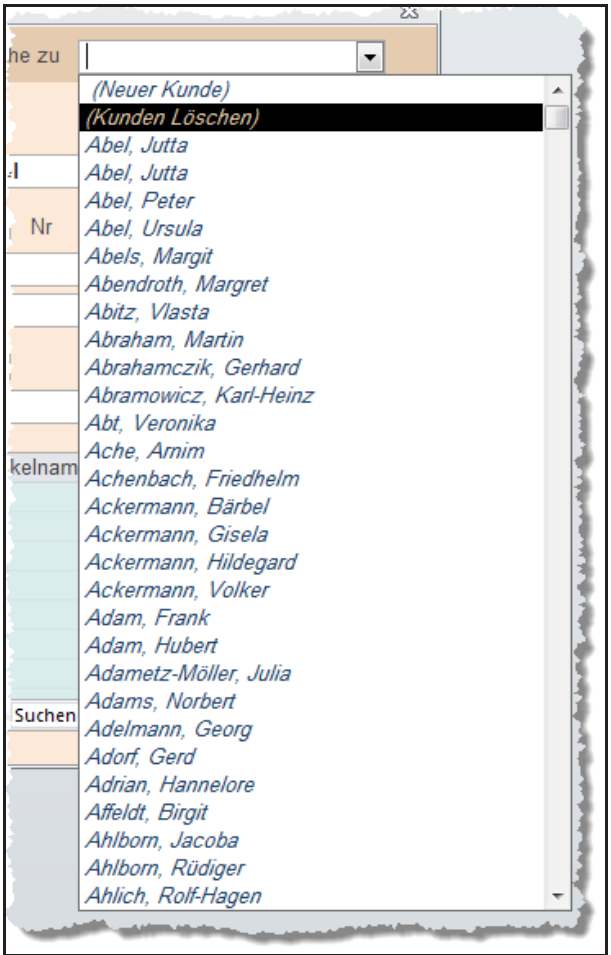



**Bild 2:** Die Combobox mit ihren zwei zusätzlichen Einträgen zum Neuanlegen und Löschen von Kunden

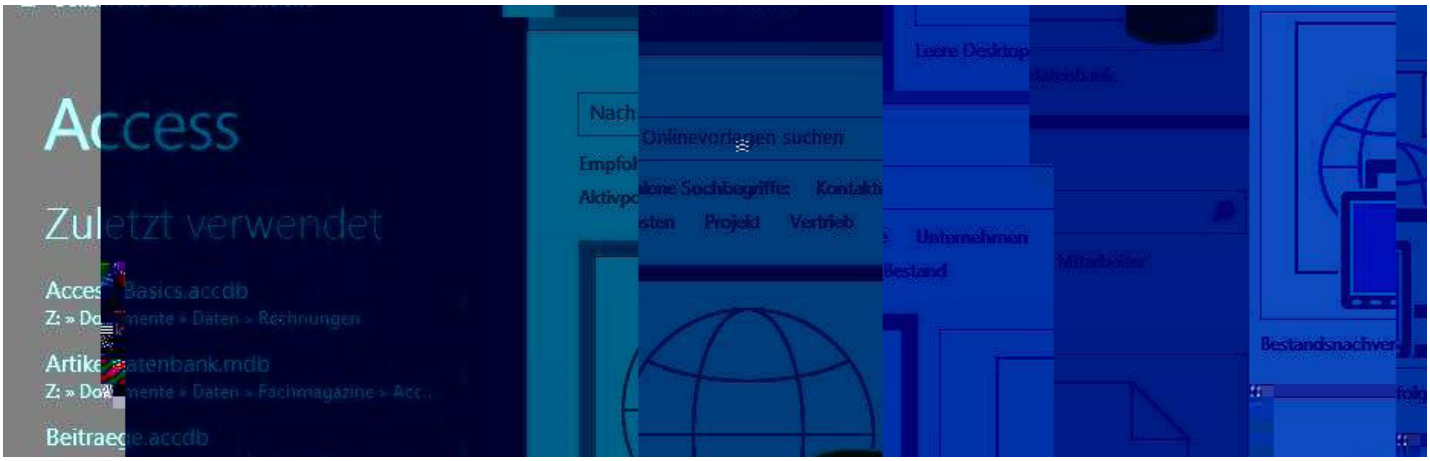
Erstaunlich, aber wahr: Nicht nur zwei, sondern auch mehrere **SELECT**-Abfragen können über die **UNION**-Klausel miteinander kombiniert werden! Die eigentliche Abfrage steht ganz unten, wobei die Kombination aus Nach- und Vorname über den Feld-**Alias Vollname** ausgegeben wird. Damit die Einzelabfragen 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

Ausgabe 03/2016

# ACCESS BASICS

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




## In diesem Heft:

**SELECT-ABFRAGEN IN UNION**
SEITE 3

**TASTENSTEUERUNG ABFANGEN**
SEITE 10

**GOOGLE- UND BING-MAPS IM FORMULAR**
SEITE 15

# 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
```

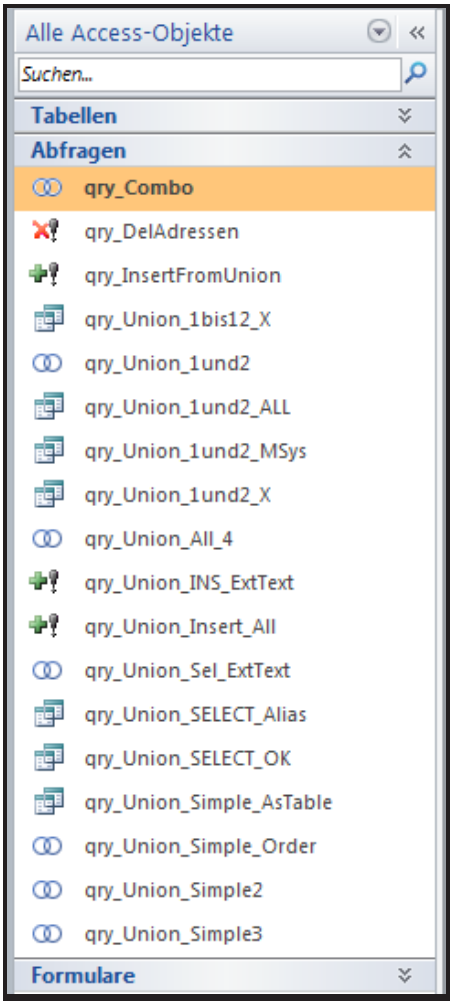



**Bild 1:** Die zwei Ringe stellen das Symbol für UNION-Abfragen im Navigationsbereich dar

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

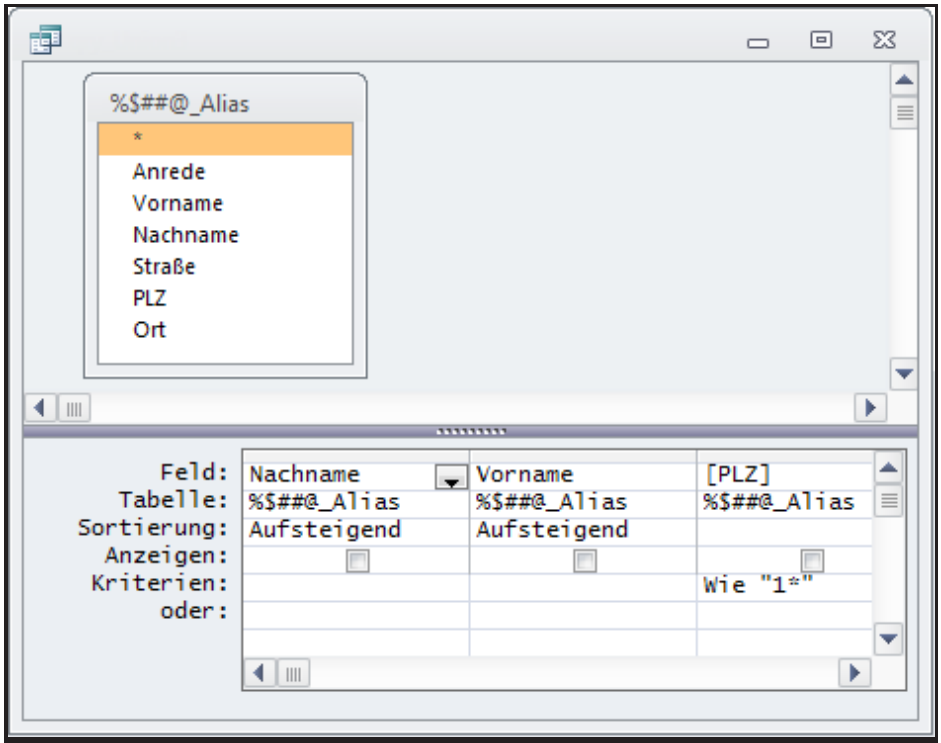



**Bild 2:** Dass es sich hier um eine UNION-Abfrage handelt, ist nicht gleich erkennbar

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
```

# 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?

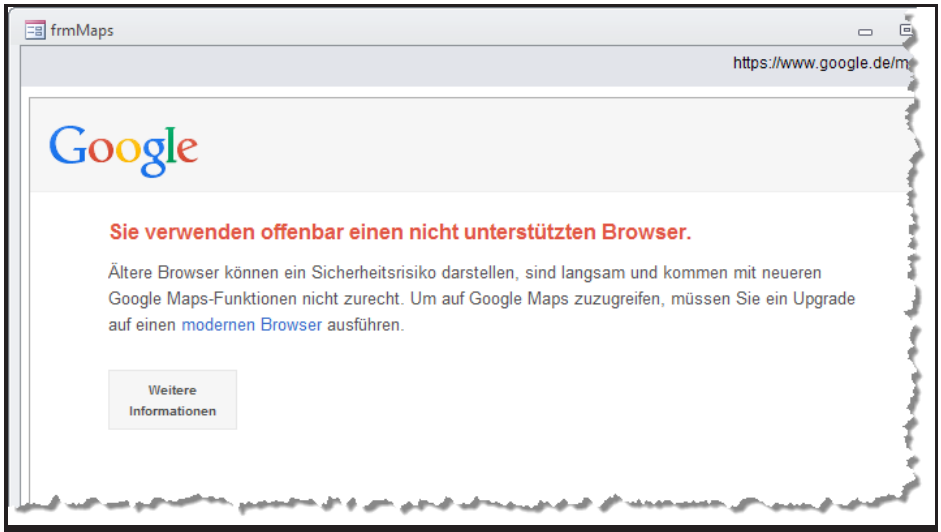



**Bild 1:** Google Maps akzeptiert das Webbrowser-Steuerelement nicht als Browser

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-

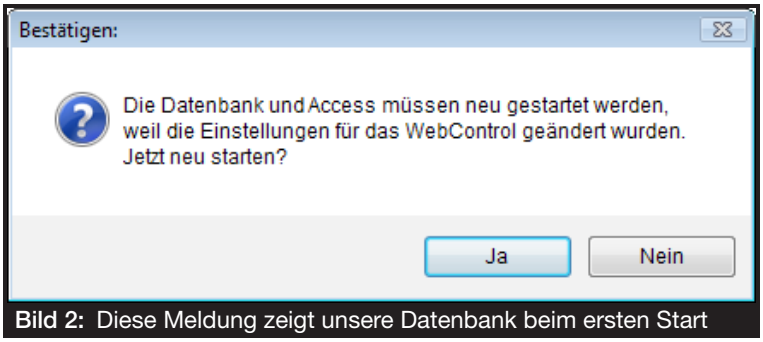



**Bild 2:** Diese Meldung zeigt unsere Datenbank beim ersten Start

Ausgabe 04/2016

# ACCESS BASICS

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




## In diesem Heft:


## ELEMENTE FÜR DIE BILDANZEIGE

Der Umgang mit Bildern in Formularen und Berichten hat für viele Entwickler hohe Priorität. Dieser Artikel zeigt die aktuellen technischen Möglichkeiten zur Anzeige von Bildern in Access-Datenbanken (ab Seite 3)

## UNTERABFRAGEN ALS FELDINHALT

Ein Abfrageergebnis als Wert eines Abfragefeldes? Dies erreichen Sie mit einer Unterabfrage, auch Subselect genannt. Alles, was Sie zu diesem Abfragekonstrukt wissen müssen, erfahren Sie ab Seite 12.

# 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

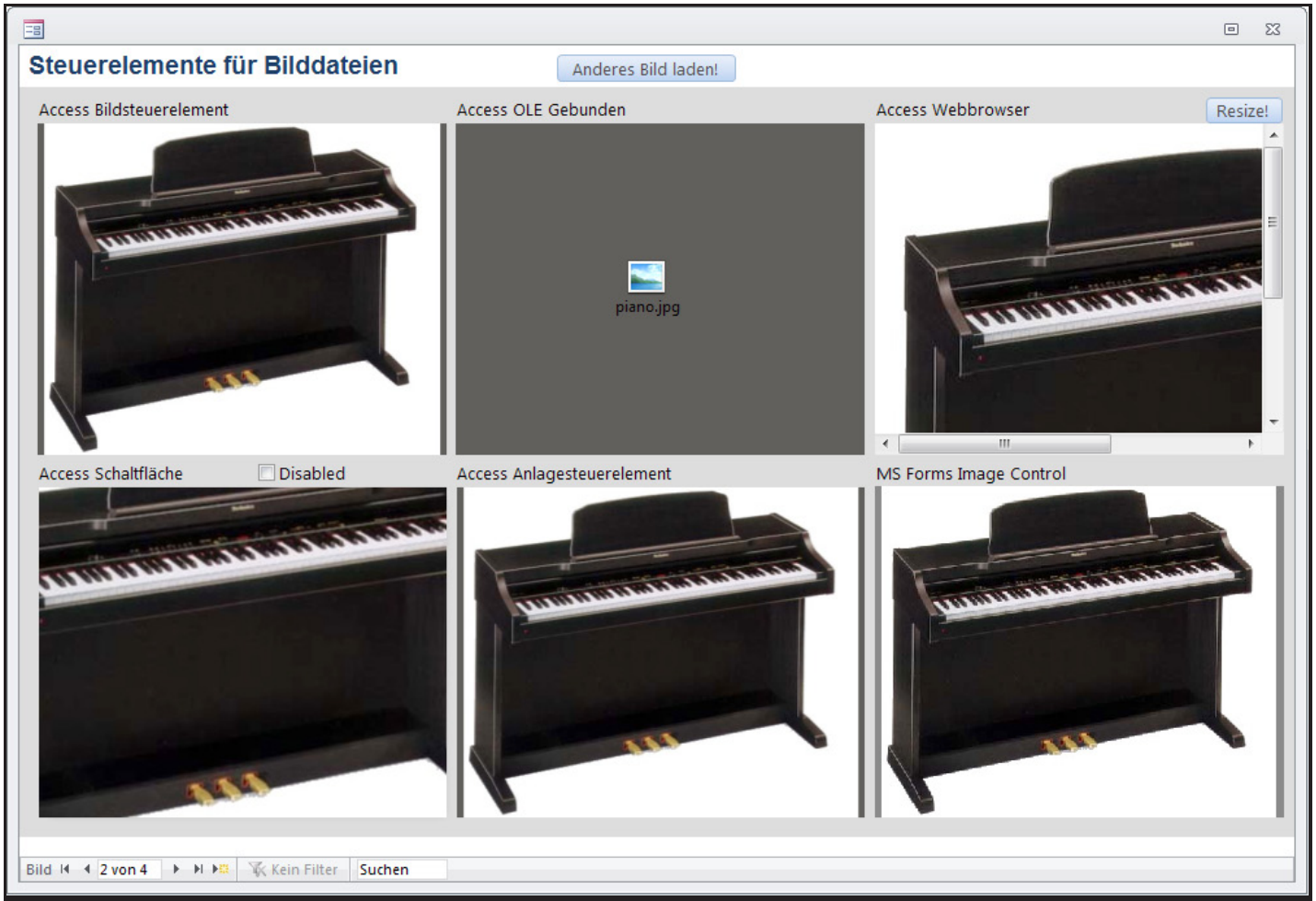



**Bild 1:** Die sechs Steuerelemente im Formular frmBilder zeigen alle dieselbe Bilddatei an – fast jedenfalls!

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.

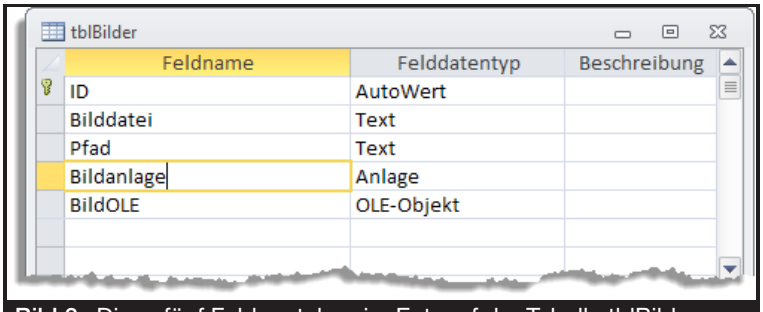



**Bild 2:** Diese fünf Felder stehen im Entwurf der Tabelle tblBilder

## 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.

# 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-

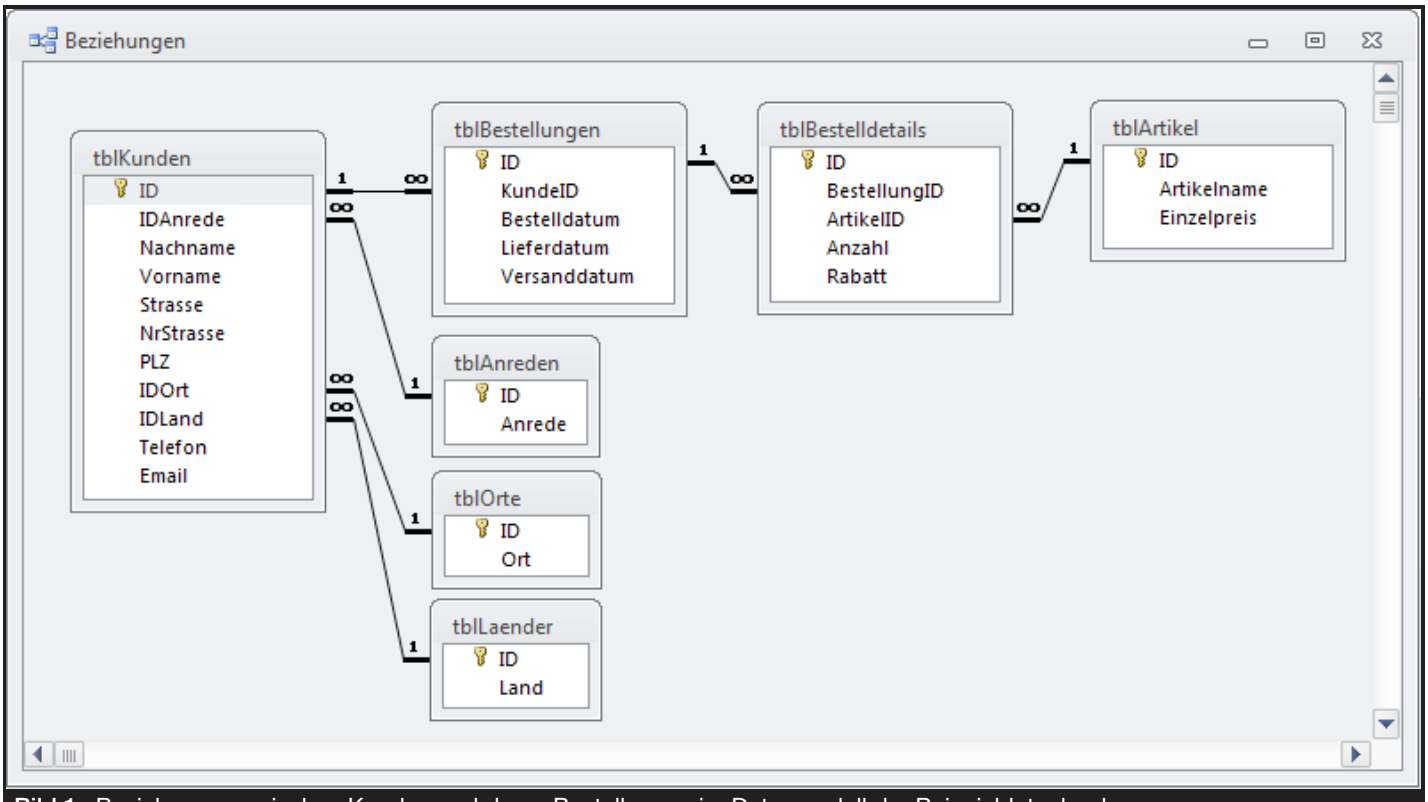



**Bild 1:** Beziehungen zwischen Kunden und deren Bestellungen im Datenmodell der Beispieldatenbank

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.

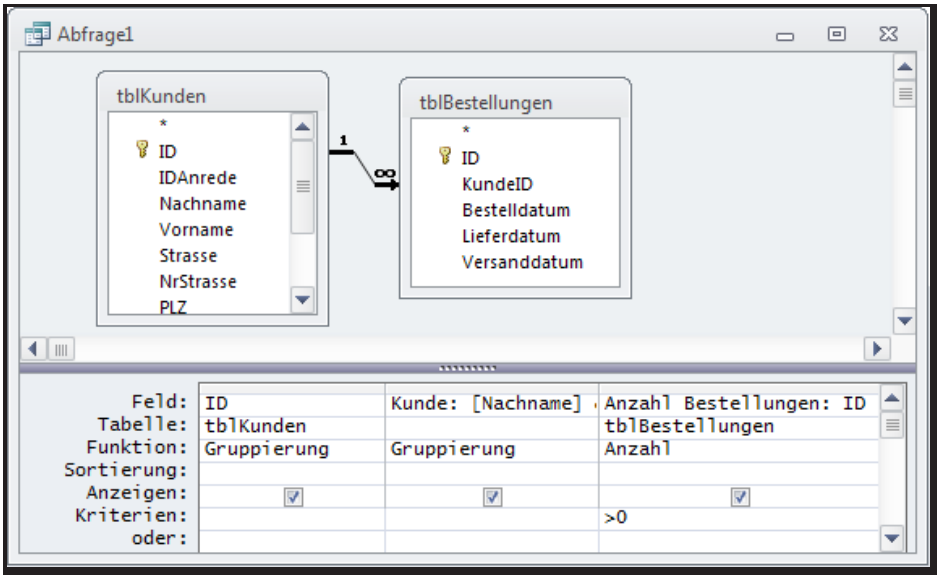



**Bild 2:** Die Abfrage gibt nur jene Kunden aus, die mindestens einmal bestellten

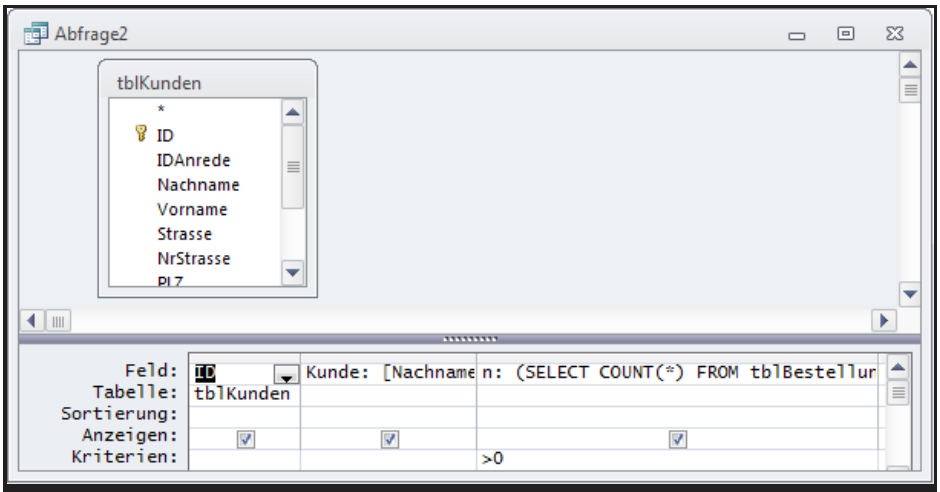



**Bild 3:** Auch hier zeigen sich nur Kunden mit Bestellungen, jedoch mit Unterabfrage

```
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**

Ausgabe 05/2016

# ACCESS BASICS

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




## In diesem Heft:


### TEXTBOXEN IM DETAIL

Das prominenteste Steuerelement in Datenbanken ist das Textfeld. Ein- und Ausgaben über datengebundene Textboxen scheinen trivial zu sein, doch es gibt auch Spezialfälle (Seite 3).

### DESIGNS FÜR STEUERELEMENTE

Ab Access 2010 können Sie Steuerelemente zentral formatieren und mit Designs versehen. Die etwas undurchsichtige Vorgehensweise stellen wir ab Seite 8 im Detail vor.

### WÄHRUNGEN RUNDEN

Im Finanzsektor sollte es keine Rundungsfehler geben. Wohin Sie daher in Ihrer Anwendung genau schauen müssen, zeigen wir Ihnen ab Seite 18.

# 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.

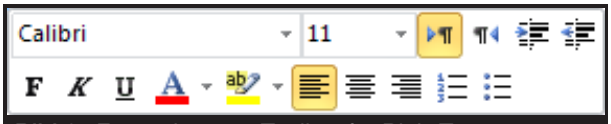



**Bild 2:** Formatierungs-Toolbar für Rich-Text

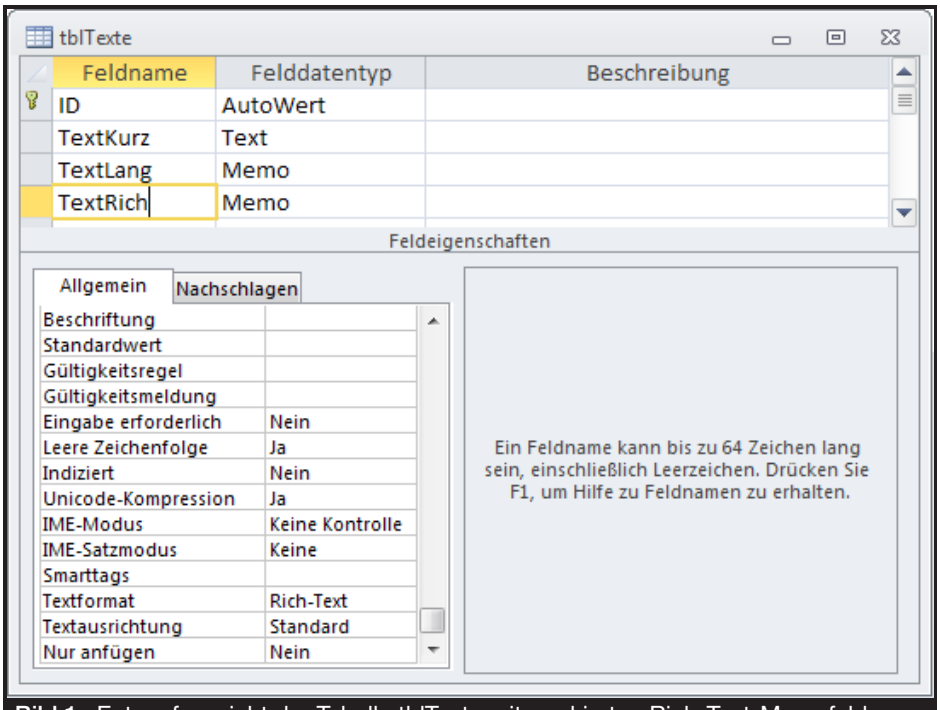



**Bild 1:** Entwurfsansicht der Tabelle tblTexte mit markierten Rich-Text-Memofeld

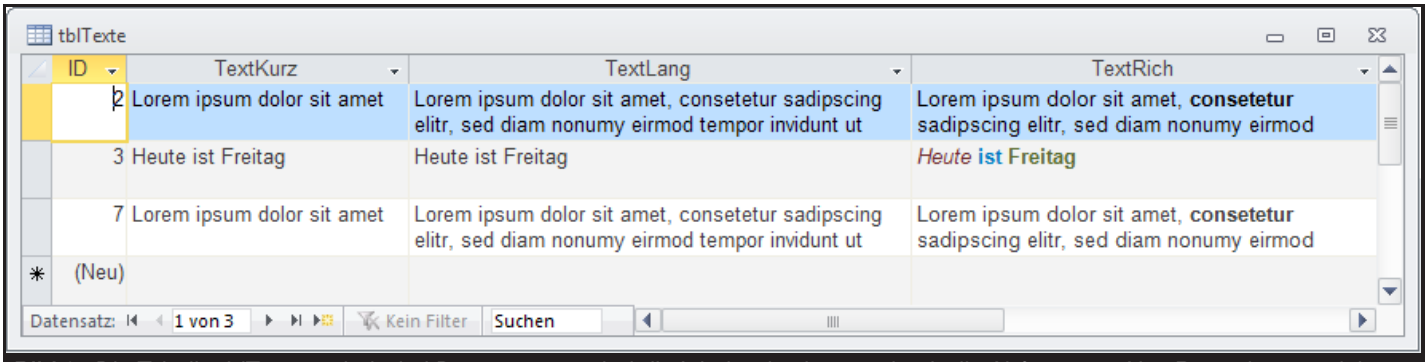



**Bild 3:** Die Tabelle tblTexte enthält drei Datensätze, wobei die Inhalte des letzten durch die Abfrage qry_NewRecord zustandekam

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?

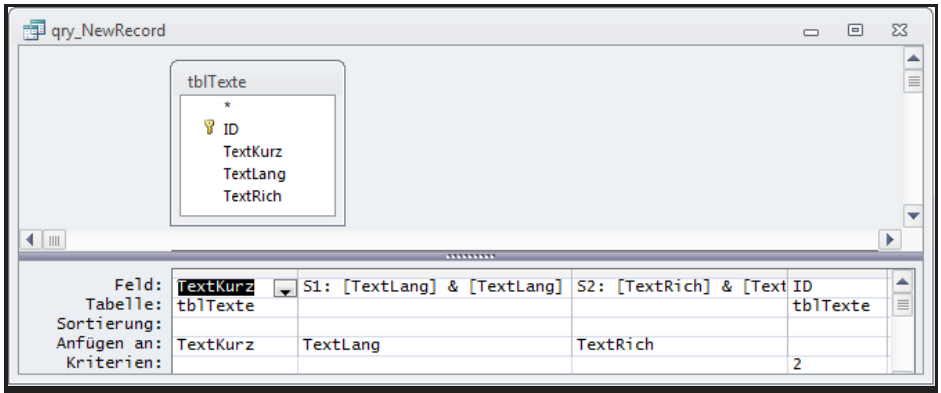



**Bild 4:** Anfügeabfrage qry_NewRecord zum Duplizieren eines Datensatzes innerhalb der gleichen Tabelle mit Vergrößern zweier Textfelder

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.

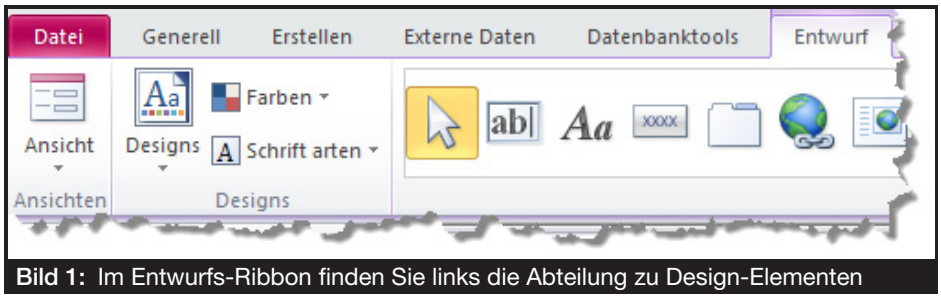



**Bild 1:** Im Entwurfs-Ribbon finden Sie links die Abteilung zu Design-Elementen

### 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**-

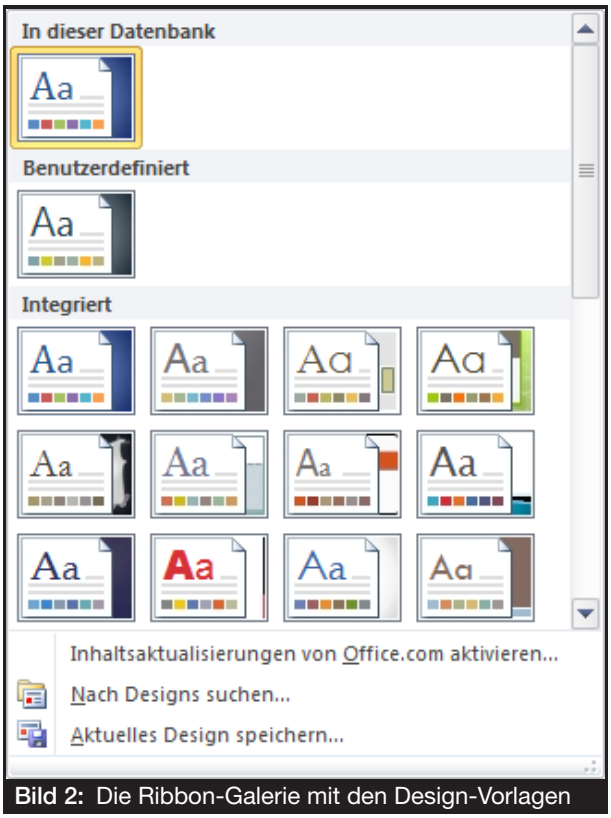



**Bild 2:** Die Ribbon-Galerie mit den Design-Vorlagen

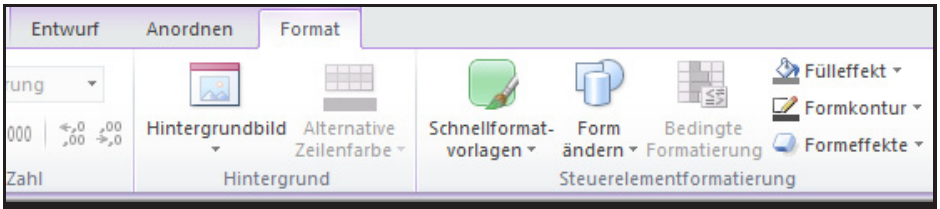



**Bild 3:** Die Designerabteilung Steuerelementformatierung im Format-Ribbon

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

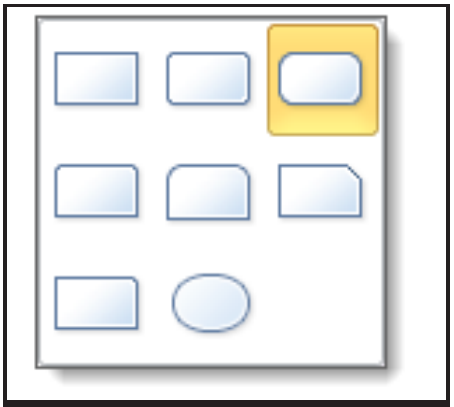


**Bild 4:** Formtypen für Buttons

auf den Farben des aktuell eingestellten **Designs**.

Für alle Steuerelemente jedoch können zumindest Füll- und Randfarben-

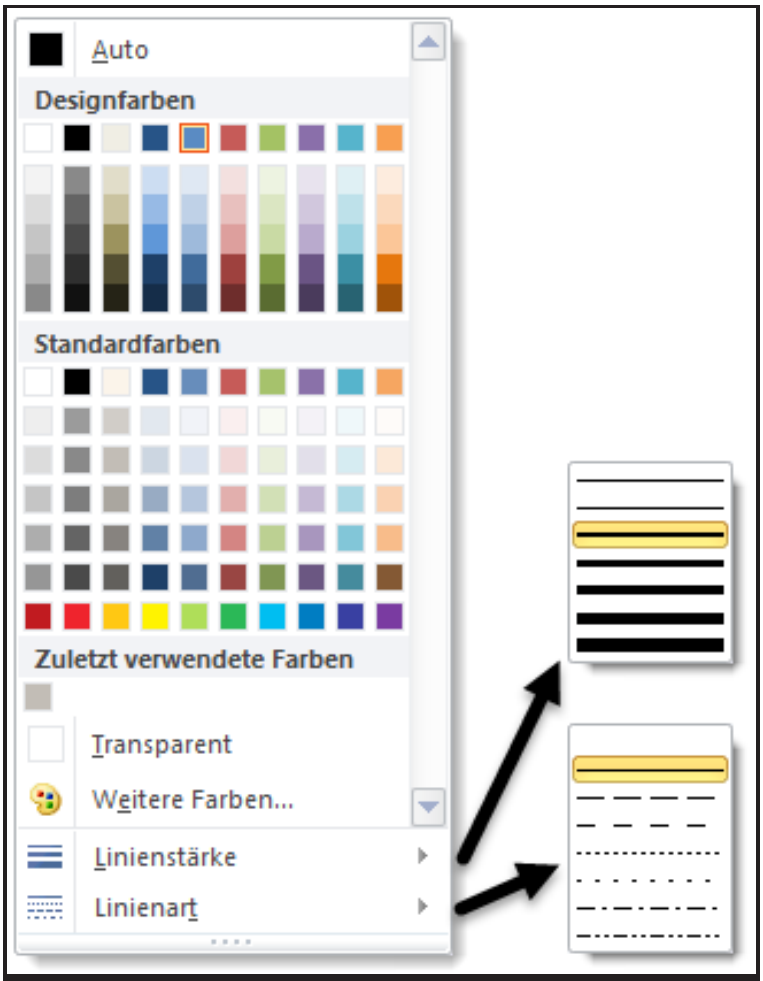



**Bild 5:** Randgestaltung über Farben und Linienarten

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-

# 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.

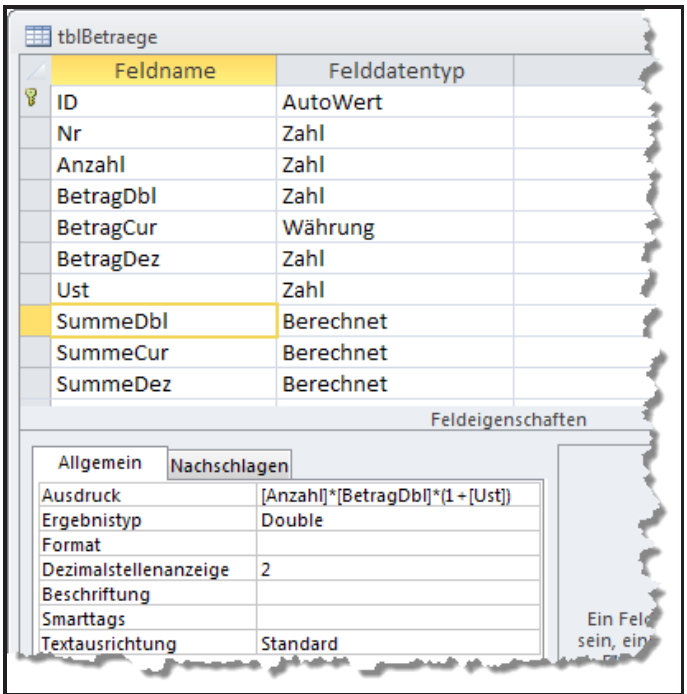



**Bild 1:** Entwurf der Tabelle tblBetraege mit ihren drei Währungsfeldern, sowie darauf bezogenen berechneten Feldern

**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 |

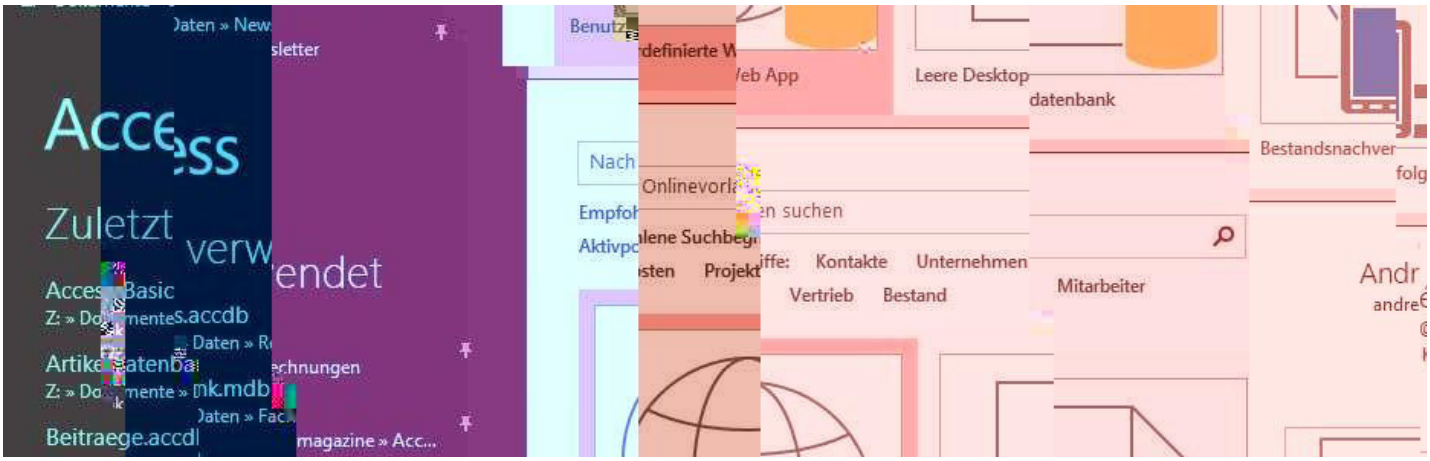
Datensatz: 1 von 10000 | Kein Filter | Suchen

**Bild 2:** Das ist die Datenblattansicht der Tabelle tblBetraege, unten mit der hinzugefügten Aggregatzeile für die Summen

Ausgabe 06/2016

# ACCESS BASICS

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




## In diesem Heft:

DATEN SICHERN ÜBER XML (S. 2)

FIXIERTE FORMULARE (S. 6)

SEITENNAVIGATION (S. 12)

DATENBANKEN VON ALTLASTEN SÄUBERN (S. 17)

# 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>"
```

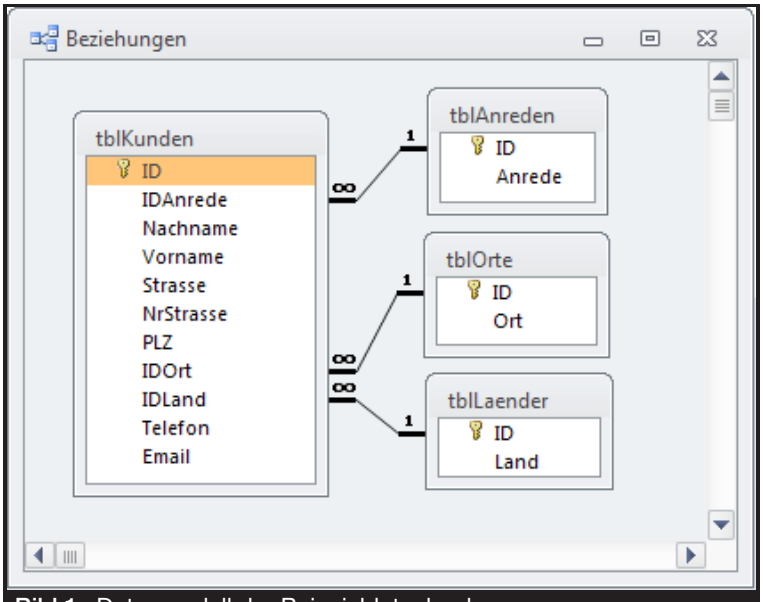


**Bild 1:** Datenmodell der Beispieldatenbank

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

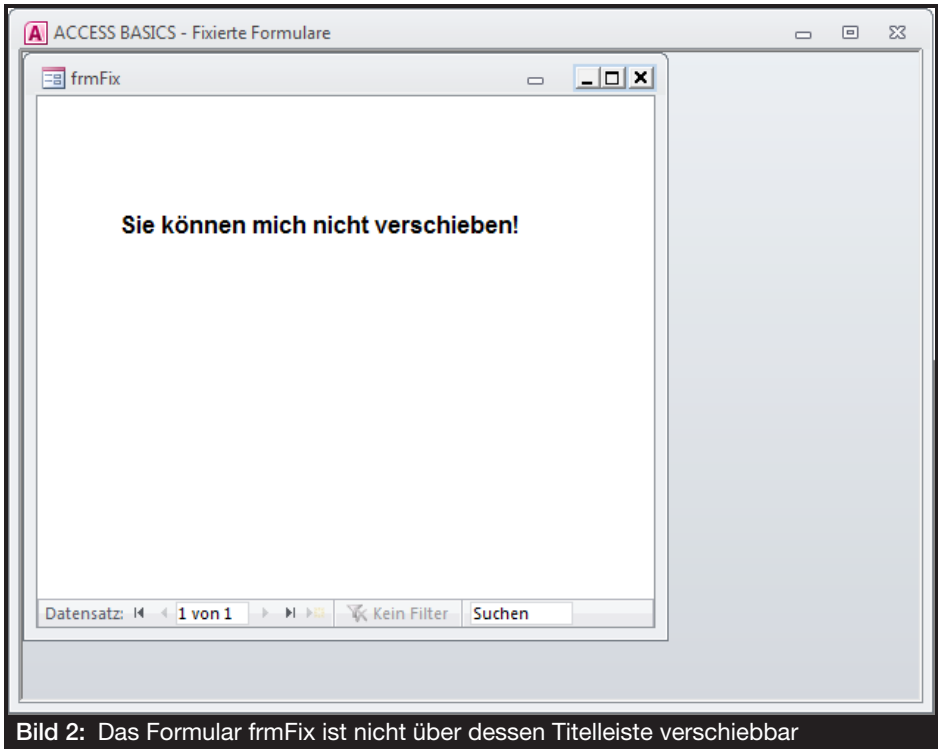



**Bild 2:** Das Formular frmFix ist nicht über dessen Titelleiste verschiebbar

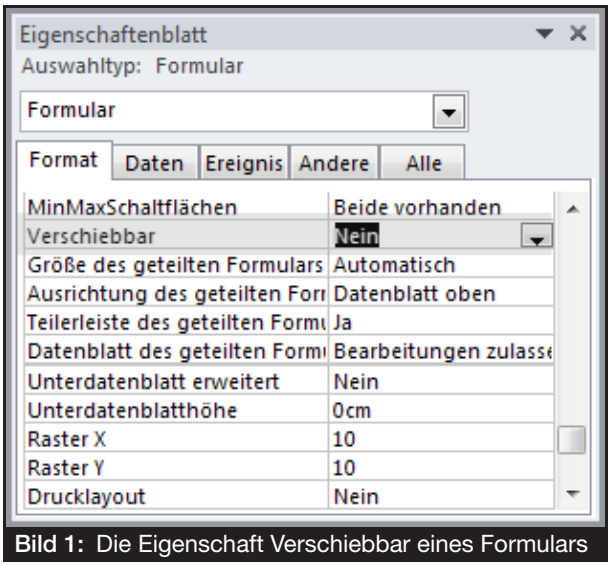



**Bild 1:** Die Eigenschaft Verschiebbar eines Formulars

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,

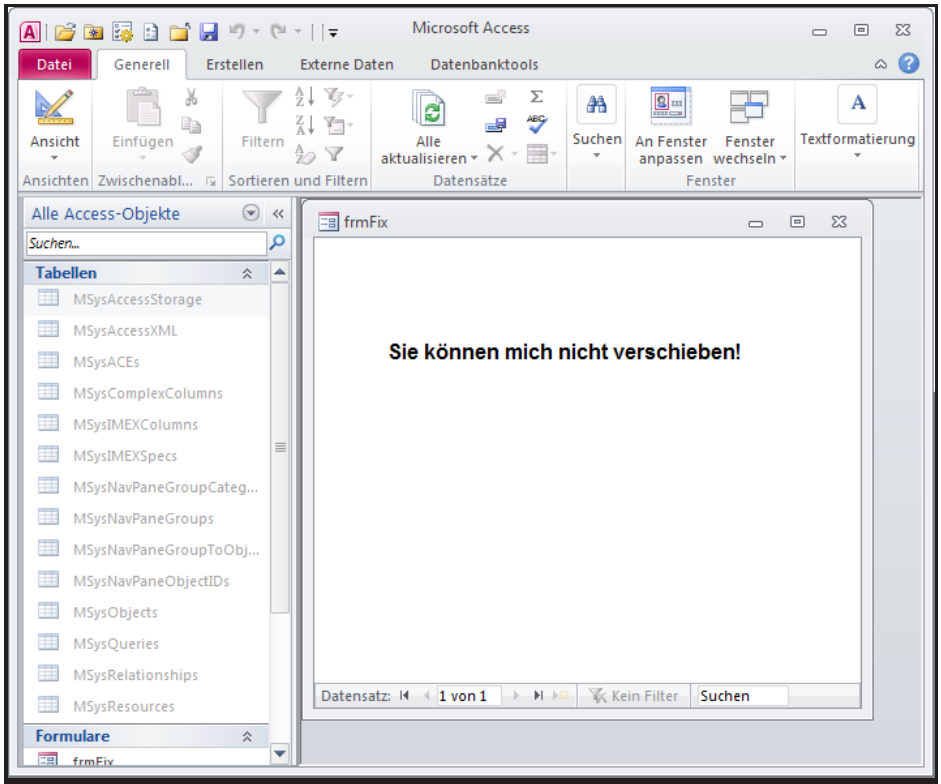



**Bild 3:** Am fixierten Modus des Formulars ändert sich auch bei eingeblendetem Ribbon und Navigationsbereich nichts

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

# 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.

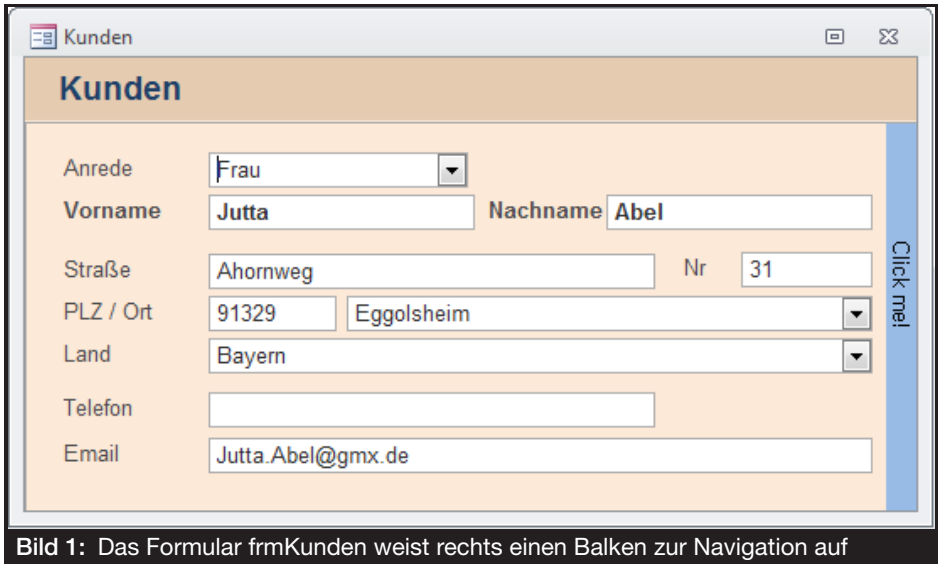



**Bild 1:** Das Formular frmKunden weist rechts einen Balken zur Navigation auf

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.

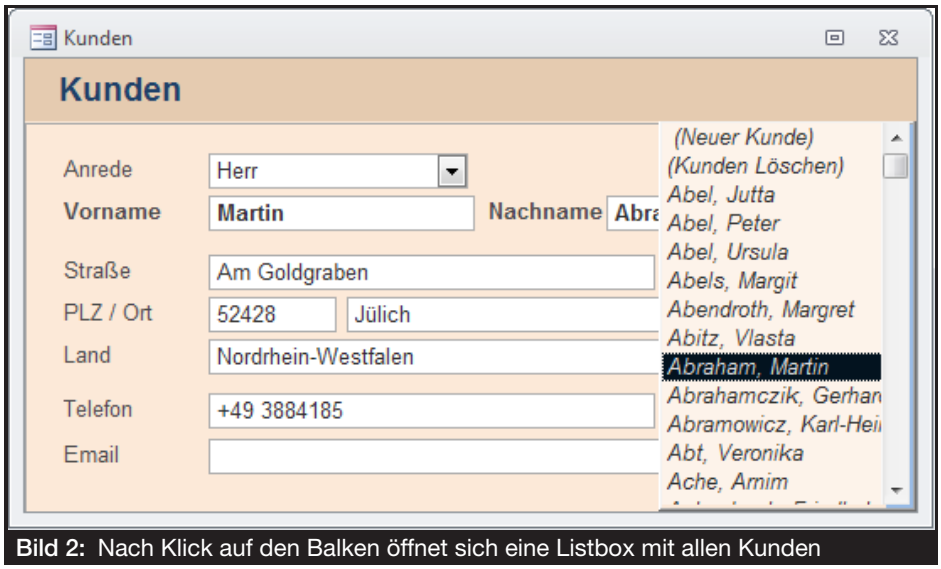



**Bild 2:** Nach Klick auf den Balken öffnet sich eine Listbox mit allen Kunden

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**.

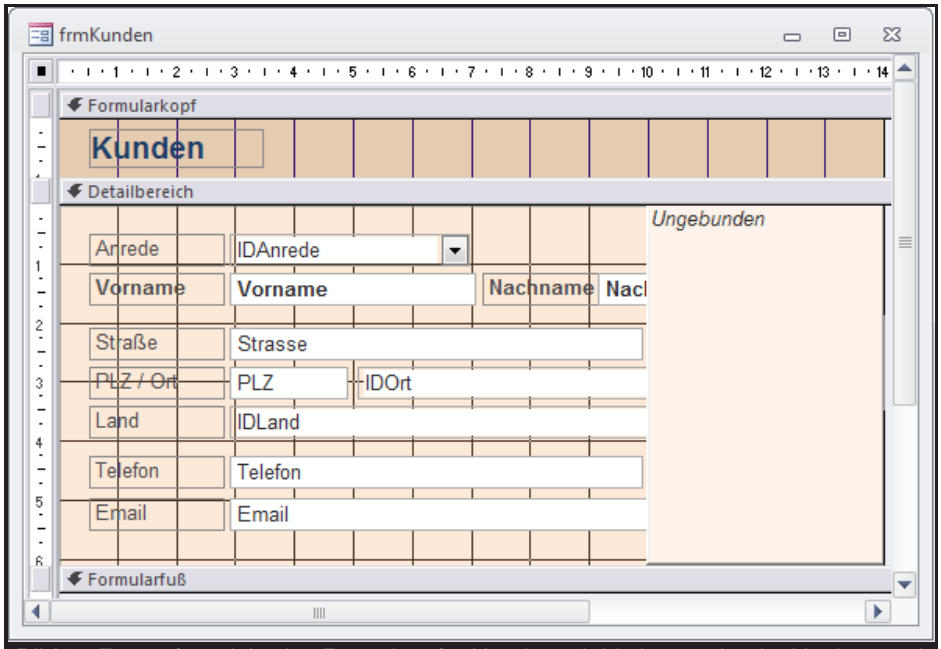



**Bild 3:** Entwurfsansicht des Formulars frmKunden mit Listbox rechts im Vordergrund

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-

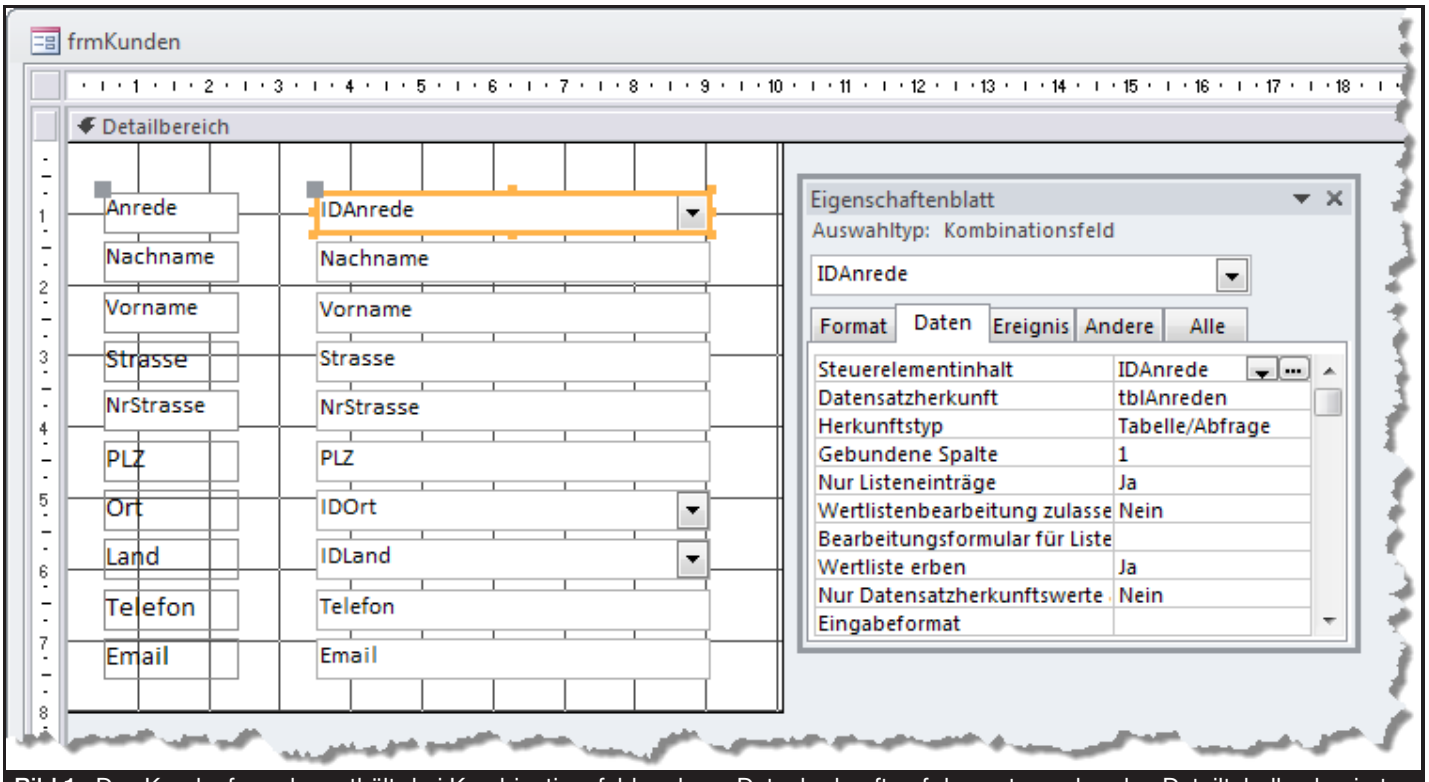



**Bild 1:** Das Kundenformular enthält drei Kombinationsfelder, deren Datenherkunft auf den entsprechenden Detailtabellen basiert

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.

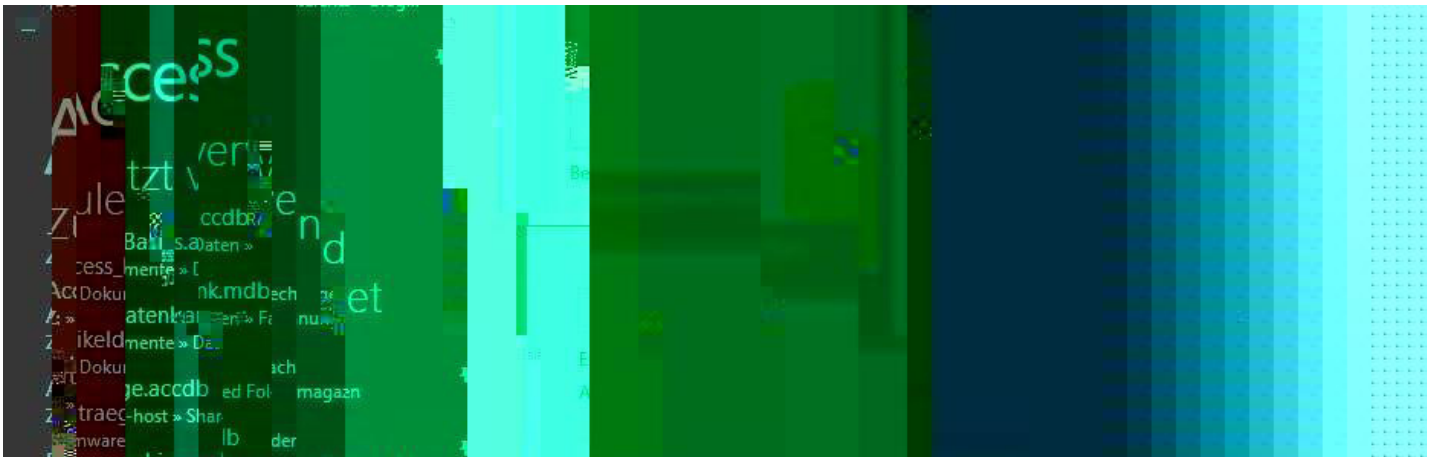
Was hat dies alles aber mit dem Müll zu tun, von dem eingangs die Rede war? Leider kommt es vor, dass besonders bei Formularen, die Sie aus anderen Datenbanken importierten, die auf sie bezogenen internen Abfragen nicht ordnungsgemäß gelöscht werden, wenn sie selbst entfernt werden. Damit finden sich 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

Ausgabe 07/2016

# ACCESS BASICS

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



In diesem Heft:


## FLYING CONTROLS (S. 3)

## ZUGRIFF AUF SERVER ÜBER ODBC (S. 9)

## RIBBON-PROGRAMMIERUNG, TEIL I (S. 17)

# 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)

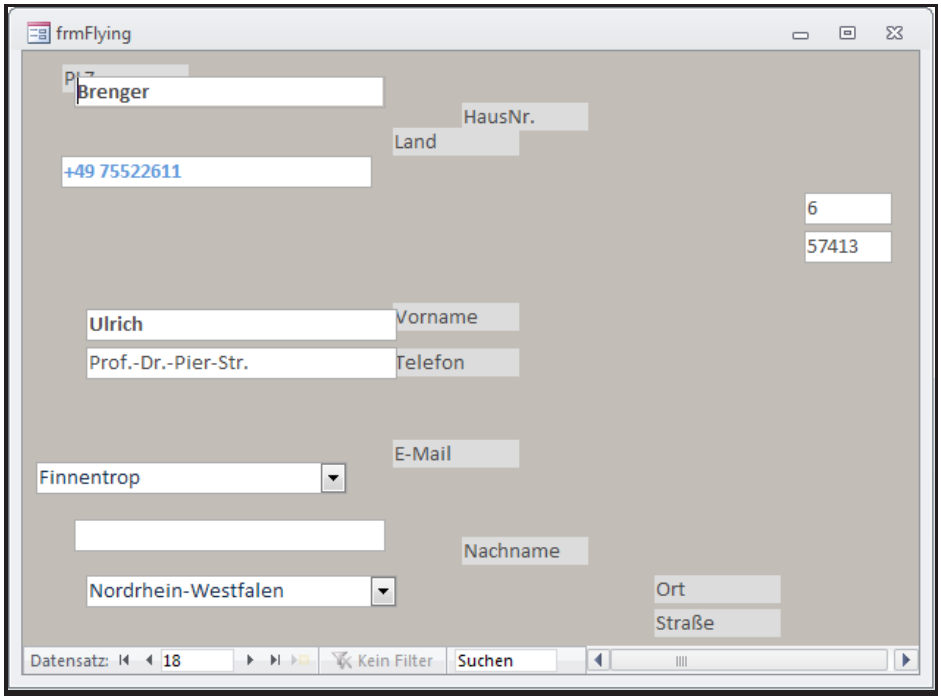



**Bild 1:** Das Formular frmFlying zeigt in der Gegend herumfliegende Steuerelemente

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

# 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-

# 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):

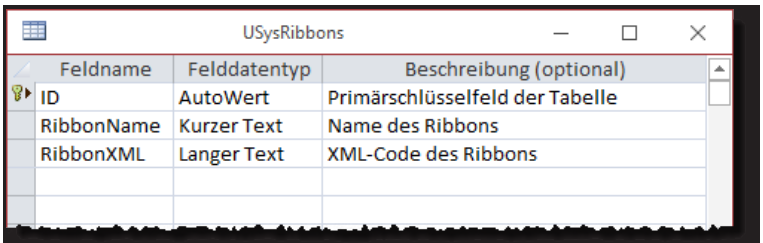



**Bild 1:** Entwurf der Tabelle **USysRibbons**

- **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.

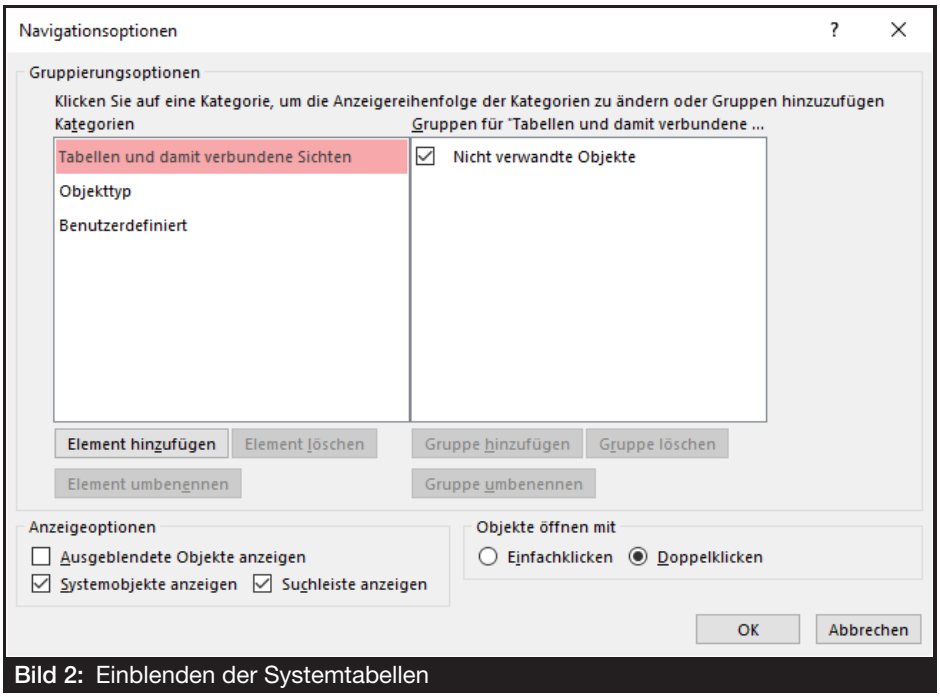



**Bild 2:** Einblenden der Systemtabellen

## Ribbon-Definition für ein einfaches Ribbon

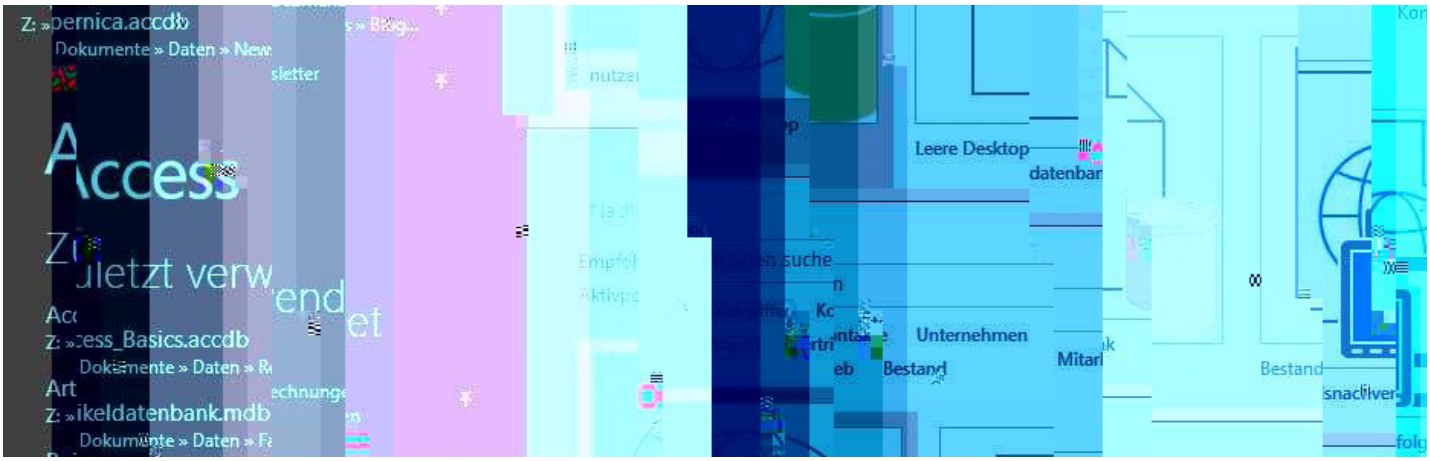
Eine der einfachsten Ribbon-Varianten besteht aus lediglich einem **tab**-Element, einem **group**-Element und einem **button**-Element. Das waren die sichtbaren Elemente, es gibt noch ein paar weitere, die nur im XML-Dokument auftauchen und für die Struktur verantwortlich sind. Geben Sie nun den folgenden 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"
```

Ausgabe 08/2016

# ACCESS BASICS

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




## In diesem Heft:

**ZUGRIFF AUF SERVER ÜBER ODBC, TEIL II (S. 3)**

**LOKALER WEBSHOP (S. 111)**

# 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**

# 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-

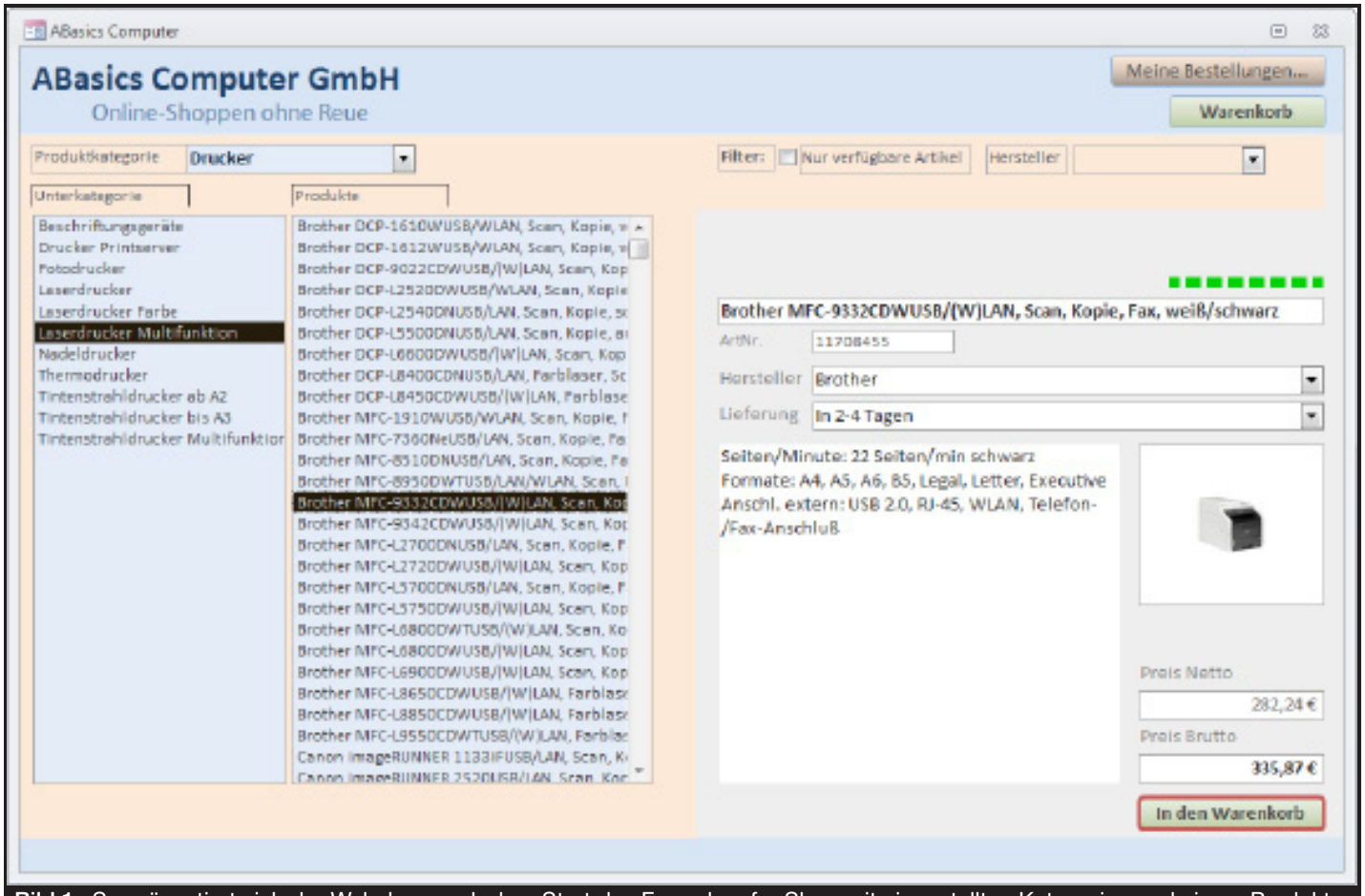



**Bild 1:** So präsentiert sich der Webshop nach dem Start des Formulars frmShop mit eingestellten Kategorien und einem Produkt

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-

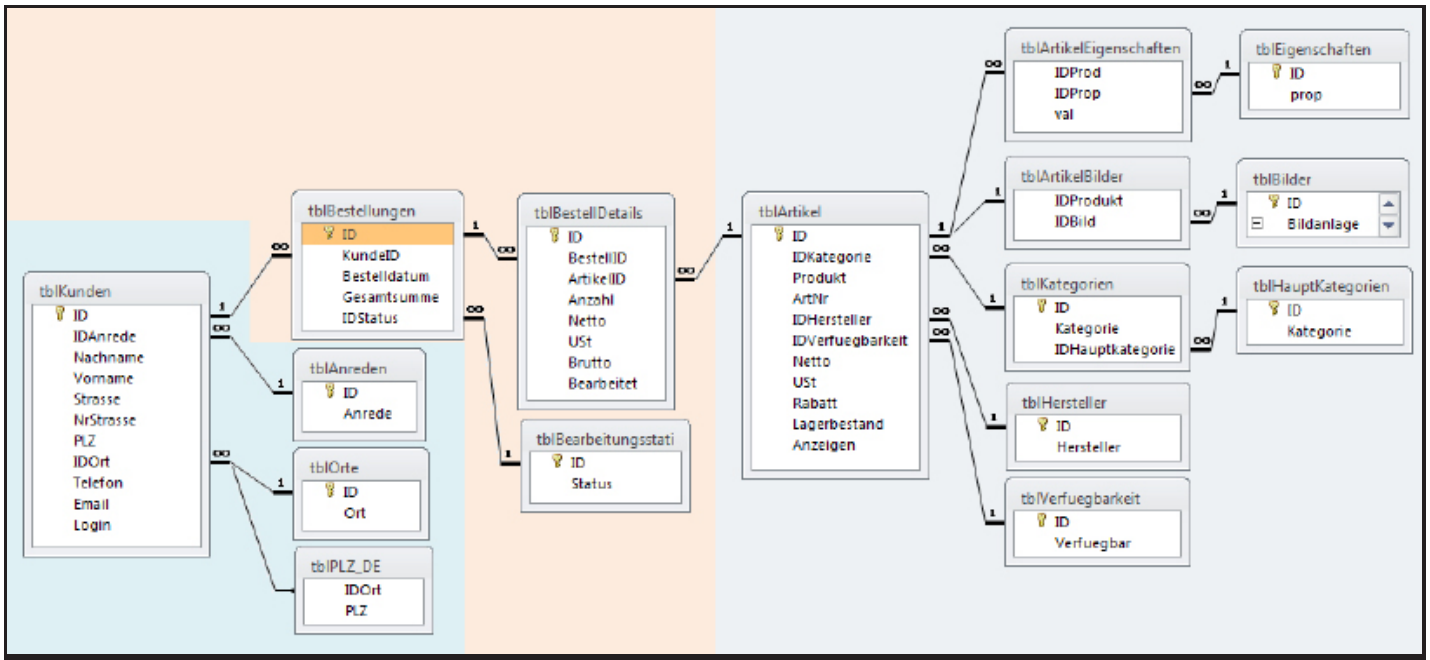
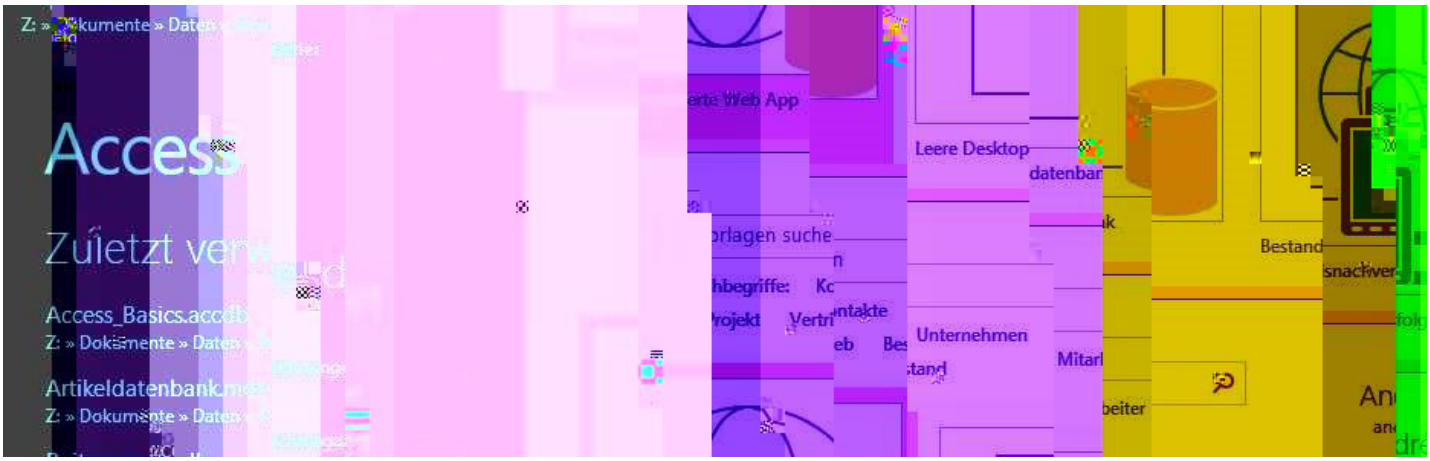



**Bild 2:** Layout des von einer Bestelldatenverwaltung abgeleiteten neuen Datenmodells des Webshops im Beziehungsfenster

Ausgabe 09/2016

# ACCESS BASICS

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




## In diesem Heft:

**Steuerelemente im Eigenbau: Range Control (S. 3)**

**Rezepte verwalten (S. 10)**

**Nährwertdatenbank (S. 17)**

# 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.

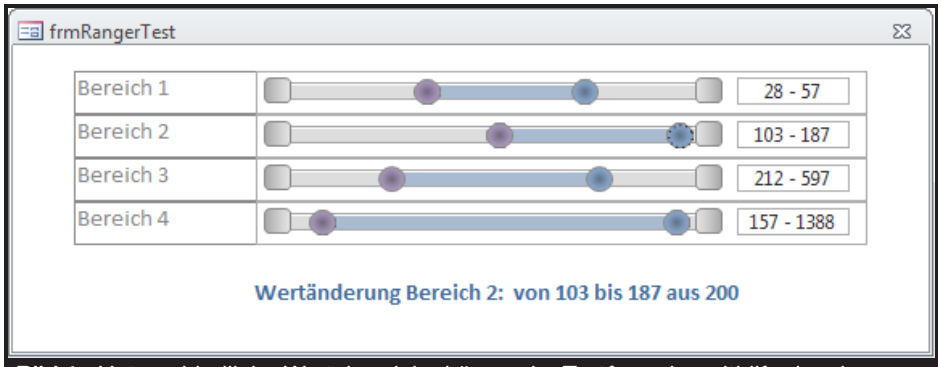



**Bild 1:** Unterschiedliche Wertebereiche können im Testformular mithilfe der vier Range Controls mit der Maus über Ziehpunkte eingestellt werden

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.

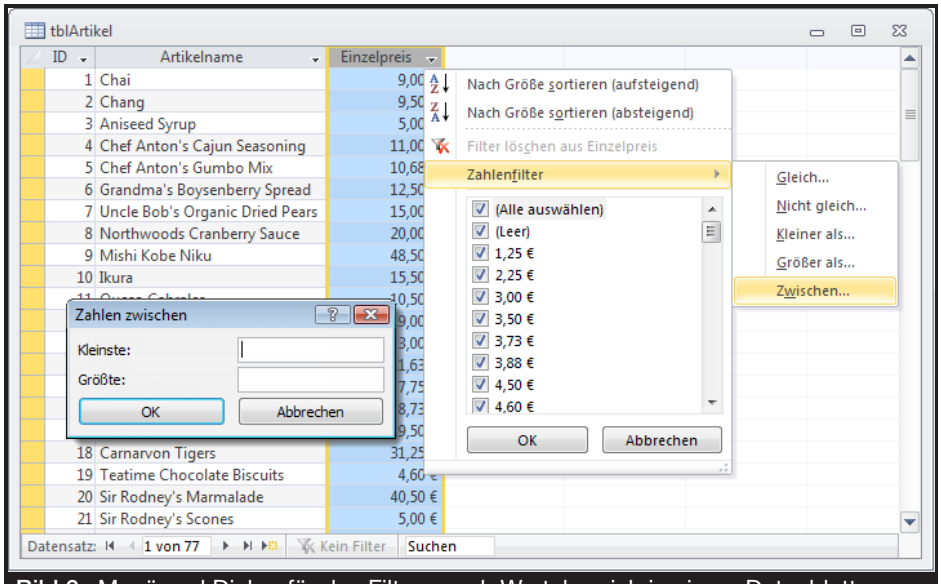



**Bild 2:** Menü und Dialog für das Filtern nach Wertebereich in einem Datenblatt

## 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.

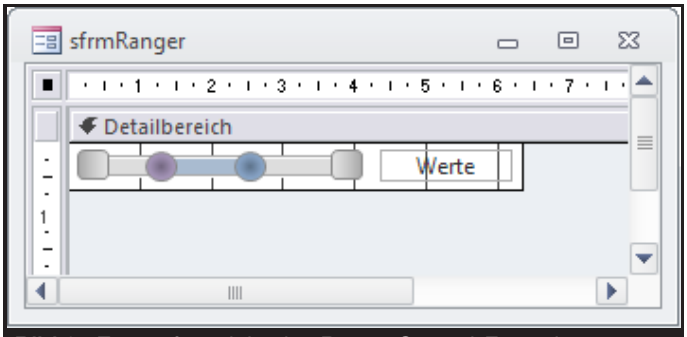


**Bild 3:** Entwurfsansicht des Range Control-Formulars

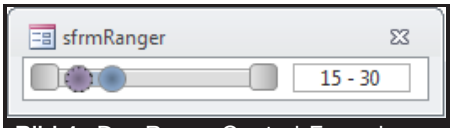


**Bild 4:** Das Range Control-Formular zur Laufzeit nach dem Öffnen

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

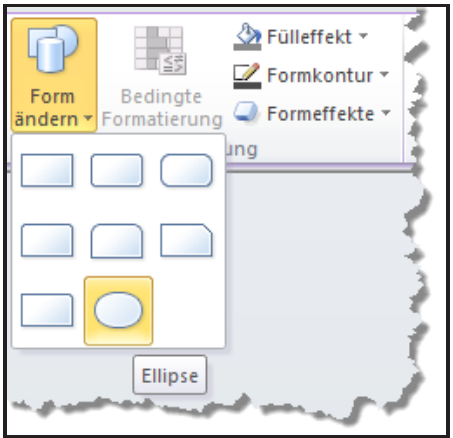


**Bild 6:** Form für die Buttons

# 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.

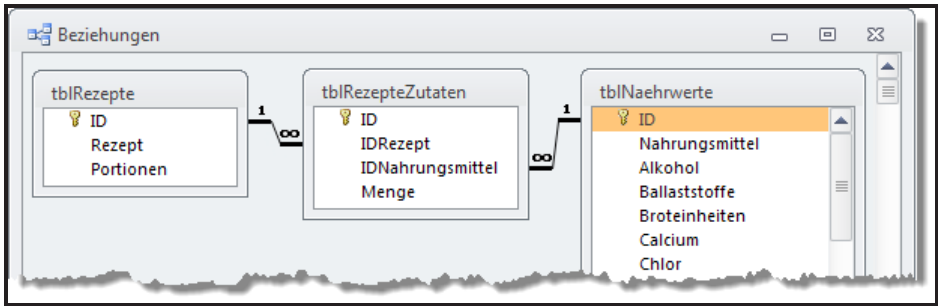
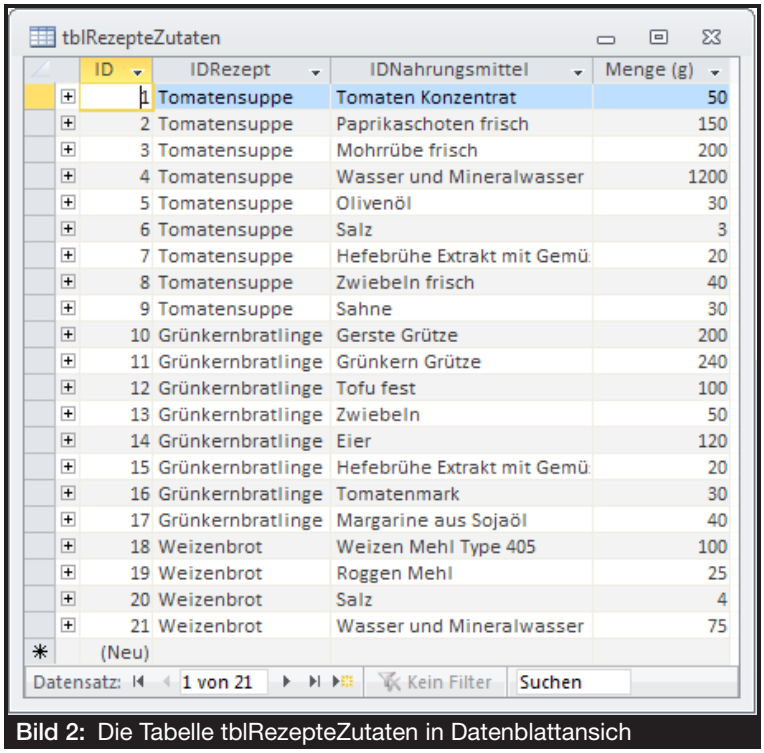



**Bild 1:** Das Modell der Rezepteverwaltung ist gleich wie das der Nährwertdatenbank

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

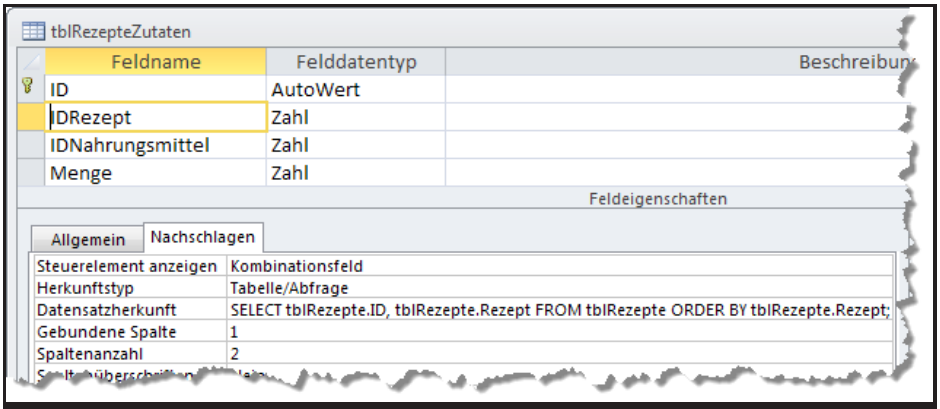



**Bild 3:** Details zum Entwurf der Tabelle tblRezepteZutaten für das Feld IDRezept

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.

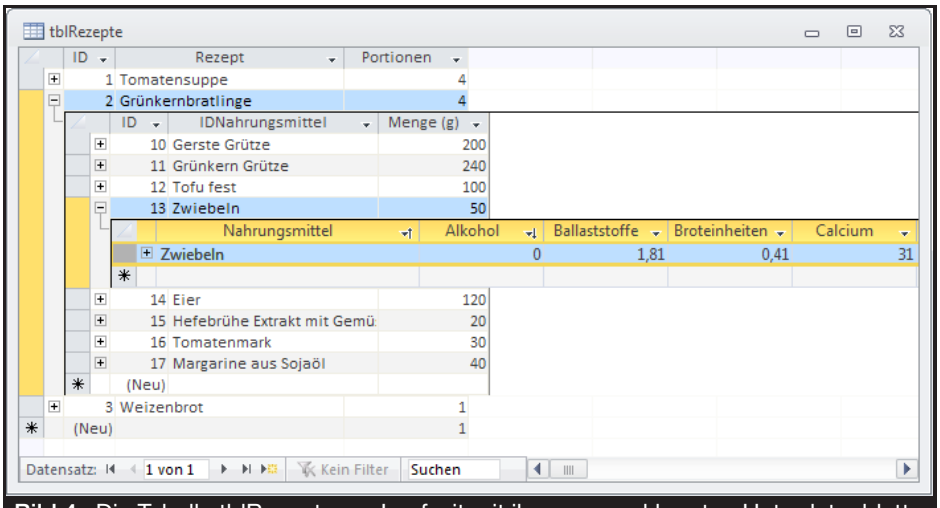



**Bild 4:** Die Tabelle tblRezepte zur Laufzeit mit ihrem ausgeklappten Unterdatenblatt

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 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.

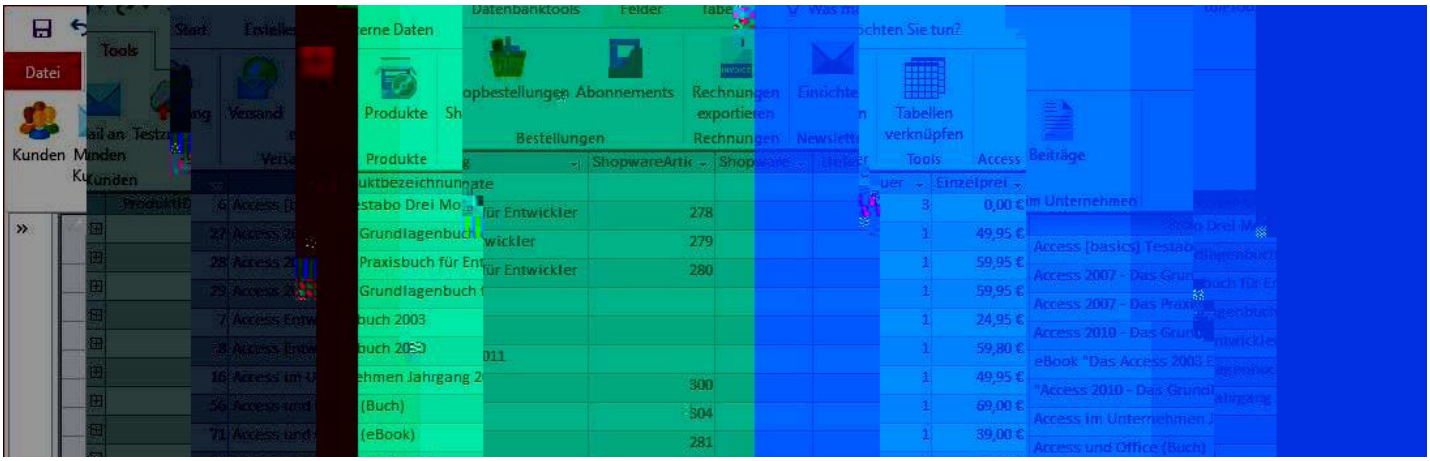
Und schließlich erwirkt ein Klick auf die Schaltfläche **Nährwertdaten des Rezepts** das Öffnen einer Abfrage, die weitere Kennzahlen zu den Inhaltsstoffen des Gesamtrezepts ausgibt. Dazu später mehr. Schauen wir uns zunächst Schritt für Schritt den Aufbau und die Code-Teile des Formulars frmRezepte an.

Das Unterformular für die Zutaten kommt aus dem Formular **sfrmRe-**

Ausgabe 10/2016

# ACCESS BASICS

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



In diesem Heft:

**Lokaler Webshop, Teil II (S. 3)**

**Datei- und Verzeichnisoperationen mit der Shell (S. 14)**

# 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

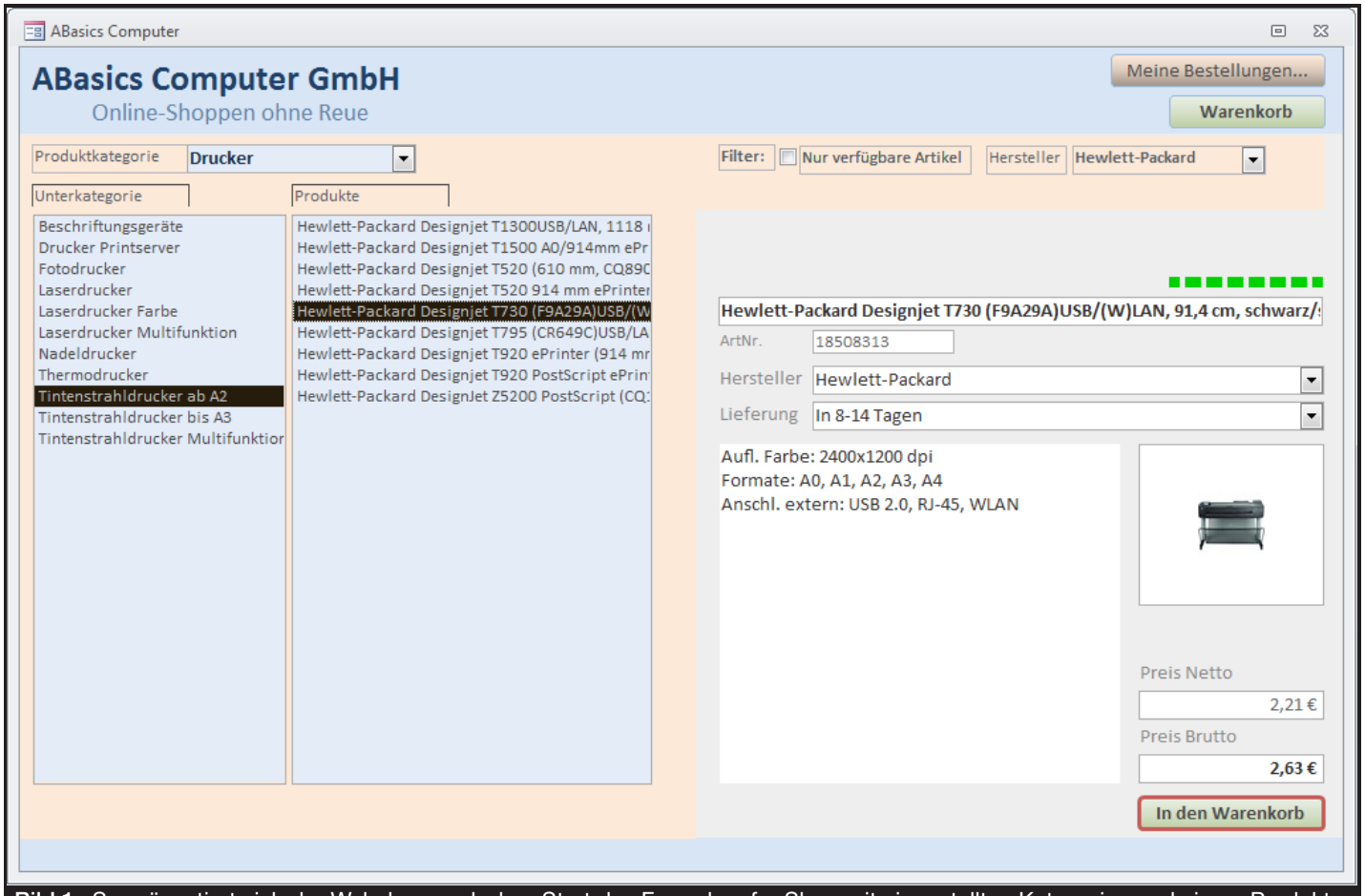



**Bild 1:** So präsentiert sich der Webshop nach dem Start des Formulars frmShop mit eingestellten Kategorien und einem Produkt

```
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.

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

```
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

Klicken Sie erneut auf den Button, so zeigt sich eine abweichende

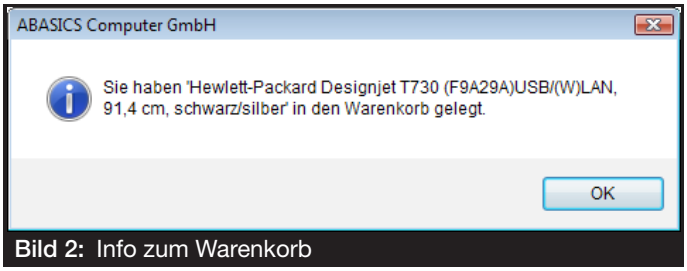



**Bild 2:** Info zum Warenkorb

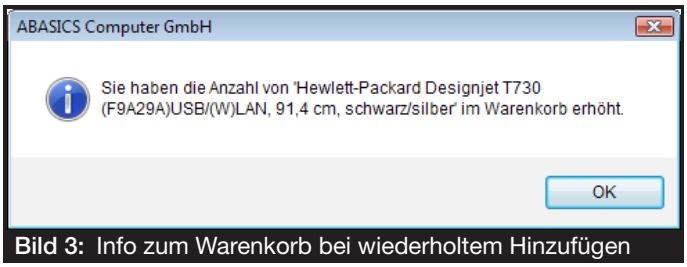



**Bild 3:** Info zum Warenkorb bei wiederholtem Hinzufügen

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

# 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.

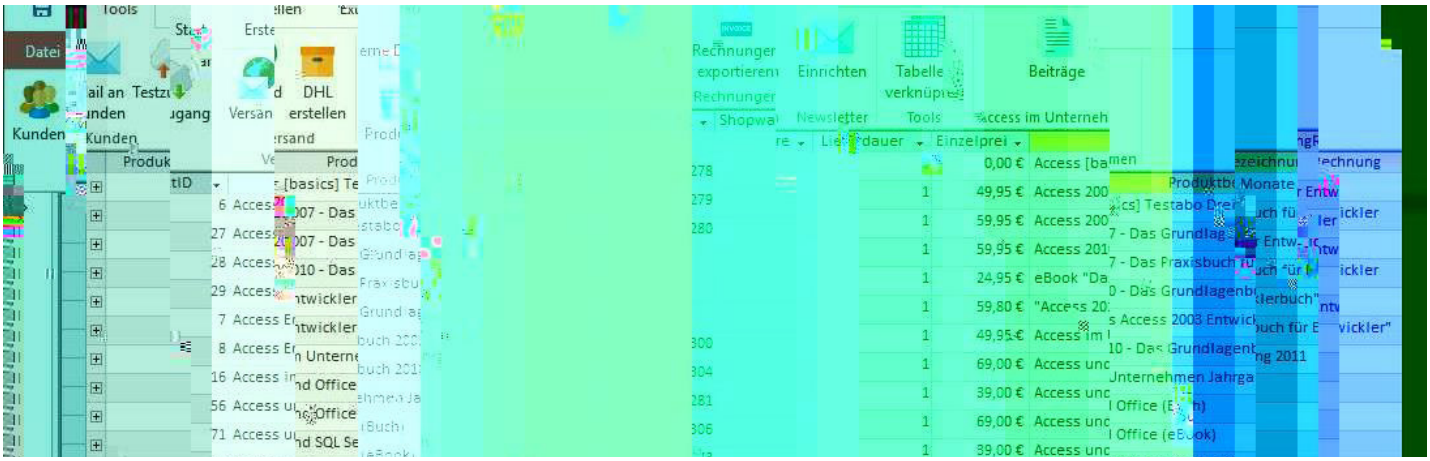
### FolderExists

Aus dem bisher Erläuterten geht die einfachste Anwendung hervor, nämlich der Test, ob ein Verzeichnis denn

Ausgabe 11-12/2016

# ACCESS BASICS

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



## In diesem Heft:


**ERFAHRUNGEN MIT LIBREOFFICE BASE (S. 3)**

**OUTLOOK-E-MAILS IMPORTIEREN (S. 11)**

**RIBBONS, TEIL II: BILDER IM RIBBON (S. 22)**

**EREIGNISSE ANDERER OBJEKTE NUTZEN (S. 27)**

**FORMULARPOSITIONEN LESEN UND SETZEN (S. 32)**

**MEHRERE INSTANZEN EINES FORMULARS (S. 38)**


André Minhorst Verlag

# 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.

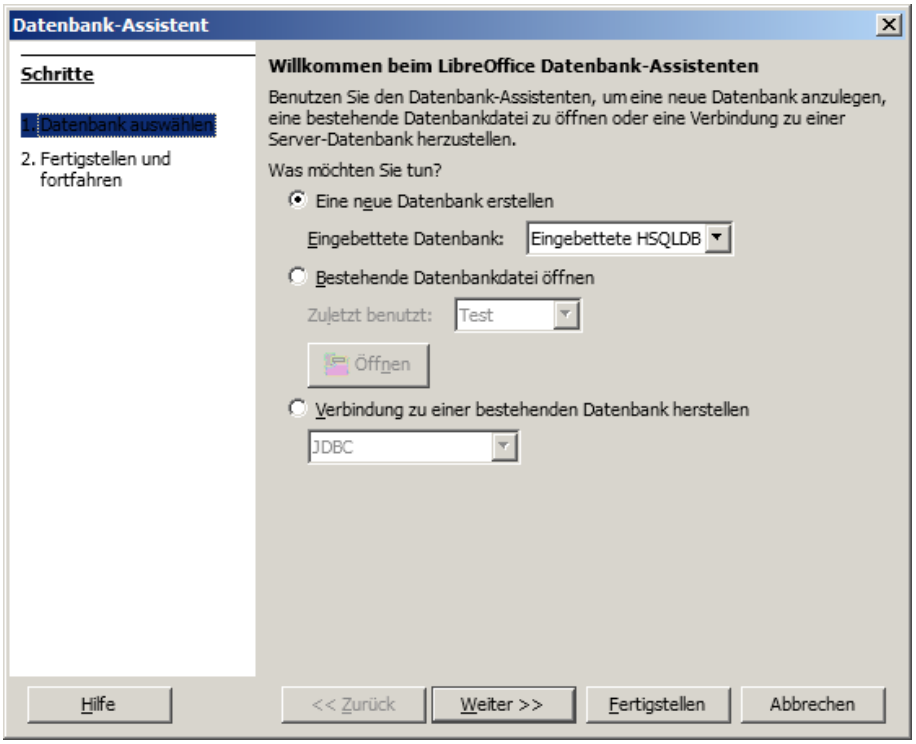



**Bild 1:** Eine neue Datenbank wird in Base immer über den Assistenten angelegt

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-

# 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.

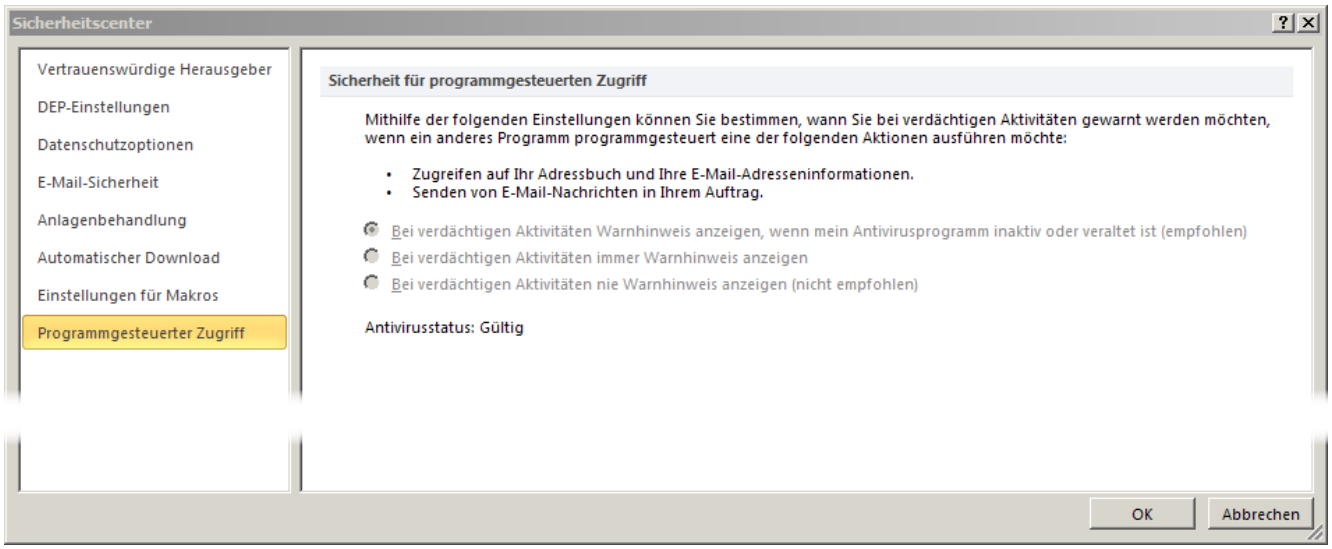



**Bild 1:** Die Sicherheitseinstellungen von Outlook beim Zugriff über Automation

### 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.

# 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**.

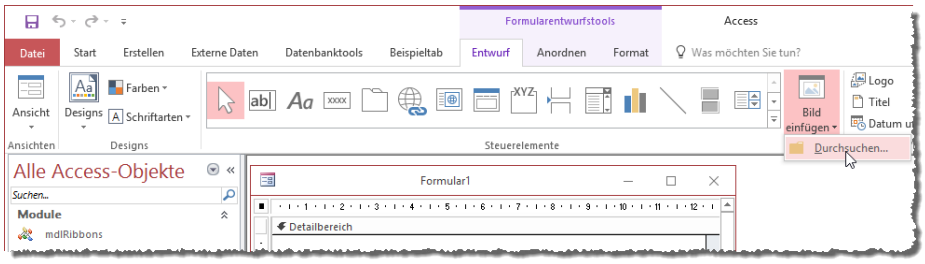



**Bild 1:** Hinzufügen einer Bilddatei per Formular-Entwurf

### 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

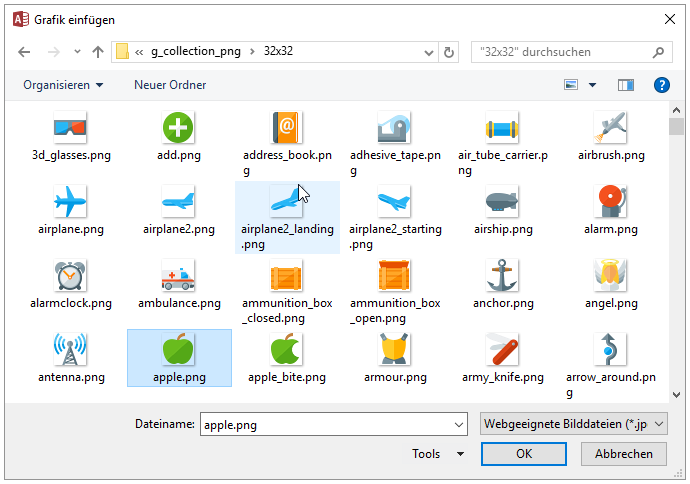



**Bild 2:** Auswahl der Bilddatei

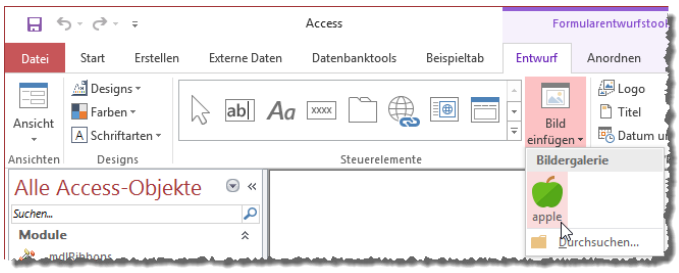



**Bild 3:** Hinzugefügte Bilddatei im Ribbon

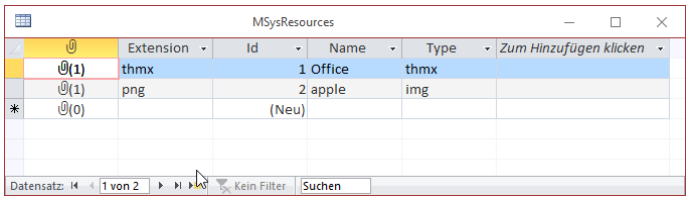



**Bild 4:** Neue Bilddatei in der Tabelle **MSysResources**

**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.

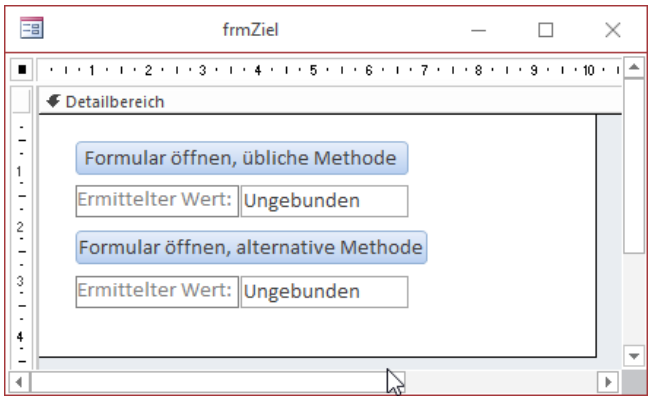



**Bild 1:** Formular, das andere Formular aufruft und vor dem Schließen deren Werte einliest

## 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
```

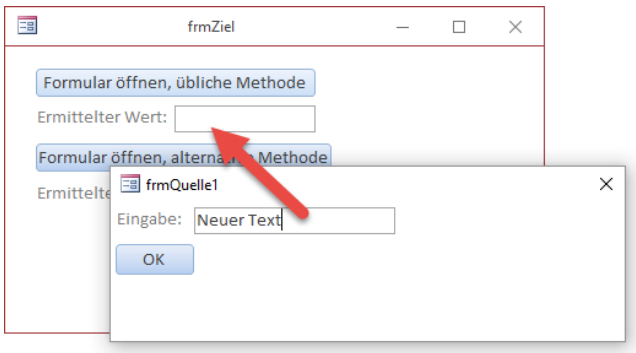



**Bild 2:** Aufruf und Übergabe des Textes von Formular zu Formular

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.

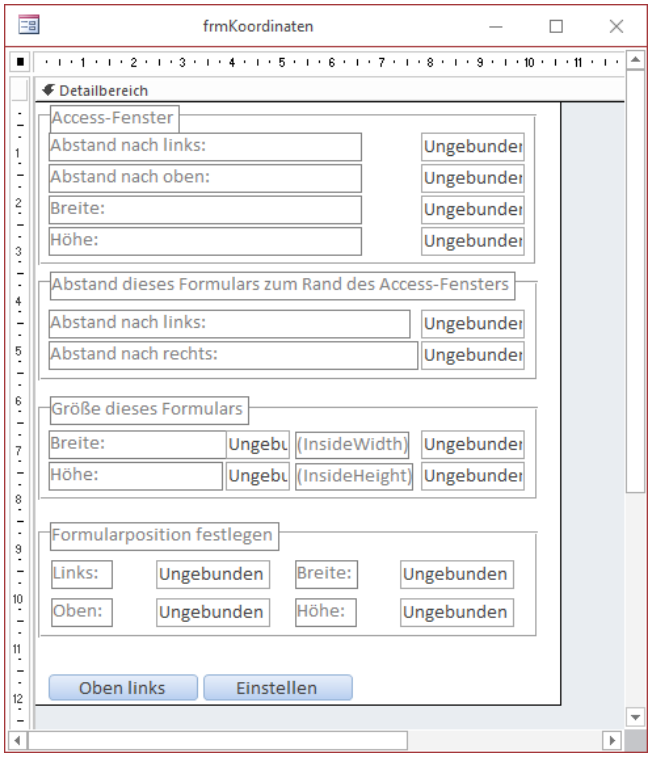



**Bild 1:** Formular zur Anzeige der aktuellen Abmessungen des Formulars und des Access-Fensters

### 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**.

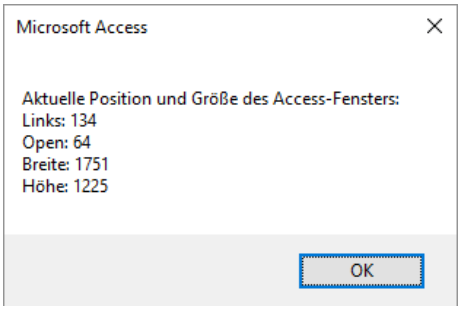



**Bild 2:** Test der Funktion zum Ermitteln von Position und Größe des Access-Fensters

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
```

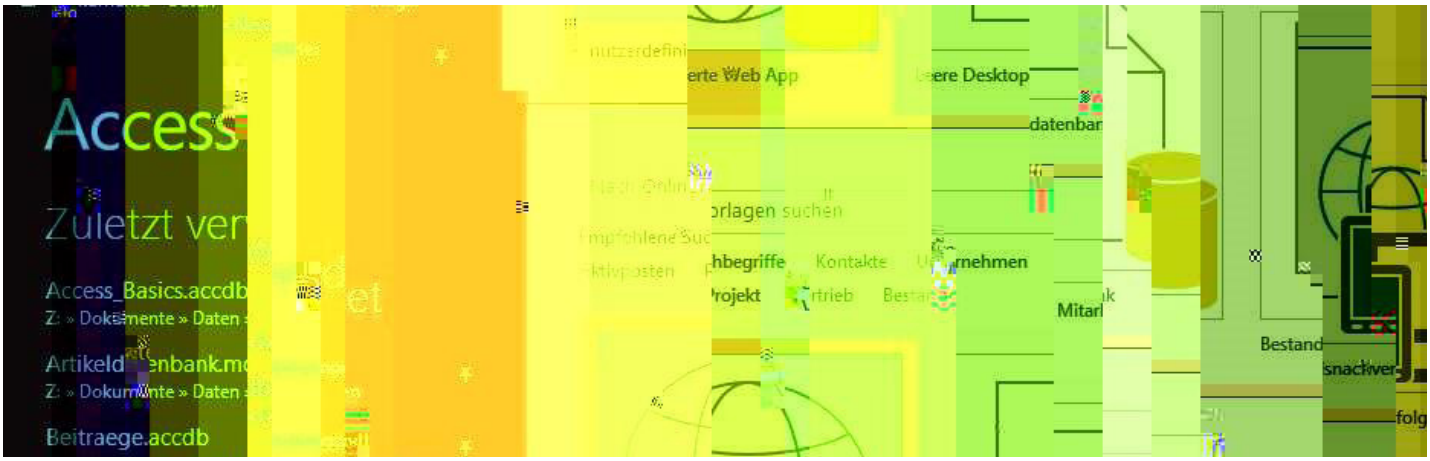
Nun kann die Funktion die API-Funktion nutzen und dieser mit **Application.hWndAccessApp** ein Handle auf die aktuelle Access-Anwendung sowie die noch leere Variable **rectMainWindow** des Typs **Rect** übergeben. Diese wird dann gefüllt. Die vier Werte des Typs **rectMainWindow** übergeben wir dann an die Rückgabeparameter der Funktion – zum Beispiel so:

```
sngLeft = rectMainWindow.Left
```

Ausgabe 01/2017

# ACCESS BASICS

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




## In diesem Heft:

**ANWENDUNGS-DESIGN MIT POSITIONIERTEN FORMULAREN (S. 3)**

**BERICHTE ZUR LAUFZEIT FILTERN (S. 13)**

**TABELLENINHALTE VERSCHLÜSSELN (S. 21)**

**RIBBONS, TEIL III: INTERAKTION IM RIBBON (S. 29)**

**PLATZHALTER IN TEXTDATEIEN FÜLLEN (S. 37)**

# 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?

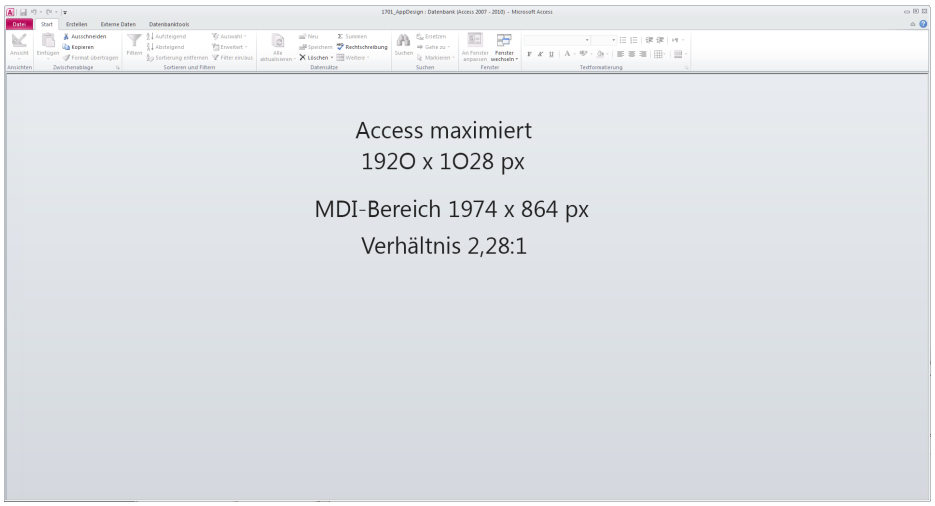



**Bild 1:** Access maximiert auf einem Laptop-Monitor ohne Navigationsbereich

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.

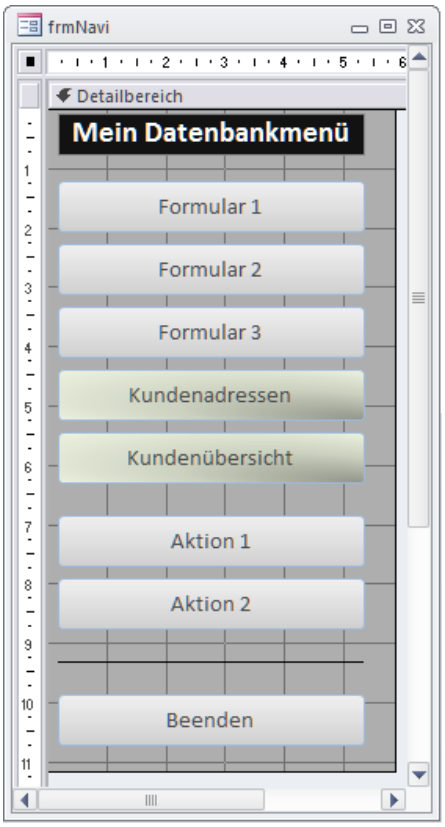



**Bild 3:** Das Switchboard-Formular frmNavi in der Entwurfsansicht

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**:

# 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**.

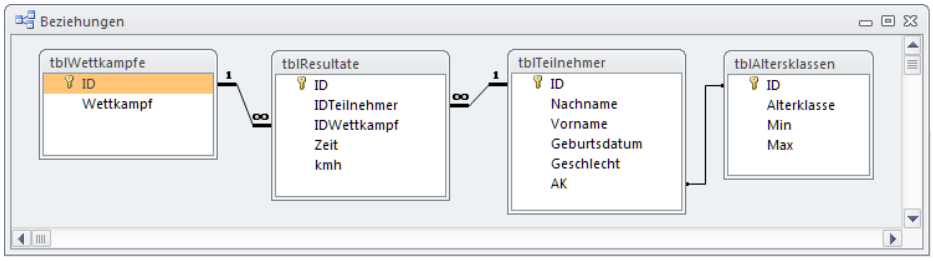



**Bild 1:** Das Datenmodell der Beispieldatenbank zu fingierten Wettkampfdaten

## 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.

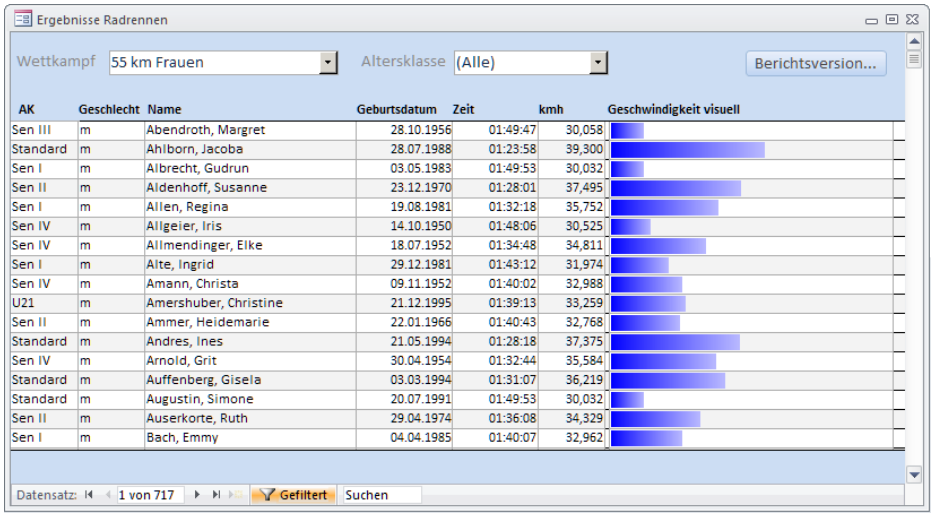
Ziel der aktuellen Beispieldatenbank ist nun nicht ein Web-Export, sondern der Ausdruck der Resultate über einen Bericht. Diese sollen wahlweise nach Wettkampfkategorien und Altersklassen filterbar sein. Das damals dazu verwendete Formular ist in der Datenbank ebenfalls enthalten (**frmErgebnisse**), 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.

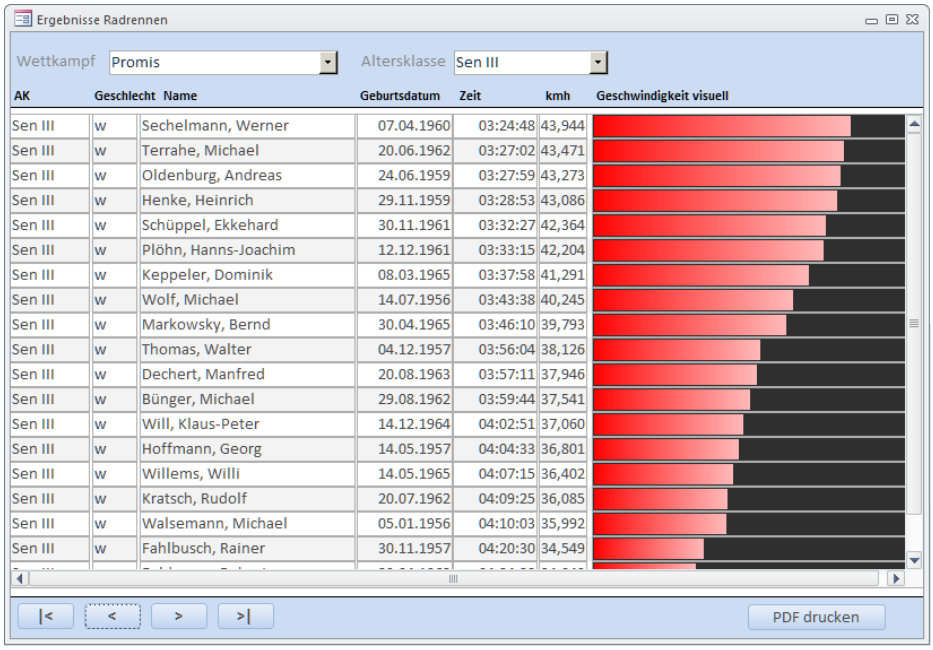
Mit Klick auf die Schaltfläche Berichtsversion... gelangen Sie zu einem anderen Formular, das jedoch einen Unterbericht einbaut hat, der die Ergebnisse im Prinzip analog darstellt, auch wenn sich die Farbgebung hier etwas unterscheidet. Dieses Formular (siehe Bild 3) und seinen Unterbericht sehen wir 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.

# 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.

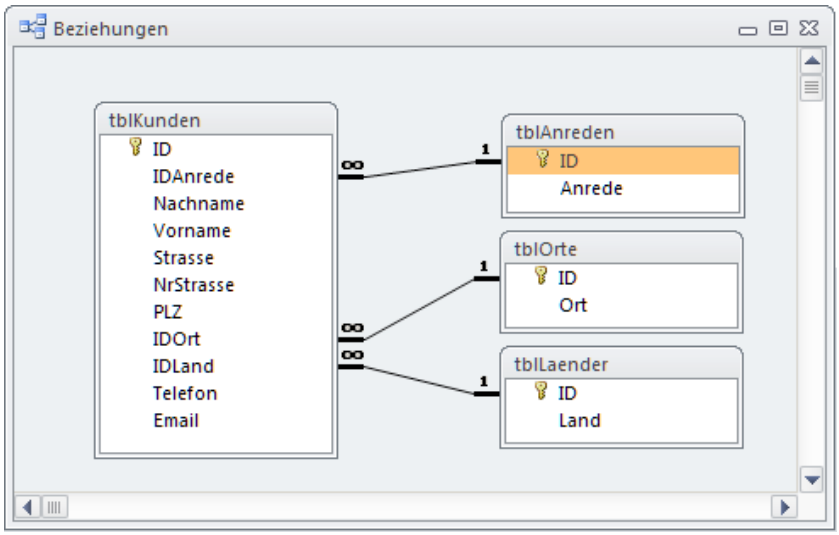



**Bild 1:** Das Datenmodell der Beispieldatenbank zu Kundenadressen

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.

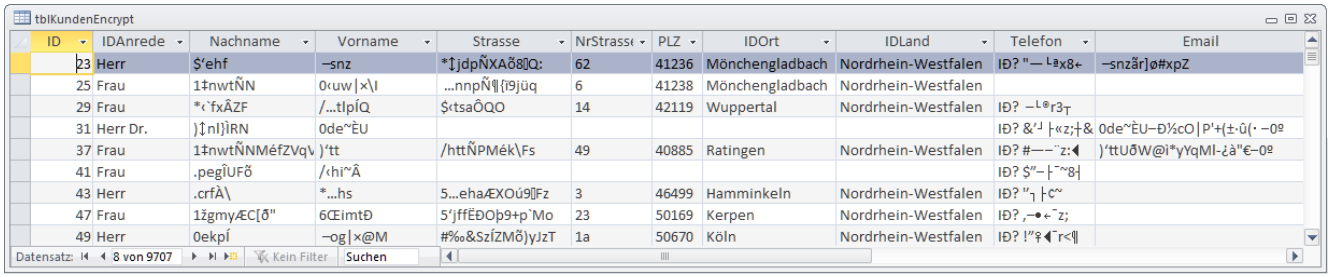
Der Ausgangszustand zeigt sich in Bild 2. Hier sind alle Adressdaten in konventioneller Weise abgespeichert. Jeder kann sie einsehen und gegebenenfalls stehlen. Dazu benötigt man noch nicht einmal eine **Office**-Installation, denn die Datenzugriffsbibliothe-

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



tblKundenEncrypt

| ID | IDAnrede | Nachname | Vorname | Strasse | NrStrasse | PLZ | IDOrt | IDLand | Telefon | Email |
|---|---|---|---|---|---|---|---|---|---|---|
| 23 | Herr | $‘ehf | –snz | *↕jdpÑXAõ8⌧Q: | 62 | 41236 | Mönchengladbach | Nordrhein-Westfalen | IÐ? "— ᴸªx8← | –snzãr]ø#xpZ |
| 25 | Frau | 1‡nwtÑN | 0‹uw\|x\I | ...nnpÑ¶{ï9jüq | 6 | 41238 | Mönchengladbach | Nordrhein-Westfalen | | |
| 29 | Frau | *‹`fxÂZF | /...tIpÍQ | $‹tsaÔQO | 14 | 42119 | Wuppertal | Nordrhein-Westfalen | IÐ? –ᴸ®r3┬ | |
| 31 | Herr Dr. | )↕nl}ìRN | 0de~ÈU | | | | | | IÐ? &’ᴶ ├«z;┼& | 0de~ÈU–Ð½cO\|P'+(±·û(· –0º |
| 37 | Frau | 1‡nwtÑNMéfZVqV | )‘tt | /httÑPMék\Fs | 49 | 40885 | Ratingen | Nordrhein-Westfalen | IÐ? #—–¨z:◀ | )‘ttUðW@ì*yYqMl-¿à"€–0º |
| 41 | Frau | .pegÎUFõ | /‹hi~Â | | | | | | IÐ? $”–├˜~8┤ | |
| 43 | Herr | .crfÀ\ | *...hs | 5...ehaÆXOú9⌧Fz | 3 | 46499 | Hamminkeln | Nordrhein-Westfalen | IÐ? ”┐├¢~ | |
| 47 | Frau | 1žgmyÆC[ð" | 6ŒimtÐ | 5‘jffËÐOþ9+p`Mo | 23 | 50169 | Kerpen | Nordrhein-Westfalen | IÐ? ,–•←˜z; | |
| 49 | Herr | 0ekpÍ | –og\|×@M | #‰&SzÍZMõ)yJzT | 1a | 50670 | Köln | Nordrhein-Westfalen | IÐ? !”⍰◀˜r<¶ | |

Datensatz: 8 von 9707 Kein Filter Suchen

**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"
->  ¾œœšŒŒß½žŒ-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

# 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>
```

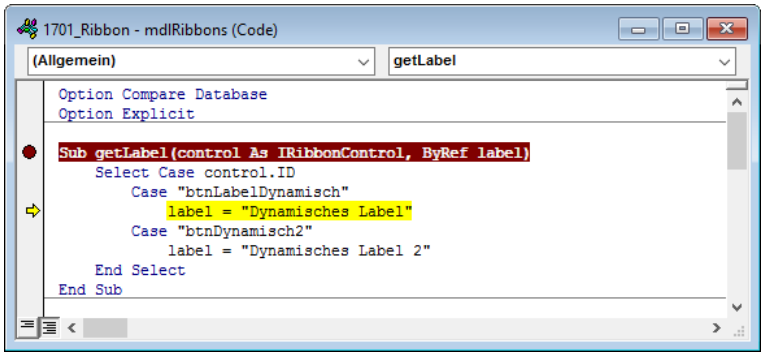



**Bild 2:** Ansteuern eines Haltepunktes in einer Callback-Prozedur

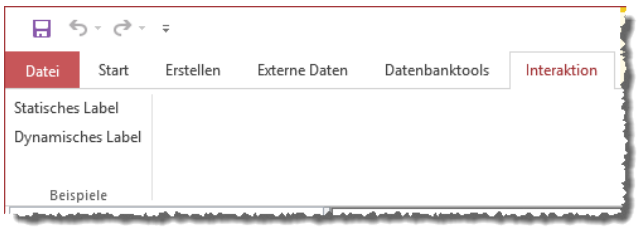



**Bild 1:** Zwei Label mit verschiedenen Beschriftungen

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**.

# 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.

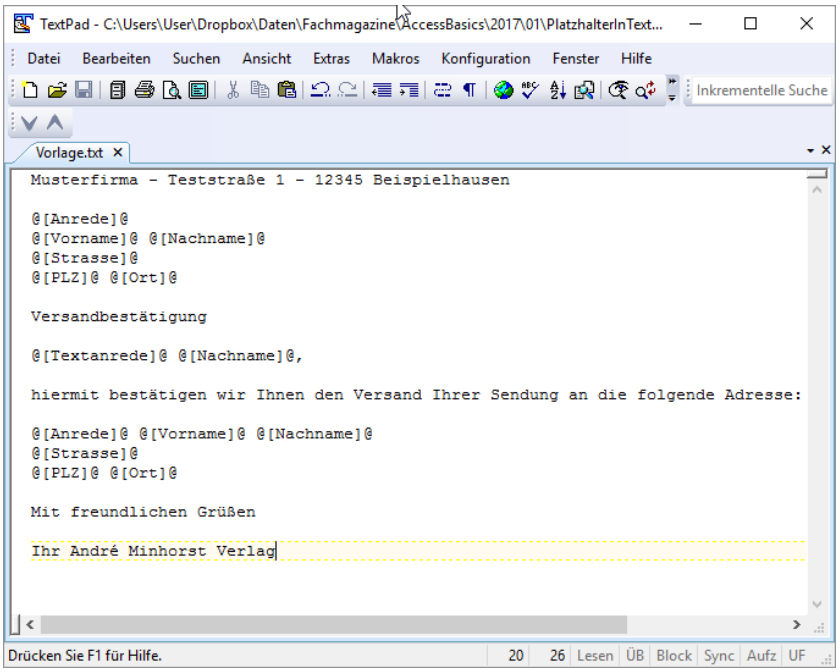



**Bild 1:** Beispielvorlage als Textdatei

**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.

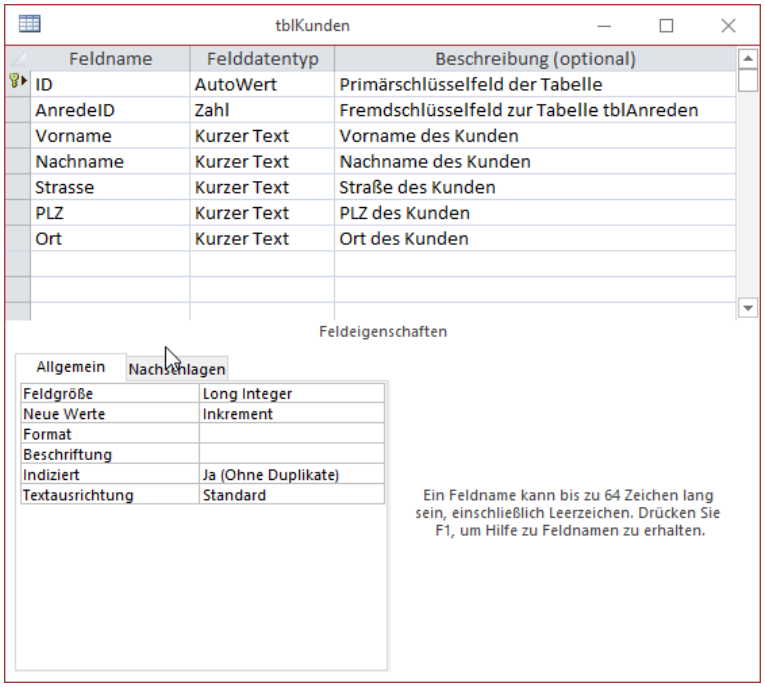



**Bild 2:** Entwurf der Tabelle **tblKunden**

## 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.

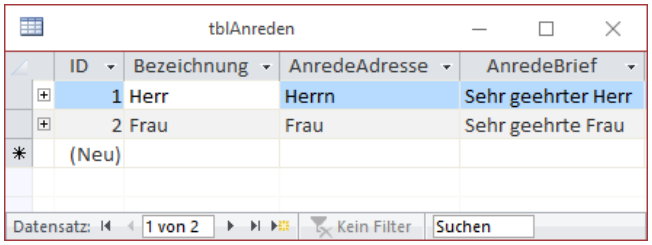



**Bild 3:** Die Tabelle **tblAnreden** mit einigen Werten

Das Ergebnis der Abfrage aus Bild 5 liefert alle Daten der Tabelle **tblKunden** plus die dazugehörigen Anreden aus der Tabelle **tblAnreden**.

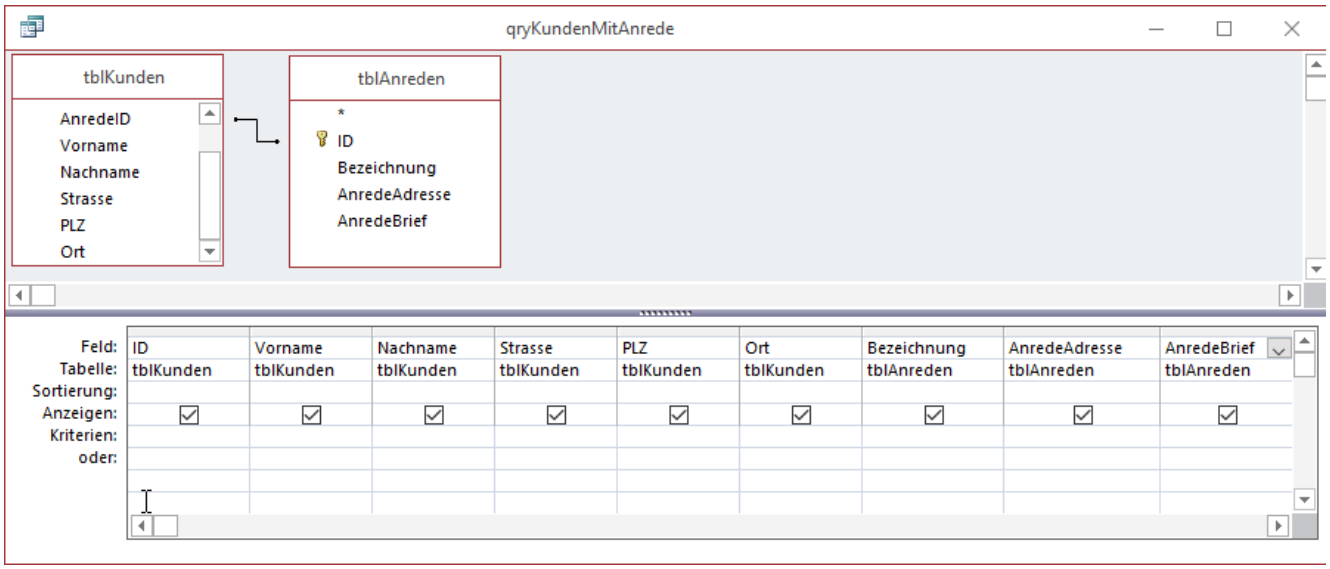
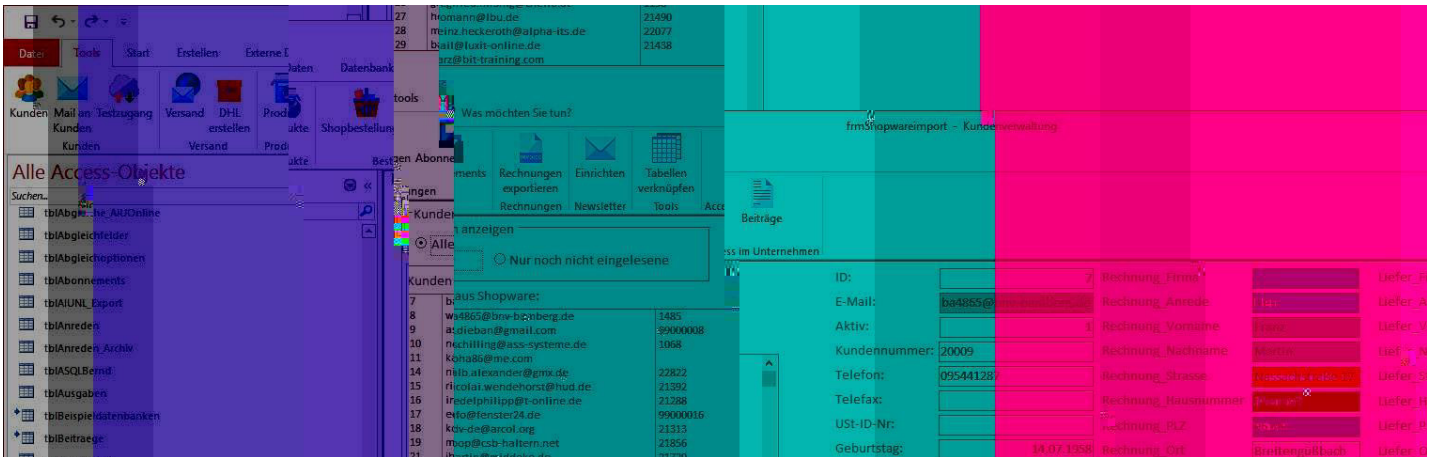



**Bild 4:** Entwurf der Abfrage **qryKundenMitAnrede**

Ausgabe 02/2017

# ACCESS BASICS

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




## In diesem Heft:

**DER WEBBROWSER ALS BILDANZEIGE (S. 3)**

**INDIZIERUNG IN DER PRAXIS (S. 15)**

**SORTIERALGORITHMEN (S. 24)**

**LOKALER WEBSHOP, TEIL III (S. 34)**

# 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

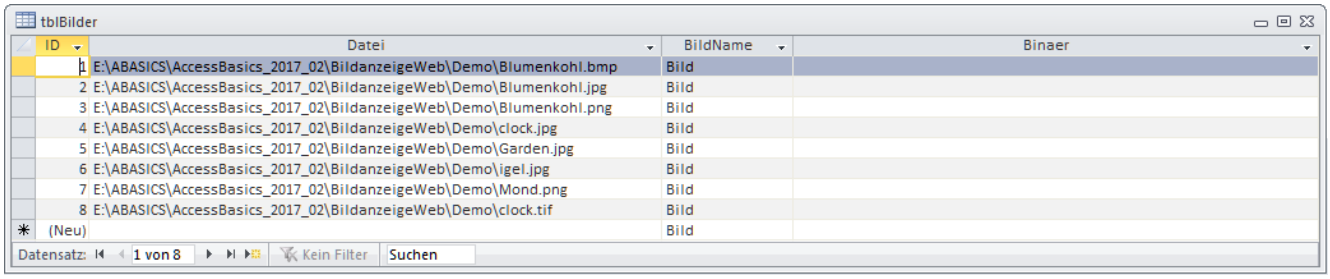



**Bild 1:** Die Tabelle **tblBilder** enthält zunächst nur die Pfade zu den Bilddateien im Feld **Datei** und einen **Default**-Namen

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.

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...

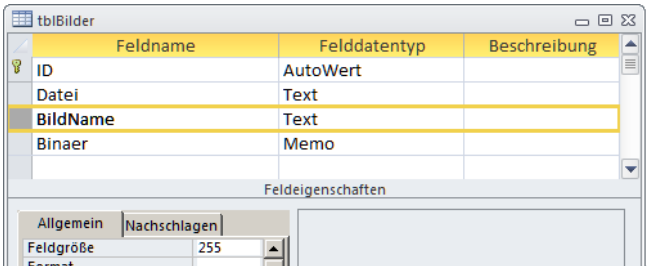



**Bild 2:** Entwurfsansicht der Tabelle **tblBilder**

## 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

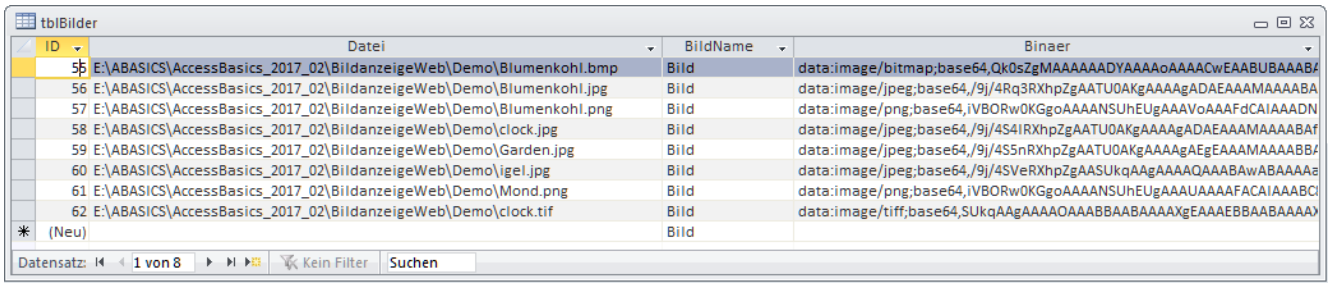



**Bild 3:** Das Memo-Feld **Binaer** füllt sich später beim Aufruf eines Bilds im Webbrowser des Formulars **frmBilder**

# 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

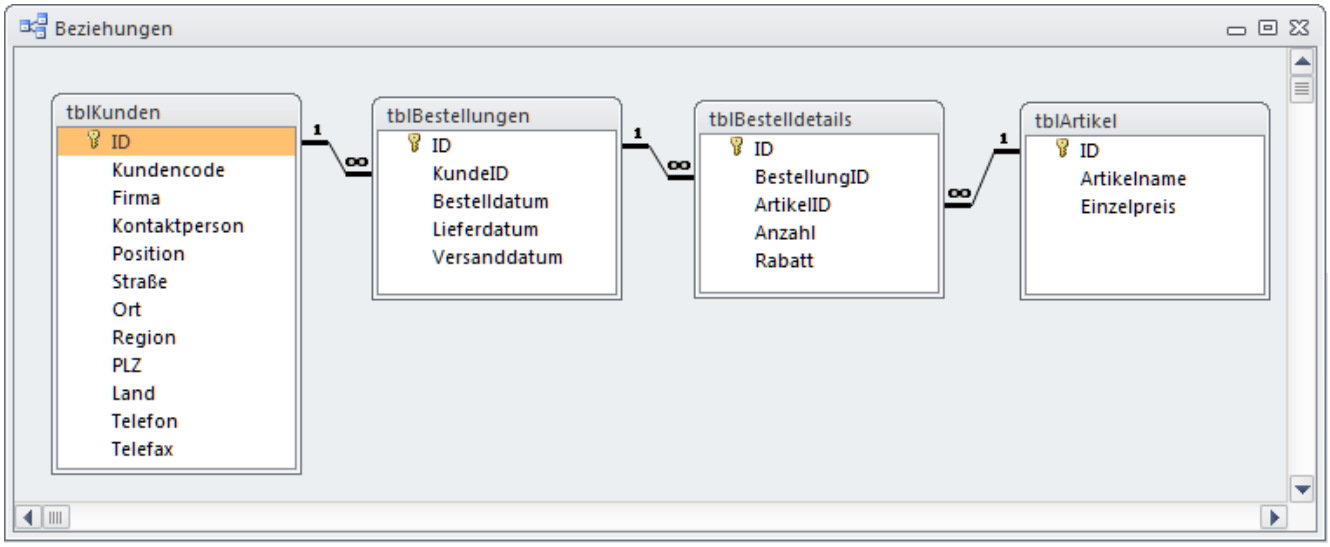



**Bild 1:** Die vier Tabellen der Kundendatenbank sind alle miteinander über Schlüsselfelder verknüpft

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.

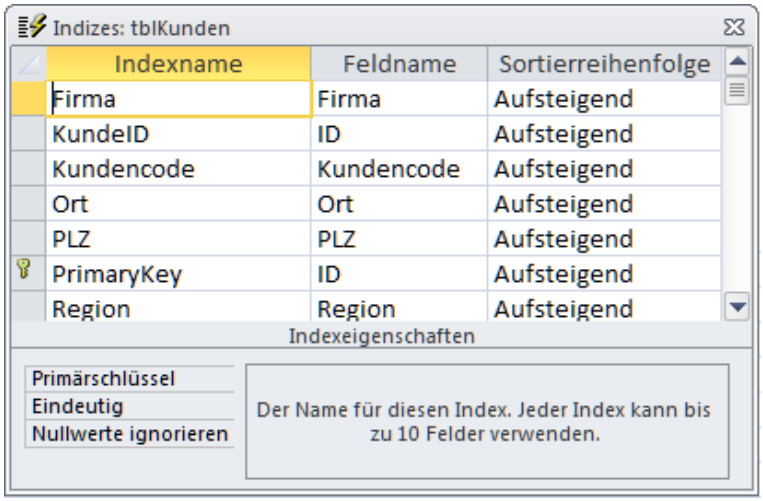



**Bild 2:** Der Indizes-Dialog zur Tabelle tblKunden im Entwurf

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
```

# 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:

# 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"
```

```
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 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.

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,

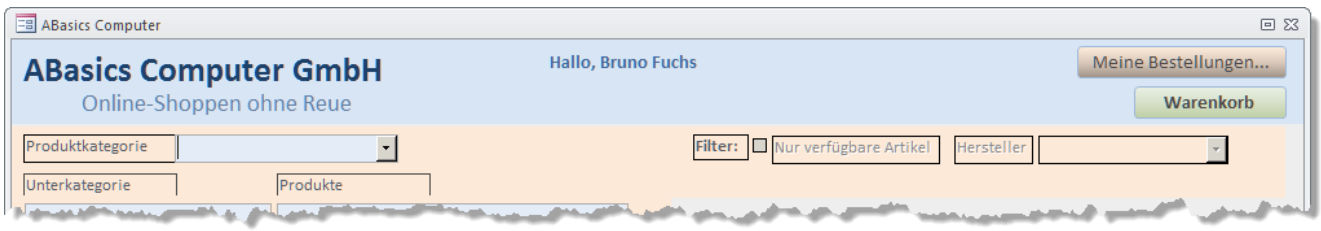



**Bild 1:** Im Kopf des Shop-Forumlar **frmShop** blendet sich eine Begrüßungformel mit dem Namen des Kunden ein

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.

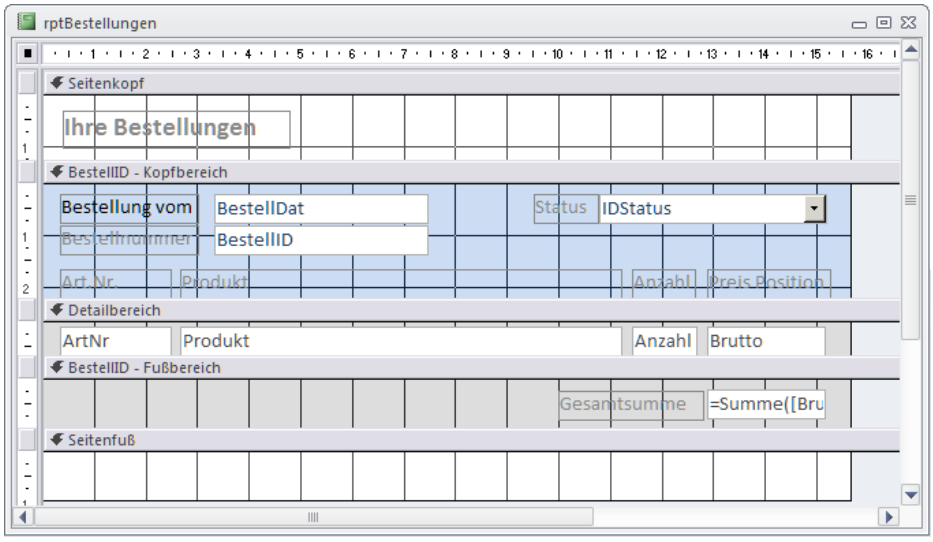



**Bild 2:** Die Entwurfsansicht des Berichts zur Einsicht in die getätigte Bestellungen

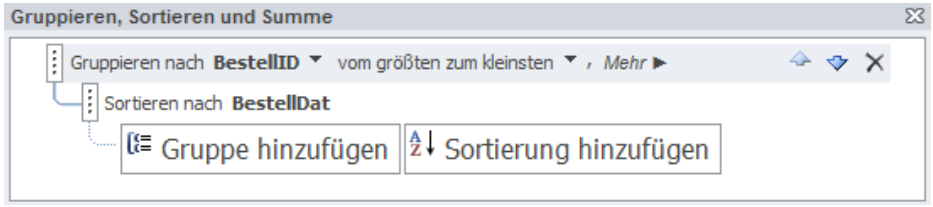



**Bild 3:** Der im Entwurf des Berichts unten eingeblendete Gruppen-Dialog

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.

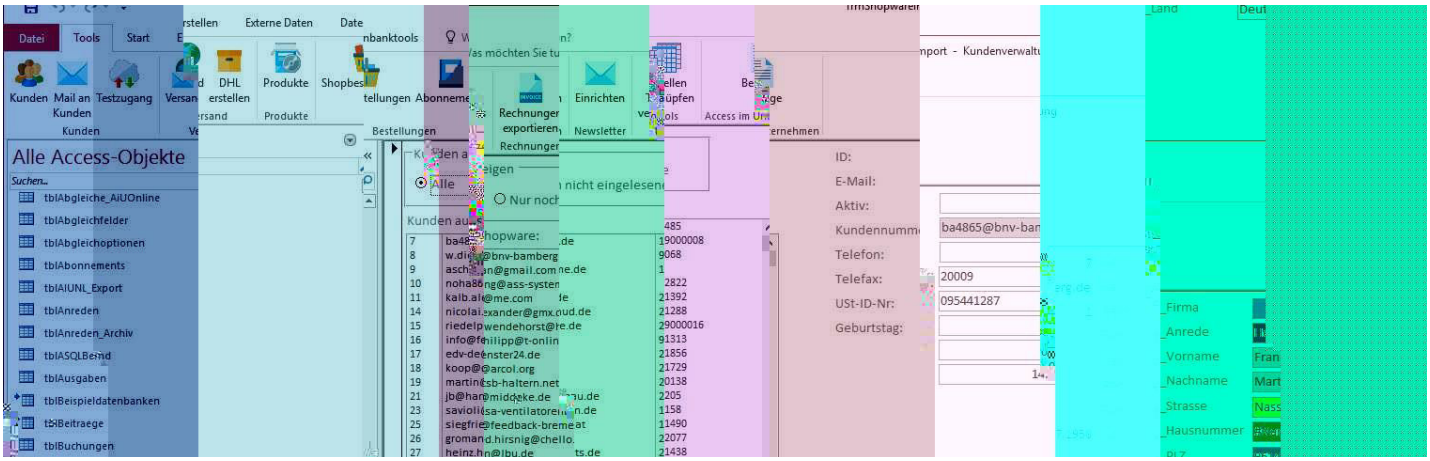
Außerdem sortieren dessen Einstellungen noch die Bestellungen nach dem Bestelldatum (**BestellDat**). Im Fußbereich wird zu jeder Bestellung noch die **Gesamtsumme** über den hinterlegten Ausdruck **Summe([Brutto])** ausgegeben.

Im Ergebnis finden wir also alle Bestellungen des eingeloggten Kunden in der Reihenfolge der Bestellungen nach Datum absteigend angezeigt.

Ausgabe 03/2017

# ACCESS BASICS

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




## In diesem Heft:

MIT BINÄRZAHLEN ARBEITEN (S. 3)

TRANSAKTIONEN IN ACCESS (S. 7)

SELEKTION IM DATENBLATT (S. 13)

INDIVIDUELLE AUSWAHL MIT EIGENER TABELLE (S. 23)

ACCESS-ADD-INS PROGRAMMIEREN (S. 28)

ACCESS-OBJEKT-ASSISTENTEN PROGRAMMIEREN, TEIL I (S. 36)

# 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).

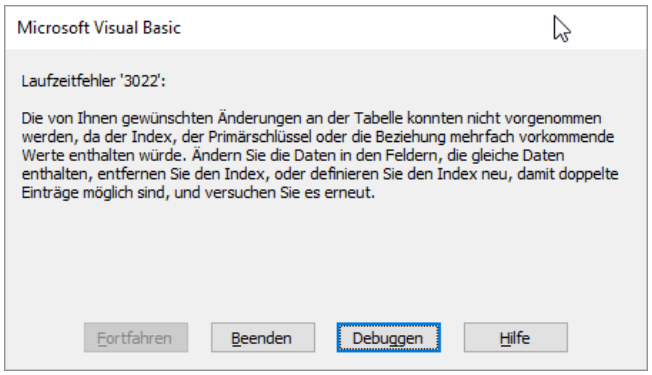



**Bild 1:** Fehlermeldung beim Anlegen eines bereits vorhandenen Kategorienamens

### 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?

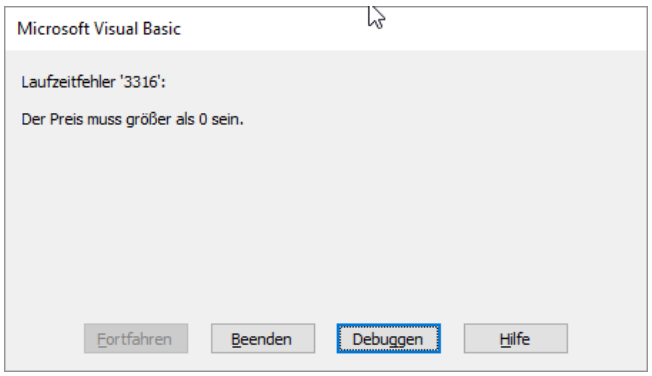



**Bild 2:** Fehlermeldung bei Verletzung der Gültigkeitsregel bei mindestem einem der betroffenen Datensätze

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

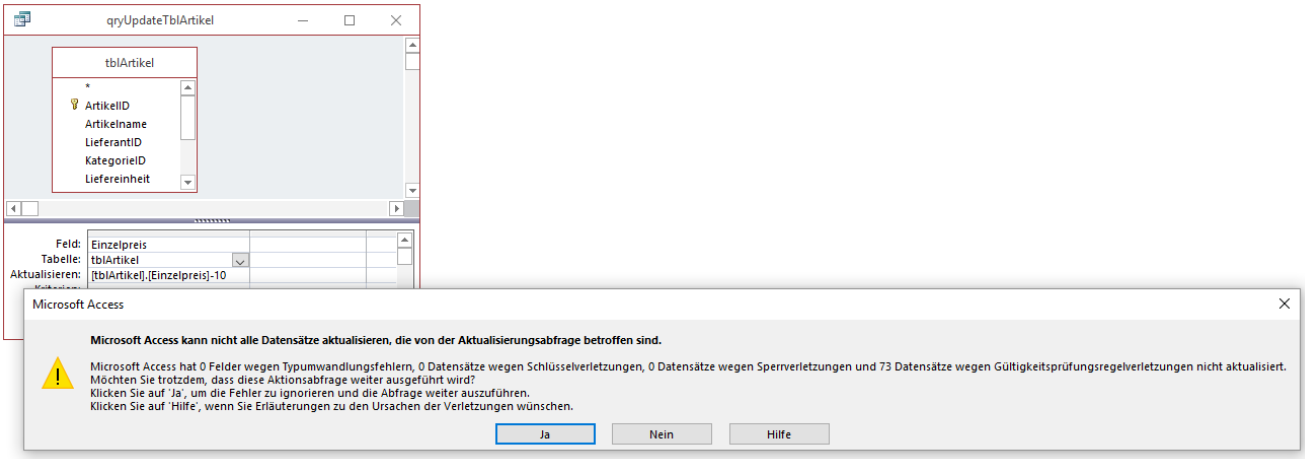



**Bild 3:** Fehlermeldung, wenn nicht alle Daten geändert werden können

# 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.

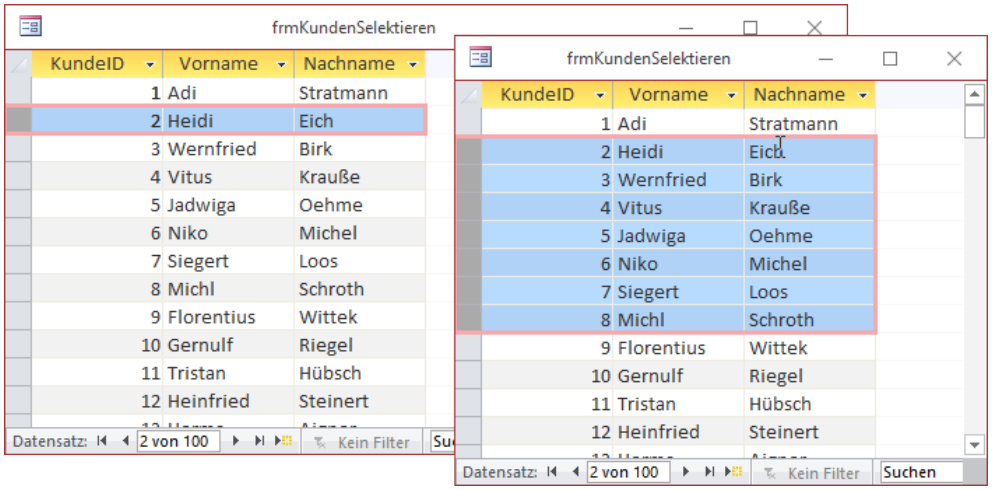



**Bild 1:** Beispiele für die Selektion von Datensätzen im Datenblatt

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.

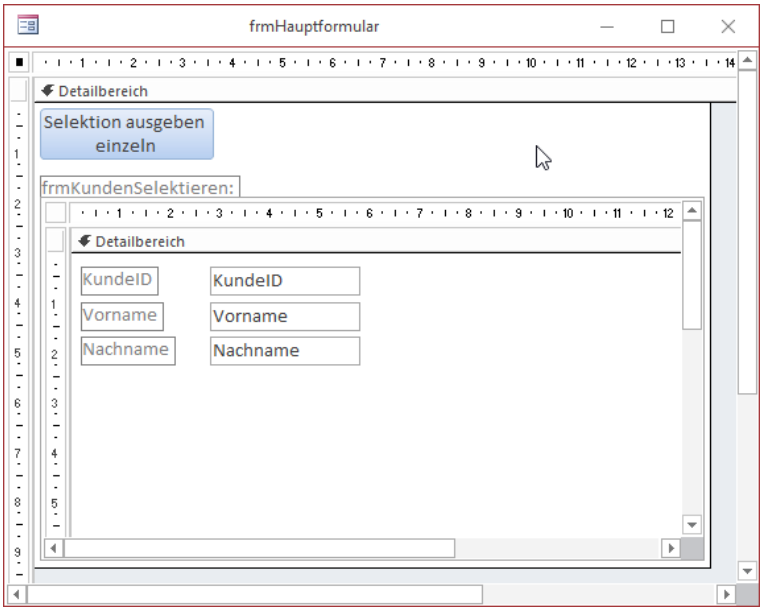



**Bild 2:** Formular zum Auswerten der aktuellen Selektion

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 _
```

# 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.

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.

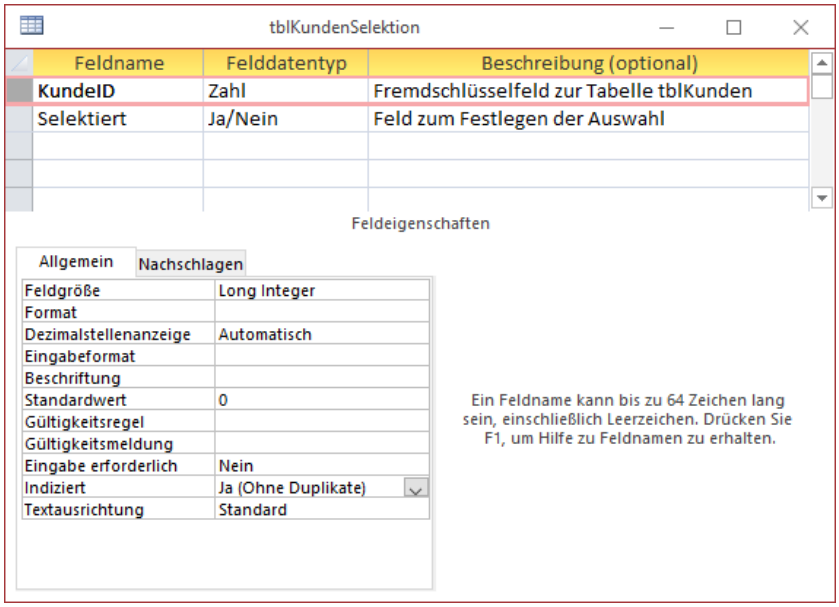



**Bild 1:** Die Tabelle mit dem Selektionsfeld

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.

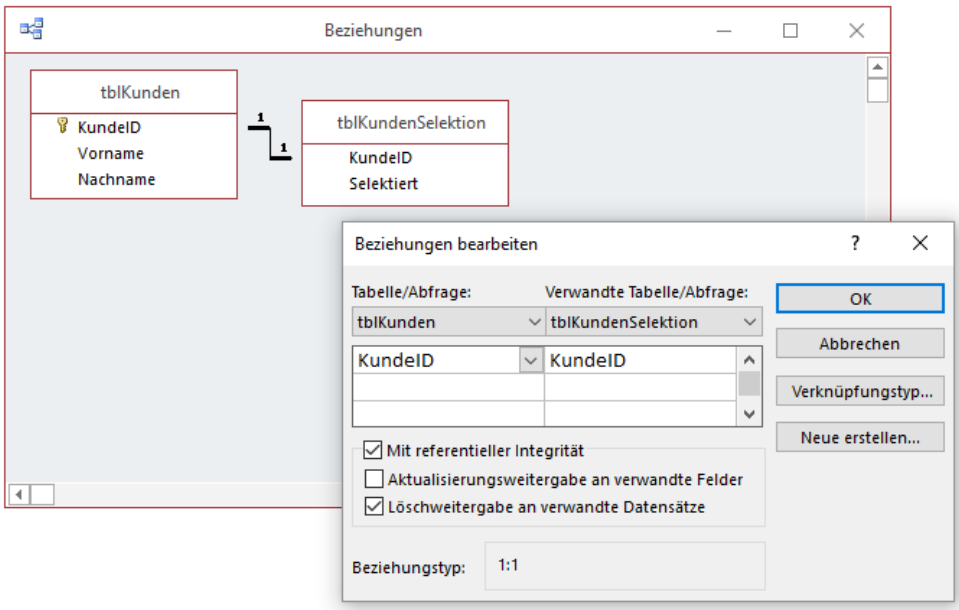



**Bild 2:** Herstellen der Beziehung zwischen den beiden Tabellen

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.

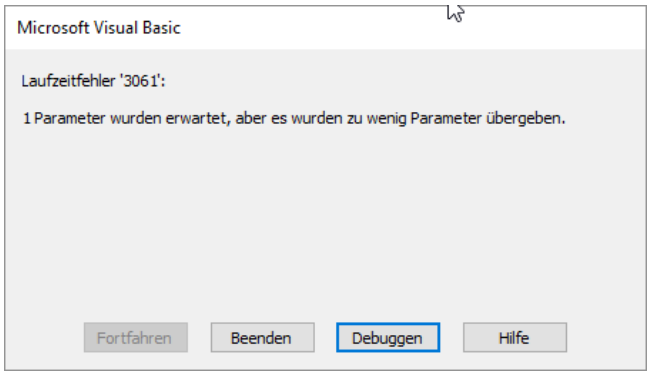



**Bild 3:** Fehler beim Versuch, auf das Feld **Selektiert** zuzugreifen

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.accdb.

**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.

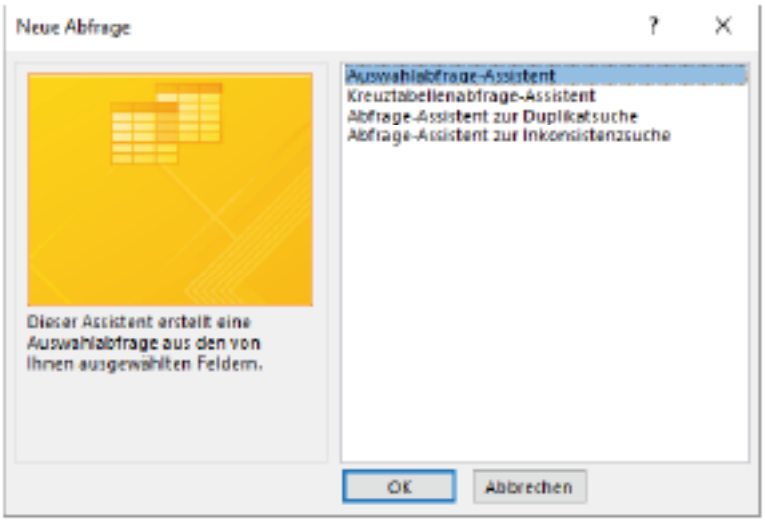



Bild 1: Übersicht der Abfrageassistenten

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.

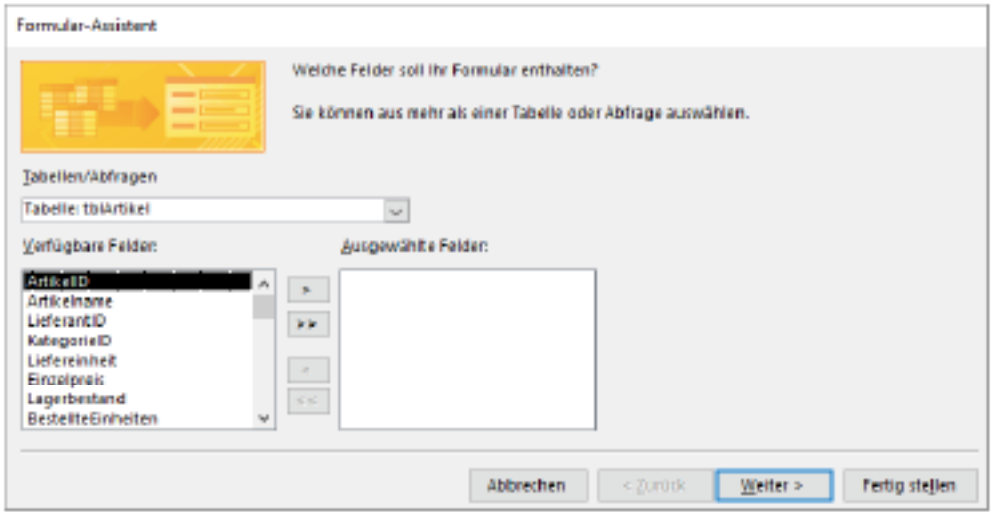



Bild 2: Der Formular-Assistent von Access

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

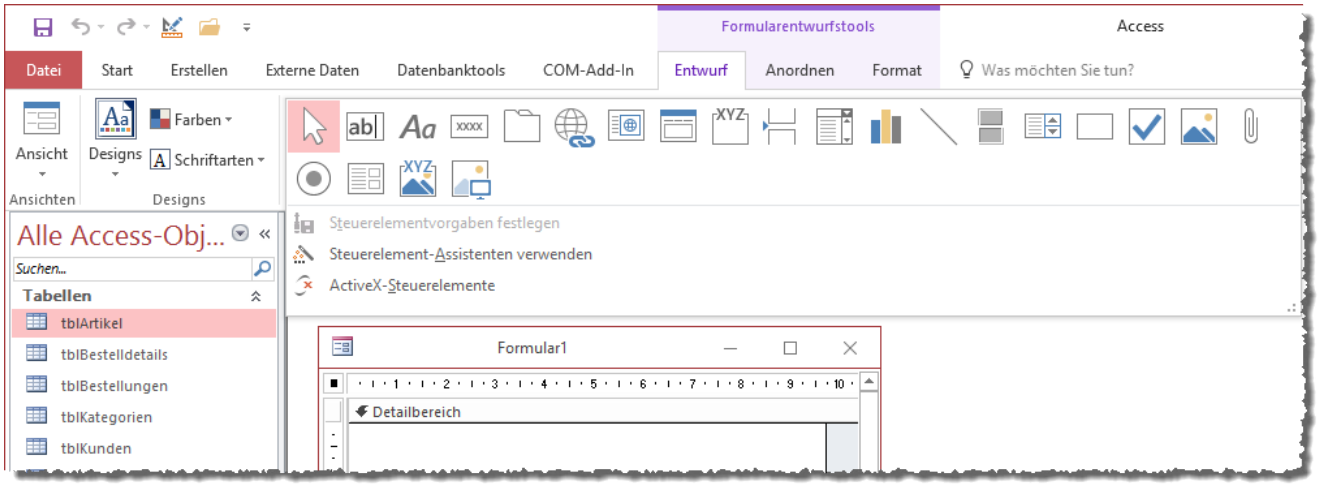



**Bild 3:** Aktivieren des Steuerelement-Assistenten

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

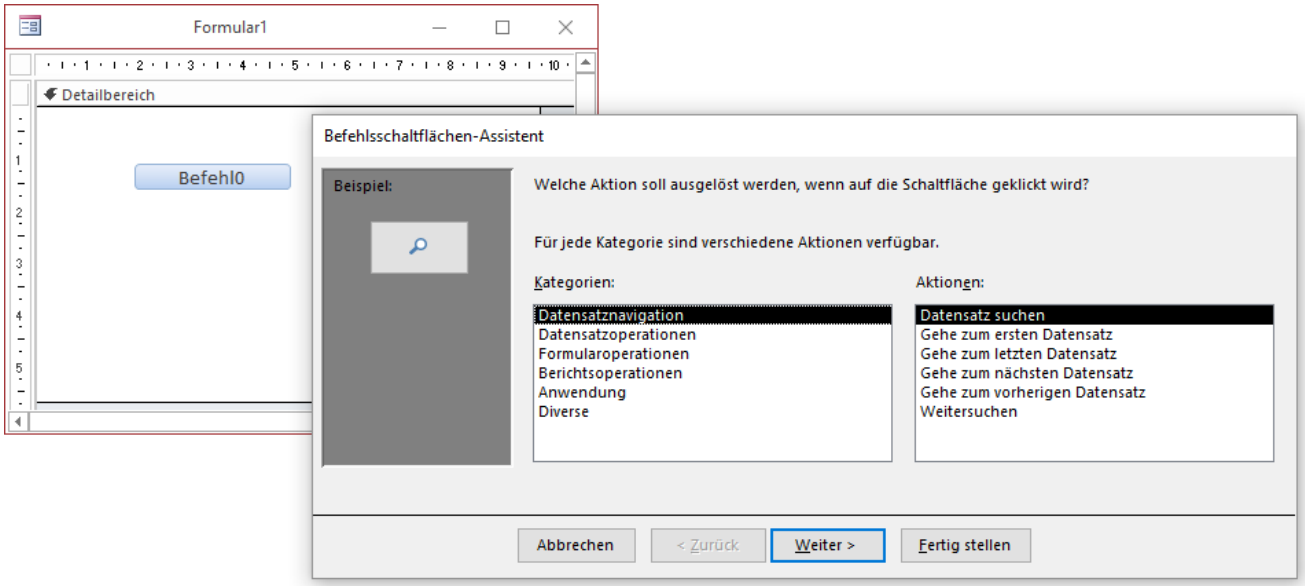



**Bild 4:** Unterstützung durch den Befehlsschaltflächen-Assistenten

# 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.

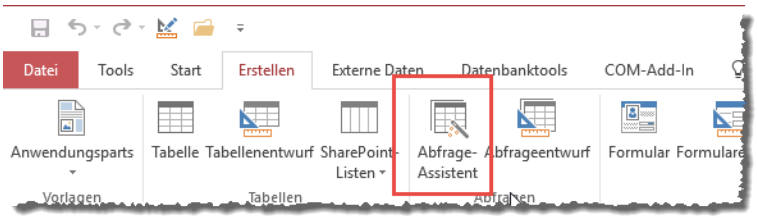



**Bild 1:** Aufruf eines Abfrage-Assistenten

## 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.

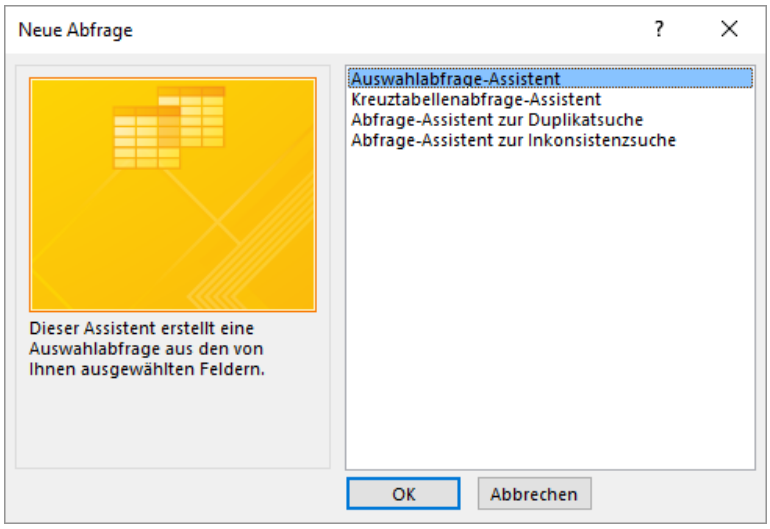



**Bild 2:** Der Dialog mit den verschiedenen Abfrage-Assistenten

## 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

Ausgabe 04/2017

# ACCESS BASICS

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



## In diesem Heft:

**TABELLEN ERSTELLEN MIT SQL UND DDL (S. 3)**

**TEXT- UND BEZEICHNUNGSFELDER (S. 19)**

**ALLE EREIGNISSE IM HAUPTFORMULAR (S. 14)**

**HANDYVERWALTUNG I: DATENMODELL (S. 24)**

**HANDYVERWALTUNG II: PERSONEN VERWALTEN (S. 33)**

# 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

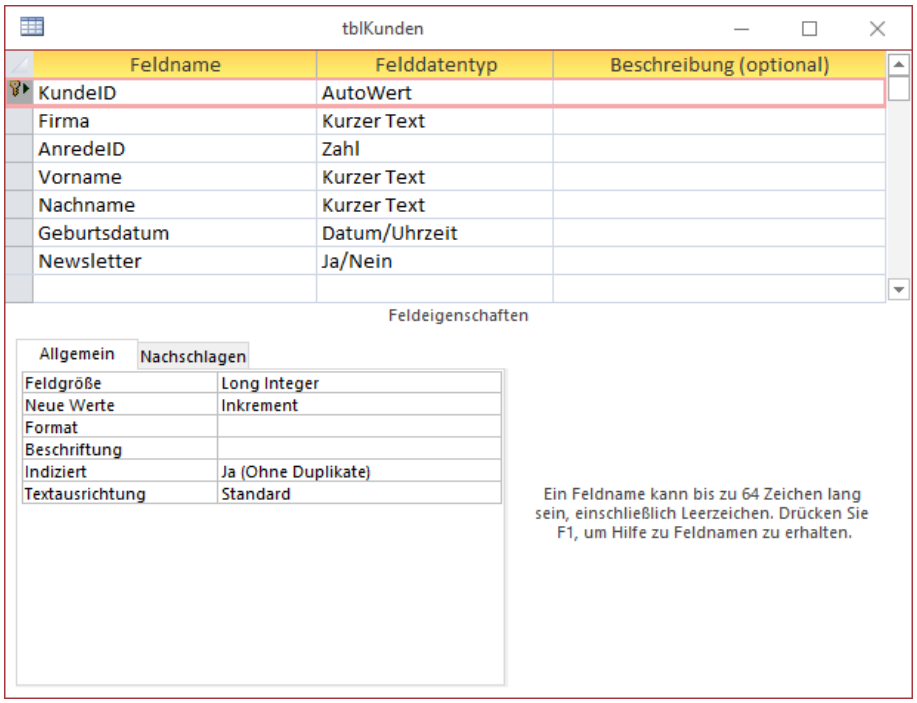



**Bild 1:** Diese Tabelle wollen wir mit VBA und SQL nachbauen.

**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.

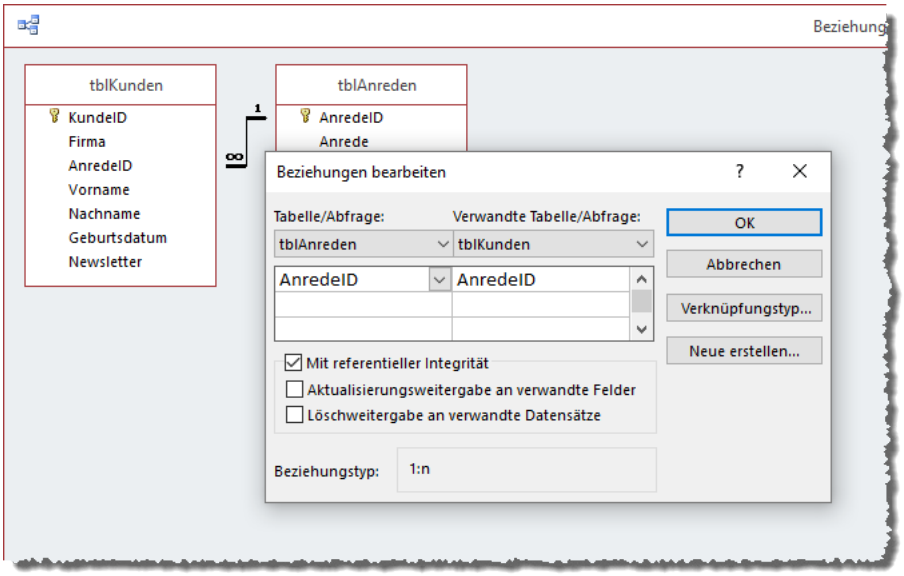



**Bild 2:** Beziehung zwischen den Tabellen **tblKunden** und **tblAnreden**

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-

# 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.

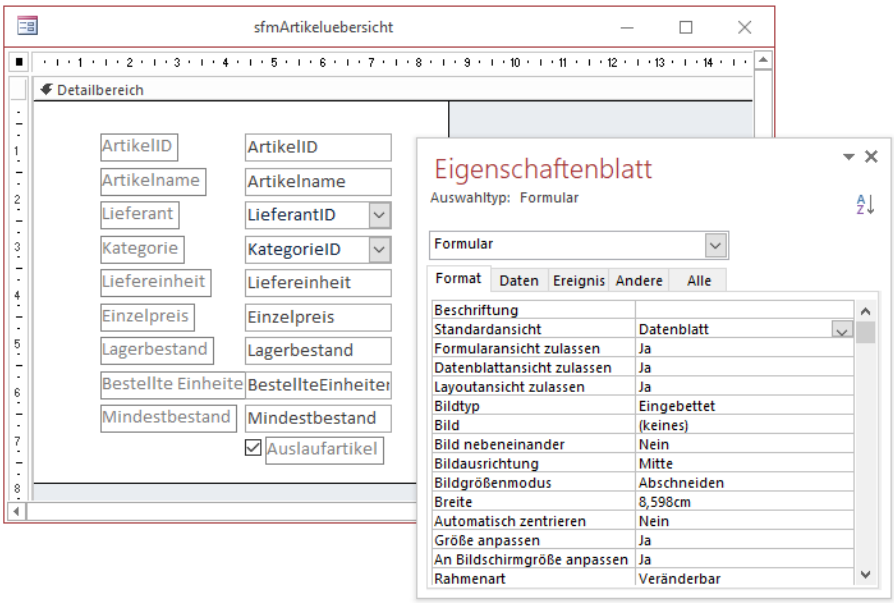



**Bild 1:** Entwurf des Unterformulars mit der Datenblattansicht

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

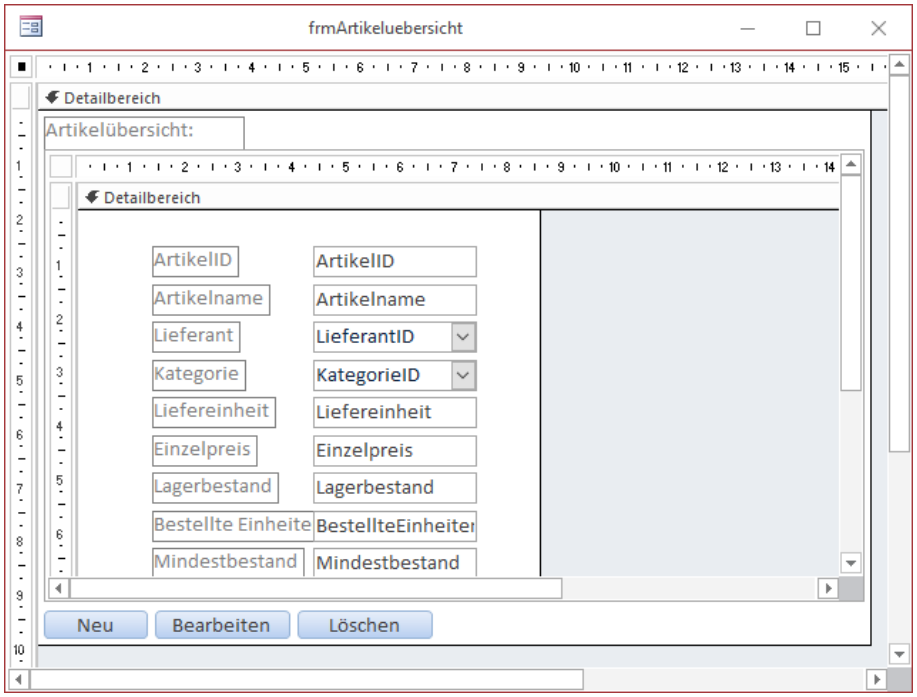



**Bild 2:** Haupt- und Unterformular in der Entwurfsansicht

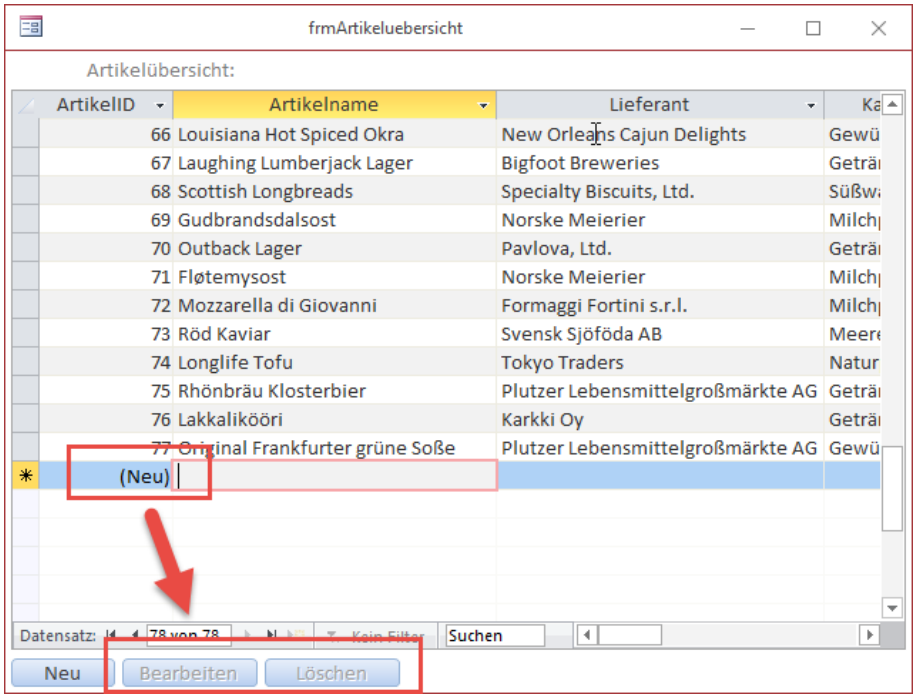



**Bild 3:** Die beiden Schaltflächen **cmdBearbeiten** und **cmdLoeschen** sollen deaktiviert sein, wenn im Unterformular kein Datensatz markiert ist.

# 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

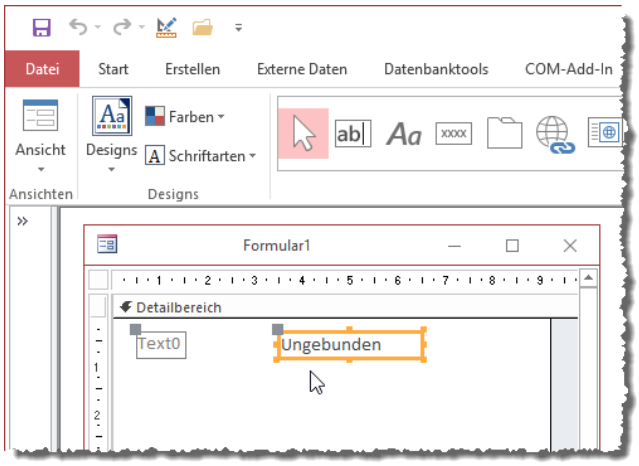



**Bild 1:** Ein frisch angelegtes Textfeld mit Bezeichnungsfeld

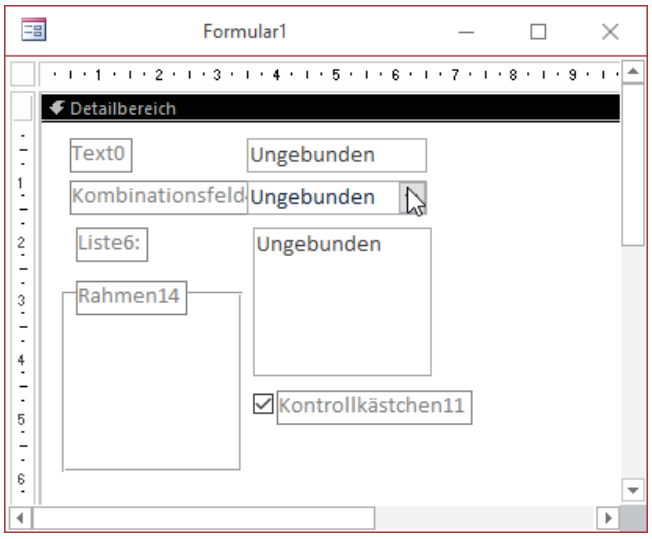



**Bild 2:** Einige Steuerelemente, zu denen automatisch ein Bezeichnungsfeld hinzugefügt wird.

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).

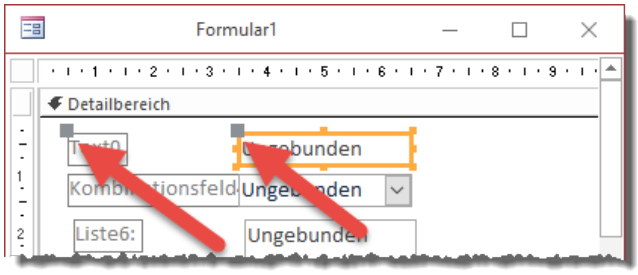



**Bild 3:** An diesen Punkten müssen Sie ziehen, wenn Sie Bezeichnungsfeld oder Steuerelement ohne den Partner umpositionieren wollen.

## 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.

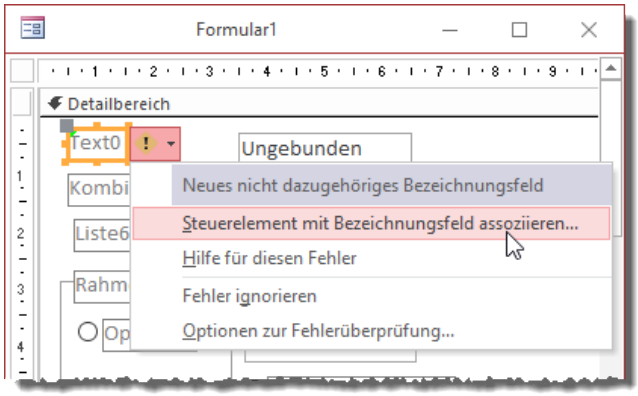



**Bild 4:** Nach dem Trennen eines Bezeichnungsfeldes von einem Steuerelement bietet Access verschiedene Möglichkeiten an.

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.

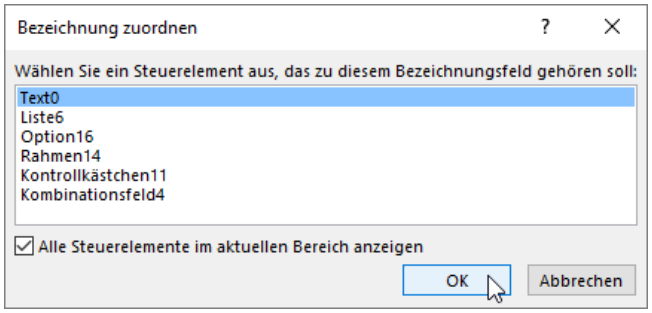



**Bild 5:** Verbinden eines Bezeichnungsfeldes mit einem Steuerelement

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

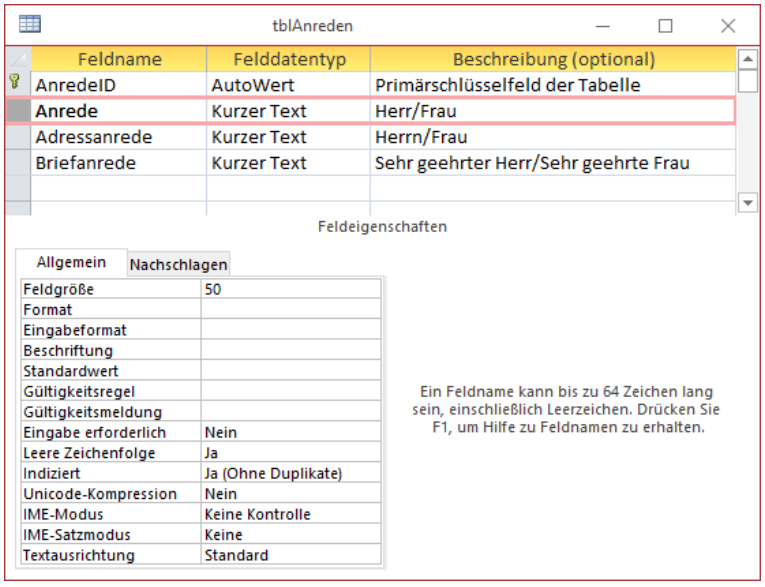



**Bild 1:** Tabelle mit den verschiedenen Anreden

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).

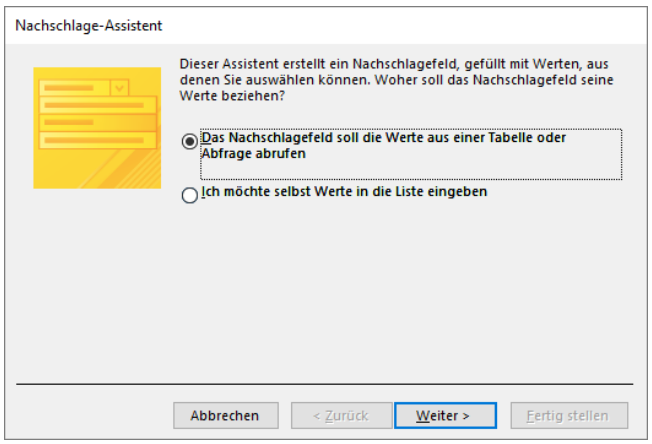



**Bild 2:** Nachschlagefeld einrichten

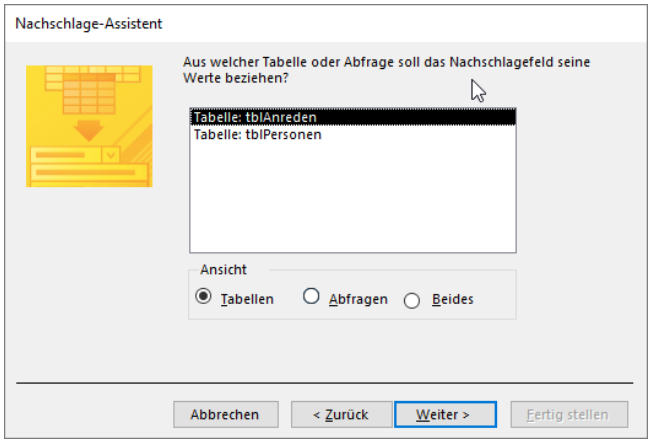



**Bild 3:** Auswahl der zu verknüpfenden Tabelle

- 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**

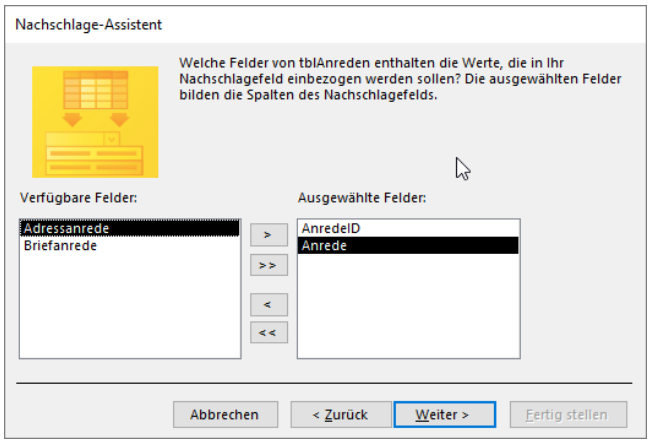



**Bild 4:** Festlegen der gewünschten Felder

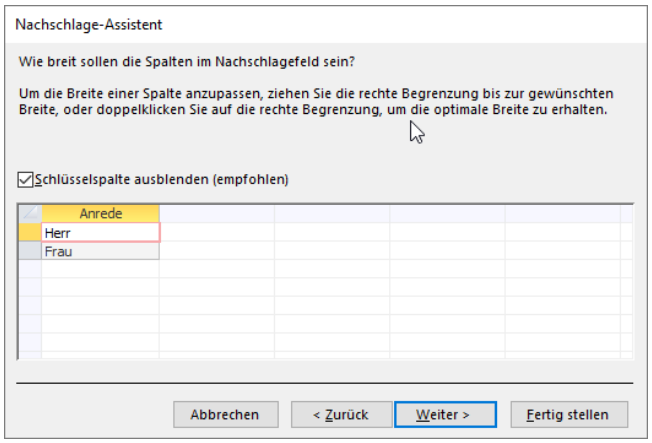



**Bild 5:** Ausblenden der Schlüsselspalte

# 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.

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).

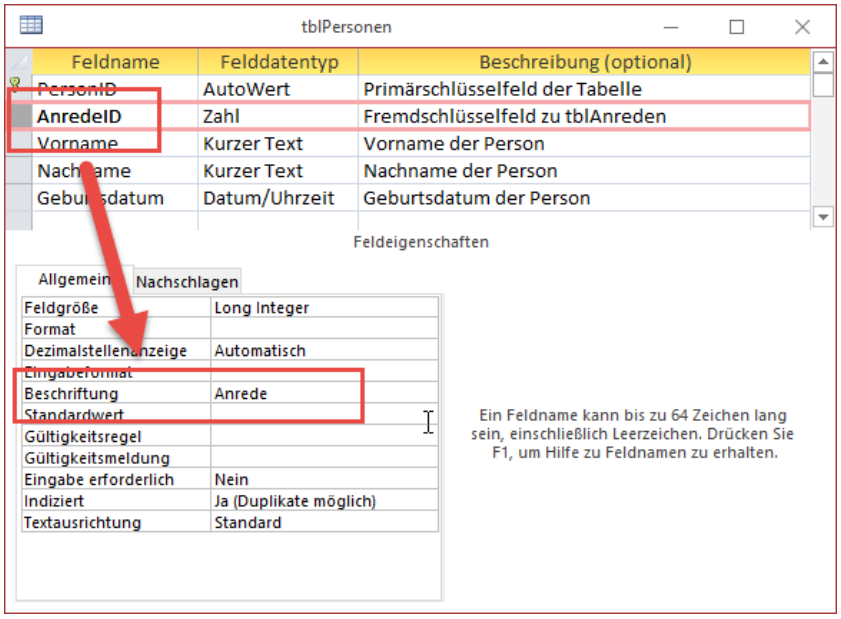



**Bild 1:** Anpassen der Beschriftung für zu erstellende Bezeichnungsfelder

Die neuen Beschriftungen werden dann übrigens gleich in der Datenblattansicht angezeigt (siehe Bild 2) – und später in allen Bezeichnungsfeldern von

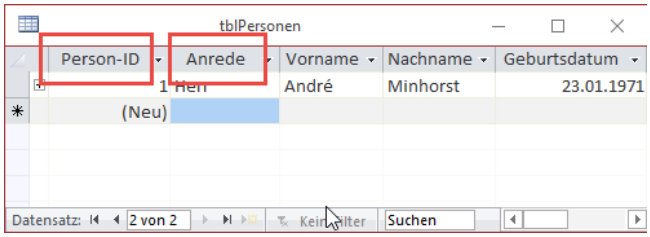



**Bild 2:** Die neuen Beschriftungen werden bereits in der Datenblattansicht angezeigt.

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.

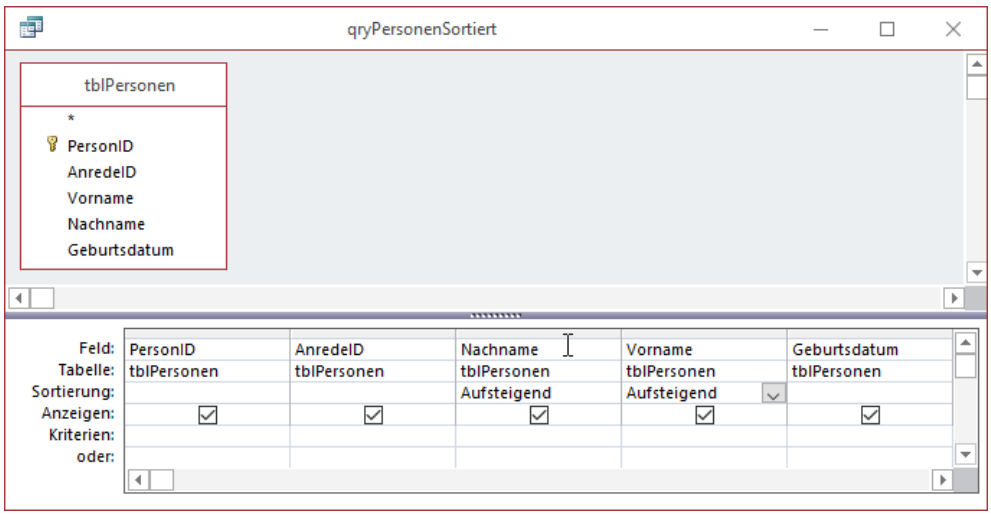



**Bild 3:** Entwurf des Unterformulars mit der Datenblattansicht

## 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

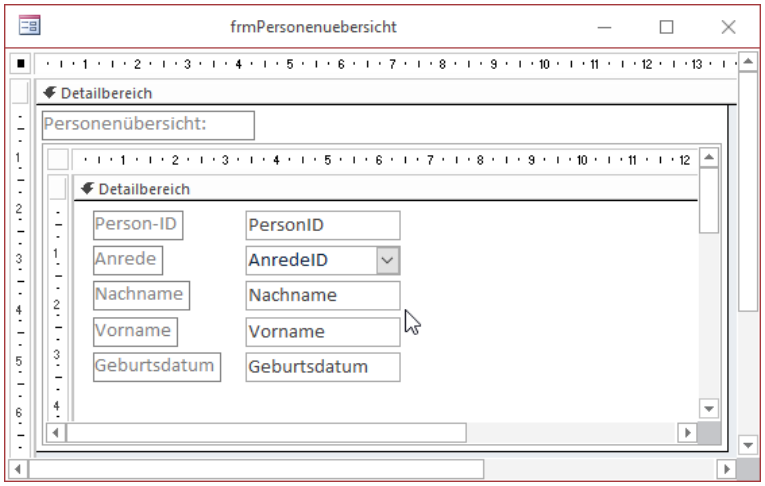
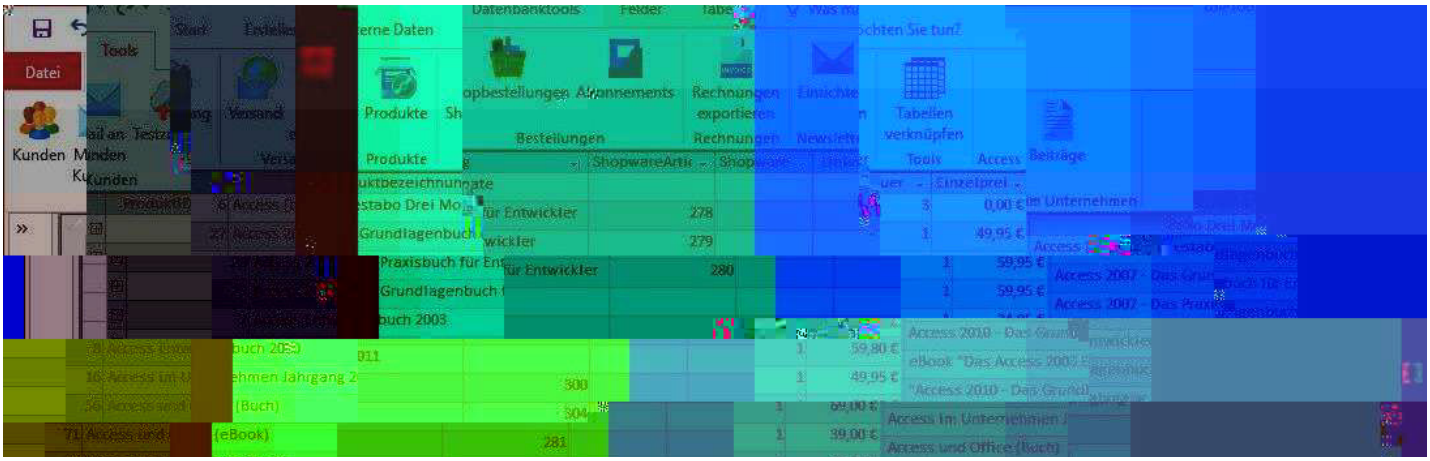



**Bild 4:** Entwurf des Unterformulars mit der Datenblattansicht

Ausgabe 05/2017

# ACCESS BASICS

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



## In diesem Heft:

MELDUNGEN BEI DATENSATZÄNDERUNGEN (S. 3)
HANDYVERWALTUNG III: PROVIDER VERWALTEN (S. 8)
STEUERELEMENTE VERANKERN (S. 17)
HANDYVERWALTUNG IV: VERTRÄGE UND SIM-KARTEN (S. 21)
DETAILFORMULAR ÖFFNEN UND ABFRAGEN (S. 34)
TIPPS UND TRICKS FÜR MEMOFELDER (S. 40)

# 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

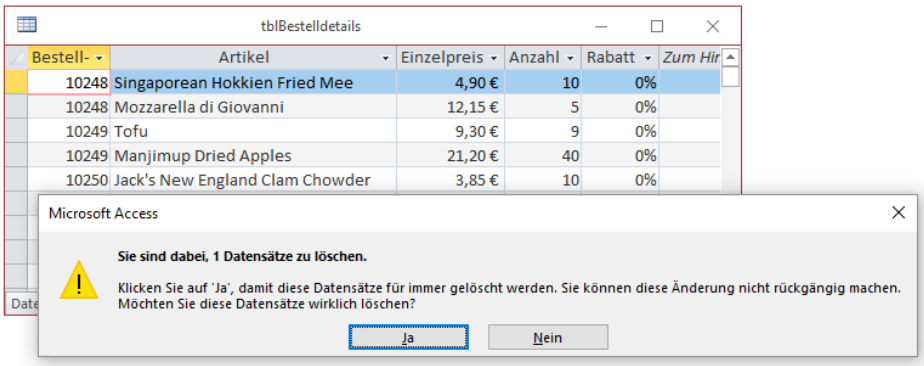



**Bild 1:** Meldung beim manuellen Löschen aus der Datenblattansicht

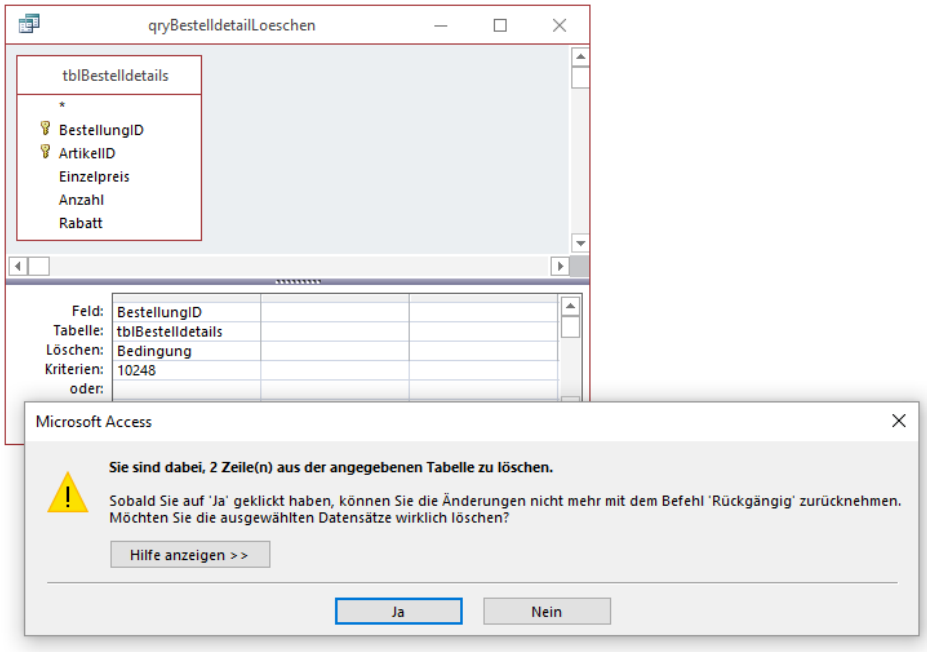



**Bild 2:** Meldung beim Löschen per Aktionsabfrage

**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.

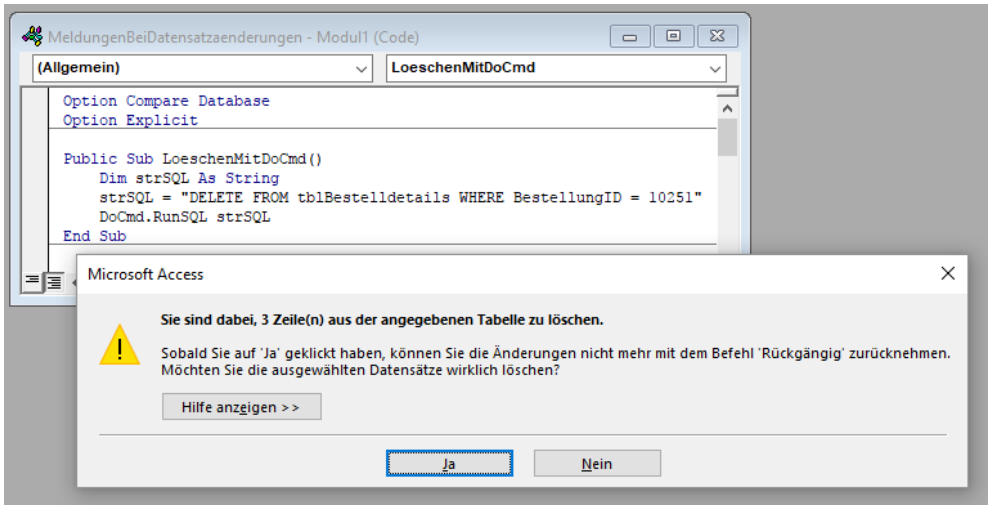



**Bild 3:** Meldung beim Löschen per **DoCmd.RunSQL**

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.

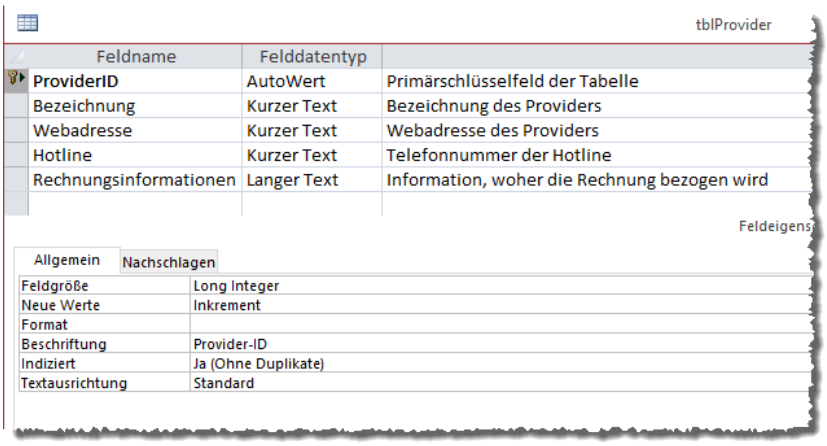



**Bild 1:** Anpassen der Beschriftung des Feldes **Provider-ID** für das Bezeichnungsfelder

## 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).

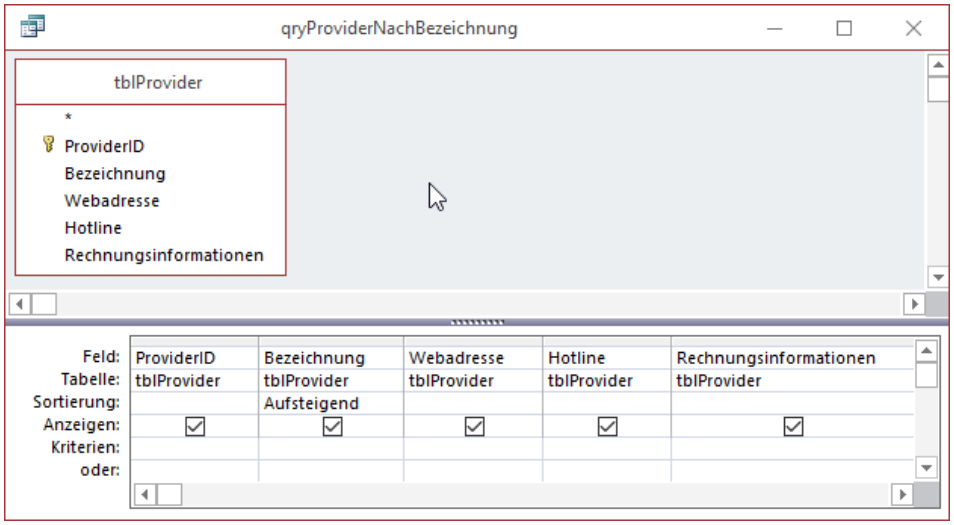



**Bild 2:** Abfrage, welche die Provider nach der Bezeichnung sortier

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).

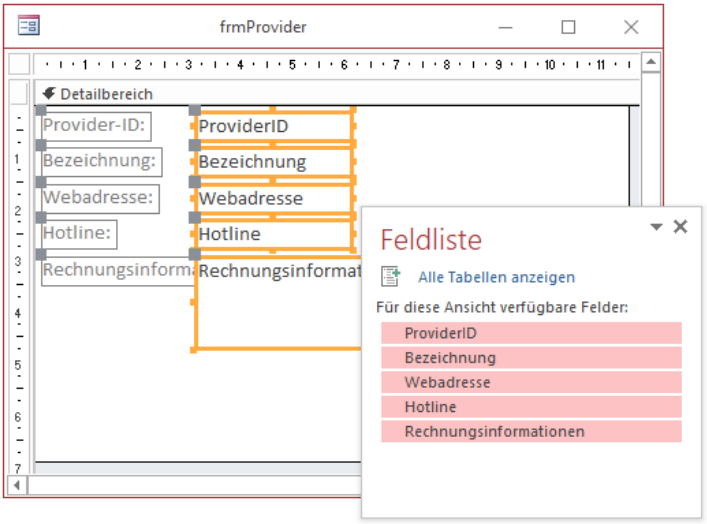



**Bild 3:** Hinzufügen der gebundenen Felder zum Formular **frmProvider**

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

# 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

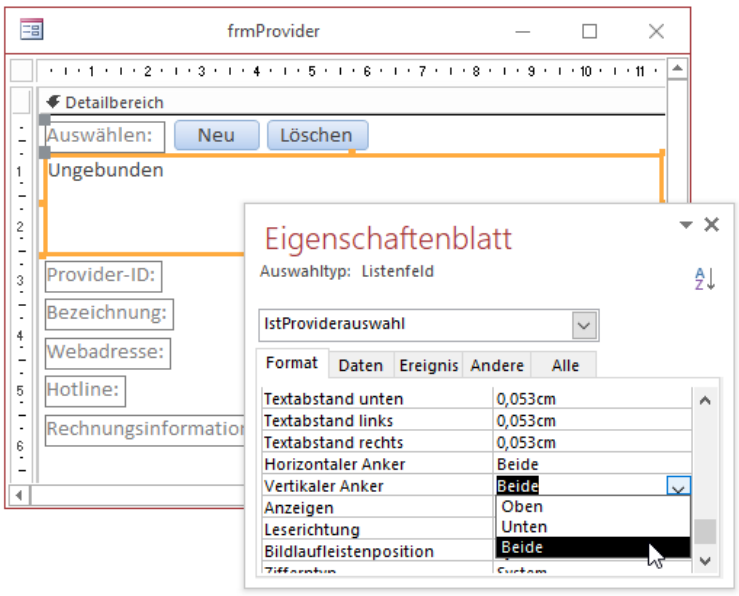



**Bild 1:** Einstellung der beiden Eigenschaften **Horizontaler Anker** und **Vertikaler Anker**

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**

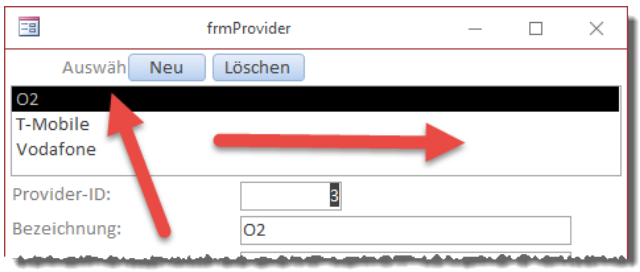



**Bild 2:** Das Bezeichnungsfeld des Listenfeldes wird beim Verbreitern des Formulars nach rechts verschoben.

- **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-

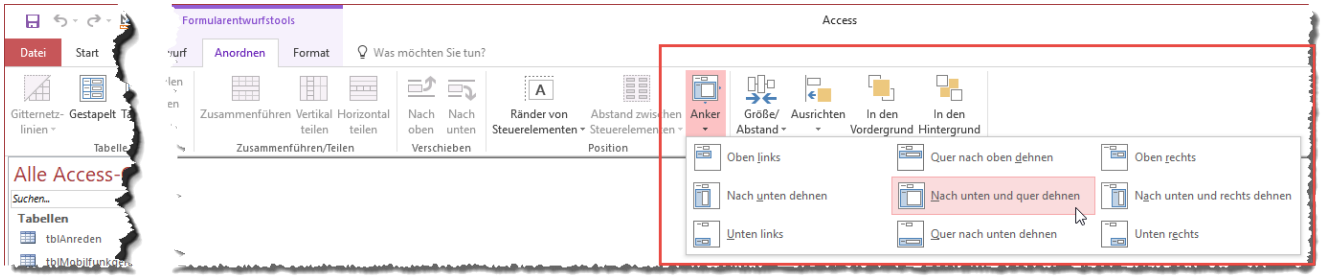



**Bild 3:** Einstellen der Verankerung über das Ribbon

# 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.

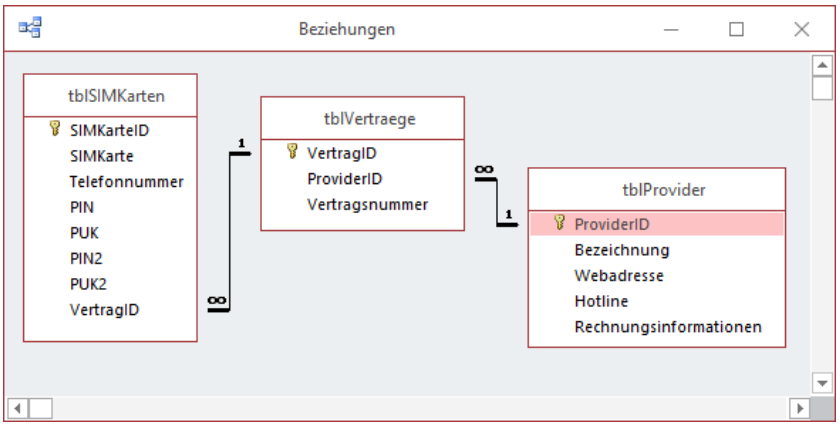



**Bild 1:** Teil des Datenmodells, den dieser Teil der Artikelreihe abdeckt

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).

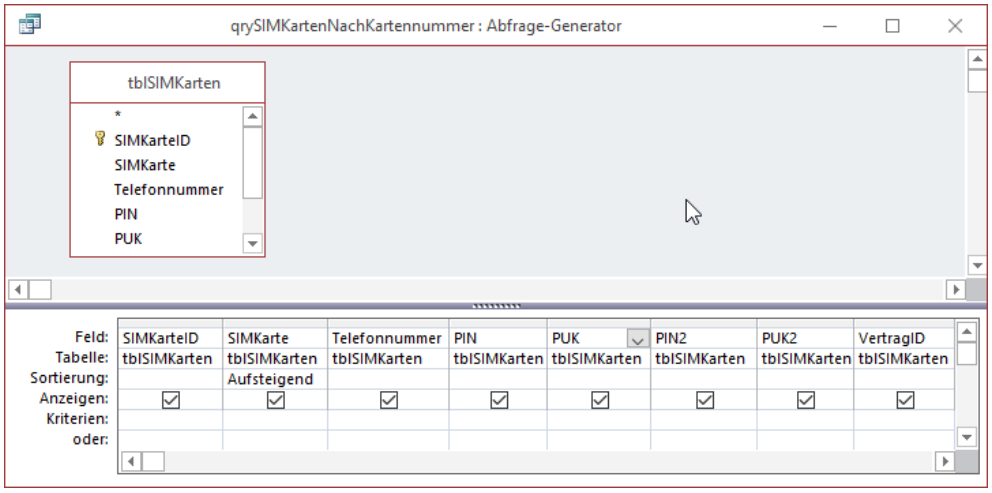



**Bild 2:** Abfrage, welche die Felder der Tabelle **tblSIMKarten** nach Kartennummer sortiert liefert

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**.

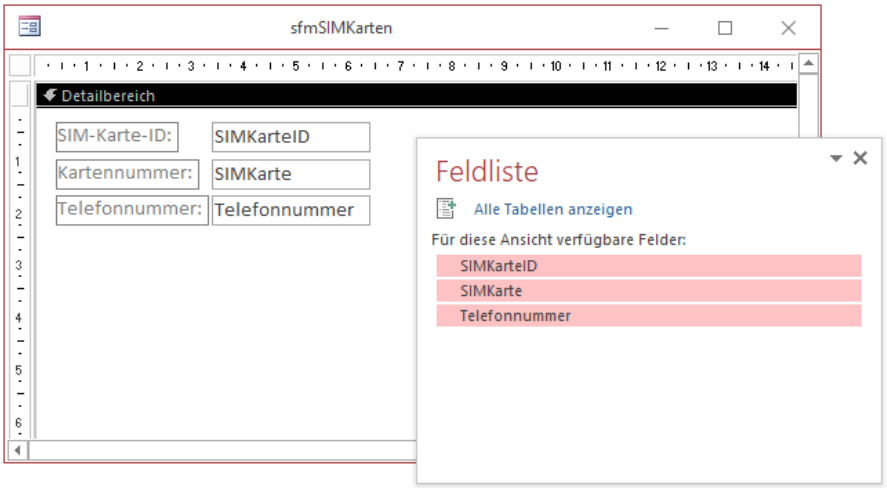



**Bild 3:** Unterformular für die Anzeige aller SIM-Karten

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

# 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).

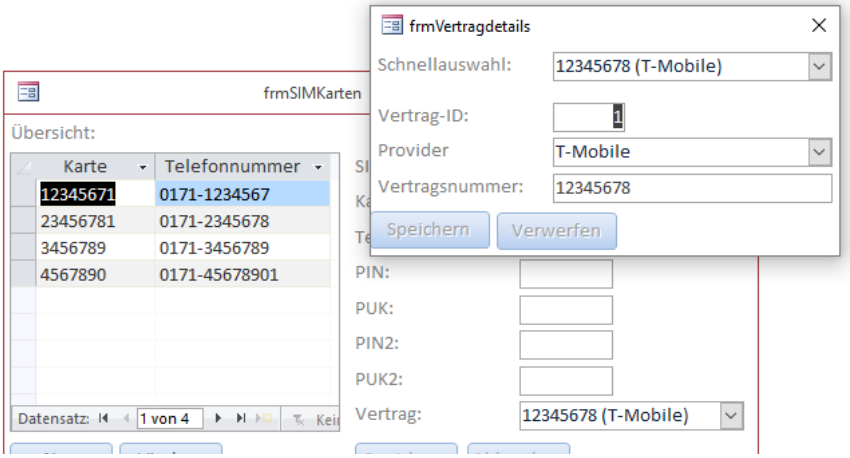



**Bild 1:** Beispiel für das Aufrufen eines Detailformulars

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
```

# 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).

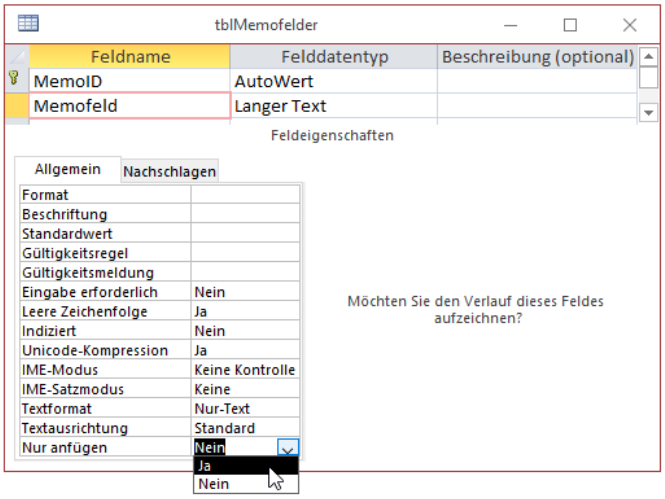



**Bild 1:** Die Eigenschaft **Nur anfügen** von Memofeldern

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).

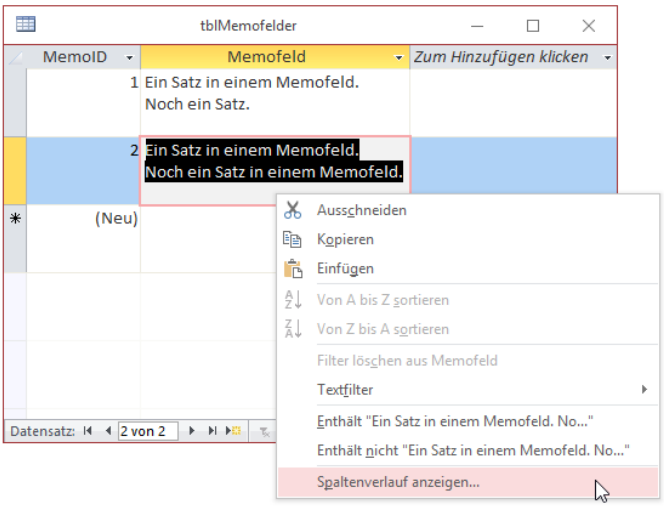



**Bild 2:** Keine Änderungen bei der Eingabe, jedoch ein neuer Kontextmenü-Eintrag

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", _
                            "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.
```

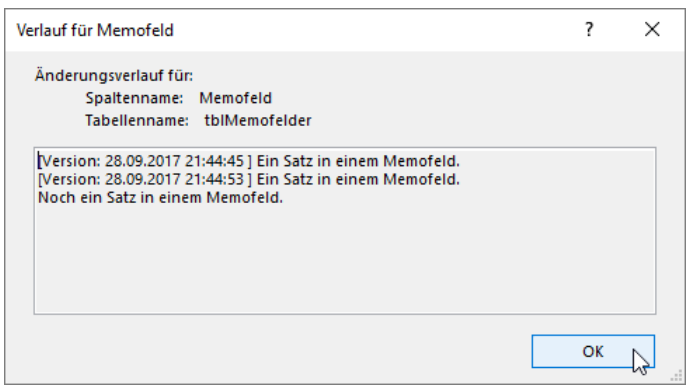
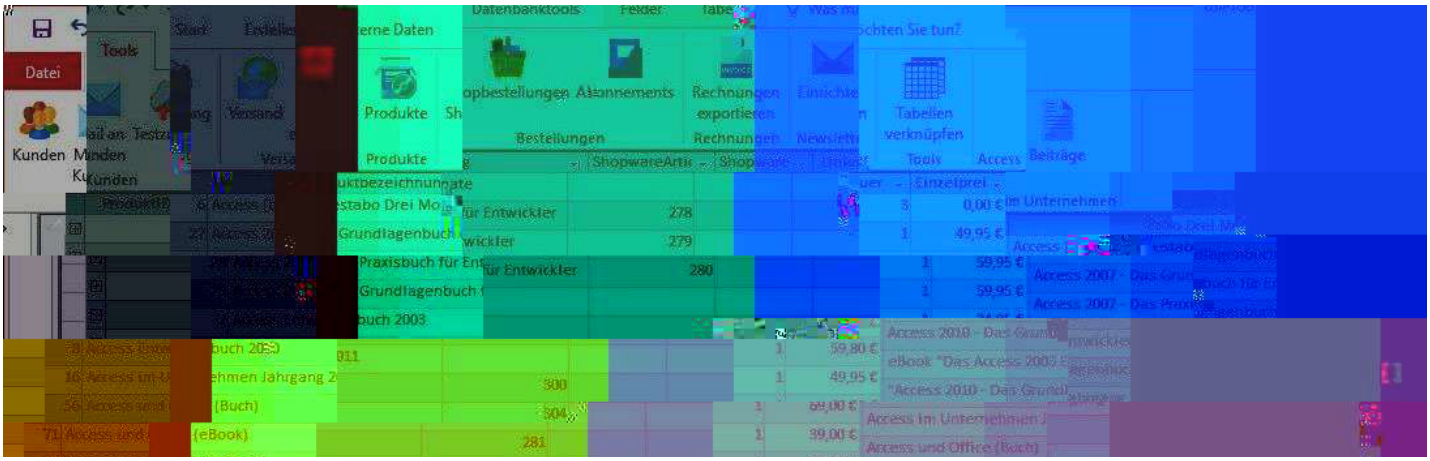



**Bild 3:** Der Verlauf in einem Memofeld

Ausgabe 06/2017

# ACCESS BASICS

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



## In diesem Heft:

SUCHE IN LOOKUPFELDERN (S. 3)

ZEILENVORSCHUB IN BERICHTEN STEUERN (S. 10)

MEHRERE DATENSÄTZE PRO ZEILE IM BERICHT (S. 16)

HAUPT- UND UNTERFORMULAR SYNCHRON (S. 25)

HANDYVERWALTUNG IV: MOBILFUNKGERÄTE VERWALTEN (S. 33)

# 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).

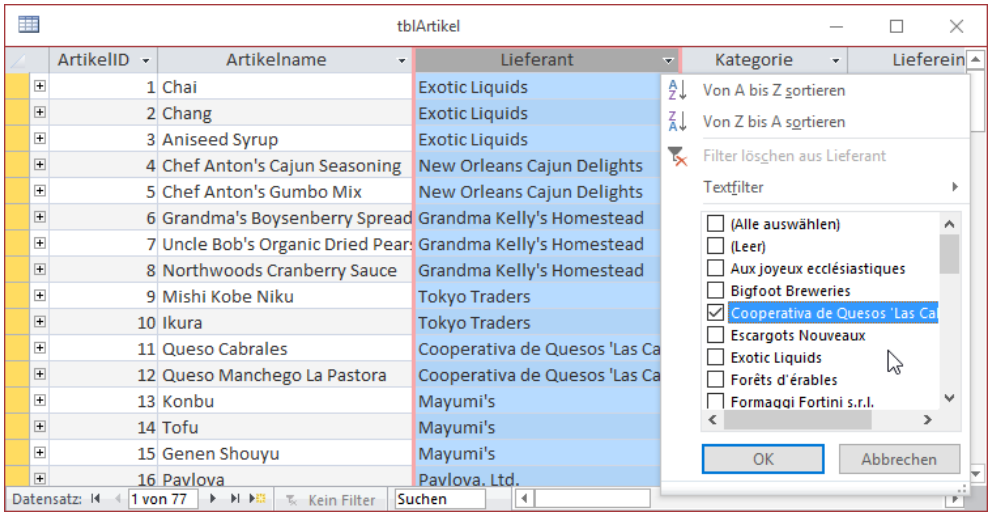



**Bild 1:** Eingebaute Suche nach einem Eintrag

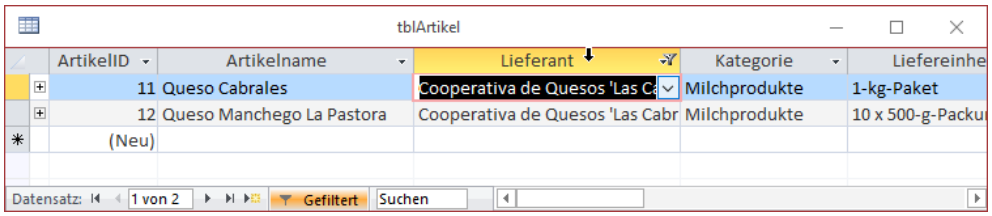



**Bild 2:** Suchergebnis mit eingebauter Suche

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**.

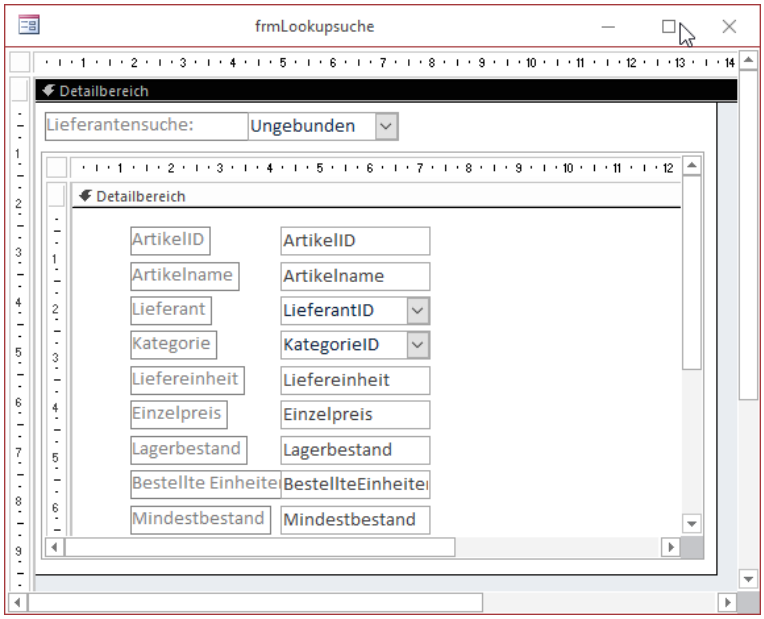



**Bild 3:** Suchformular mit Kombinationsfeld

## 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

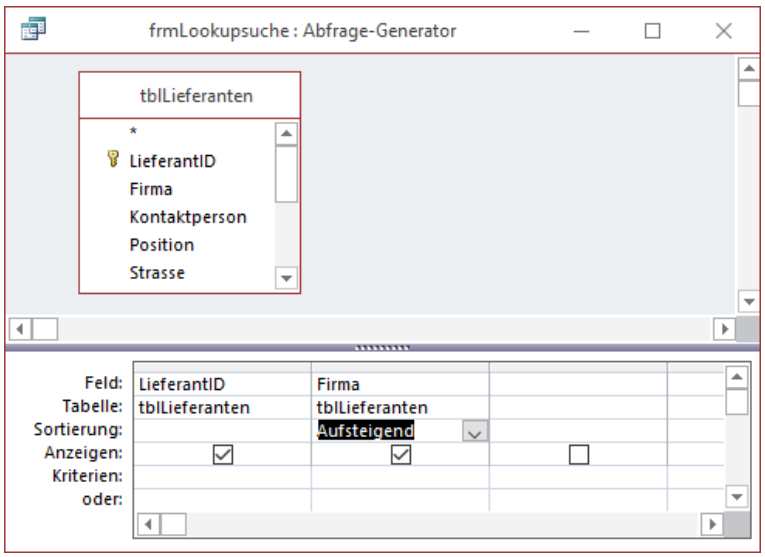



**Bild 4:** Abfrage als Datensatzherkunft für das Such-Kombinationsfeld

# 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 1708_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).

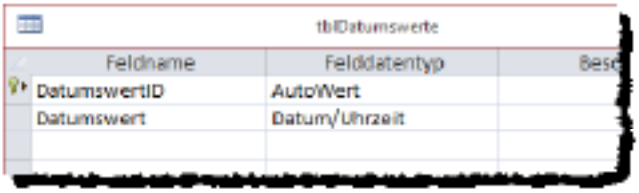



Bild 1: Entwurf der Datumstabelle

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.

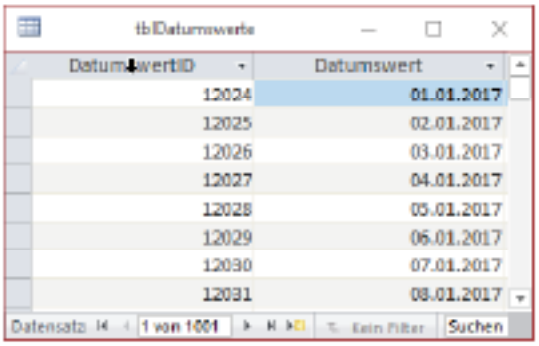



Bild 2: Gefüllte Datumstabelle

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 noch ganz links oben. Ändern Sie seine Größe so, dass es sieben Mal nebeneinander in den Bericht passt (siehe Bild 3).

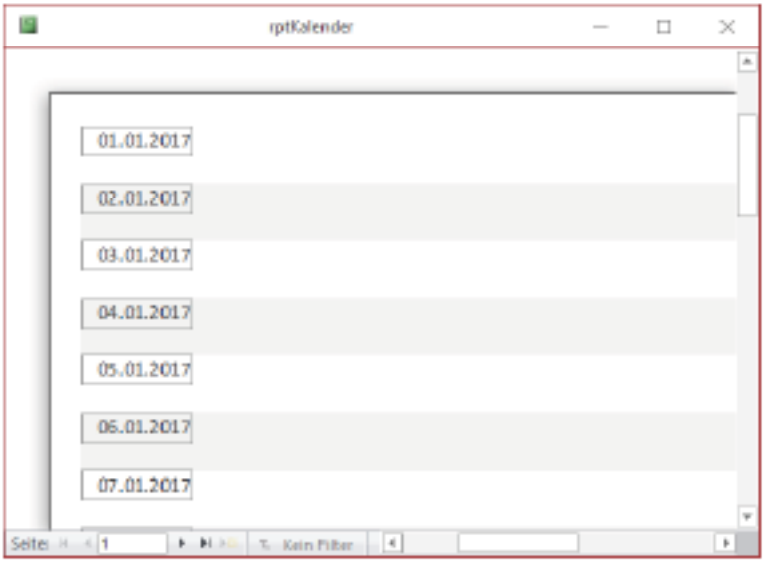



Bild 4: Bericht in der herkömmlichen Ansicht mit einem Datensatz je Zeile

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).

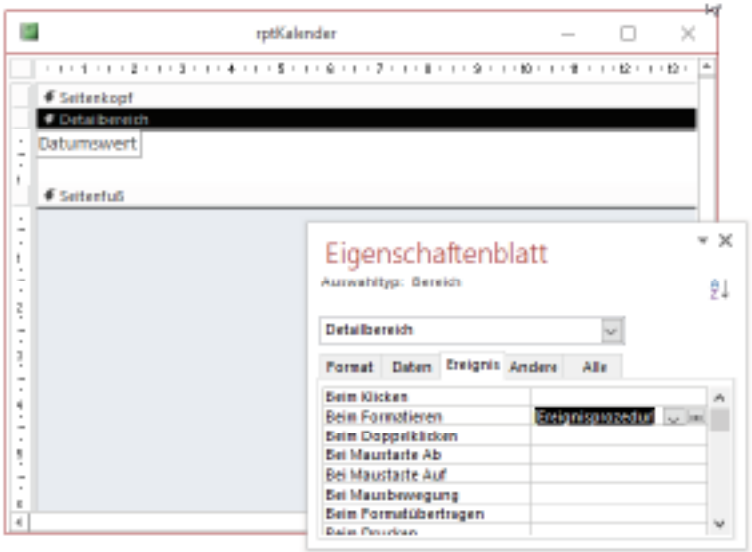



Bild 5: Einfügen einer Ereignisprozedur für Beim Formatieren

# 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:

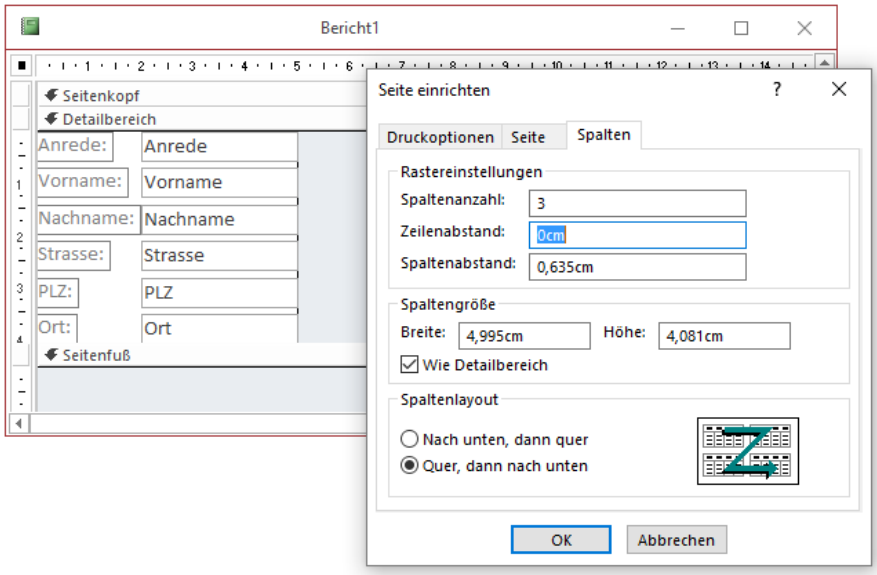



**Bild 1:** Einstellungen für einen mehrspaltigen Bericht

- 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

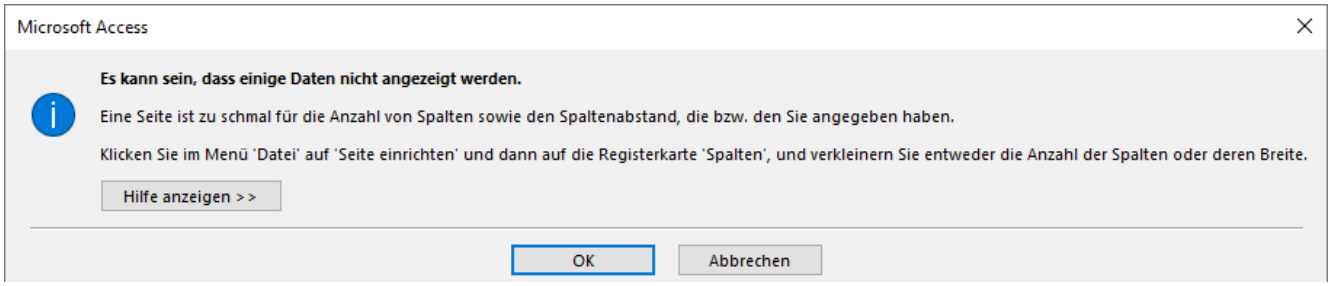



**Bild 2:** Meldung, wenn die Summe der Spaltenbreiten die Breite des Berichts überschreitet

Anzahl der Spalten mit der Eigenschaft **Spaltenanzahl** auf 3 ein. Unter **Spaltenlayout** behalten wir die Einstellung **Quer, dann nach unten** bei.

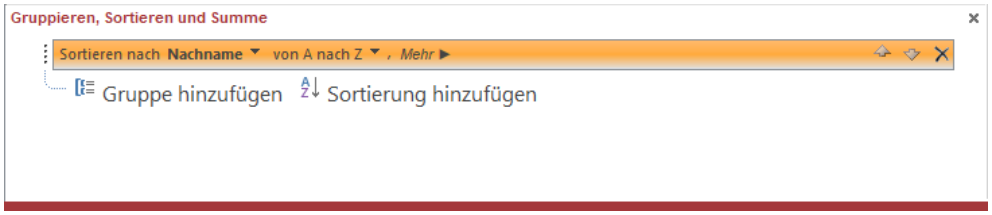



**Bild 3:** Einstellungen für einen mehrspaltigen Bericht

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

# 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.

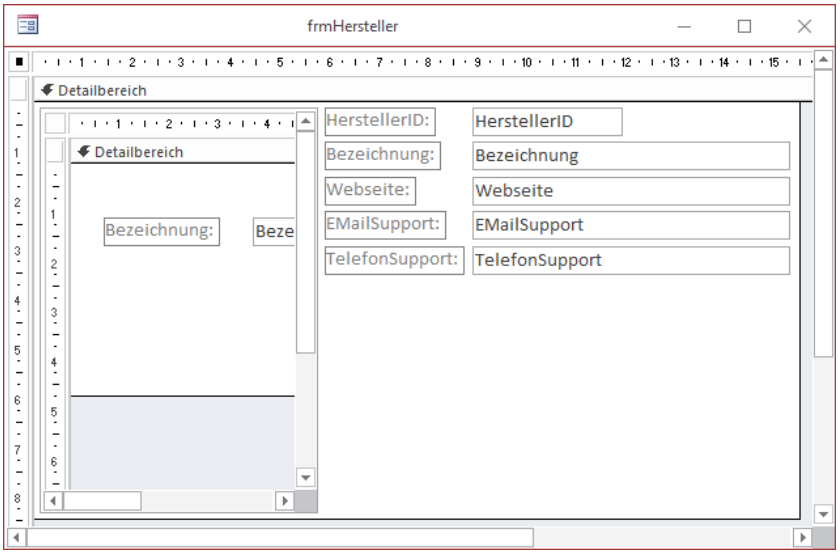



**Bild 1:** Haupt- und Unterformular zur Anzeige der Daten aus der gleichen Datenherkunft

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

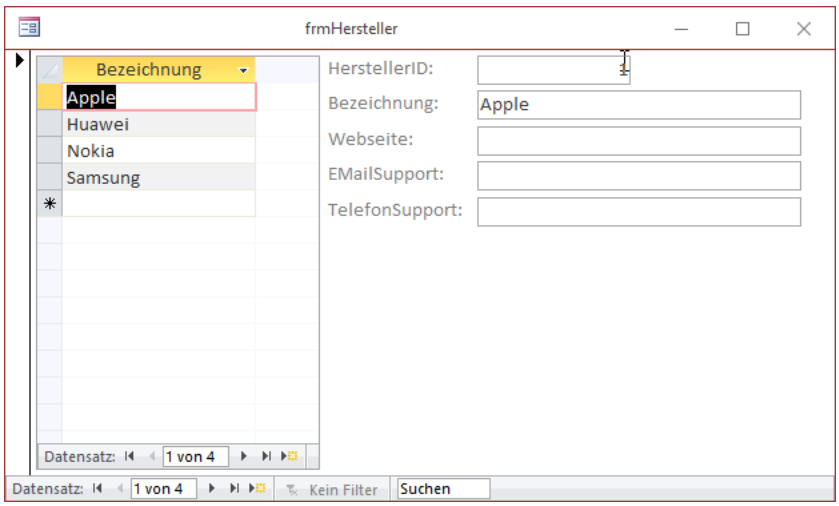



**Bild 2:** Beim Anzeigen erscheinen die Datensätze im Haupt- und Unterformular synchron.

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 =
```

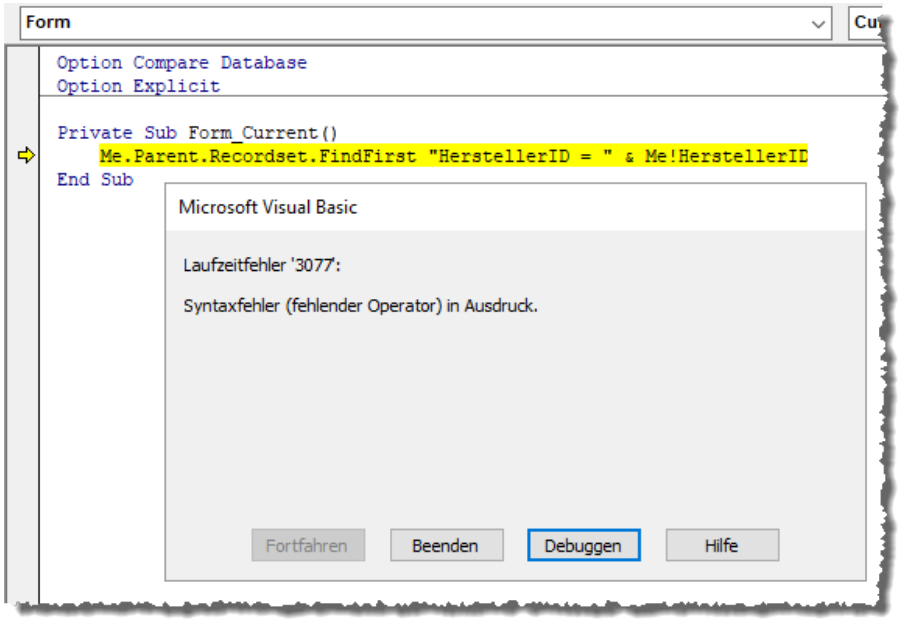



**Bild 3:** Fehler beim Anzeigen eines neuen, leeren Datensatzes

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:

# 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.

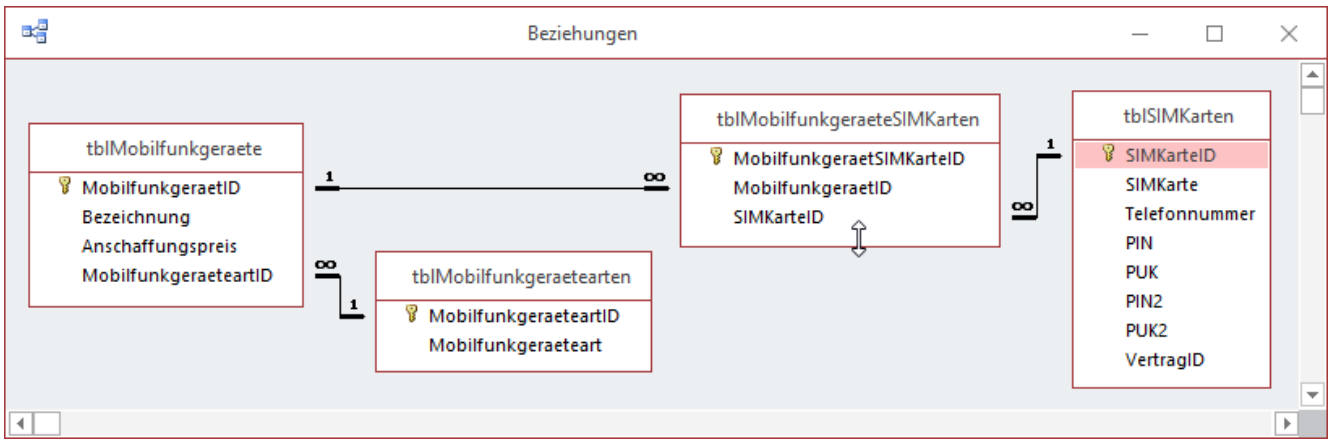



**Bild 1:** Der Teil des Datenmodells, um den wir uns in diesem Artikel kümmern

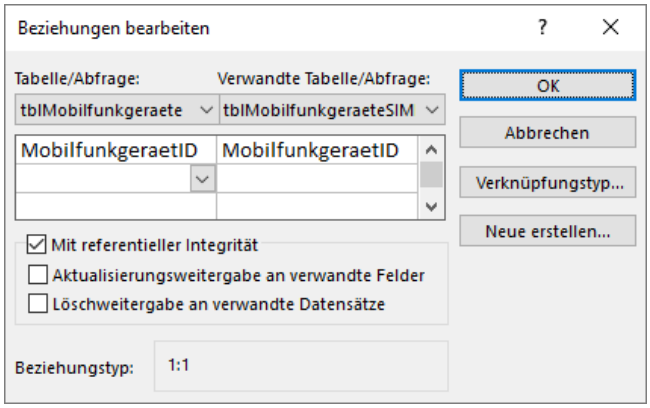



**Bild 2:** Die Beziehung wird als 1:1-Beziehung eingetragen.

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

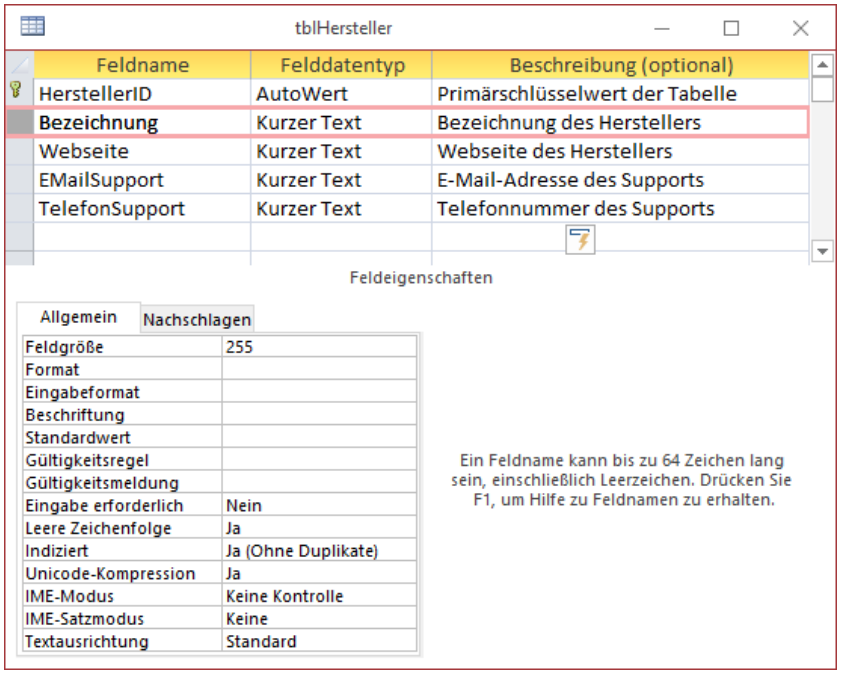



**Bild 3:** Die Beziehung wird als 1:1-Beziehung eingetragen.



# Access [basics] A-Z

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

# Access [basics] A-Z

## Schritt für Schritt zum Erfolg mit Microsoft Access!

Dieses Dokument ist ein Begleitdokument zum Magazin **Access [basics]**. Es enthält die wirklichen **[basics]** für die tägliche Arbeit mit Access. Hier beschreiben wir die Schritte, die beispielsweise zum Erstellen einer Tabelle, zum Hinzufügen von Feldern, zum Definieren einer Abfrage oder zum Bauen eines Formulars nötig sind. Dieses Dokument wird ständig erweitert. In den regulären Ausgaben von **Access [basics]** finden Sie jeweils Verweise in der Form →*[basics]* **Tabelle erstellen**. Das bedeutet: Im **[basics]**-Dokument gibt es einen Abschnitt mit der Überschrift **Tabelle erstellen**, der weitere Informationen zum Durchführen einer Aktion liefert. Nach einer Weile brauchen Sie diese Schritte nicht mehr nachzuschlagen und können dies beim Lesen der eigentlichen Artikel in **Access [basics]** auslassen, und wir brauchen uns nicht die Finger wund zu tippen, um immer wieder die gleichen Schritte zu beschreiben. In diesem Dokument gehen wir auf die Versionen 2003 und älter sowie 2007 und jünger ein, da Microsoft mit Access 2007 einige Neuerungen eingeführt haben, die wesentliche Änderungen in der Benutzeroberfläche nach sich zogen und somit unterschiedliche Vorgehensweisen zum Durchführen vieler Aktionen bewirkte.

## Funktionsweise für Papierfreunde

Wir haben bei der Gestaltung die Möglichkeit berücksichtigt, dass Sie die **[basics]** ausdrucken. Daher sind diese erstens alphabetisch sortiert und enthalten oben rechts die ersten beiden Buchstaben der jeweiligen Themenbezeichnung. Sie brauchen dann nur schnell durchzublättern, um zum gewünschten Artikel zu gelangen.

Außerdem soll die **[basics]**-Sammlung kontinuierlich erweitert werden. Erstens kommen mit jeder Ausgabe neue Themen hinzu, und zweitens erweitern wir gegebenenfalls auch einzelne Themen. Beispiele dafür sind →*[basics]* **Tastenkombinationen** oder →*[basics]* **Konventionen**. Damit Sie nur das jeweilige Thema neu ausdrucken müssen, beginnen wir jedes Thema auf einer neuen Seite. So können Sie leicht neue Themen zwischen zwei anderen Themen einfügen oder erweiterte Themen ersetzen.

## Neue Funktionen

Ab dieser Version können Sie über einen Klick auf **->Inhaltsverzeichnis** im Seitenkopf zum Inhaltsverzeichnis zurückspringen.

Außerdem finden Sie alle Informationen zum Erstellungsdatum und zum Datum der letzten Änderung im Inhaltsverzeichnis. Änderungen seit der letzten Version werden farbig hervorgehoben.

# Inhaltsverzeichnis


| Thema | Hinzugefügt | Zuletzt geändert |
|---|---|---|
| 1:n-Beziehung anlegen | 3/2010 | |
| Abbrechen-Schaltfläche erstellen | 2/2010 | |
| Abfrage erstellen | 3/2010 | |
| Abfrage-Eigenschaften | 6/2010 | |
| Aktivierreihenfolge von Steuerelementen | 2/2010 | |
| Aktualisierungsweitergabe festlegen | 3/2010 | |
| Autowertfeld anlegen | 2/2010 | |
| Bericht erstellen | 4/2010 | |
| Berichtsansichten | 4/2010 | |
| Berichtsbereiche | 4/2010 | |
| Beschriftungen von Tabellenfeldern einstellen | 2/2010 | |
| Beziehungen-Fenster | 3/2010 | |
| Bilder unter Access 2007 und jünger | 6/2010 | |
| Datum/Zeit-Feld anlegen | 2/2010 | |
| Datumsfeld im Bericht anlegen | 4/2010 | |
| Eingabeformat-Eigenschaft | 9/2010 | |
| Ereignisprozedur hinzufügen | 2/2010 | |
| Felddatentypen | 9/2010 | |
| Format-Eigenschaft | 9/2010 | |
| Formular maximiert anzeigen | 6/2010 | |
| Formularansichten | 2/2010 | |
| Formularbereiche | 2/2010 | |
| Formularelemente referenzieren | 8/2010 | |
| Gebundenes Formular anlegen | 2/2010 | |
| Gebundene Steuerelemente zu Formularen und Berichten hinzufügen | 4/2010 | |
| Gruppierungen anlegen | 4/2010 | |
| Gültigkeitsbereich von VBA-Programmen | 6/2010 | |
| Haupt- und Unterformular mit verknüpften Daten | 3/2010 | |
| Index für ein Tabellenfeld festlegen | 3/2010 | |
| Ja/Nein-Feld anlegen | 2/2010 | |
| Keine Dateneingabe via Tabelle oder Abfrage | 3/2010 | |
| Konventionen | 2/2010 | |
| Löschweitergabe festlegen | 3/2010 | |
| Nachschlagefeld für eine 1:n-Beziehung einrichten | 3/2010 | |
| Nachschlagefeld für Wertliste einrichten | 2/2010 | |
| OK- und Abbrechen-Schaltfläche hinzufügen | 2/2010 | |
| OK-Schaltfläche erstellen | 2/2010 | |
| Optionen einstellen | 6/2010 | |
| Primärschlüsselfeld festlegen | 2/2010 | |
| Referentielle Integrität festlegen | 3/2010 | |
| Seitenränder eines Berichtes einstellen | 4/2010 | |

| Thema | Hinzugefügt | Zuletzt geändert |
|---|---|---|
| Seitenzahl zu Bericht hinzufügen | 4/2010 | |
| Standardwerte für Tabellenfelder einstellen | 2/2010 | |
| Startformular einer Anwendung festlegen | 6/2010 | |
| Steuerelemente ausrichten und skalieren | 2/2010 | |
| Tabelle erstellen | 3/2010 | |
| Tabellenentwurf-Einstellungen anpassen | 2/2010 | |
| Tastenkombinationen | 2/2010 | 1/2011 |
| Textfeld anlegen | 2/2010 | |
| Unterformular zu Formular hinzufügen | 3/2010 | |
| VBA-Befehle: MsgBox | 6/2010 | |
| VBA-Editor öffnen | 6/2010 | |
| VBA-Modul anlegen | 6/2010 | |
| VBA-Routine schreiben | 6/2010 | |
| VBA-Prozedur ausführen | 6/2010 | |
| Zahlenfeld anlegen | 3/2010 | |

# 1:n-Beziehung anlegen

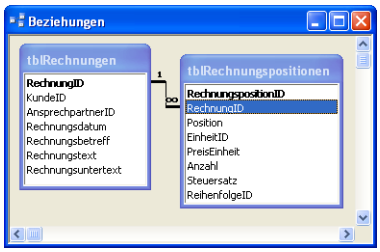
Wenn Sie eine 1:n-Beziehung zwischen Tabellen herstellen möchten, von denen keine eine Lookup-Tabelle mit Detaildaten ist, können Sie meist auf ein Nachschlagefeld verzichten. Ein gutes Beispiel sind Rechnungen und Rechnungspositionen: Jede Rechnungsposition speichert zwar in ihrem Datensatz die Nummer der Rechnung, zu der sie gehört, aber diese Nummer wird nie in einem Formular oder einem Bericht auftauchen. Solche Beziehungen werden anders dargestellt, beispielsweise durch die Anzeige aller verknüpften Positionen einer Rechnung im Unterformular eines Formulars oder im Detailbereich eines Berichts, dessen Berichtskopf die eigentlichen Rechnungsdaten anzeigt. Dies läuft darauf hinaus, dass es keinen Sinn macht, für solche eine Beziehung ein Nachschlagefeld einzurichten: Sie legen zwar eine Rechnungsposition zu einer Rechnung an, was aber implizit geschieht. Es wird nie passieren, dass man einer Rechnungsposition eine neue Rechnung zuweist, wofür wiederum ein Nachschlagefeld interessant wäre.

Also zeigen wir einfach nur, wie Sie eine 1:n-Beziehung zwischen zwei Tabellen anlegen, ohne dies über den Nachschlage-Assistenten zu erledigen.

- Legen Sie zunächst die Detailtabelle an, also die Tabelle, welche die zu verknüpfenden Datensätze enthält – in unserem Beispiel die Tabelle mit den Rechnungen. Das Primärschlüsselfeld soll beispielhaft **RechnungID** heißen.
- Legen Sie dann die Mastertabelle an, also diejenige Tabelle, welche das Fremdschlüsselfeld enthält, dass den Verweis auf den Primärschlüsselwert der anderen Tabelle aufnimmt – hier die Tabelle mit den Rechnungspositionen. Das Primärschlüsselfeld heißt **RechnungspositionID**, das Fremdschlüsselfeld heißt **RechnungID** und hat den gleichen Datentyp wie das Primärschlüsselfeld der zu verknüpfenden Tabelle, also Zahl.
- Öffnen Sie dann das Beziehungenfenster (Access 2003 und älter: Menüeintrag **Extras|Beziehungen**, Access 2007: Ribboneintrag **Datenbanktools|Einblenden/Ausblenden|Beziehungen**).
- Ziehen Sie die betroffenen Tabellen, zum Beispiel **tblRechnungen** und **tblRechnungspositionen**, aus dem Datenbankfenster (Access 2003 und älter) beziehungsweise dem Navigationsbereich (ab Access 2007) in das **Beziehungen**-Fenster.
- Ziehen Sie das Feld Fremdschlüsselfeld **RechnungID** aus der Tabelle **tblRechnungspositionen** per Drag and Drop auf das Primärschlüsselfeld **RechnungID** der Tabelle **tblRechnungen**.

Es erscheint nun der **Beziehung bearbeiten**-Dialog, mit dem Sie weitere Eigenschaften der Beziehung festlegen können (siehe auch →*[basics]* **Referentielle Integrität festlegen**, →*[basics]* **Aktualisierungsweitergabe festlegen** und →*[basics]* **Löschweitergabe festlegen** ).

Anschließend erscheint im Beziehungsfenster ein Pfeil, der die Art der Beziehung charakterisiert:




Eine 1:n-Beziehung im Beziehungen-Fenster

# Abbrechen-Schaltfläche erstellen

Eine **Abbrechen**-Schaltfläche dient normalerweise dazu, ein gebundenes Formular zu verlassen, ohne die durchgeführten Änderungen zu speichern. Dazu legen Sie die folgende Ereignisprozedur für das Ereignis **Beim Klicken** einer Schaltfläche etwa namens **cmdAbbrechen** an (→**Ereignisprozedur hinzufügen**):

```
Private Sub cmdAbbrechen_Click()
    Me.Undo
    DoCmd.Close acForm, Me.Name
End Sub
```

Die erste Anweisung dieser Routine macht die Änderungen rückgängig, die seit dem letzten Speichern am angezeigten Datensatz vorgenommen wurden. Dies betrifft nicht solche Änderungen, die Sie an den Daten eines Unterformulars mit verknüpften Daten durchgeführt haben – diese sind spätetens nach dem Wechseln zum Hauptformular und somit beim Anklicken eines seiner Steuerelemente (hier der **Abbrechen**-Schaltfläche) gespeichert und können nicht mehr rückgängig gemacht werden.

Die zweite Anweisung schließt das Formular.

### Abbrechen-Klick per Eingabetaste

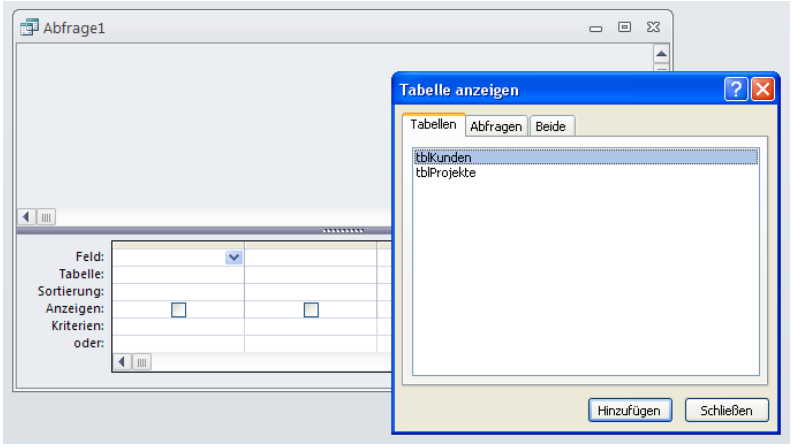
Der Benutzer kann die durch den Klick auf die **Abbrechen**-Schaltlfäche ausgelöste Ereignisprozedur auch über die **Esc**-Taste aufrufen. Dazu stellen Sie einfach die Eigenschaft **Abbrechen** der Schaltfläche **cmdAbbrechen** auf **Ja** ein. Ein Klick auf die **Esc**-Taste löst nun die gleiche Ereignisprozedur aus wie ein Mausklick auf **cmdAbbrechen** und schließt somit das Formular nach vorherigem Rückgängigmachen der seit dem letzten Speichern durchgeführten Änderungen.

# Abfrage erstellen

Eine Abfrage erstellen Sie entsprechend der Access-Version auf verschiedene Arten:

**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 zeigt Access die Abfrage-Entwurfsansicht an und bietet mit dem Dialog **Tabelle anzeigen** die Möglichkeit, die als Datenherkunft zu verwendende Tabelle zum Abfrageentwurf hinzuzufügen.

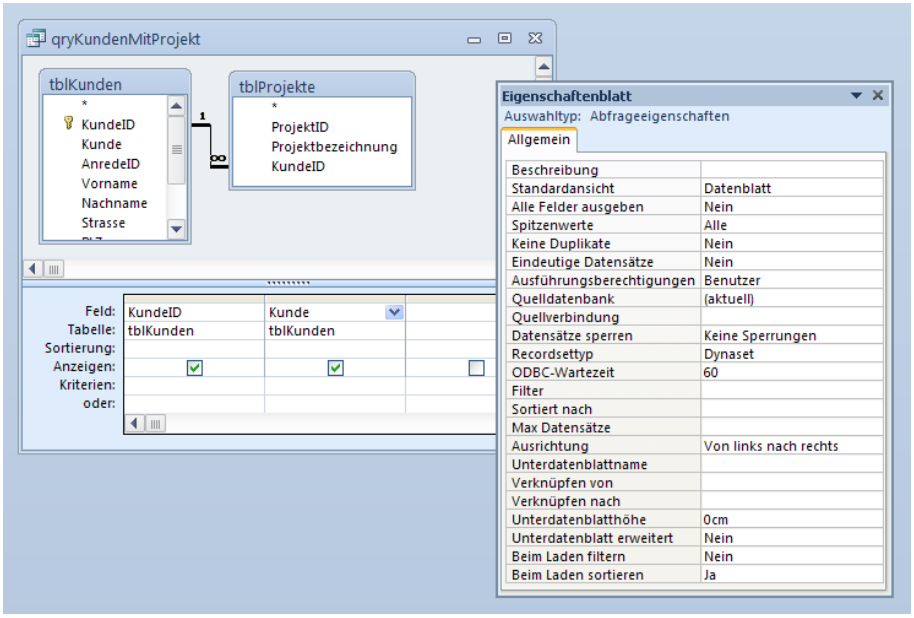
Nach dem Hinzufügen finden sich die ausgewählten Tabellen beziehungsweise Abfragen, die als Datenherkunft der Abfrage dienen, im Abfrageentwurf wieder.

Ziehen Sie dann alle Felder der als Datenherkunft angegebenen Tabellen, deren Inhalt die Abfrage anzeigen soll, in den unteren Bereich der Entwurfsansicht, dem sogenannten Entwurfsraster.

# Abfrage-Eigenschaften

**Der Abfrageentwurf bietet eine ganze Reihe Eigenschaften, mit denen Sie das Verhalten einer Abfrage steuern können. Dieser Artikel stellt die wichtigsten Eigenschaften vor.**

Die Abbildung zeigt das Eigenschaftenblatt einer typischen Abfrage. Die folgende Liste der Beschreibungen dieser Eigenschaften wird im Laufe derZeit ergänzt, wenn eine der Eigenschaften in einem Artikel von Access [basics] verwendeet wird.




## Keine Duplikate

Diese Eigenschaft stellt sicher, dass Duplikate der angezeigten Daten ausgeblendet werden. Die Betonung liegt dabei im Gegensatz zur Eigenschaft **Eindeutige Datensätze** auf "angezeigte Daten".

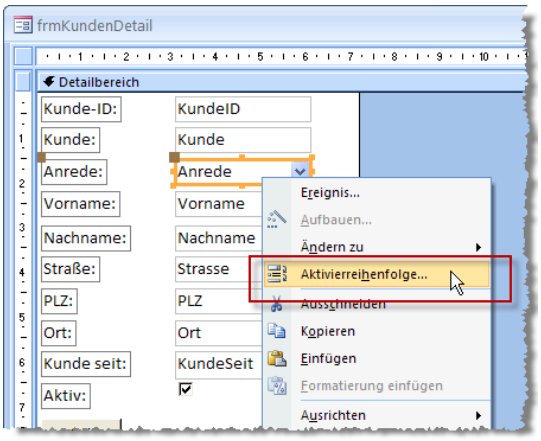
## Eindeutige Datensätze

Diese Eigenschaft sorgt dafür, dass Datensätze, die bezogen auf alle Felder der Datenherkunft komplett gleich sind, nur einmal angezeigt werden.

# Aktivierreihenfolge von Steuerelementen

Beim Erstellen eines Formulars könnten Dinge schiefgelaufen sein, durch die das Formular sich nicht wie gewünscht verwenden lässt. So könnte es sein, dass die Steuerelemente durch Betätigen der **Tab**-Taste nicht in der Reihenfolge durchlaufen werden, wie Sie es sich eigentlich vorstellen.

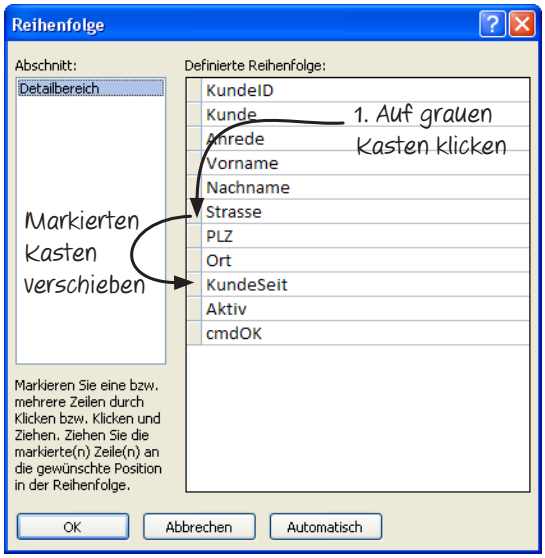
Das liegt dann möglicherweise daran, dass Sie die Steuerelemente nicht auf einen Rutsch aus der Feldliste in den Detailbereich des Formulars gezogen oder nachträglich deren Reihenfolge geändert haben.




Access achtet nämlich normalerweise darauf, in welcher Reihenfolge der Entwickler Steuerelemente zum Formular hinzufügt – und in dieser Reihenfolge werden diese dann etwa per **Tab**-Taste durchlaufen. Sie können diese Reihenfolge allerdings nachträglich ändern. Dazu klicken Sie einfach mit der rechten Maustaste auf eines der Steuerelemente und wählen aus dem Kontextmenü den Eintrag **Aktivierreihenfolge** aus.

Der Dialog **Reihenfolge** zeigt nun alle im Formular enthaltenen Steuerelemente an. Wenn Sie die Aktivierreihenfolge ändern möchten, gehen Sie in zwei Schritten vor:

- Klicken Sie mit der Maus auf den grauen Kasten links vom zu verschiebenden Eintrag, sodass Access den betroffenen Eintrag schwarz hinterlegt.
- Klicken Sie dann erneut auf den grauen Kasten, halten Sie die Maustaste dabei gedrückt, und ziehen Sie den Eintrag auf den Zwischenraum der beiden Einträge, zwischen die der aktuelle Eintrag rücken soll.




Falls Sie ein Steuerelement komplett aus der Aktivierreihenfolge herausnehmen möchten, brauchen Sie nur seine Eigenschaft **In Reihenfolge** auf **Nein** einzustellen. Dazu öffnen Sie das Formular in der Entwurfsansicht, wählen im Eigenschaftsfenster die Registerseite **Andere** aus und stellen dort die entsprechende Eigenschaft ein.

# Aktualisierungsweitergabe festlegen

Diese Eigenschaft stellen Sie im gleichen Dialog ein, in dem Sie auch referentielle Integrität zwischen zwei Feldern verknüpfter Tabellen festlegen (→*[basics]* **Referentielle Integrität festlegen**). Diese ist auch Voraussetzung für die Aktualisierungsweitergabe.

Die Aktivierung der Aktualisierungsweitergabe sorgt dafür, dass bei Änderungen an den Werten des Primärschlüsselfelds der einen verknüpften Tabelle auch die entsprechenden Werte im Fremdschlüsselfeld der anderen Tabelle geändert werden.

# Autowertfeld anlegen

Ein Autowertfeld legen Sie an, indem Sie den Namen des gewünschten Feldes in die Spalte **Feldname** einer leeren Zeile einer Tabelle in der Entwurfsansicht schreiben. Der **Felddatentyp** erhält den Wert **Autowert**.

Autowertfelder sind prinzipiell Zahlen- oder Textfelder, für die Access beim Anlegen eines neuen Datensatzes automatisch einen noch nicht verwendeten Wert einträgt. Standardmäßig verwendet Access Zahlen, um ein Autowertfeld zu füllen – dann ist als Feldgröße **Long Integer** eingestellt. Die Zahl für einen neuen Datensatz ist dann um eins größer als die bisher größte verwendete Zahl.

Sie können als Feldgröße auch Replikations-ID einstellen. Access legt dann Werte im GUID-Format an, die etwa so aussehen: **{A5036FF7-6804-4603-97B9-402DA4D3D9D3}**

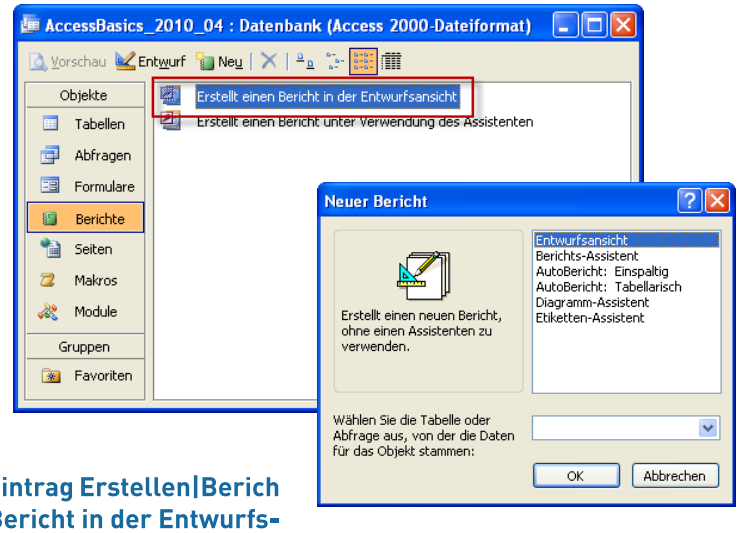
# Bericht erstellen

Die Basis für das Erstellen eines Berichtes ist das Anlegen eines leeren Berichtes und das Öffnen dieses Berichtes in der Entwurfsansicht. Gegebenenfalls möchten Sie den Bericht vielleicht noch unter dem gewünschten Namen speichern – dies erledigen Sie ganz einfach durch Betätigen der Tastenkombination **Strg + S**. Beachten Sie bei der Namensvergabe die gängigen →*[basics]* **Konventionen**. Typischerweise weisen Sie dem Bericht anschließend gleich die Datenherkunft zu, aus der er die anzuzeigenden Daten beziehen soll – der Rest ist je nach Bericht individuell.

Einen Bericht in der Entwurfsansicht erstellen Sie abhängig von der Access-Version auf folgende Weise:

**2003 und älter:** Wählen Sie im Datenbankfenster den Eintrag **Erstellt einen Bericht in der Entwurfsansicht** aus oder öffnen Sie den Dialog **Neuer Bericht** über den Menüeintrag **Einfügen|Bericht** und wählen Sie dort den Eintrag **Entwurfsansicht** aus.

Sie können auch im Register **Tabellen** oder **Abfragen** des Datenbankfensters eine Datenherkunft für den Bericht auswählen und dann den Menübefehl Einfügen|Bericht betätigen: Der Dialog **Neuer Bericht** zeigt dann gleich die gewählte Datenherkunft im entsprechenden Kombinationsfeld an.



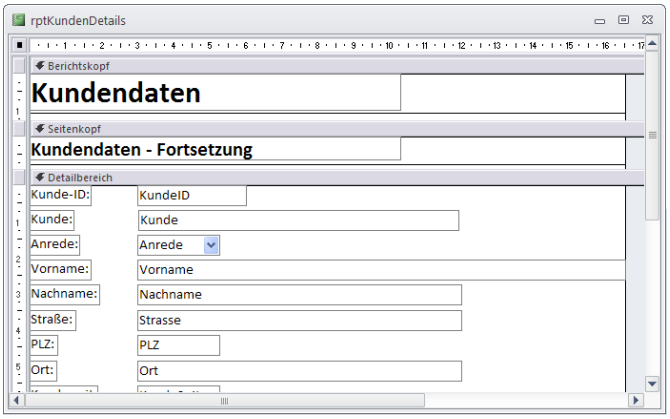

**2007 und jünger: Verwenden Sie den Ribbon-Eintrag Erstellen|Berichte|Berichtsentwurf, um einen neuen, leeren Bericht in der Entwurfsansicht zu öffnen.**

Nach dem Erstellen des Berichts und der Auswahl der Datenherkunft können Sie die gewünschten Steuerelemente im Bericht platzieren.

# Berichtsansichten

Bis Access 2003 gab es drei verschiedene Berichtsansichten, ab 2007 ist dann eine weitere hinzugekommen und eine Ansicht wurde grundlegend überarbeitet.

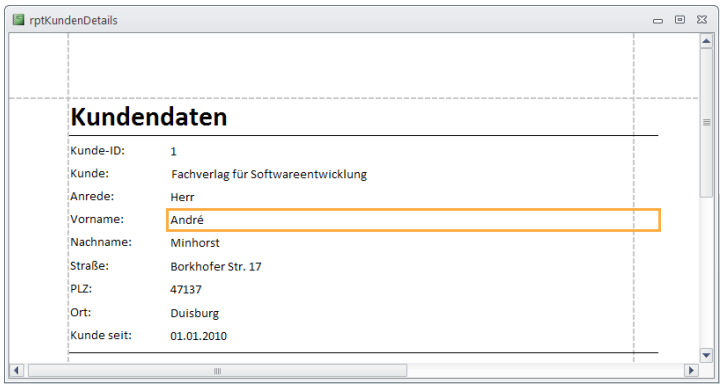
Dies sind die Ansichten von Access 2003 und älter:




- **Entwurfsansicht**: Diese Ansicht dient dem Entwurf des Berichts. Hier können Sie die Größe der Bereiche einstellen, Bereiche hinzufügen und wieder entfernen und Steuerelemente einfügen, positionieren und anpassen.
- **Seitenansicht**: Diese Ansicht gibt das Bild des Berichts beim Ausdrucken oder beim Exportieren in eine PDF-Datei wieder. Sie können dort genau betrachten, wie der gedruckte Bericht aussehen wird.
- **Layoutvorschau**: Dies ist eine schnelle Variante der Seitenansicht. Gelegentlich zeigen Berichte große Datenmengen an und muss unter Umständen den kompletten Bericht vor dem Anzeigen der ersten Seite rändern – das ist beispielsweise der Fall, wenn die erste Seite bereits die gesamte Seitenzahl des Berichts anzeigen soll. Die Layoutansicht zeigt nicht alle Datensätze der Datenherkunft an und und enthält weitere Einschränkungen. Für ein repräsentatives Ergebnis sollten Sie daher eher die Seitenansicht mit einer entsprechend gefilterten Datenherkunft öffnen, um ein Bild vom Aufbau des Berichts zu erhalten.

Mit Access 2007 kamen dann die Berichtsansicht hinzu und die Layoutansicht wurde überarbeitet:

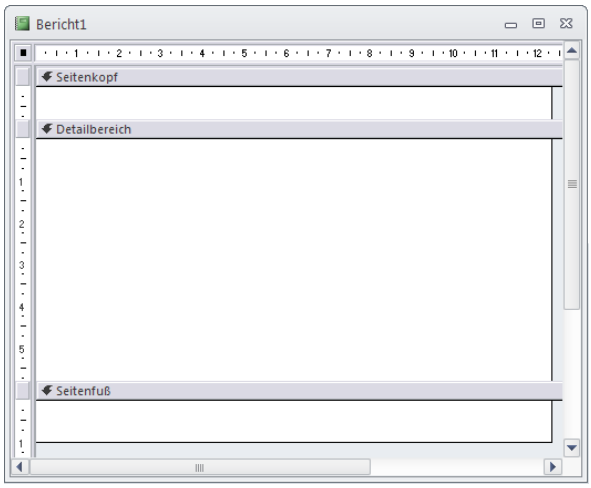
- **Berichtsansicht**: Dies ist ein Hybrid aus einem Bericht und einem Formular. Sie können dort Schaltflächen und andere interaktive Elemente unterbringen, eine direkte Bearbeitung der Daten ist jedoch leider nicht möglich. Die Berichtsvorschau liefert prinzipiell den gleichen Inhalt wie die Seitenansicht, allerdings in einer Endlosansicht.
- **Layoutvorschau**: Diese überarbeitete Ansicht ist eine Mischung aus der Entwurfsansicht und der Seitenansicht. Die Steuerelemente werden mit den Daten der Datenherkunft gefüllt, damit Sie erkennen können, ob diese überall groß genug ausgelegt sind. Sie können direkt in der Layoutansicht in den Entwurf des Berichts eingreifen und beispielsweise die Position oder die Größe von Steuerelementen einstellen.



# Berichtsbereiche

Ein Bericht kann eine ganze Menge Bereiche enthalten. Wieviele Sie verwenden, hängt von den anzuzeigenden Daten und dem gewünschten Berichtsaufbau ab.

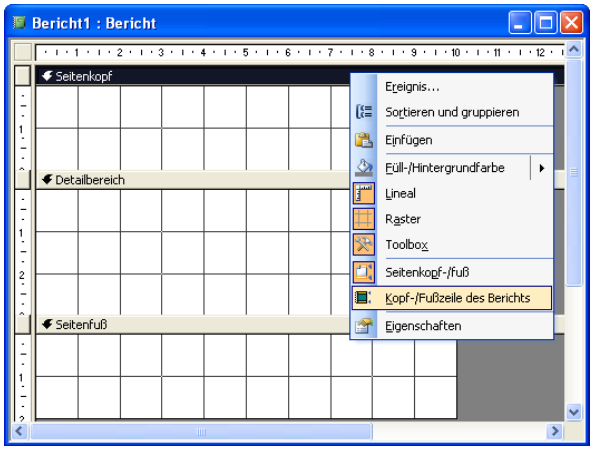
Ein frisch angelegter Bericht enthält standardmäßig drei Bereiche:




- Seitenkopf: Dieser Bereich wird auf jeder Seite oben angezeigt. In einem Seitenkopf fügen Sie beispielsweise das Datum oder die Seitenzahl ein (kann auch im Seitenfuß geschehen). Bei Berichten, die ihre Daten in tabellarischer Form anzeigen, bringen Sie oft die Spaltenüberschriften im Seitenkopf unter, damit diese auf jeder Seite oben über den eigentlichen Daten erscheinen.
- Detailbereich: Dieser Bereich wird für jeden Datensatz der Datenherkunft einmal angezeigt und dabei zwischen den übrigen Bereichen angeordnet – standardmäßig also zwischen Seitenkopf und Seitenfuß.
- Seitenfuß: Dieser Bereich wird am Ende jeder Seite angezeigt. Hier bringen Sie Informationen wie das Datum, den Dokumentnamen oder auch die Seitenzahl unter. Gern werden hier auch berechnete Werte auf Basis der im Detailbereich angezeigten Daten untergebracht – zum Beispiel die Summe von Bestellpositionen.

## Berichtskopf- und Fuß

Sie können einem Bericht zusätzlich einen Berichtskopf und einen Berichtsfuß zuweisen:

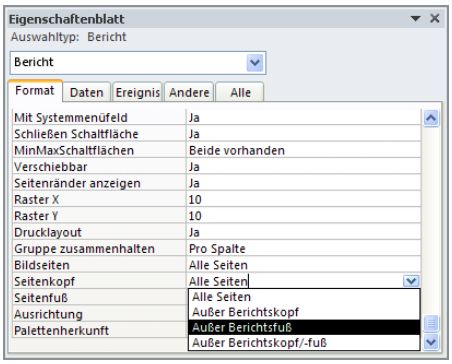



**2003 und älter:** Öffnen Sie den Bericht in der Entwurfsansicht und wählen Sie den Menüeintrag **Ansicht|Kopf-/Fußzeile** des Berichts aus oder betätigen Sie den Kontextmenü-Eintrag **Berichtskopf/-fuß**. Achtung: Dieser Kontextmenü-Eintrag steht nur zur Verfügung, wenn Sie vorher einen der Berichtsköpfe in der Entwurfsansicht angeklickt haben, also beispielsweise **Seitenkopf** wie in der Abbildung rechts.

**2007 und jünger: Öffnen Sie den Bericht in der Entwurfsansicht und betätigen Sie den Kontextmenü-Eintrag Berichtskopf/-fuß.**

## Berichtskopf und -fuß und Seitenkopf und -fuß im Zusammenspiel

Dummerweise wird der Seitenfuß immer ganz unten auf der Seite angezeigt – auch, wenn die Seite einen Berichtsfuß enthält. Sie sollten also beispielsweise dafür sorgen, dass der Seitenfuß nur auf allen Seiten angezeigt wird, die keinen Berichtsfuß enthalten – also auf allen außer der letzten. Für solche Fälle hält Access spezielle Eigenschaften vor, zum Beispiel **Seitenkopf** und **Seitenfuß**. Mit beiden können Sie festlegen, ob der jeweilige Bereiche Seitenkopf auf Seiten, die den Berichtskopf und/oder -fuß enthalten, nicht angezeigt werden sollen. Um die Berichtseigenschaften zu bearbeiten, wählen Sie im Kombinationsfeld oben im Eigenschaftsfenster den Eintrag **Bericht** aus.



## Gruppenkopf und -fuß

Wenn Ihr Bericht Gruppierungen enthält, können Sie für diese ebenfalls jeweils einen Kopf- und einen Fußbereich festlegen. Mehr zu diesem Thema erfahren Sie unter →*[basics]* **Gruppierungen anlegen**.

# Beschriftungen von Tabellenfeldern einstellen

Früher oder später werden Sie Formulare oder Berichte auf Basis einer Tabelle als Datenherkunft erstellen. Dabei werden Sie die Felder im Formular- oder Berichtsentwurf am einfachsten anlegen, indem Sie diese aus der Feldliste in den entsprechenden Bereich der Entwurfsansicht des Zielobjekts ziehen. Access legt dann für die meisten Steuerelementtypen automatisch Beschriftungsfelder an. Als Beschriftung verwendet es dabei standardmäßig die Bezeichnung des Feldes der Datenherkunft, an welches das Steuerelement gebunden ist. Um den gängigen Konventionen zu genügen, werden diese beispielsweise keine Umlaute, kein scharfes S, keine Leerzeichen und keine Trennstriche enthalten, also etwa **Aenderungsdatum**, **Aufmass** oder **AnzahlGebinde** heißen.

Sie müssten den Text der Bezeichnungsfelder dann händisch ändern. Mit ein wenig Vorarbeit sparen Sie hier Zeit, zumindest wenn mehr als ein Formular an die entsprechende Tabelle gebunden wird. Dazu tragen Sie einfach die Bezeichnung, die in Formularen und Berichten auftauchen soll, für die Eigenschaft **Beschriftung** der einzelnen Felder der Tabelle ein.

# Beziehungen-Fenster

Das Beziehungen-Fenster zeigt die Beziehungen zwischen den aktuell angezeigten Tabellen an und kann auch zum Verwalten dieser Beziehungen verwendet werden – also zum Anlegen neuer Beziehungen, zum Ändern ihrer Eigenschaften oder zum Löschen vorhandener Beziehungen.

Das Beziehungen-Fenster öffnen Sie in Abhängigkeit von der Access-Version wie folgt:

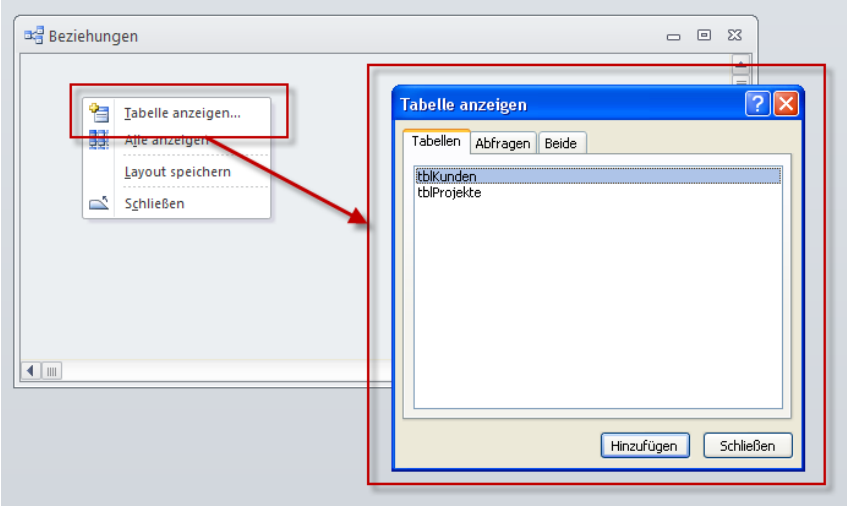
**2003 und älter:** Wählen Sie den Menüeintrag **Extras|Beziehungen**.

**2007 und jünger: Klicken Sie im Ribbon auf den Eintrag Datenbanktools|Beziehungen.**

## Tabelle hinzufügen

Es gibt mehrere Möglichkeiten zum Hinzufügen von Tabellen zum Beziehungen-Fenster.

- Um eine Tabelle über den dafür vorgesehenen Dialog hinzuzufügen, klicken Sie mit der rechten Maustaste in das Beziehungen-Fenster und wählen den Kontextmenüeintrag **Tabellen anzeigen...** aus. Im nun erscheinenden Dialog **Tabelle anzeigen** klicken Sie entweder doppelt auf den Namen der Tabelle, die Sie zum Beziehungen-Fenster hinzufügen möchten, oder Sie markieren eine oder mehrere Tabellen und klicken anschließend auf die Schaltfläche **Hinzufügen**.



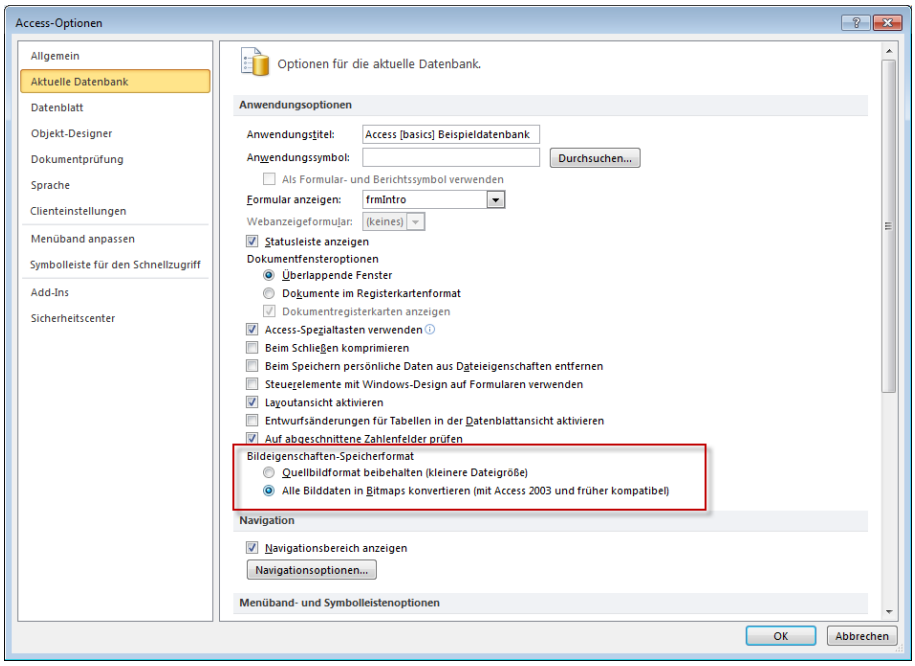

- Alternativ können Sie dies je nach Access-Version gleich vom Datenbankfenster (bis Access 2003) oder vom Navigationsbereich (ab Access 2007) erledigen. Dazu zeigen Sie ebenfalls das Beziehungen-Fenster an, öffnen aber nicht den **Tabelle anzeigen**-Dialog. Stattdessen machen Sie im Datenbankfenster beziehungsweise im Navigationsbereich die Tabellen sichtbar, die Sie zum Beziehungenfenster hinzufügen möchten, und ziehen diese einfach per Drag and Drop in das Beziehungen-Fenster.

## Beziehung hinzufügen****

# Bilder unter Access 2007 und jünger

Ab Access 2007 gibt es wesentlich bessere Möglichkeiten für den Umgang und den Einsatz von Bildern. Das Anlage-Feld kann zum Speichern von Bildern genutzt werden, und auch der Zugang zur Benutzerfläche ist wesentlich einfacher geworden: So lassen sich Bilder etwa auf Schaltflächen platzieren und statt oder neben einer Beschriftung verwenden.

Damit dies reibungslos klappt, müssen Sie eine spezielle Einstellung in den Access-Optionen für die aktuelle Datenbank vornehmen. Diese Option heißt **Bildeigenschaften-Speicherformat** und kann die beiden Optionen **Quellbildformat beibehalten (kleinere Dateigröße)** oder **Alle Bilddaten in Bitmaps konvertieren (mit Access 2003 und früher kompatibel)**:




Die zweite Option ist standardmäßig eingestellt und entspricht dem Verhalten unter älteren Access-Versionen. Dort konnten Sie Bilddateien etwa in einem OLE-Feld einer Tabelle speichern, aber es wurde dabei in ein proprietäres Format gebracht. Der größte Nachteil dieses Formats war, dass dies die Bildgröße stark wachsen ließ.

Aus diesem Grund und weil diese Option nicht mit wichtigen Bilddateitypen wie **.png** zusammenarbeitet sollten Sie hier die erste Option auswählen. Damit wächst die Datenbank beim Speichern von Bilddateien nur um die tatsächliche Größe der Bilddatei und Sie können alle gängigen Dateitypen verwenden.

# Datum/Zeit-Feld anlegen

Ein Datumsfeld legen Sie an, indem Sie in die Spalte **Feldname** einer leeren Zeile in der Entwurfsansicht einer Tabelle den Feldnamen eintragen und den Datentyp auf **Datum/Zeit** festlegen.

Desweiteren gibt es nicht viel zu beachten, außer dass es hilfreich sein kann, die Art der internen Speicherung von Daten zu kennen. Access speichert Daten nämlich als Double-Felder, also in einem Zahlenformat mit Vor- und Nachkommastellen. Dabei werden das Datum im Vorkommabereich und die Zeit im Nachkommabereich untergebracht.

Das erste mögliche Datum ist der **31.12.1899**, was dem Double-Wert **1** entspricht. Eine Ganzzahl entspricht einem Tag. Der **1.1.2010** hat somit den Zahlenwert **40.179**. Bei der Uhrzeit ist das anders: Dort entspricht der Zahlenwert **0** der Zeit **0:00 Uhr**. Der Zahlenwert **1** entspricht der gleich Uhrzeit, allerdings einen Tag später. Das scheint logisch: Wenn die Addition von **1** einem Tag entspricht, sind das 24 Stunden. Alles, was sich hinter dem Komma befindet, sind also Bruchteile eines Tages. **0,5** entspricht **12:00 Uhr**, **0,25** entspricht **6:00 Uhr**.

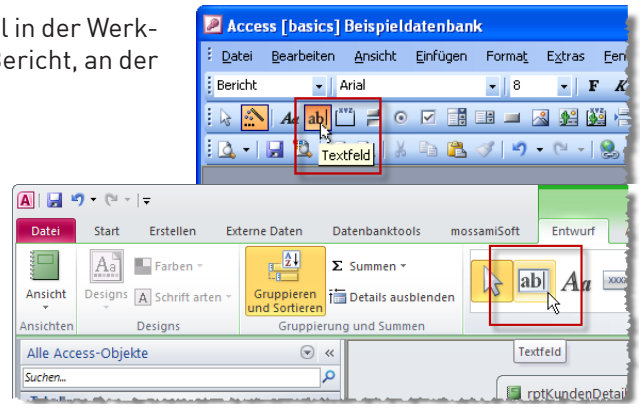
# Datumsfeld im Bericht anlegen

Ein Datumsfeld in einem Bericht ist nichts weiter als ein Textfeld, dessen Steuerelementinhalt eine Funktion zum Ermitteln des aktuellen Datums enthält.

Solch ein Feld findet meist im Seitenfußbereich eines Berichts Platz, Sie können es aber auch in anderen Bereichen unterbringen.

Fügen Sie dem Bericht zunächst ein Textfeld hinzu:

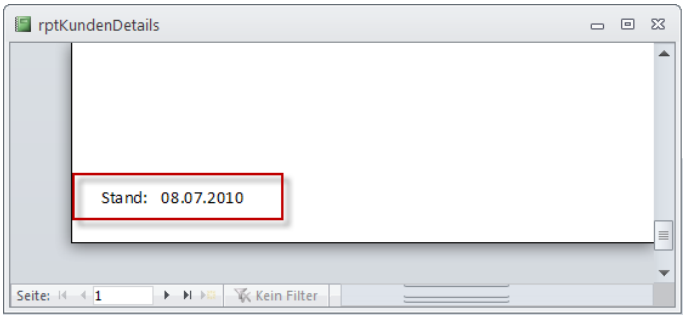
**2003 und älter:** Klicken Sie erst auf das Textfeld-Symbol in der Werkzeug-Symbolleiste und anschließend auf die Stelle im Bericht, an der Sie das Textfeld einfügen möchten.




**2007 und jünger: Betätigen Sie den Ribbon-Eintrag Entwurf|Steuerelemente|Textfeld und klicken Sie dann auf die gewünschte Stelle im Berichtsentwurf.**

Anschließend löschen Sie, wenn Sie nur das Datum anzeigen möchten, das mit dem Textfeld automatisch angelegte Bezeichnungsfeld. Vielleicht möchten Sie das Bezeichnungsfeld aber auch für einen Text wie **Stand:** oder dergleichen nutzen – dann behalten Sie deses natürlich im Bericht und stellen die Eigenschaft **Beschriftung** entsprechend ein.

Anschließend weisen Sie der Eigenschaft **Steuerelementinhalt** des Textfeldes den Wert **=Datum()** zu und stellen außerdem die Eigenschaft **Name** auf einen Wert wie **txtDatum** ein. Richten Sie Bezeichnungsfeld und Textfeld wie gewünscht aus und schieben Sie dabei das Textfeld soweit an das Bezeichnungsfeld heran, dass beide einen scheinbar zusammenhängenden Text wie in der Abbildung rechts ergeben.



# Eingabeformat-Eigenschaft

Die folgende Tabelle beschreibt alle Platzhalter für die Eigenschaft **Eingabeformat** von Feldern in Tabellen.

| Platzhalter | Beschreibung |
|---|---|
| 0 | Ziffern 0 bis 9. Vorzeichen (+, -) sind nicht erlaubt. Die Eingabe ist erforderlich. |
| 9 | Ziffern oder Leerzeichen. Vorzeichen sind nicht erlaubt. Die Eingabe ist optional. |
| # | Ziffern mit und ohne Vorzeichen sowie Leerzeichen, die Access jedoch beim Speichern wieder entfernt. |
| L | Buchstaben von a bis Z. Die Eingabe ist erforderlich. |
| ? | Buchstaben von a bis Z. Die Eingabe ist optional. |
| A | Buchstaben oder Ziffern. Die Eingabe ist erforderlich. |
| a | Buchstaben oder Ziffern. Die Eingabe ist optional. |
| & | Beliebiges Zeichen oder Leerzeichen. Die Eingabe ist erforderlich. |
| C | Beliebiges Zeichen oder Leerzeichen. Die Eingabe ist optional. |
| . , :; – / | Platzhalter für Dezimal-, Tausender-, Datums- und Zeit-Trennzeichen aus den Ländereinstellungen der Systemsteuerung. |
| < | Wandelt alle Buchstaben in Kleinbuchstaben um. |
| > | Wandelt alle Buchstaben in Großbuchstaben um. |
| ! | Zeigt das Eingabeformat nicht von links nach rechts, sondern von rechts nach links an. |
| \ | Das folgende Zeichen wird als Literal angezeigt. |

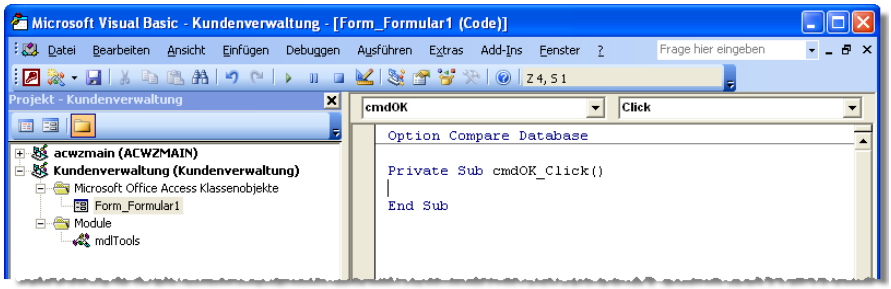
# Ereignisprozedur hinzufügen

Um einem Formular, einem Bericht oder einem Steuerelement eine Ereignisprozedur hinzuzufügen, brauchen Sie zunächst Zugriff auf das Eigenschaftsfenster (Schaltfläche **F4**). Markieren Sie dann das betroffene Element. In einem Formular kann dies etwa das Formular selbst (Klick auf den grauen Hintergrundbereich oder Auswahl des Eintrags **Formular** im Kombinationsfeld oben im Eigenschaftsfenster), einer der Formularbereiche (Klick auf die Leiste mit der Bereichsbezeichnung oder Auswahl des Bereichs im Eigenschaftsfenster) oder eines der enthaltenen Steuerelemente sein.



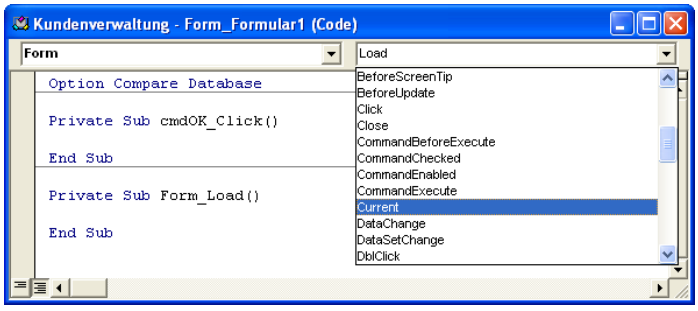

Wechseln Sie dann im Eigenschaftsfenster zur Registerseite **Ereignis**. Wählen Sie den Eintrag **[Ereignisprozedur]** für die gewünschte Ereigniseigenschaft aus und klicken Sie auf die Schaltfläche rechts neben dem Eigenschaftsfeld.

Es erscheint der VBA-Editor, der gleich das Klassenmodul für das betroffene Formular oder den Bericht anzeigt und den leeren Prozedurrumpf zum Einfügen der notwendigen Anweisungen anbietet. Der folgende Screenshot zeigt im linken Bereich den Projekt-Explorer an, den Sie mit **Strg + R** einblenden.




Die Klassenmodule mit den Ereignisprozeduren für die einzelnen Formulare und Berichte finden Sie dort unter dem Eintrag **Microsoft Office Access Klassenobjekte**. Ein Formular-Klassenobjekt erhält als Namen immer das Präfix **Form_** plus dem Formularnamen, hier **Formular1**. Klicken Sie später auf einen solchen Eintrag, um zum Klassenmodul eines bestimmten Formulars oder Berichts zu gelangen (der Namen der Klassenmodule von Berichten beginnt mit **Report_**).

Alternativ zum Anlegen einer Ereignisprozedur über das Eigenschaftsfensters des jeweiligen Objekts können Sie dies auch direkt im VBA-Editor erledigen (möglicherweise besitzt das Formular oder der Bericht aber noch gar kein Klassenmodul – dies holen Sie dann durch Einstellen der Eigenschaft **Enthält Modul** des jeweiligen Objekts nach). Dazu öffnen Sie zunächst das Klassenmodul des gewünschten Objekts. Wählen Sie dann im linken Kombinationsfeld des Codefensters den Bereich oder das Steuerelement aus, welches das Ereignis auslösen soll. Der VBA legt danach automatisch das Standardereignis für dieses Element an. Wenn Sie ein anderes anlegen wollen, können Sie dieses wieder entfernen und im rechten Kombinationsfeld das richtige Ereignis auswählen. In folgendem Screenshot wird so etwa das Ereignis **Beim Anzeigen** eines Formulars angelegt, dass bei jedem Datensatzwechsel ausgelöst wird.




Für einwandfreie Funktion einer Ereignisprozedur für Formulare, Berichte oder Steuerelemente muss die entsprechende Eigenschaft im Eigenschaftsfenster den Eintrag **[Ereignisprozedur]** enthalten und die Ereignisprozedur mit entsprechendem Prozedurkopf im betroffenen Klassenmodul vorliegen.

Eine so angelegte Ereignisprozedur können Sie nun mit den benötigten Anweisungen füllen.

# Felddatentypen

Die folgende Tabelle enthält die Felddatentypen von Access:

| Datentyp | Beschreibung | Größe |
|---|---|---|
| AutoWert | speichert eindeutige, fortlaufende Zahlen, die Access selbst zuordnet | 4 Byte |
| Text | speichert Text, Zahlen und Sonderzeichen | 1–255 Zeichen |
| Memo | speichert lange Texte mit Zahlen und Sonderzeichen | 1–64.000 Zeichen |
| Zahl | speichert numerische Daten | 1, 2, 4 oder 8 Byte je nach Zahlentyp |
| Datum/Zeit | speichert Datums- und Zeitangaben | 8 Byte |
| Währung | speichert Zahlen mit vier Nachkommastellen und Währungszeichen (DM) | 8 Byte |
| Ja/Nein | speichert logische Werte wie Ja/Nein, True/False oder Ein/Aus | 1 Bit |
| OLE-Objekt | speichert OLE-Objekte aus anderen Windows-Anwendungen | max. 1 GByte |
| Hyperlink | speichert E-Mail-, Internet-Adressen oder Positionen in der Datenbank, die per Mausklick angesprungen werden | max. 8.192 Zeichen |
| Nachschlage-Assistent | übernimmt Werte aus einer anderen Tabelle | Größe des Primärschlüssels der anderen Tabelle |
| Anlage | speichert Dateien | max. 1 GByte |
| Berechnet | speichert berechnete Ergebnisse | |

# Format-Eigenschaft

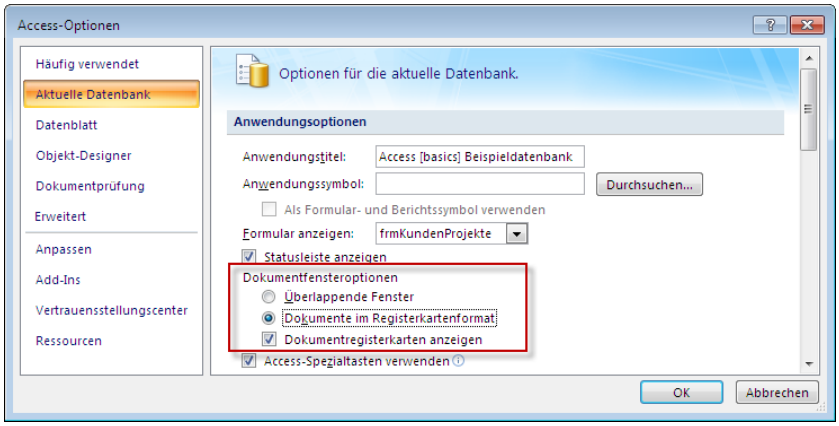
Die folgende Tabelle zeigt die Format-Eigenschaften der verschiedenen Datentypen:

| Datentyp | Standardformat | Beschreibung | Beispiel |
|---|---|---|---|
| Datum/ Zeit | Standarddatum | Je nach Feldinhalt zeigt Access nur das Datum, nur die Zeit oder beides an | 01.01.2011 oder 15:04:30 oder 01.01.2011 15:04:30 |
| | Datum, lang | Langformat aus den Ländereinstellungen der Systemsteuerung | Samstag, 1. Januar 2011 |
| | Datum, mittel | Zeigt die ersten drei Buchstaben des Monats an | 01. Jan. 11 |
| | Datum, kurz | Kurzformat aus den Ländereinstellungen der Systemsteuerung. Daten vom 1.1.00 bis 31.12.29 gehören zum 21. Jahrhundert. Daten vom 1.1.30 bis 31.12.99 zählt Access zum 20. Jahrhundert. | 01.01.11 |
| | Zeit, lang | Langformat aus den Ländereinstellungen der Systemsteuerung | 15:04:30 |
| | Zeit, 12 Std | Drei Uhr vor- und nachmittags besitzen dasselbe Format | 03:04 |
| | Zeit, 24 Std | Unterteilt die Uhrzeit in 24 Stunden, sodass Vor- und Nachmittag erkennbar sind | 15:04 |
| Ja/Nein | Ja/Nein | Erlaubt die Werte Ja oder Nein | Ja |
| | Wahr/Falsch | Erlaubt die Werte Wahr oder Falsch | Wahr |
| | Ein/Aus | Erlaubt die Werte Ein oder Aus | Ein |
| Zahl | Allgemeine Zahl | Zeigt die Zahl wie eingegeben an | 123,45 |
| | Währung | Zeigt den Punkt als Tausender-Trennzeichen, zwei Nachkommastellen und das in den Ländereinstellungen festgelegte Währungszeichen an. Negative Werte stehen in Klammern | 123,45 € |
| | Euro | Zeigt den Punkt als Tausender-Trennzeichen, zwei Nachkommastellen und das Euro-Zeichen an. Negative Werte stehen in Klammern | |
| | Festkommazahl | Zeigt eine Zahl standardmäßig mit zwei Nachkommastellen an | 123,45 |
| | Standardzahl | Zeigt eine Zahl mit Tausender-Trennzeichen und standardmäßig zwei Nachkommastellen an. | 1.234,50 |
| | Prozentzahl | Multipliziert den Wert mit 100, zeigt zwei Nachkommastellen an und fügt ein Prozentzeichen (%) hinzu. So wird der Wert 0,35 zu 35%, während die Zahl 35 zu 3500% wird. | 35% |
| | Exponentialzahl | Schreibweise für Exponentialzahlen. Das E markiert die Zehnerpotenz. | 3,5E+3 |

# Formular maximiert anzeigen

Zum maximierten Öffnen von Formularen gibt es ab Access 2007 neue Möglichkeiten. Dieser Artikel stellt die verschiedenen Varianten vor.

Vielleicht soll das Formular auch gleich maximiert angezeigt werden. Unter Access ab Version 2007 ist dies ganz einfach. Sie brauchen einfach nur die Option **Dokumentfensteroptionen** auf den Wert **Dokumente im Registerkartenformat** einzustellen. Gegebenenfalls deaktivieren Sie mit der Option **Dokumentregisterkarten anzeigen** noch die Registerlaschen. Allerdings zeigt Access dann alle Formulare maximiert an.




**Formulare maximieren dank Registerkartenansicht**

Wenn Sie das nicht wollen oder ohnehin Access 2003 oder älter verwenden, können Sie ein Formular gleich nach dem Öffnen maximieren. Dazu legen Sie eine Ereignisprozedur für das Ereignis Beim Aktivieren an, die lediglich den folgenden Befehl enthält:

```
DoCmd.Maximize
```

# Formularansichten

Genau wie beiden Tabellen gibt es auch bei den Formularen verschiedene Ansichten. Bis zum Erscheinen von Access 2007 gab es die folgenden Ansichten:

- **Entwurfsansicht**: Nur für den Entwickler, erlaubt das Editieren des Formulars und der Steuerelemente sowie das Einstellen der Eigenschaften.
- **Datenblattansicht**: Entspricht im Wesentlichen der Datenblattansicht für Tabellen, wobei allerdings nur die Felder angezeigt werden, die in der Entwurfsansicht in Form eines Steuerelements in den Detailbereich des Formulars gezogen wurden. In der Datenblattansicht können nur Textfelder, Kombinationsfelder und Kontrollkästchen angezeigt werden. Es werden nur Steuerelemente aus dem Detailbereich angezeigt.
- **Endlosansicht**: Ähnlich wie die Datenblattansicht. Es können mehrere Datensätze untereinander angezeigt werden, allerdings ist das Layout der Steuerelemente anpassbar. Bereiche wie der Formularkopf und der Formularfuß können zum Anzeigen einer Überschrift oder von Schaltflächen etwa zum Schließen oder Abbrechen verwendet werden.
- **Formularansicht**: Es wird nur jeweils ein Datensatz angezeigt. Das Layout der Steuerelemente kann wie in der Endlosansicht frei gestaltet werden, auch Formularkopf und Formularfuß können genutzt werden.

Es gibt seit Access 2002 auch noch eine **PivotTable-Ansicht** und eine **PivotChart-Ansicht** und seit Access 2007 eine **Layoutansicht**. Diese wollen wir jedoch hier zunächst nicht weiter berücksichtigen.

Beim Öffnen eines Formulars per Doppelklick auf den Formularnamen im Datenbankfenster (Access 2003 und älter) beziehungsweise im Navigationsbereich (Access 2007 und jünger) wird das Formular in der Ansicht geöffnet, die in seiner Eigenschaft **Standardansicht** festgelegt ist. Um ein Formular in der Entwurfsansicht zu öffnen, klicken Sie mit der rechten Maustaste auf den Formularnamen und wählen den Kontextmenüeintrag **Entwurfsansicht** aus.

Tastenkombinationen zum Wechseln zwischen den verschiedenen Formularansichten finden Sie unter →*[basics]* **Tastenkombinationen**.

# Formularbereiche

Formulare haben verschiedene von oben nach unten angeordnete Bereiche, die je nach der verwendeten Ansicht angezeigt werden:

- **Detailbereich**: Dieser Bereich enthält in der Regel die eigentlichen Daten. Er wird in der Formularansicht einmal angezeigt. In der Endlosansicht wird er für so viele Datensätze angezeigt, wie ins Formular passen (abzüglich Formularkopf und -fuß). In der Datenblattansicht erscheinen ausschließlich die im Detailbereich angelegten gebundenen Steuerelemente. (die Betonung liegt auf »gebunden«!) Standardmäßig ist dies der einzige Bereich, der beim Erstellen eines neuen Formulars angelegt wird.
- **Formularkopf**: Der Formularkopf wird einmalig zu Beginn des Formulars angezeigt. Er kann beispielsweise ein Firmenlogo, allgemeine Informationen oder Steuerelemente enthalten – beispielsweise zum Filtern der im Detailbereich angezeigten Daten. Dieser Bereich erscheint nicht in der Datenblattansicht.
- **Formularfuß**: Der Formularfuß erscheint einmalig am unteren Ende des Formulars. Dort bringt man gern Schaltflächen wie **OK** oder **Abbrechen** unter, aber es können dort auch Steuerelemente mit Berechnungsergebnissen bezogen auf die Daten des Detailbereichs Platz finden. Dieser Bereich erscheint nicht in der Datenblattansicht.
- **Seitenkopf** und **Seitenfuß**: Obwohl es zum Ausdrucken von Daten eigentlich den Objekttyp **Bericht** gibt, hat Microsoft auch die Möglichkeit zum Drucken von Berichten offen gehalten. Falls Sie dies tun, können Sie den Seitenkopf und den Seitenfuß nutzen, deren Inhalte auf jeder ausgedruckten Seite je einmal am oberen und am unteren Seitenende erscheinen.

Diese Information dient nur der Vollständigkeit. Formulare druckt man nicht aus, also brauchen Sie auch keinen Seitenkopf und keinen Seitenfuß. Und wehe, doch!

**Bereiche ein- und ausblenden**

Wie erwähnt, zeigt Access in neuen Formularen nur den Detailbereich an.

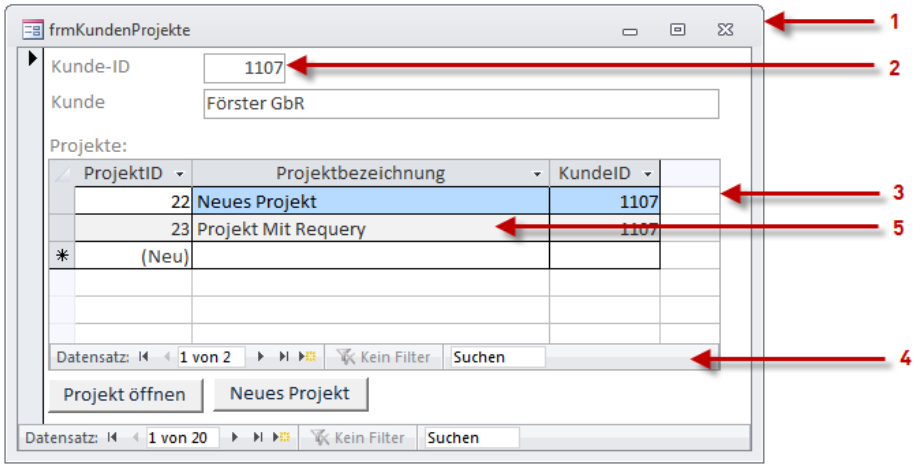
# Formularelemente referenzieren

Formulare besitzen Steuerelemente wie Textfelder, Kombinationsfelder und Schaltflächen, die direkt auf dem Formular platziert werden. Es gibt aber auch Unterformulare, und diese enthalten wiederum Steuerelemente. All diese Elemente werden Sie früher oder später per VBA referenzieren wollen – beispielsweise, um den Inhalt eines Listenfeldes nach dem Primärschlüsselfeld des im Formular angezeigten Datensatzes filtern zu können. Dieser Beitrag zeigt, wie Sie auf die Elemente von Formularen zugreifen.

## Formulare referenzieren

Zunächst einmal müssen Sie das Formular selbst referenzieren. Die folgenden Anweisungen geben der Einfachheit halber den Namen des Formulars aus. Das Formular muss geöffnet sein, um per VBA erreichbar zu sein.

Im Beispiel verwenden wir das Formular **frmKundenProjekte**. Es enthält ein Hauptformular und ein Unterformular, beide mit einigen Steuerelementen gefüllt. Im Folgenden finden Sie die Verweise auf die in der folgenden Abbildung mit Pfeilen markierten Elemente:




Auf das Formular selbst (1) greifen Sie per VBA über die **Forms**-Auflistung zu. Auflistungen enthalten immer eine **Item**-Eigenschaft, über die man standardmäßig auf ihre Elemente zugreift. Dazu gibt man entweder den Namen des Objekts oder seine Ordinalzahl an und gibt den Wert mit **Debug.Print** etwa im Direktfenster aus (natürlich können Sie statt auf **Name** auf beliebige andere Eigenschaften zugreifen):

```
Debug.Print Forms.Item("frmKundenProjekte").Name
Debug.Print Forms.Item(0).Name
```

**Item** ist aber die Standardeigenschaft des **Forms**-Objekts, weshalb Sie es auch weglassen und folgendermaßen auf ein Formular zugreifen können:

```
Debug.Print Forms("frmKundenProjekte").Name
```

In der Regel werden Sie jedoch über den Namen auf ein Formular zugreifen. Es gibt noch eine vereinfachte Schreibweise für **Forms("frmKundenProjekte")**:

```
Debug.Print Forms!frmKundenProjekte.Name
```

Nun greifen wir auf ein Steuerelement im Formular zu (2). Wir möchten beispielsweise den im Feld **KundeID** enthaltenen Wert ermitteln. Dieser wird im gebundenen Textfeld **txtKundeID** angezeigt. Wir beginnen wieder bei der ausführlichsten Variante:

```
Debug.Print Forms.Item("frmKundenProjekte").Controls.Item("txtKundeID").Value
```

Das Element **frmKundenProjekte** der **Forms**-Auflistung enthält also eine weitere Auflistung namens **Controls**, die wiederum den Zugriff auf alle im Formular enthaltenen Steuerelemente erlaubt – die wiederum über eine **Item**-Eigenschaft referenziert werden können. Und den angezeigten Wert liefert schließlich die Value-Eigenschaft. Wir vereinfachen schrittweise:

- **Item** entfernen:

```
Debug.Print Forms.("frmKundenProjekte").Controls.("txtKundeID").Value
```

- **Controls** entfernen

```
Debug.Print Forms("frmKundenProjekte")("txtKundeID").Value
```

- Klammern und Anführungszeichen entfernen und Ausrufezeichen verwenden

```
Debug.Print Forms!frmKundenProjekte!txtKundeID.Value
```

- Und **Value** als Standardeigenschaft des Steuerelements entfällt auch noch:

```
Debug.Print Forms!frmKundenProjekte!txtKundeID
```

Auf die gleiche Weise wollen wir nun noch auf ein Steuerelement im Unterformular zugreifen (5). Dazu müssen Sie bedenken, dass das Unterformularsteuerelement und das darin enthaltene Formular zwei verschiedene Elemente sind.

Zunächst wollen wir den Namen des Unterformularsteuerelements erfahren (3):

```
Debug.Print Forms.Item("frmKundenProjekte").Controls("sfm").Name
```

Den Namen des im Unterformularsteuerelement enthaltenen Formulars (4) erhalten Sie so:

```
Debug.Print Forms.Item("frmKundenProjekte").Controls("sfm").Form.Name
```

Schließlich greifen wir auf das im Unterformular enthaltene Steuerelement zu (5) und geben seinen Wert aus:

```
Debug.Print Forms.Item("frmKundenProjekte").Controls("sfm").Form.Controls("txtProjektID").Value
```

In der einfachsten Schreibweise lautet der Ausdruck so:

```
Debug.Print Forms!frmKundenProjekte!sfm.Form!txtProjektID
```

**frmKundenProjekte**, **sfm** und **txtProjektID** sind jeweils benutzerdefinierte Elemente. Sie können mit dem Ausrufezeichen adressiert werden. **Form** hingegen ist eine Eigenschaft des durch **Forms!frmKundenProjekte!sfm** referenzierten Unterformularsteuerelements und muss daher mit vorangestelltem Punkt eingegeben werden.

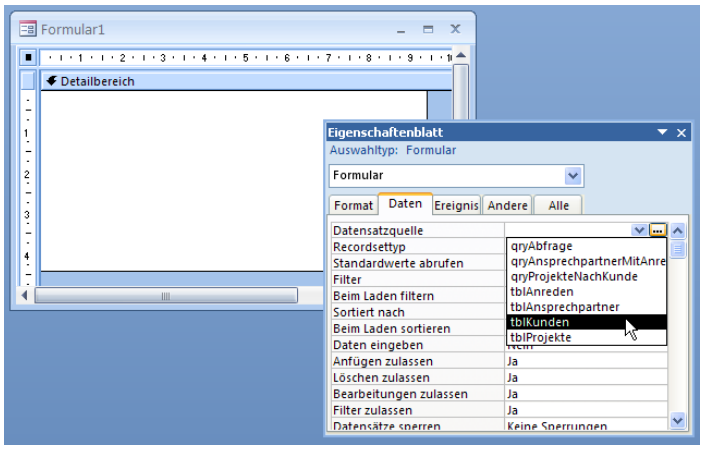
# Gebundenes Formular anlegen

Ein gebundenes Formular ist ein Formular, das einen Datensatz einer Tabelle oder einer Abfrage übersichtlich anzeigt (Detailformular) oder eine Liste der Datensätze der zugrunde liegenden Datenherkunft liefert (Übersichtsformular). Grundlage für beide ist ein neues gebundenes Formular, dass Sie wie folgt anlegen:

**2003 und älter:** Wählen Sie den Menüeintrag **Einfügen|Formular** oder wechseln Sie im Datenbankfenster zur Registerseite **Formulare** und klicken dort auf die Schaltfläche **Neu**. Wählen Sie dann im Dialog **Neues Formular** die gewünschte Herkunftstabelle oder -abfrage aus und klicken Sie auf **OK**. Wenn das Datenbankfenster gerade die Herkunftstabelle oder -abfrage anzeigt und Sie dann den Menüeintrag **Einfügen|Formular** auswählen, ist dieser Eintrag im Dialog **Neues Formular** vorausgewählt. In jedem Fall zeigt Access im Anschluss ein neues, leeres Formular in der Entwurfsansicht an.

**2007 und jünger:** Klicken Sie einfach auf den Ribbon-Eintrag **Erstellen|Formularentwurf**. Access zeigt dann ein neues, leeres Formular in der Entwurfsansicht an.

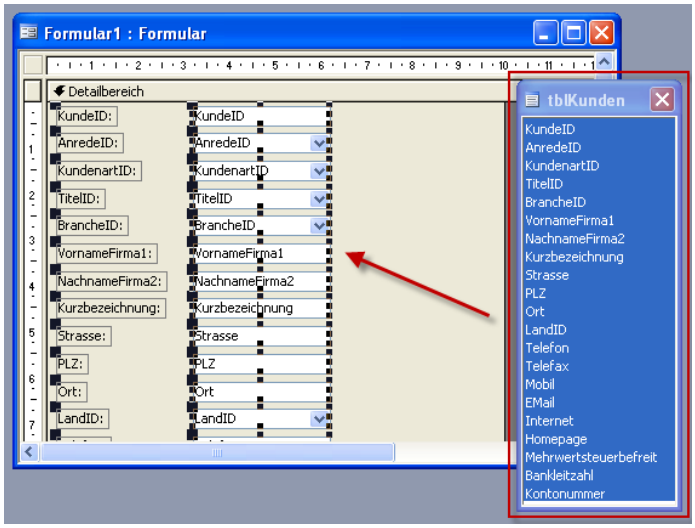
## Datenherkunft festlegen




Je nachdem, wie Sie das Formular angelegt haben, wurde seine Datenherkunft bereits festgelegt. Für alle Fälle zeigen wir nun, wie dies manuell funktioniert. Dazu zeigen Sie das Formular in der Entwurfsansicht an und aktivieren das Eigenschaftsfenster (entweder mit der Schaltfläche **F4** oder durch einen Doppelklick auf den Formularhintergrund). Wechseln Sie zur Registerseite **Daten** und wählen Sie für die Eigenschaft **Datenherkunft** die gewünschte Tabelle oder Abfrage aus.

## Felder in den Entwurf ziehen

Ziehen Sie nun die benötigten Felder in die Entwurfsansicht, und zwar in den Detailbereich. Je nach weiterer Verfahrensweise gelangen einige Steuerelemente in die übrigen Bereiche, fürs erste reicht dies jedoch. Dazu aktivieren Sie zunächst die Feldliste:




**2003 und älter:** Menüeintrag **Ansicht|Feldliste** – ist erst nach der Auswahl einer Datenherkunft verfügbar.

**2007 und jünger**: Betätigen Sie den Ribboneintrag **Entwurf|Tools|Vorhandene Felder einfügen**, markieren Sie die benötigten Felder und ziehen Sie diese per Drag and Drop in den Detailbereich.

### Formular speichern

Speichern Sie das Formular dann – am einfachsten gelingt dies über die Tastenkombination **Strg + S**. Geben Sie hier den geplanten Formularnamen an. Das Formular erscheint nun unter diesem Namen in der Liste der Formular im Datenbankfenster (Access 2003 und älter) beziehungsweise im Navigationsbereich (Access 2007 und jünger).

# Gebundene Steuerelemente zu Formularen und Berichten hinzufügen

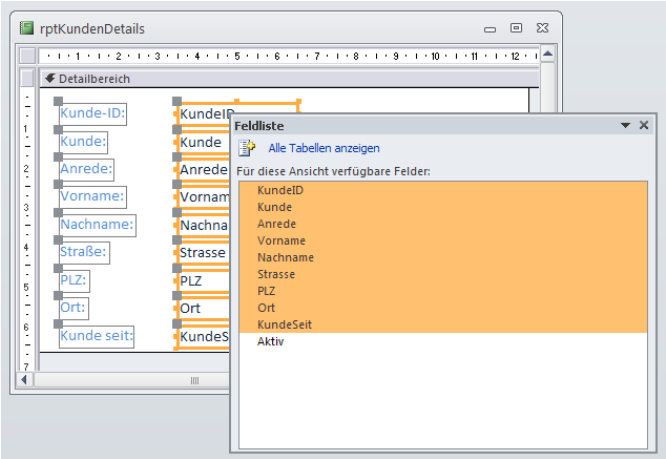
Bevor ein Formular oder ein Bericht überhaupt irgendetwas anzeigt, müssen Sie diesem entsprechende Steuerelemente hinzufügen. Dies gelingt auf verschiedene Arten, die wir in den folgenden Abschnitten erläutern werden.

## Gebundene Steuerelemente anlegen

In den meisten Fällen werden Sie einem Formular oder Bericht gebundene Steuerelemente hinzufügen. Gebundene Steuerelemente sind Steuerelement wie Textfelder, Kombinationsfelder, Listenfelder oder Kontrollkästchen, die Daten aus einer Tabelle oder Abfrage anzeigen, die Sie dem Formular oder Bericht zuvor als Datenherkunft zugewiesen haben.

Diese Steuerelemente unterscheiden sich von ungebundenen Steuerelementen allein dadurch, dass ihre Eigenschaft **Steuerelementinhalt** mit dem Namen eines der Felder der Datenherkunft versehen wird.

Die einfachste Möglichkeit zum Anlegen solcher Steuerelemente besteht darin, sie einfach aus der Feldliste in den gewünschten Bereich der Entwurfsansicht zu ziehen.




Die Feldliste machen Sie je nach Access-Version auf eine bestimmte Art und Weise sichtbar. Voraussetzung ist, dass Sie zuvor zur Entwurfsansicht des Formulars oder Berichts umgeschaltet haben:

**2003 und älter:** Betätigen Sie den Menübefehl **Ansicht|Feldliste**. Achtung: Diese Befehl ist nur aktiviert, wenn Sie der Eigenschaft Datenherkunft des Formulars oder Berichts eine Tabelle oder Abfrage mit den gewünschten Felder zu gewiesen haben.

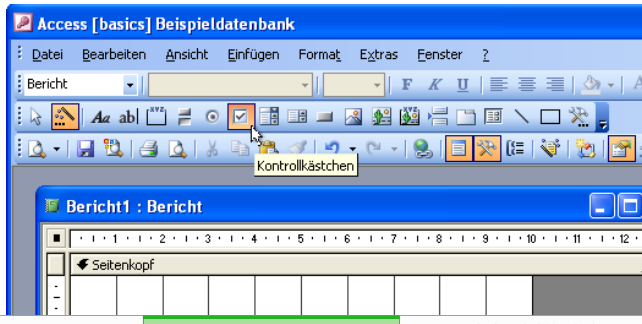
**2007 und jünger**: Wählen Sie den Ribbon-Eintrag **Entwurf|Tools|Vorhandene Felder hinzufügen** aus. Diese Eintrag ist immer vorhanden, auch wenn Sie noch keine Datenherkunft für das Formular oder den Bericht ausgewählt haben. Die Feldliste zeigt dann allerdings keine Felder, sondern lediglich den Eintrag **Alle Tabellen anzeigen** an. Sie können dort aus allen Tabellen die gewünschten Felder auswählen; Access stellt dann eine Abfrage als Datenherkunft zusammen, die alle zum Entwurf des Formulars oder Berichts hinzugefügten Felder beinhaltet.

Sie können auch auf diese Weise auch gleich mehrere Steuerelemente auf einen Rutsch zur Entwurfsansicht hinzufügen: Dazu markieren Sie einfach alle gewünschten Elemente der Feldliste und ziehen diese in den Entwurf des Formulars oder Berichts. Halten Sie die Umschalttaste gedrückt und markieren Sie nacheinander mit der Maus das erste und das letzte Elemente mehrerer aufeinanderfolgender Elemente oder klicken Sie die gewünschten Elemente einfach bei gedrückter **Strg**-Taste zusammen.

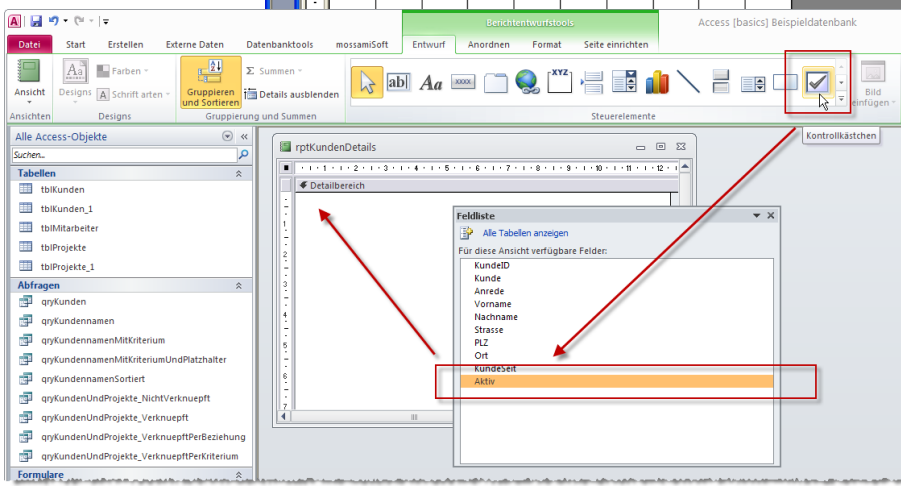
## Steuerelementtyp festlegen

Wenn Sie ein Feld aus der Feldliste in den Entwurf eines Formulars oder Berichts ziehen, legt Access in der Regel ein Textfeld an, dass an das zugrunde liegende Tabellen- oder Abfragefeld gebunden wird (dazu wird die Eigenschaft Steuerelementinhalt auf den entsprechenden Feldnamen eingestellt). Es kann jedoch auch vorkommen, dass Access automatisch ein anderes Steuerelement anlegt – beispielsweise, wenn Sie dieses Steuerelement im Tabellenentwurf als Nachschlagefeld definiert haben (→*[basics]* **Nachschlagefeld für eine 1:n-Beziehung einrichten**). Access legt dann automatisch ein Kombinationsfeld an. Ein anderes Beispiel für alternative Steuerelemente beim Anlegen aus der Feldliste heraus sind **Ja/Nein**-Felder (→*[basics]* **Ja/Nein-Feld anlegen**). Hier legt Access automatisch ein Kontrollkästchen an, dass an das betreffende Feld der Datenherkunft gebunden ist.

Sie können beim Anlegen eines gebundenen Steuerelements aber auch selbst festlegen, welchen Datentyp dieses haben soll. Das Formular oder der Bericht sollten aktuell in der Entwurfsansicht angezeigt werden:




**2003 und älter:** Aktivieren Sie über den Menübefehl **Ansicht|Toolbox** die Werkzeugleiste, die Sie auch in der Abbildung rechts sehen. Klicken Sie dort auf das gewünschte Steuerelement, beispielsweise das Kontrollkästchen. Ziehen Sie anschließend wie oben beschrieben das betreffende Feld aus der Feldliste in den Entwurf des Formulars oder Berichts. Es wird nun ein Steuerelement mit dem gewünschten Typ und mit dem entsprechenden Feld als Steuerelementinhalt angelegt.




**2007 und jünger: In neueren Access-Versionen klicken Sie das gewünschte Steuerelement im Ribbon unter Entwurf|Steuerelemente an, markieren dann das betreffende Feld aus der Feldliste und legen dieses dann im Entwurf des Formulars oder Berichts an.**

## Erst anlegen, dann binden

Die dritte Möglichkeit zum Anlegen eines gebundenen Steuerelements ist die, zunächst ein Steuerelement des gewünschten Typs zum Formular- oder Berichtsentwurf hinzuzufügen und anschließend seine Eigenschaft Steuerelementinhalt auf eines der Felder der unter Datenherkunft angegebenen Tabelle oder Abfrage einzustellen. Dies ist aber vermutlich die umständlichste Variante.

## Achtung: Steuerelemente benennen!

Steuerelemente, die Sie auf eine der ersten beiden Methoden anlegen, erhalten als Namen standardmäßig den gleichen Namen wie das Feld, an das sie gebunden sind. Wenn Sie ein Steuerelement erst anlegen und dann an ein Feld binden, erhält es als Namen automatisch einen Wert wie **Text27** oder **Kombinationsfeld7**. Beides ist nicht gut, denn Sie werden früher oder später per VBA oder auf andere Weise sowohl auf Steuerelemente als auch auf deren Inhalt, also das Feld, das als Steuerelementinhalt angegeben wurde, zugreifen. Wenn dann sowohl das Steuerelement als auch das Feld den gleichen Namen haben, ist nicht eindeutig, ob Sie auf das Steuerelement oder das Feld zugreifen. Gewöhnen Sie sich daher an, den Namen von Steuerelementen mit einem geeigneten Präfix zu versehen (→*[basics]*Konventionen).

# Gruppierungen anlegen

Sie können die Daten eines Berichts nach verschiedenen Kriterien gruppieren, zum Beispiel die Artikel nach der Kategorie oder Personen nach Eigenschaften wie Geschlecht et cetera. Dazu legen Sie eine Gruppierung an und stellen weitere Eigenschaften ein, die beispielsweise angeben, ob jede Gruppe einen eigenen Kopf- und/oder Fußbereich enthalten soll.

*****

# Gültigkeitsbereich von VBA-Programmen

Je nach dem Ort und der Deklaration kann ein VBA-Programm von verschiedenen Orten aus aufgerufen werden. Dieser Artikel stellt diese Gültigkeitsbereiche vor.

Für den Gültigkeitsbereich eines VBA-Programms sind zwei Faktoren ausschlaggebend:

- Ist das VBA-Programm als Public oder Private deklariert?
- Befindet sich das VBA-Programm in einem Standardmodul oder in einem Klassenmodul?

## Universelle Gültigkeit

Ein VBA-Programm kann von überall aus aufgerufen werden, wenn es sich in einem Standardmodul befindet und es als öffentliches VBA-Programm deklariert wurde. Letzteres erreichen Sie durch Voranstellen des Schlüsselworts Public oder ohne Verwendung eines Schlüsselworts:

```
Public Sub OeffentlicheProzedur()
    ...
End Sub
```

oder

```
Sub Oeffentliche Prozedur()
    ...
End Sub
```

## Modulweite Gültigkeit

Wenn Sie ein VBA-Programm mit dem Schlüsselwort **Private** deklarieren, kann es nur von innerhalb des Moduls aufgerufen werden, in dem es sich befindet. Dabei spielt es keine Rolle, ob das VBA-Programm sich in einem Standardmodul oder in einem Klassenmodul befindet:

```
Private Sub NichtOeffentlicheProzedur()
    ...
End Sub
```

## Eingeschränkte Gültigkeit in Klassenmodulen

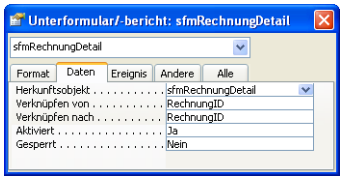
Wenn Sie ein VBA-Programm in einem Klassenmodul mit dem Schlüsselwort **Public** deklarieren, können Sie dieses auch von außerhalb des Klassenmoduls aufrufen. Hier gelten allerdings spezielle Regeln, die wir zu einem späteren Zeitpunkt ergänzen.

# Haupt- und Unterformular mit verknüpften Daten

Meist sollen Haupt- und Unterformular Daten aus verknüpften Tabellen darstellen. Dabei kann es sich beispielsweise um Rechnungen im Hauptformular und Rechnungspositionen im Unterformular handeln. Das Unterformular von Access, in dem das als Unterformular verwendete Formular eingebettet wird, bietet die Möglichkeit, die Daten des Unterformulars mit denen des Hauptformulars zu synchronisieren.

Zunächst fügen Sie das eine Formular als Unterformular in das Hauptformular ein (→*[basics]* **Unterformular zu Formular hinzufügen**). Wenn die als Datenherkunft der beiden Formulare verwendeten Tabellen bereits über eine Beziehung über ein Fremdschlüsselfeld verfügen (so wie in →*[basics]* **1:n-Beziehung anlegen** ), erledigt das Hinzufügen des Unterformulars per Drag and Drop bereits die nötige Arbeit.

Anderenfalls müssen Sie selbst Hand anlegen: Aktivieren Sie das Unterformularsteuerelement, was am einfachsten geht, indem Sie ausgehend von einem leeren Bereich des Hauptformulars mit der Maus einen Rahmen aufziehen, der als einziges Steuerelement das Unterformularsteuerelement berührt. Das Eigenschaftsfenster sollte dann auf der Registerseite **Daten** zwei Eigenschaften namens **Verknüpfen von** und **Verknüpfen nach** anzeigen (Sie können das Unterformularsteuerelement auch über das Eigenschaftsfenster auswählen – allein das Anklicken zum Auswählen schlägt manchmal fehl, weil allein ein weiterer Klick bereits das enthaltene Unterformular aktiviert).




Die beiden Eigenschaften **Verknüpfen von** und **Verknüpfen nach** sorgen für eine Synchronisierung der Daten von Haupt- und Unterformular.

Falls dies noch nicht automatisch geschehen ist, müssen Sie für diese beiden Eigenschaften je ein Feld der Datenherkunft des Haupt- und des Unterformulars auswählen. Für die Eigenschaft **Verknüpfen von** tragen Sie das Fremdschlüsselfeld der Tabelle ein, die als Datenherkunft des Unterformulars dient. Für die Eigenschaft **Verknüpfen nach** verwenden Sie das Primärschlüsselfeld der Datenherkunft des Hauptformulars.

# Index für ein Tabellenfeld festlegen

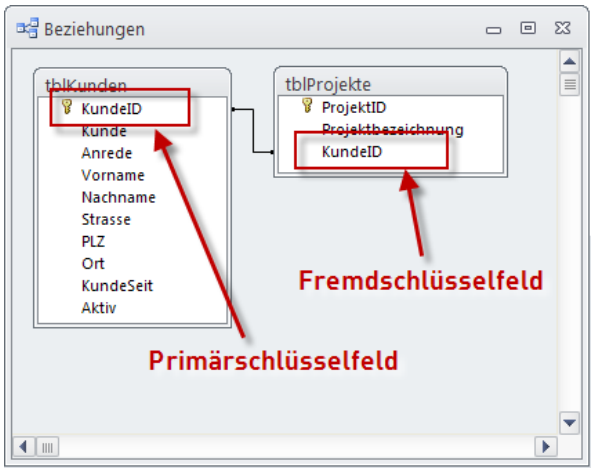
Access bietet verschiedene Index- beziehungsweise Schlüsselarten an, die Sie über die Festlegung bestimmter Eigenschaften genau einstellen können.

Grundsätzlich dient das Festlegen eines Index für ein Feld dazu, dass Access zusätzliche Informationen zu den Inhalten des indizierten Feldes speichert, um später bei der Suche nach Werten in diesem Feld oder beim Sortieren von Datensätzen nach den Werten dieses Feldes schneller ein Ergebnis liefern zu können.

## Primärindex anlegen

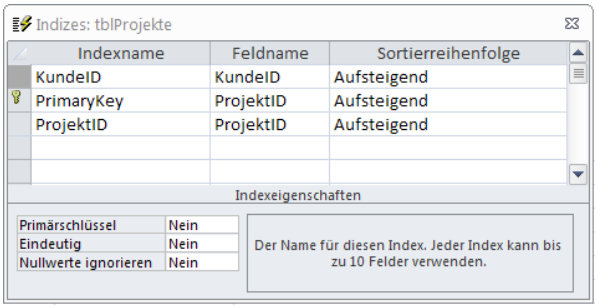
Am einfachsten legen Sie einen Primärindex beziehungsweise einen Primärschlüssel an. Wie dies funktioniert, zeigt →*[basics]* **Optionen einstellen**. Mit einem Primärschlüssel definieren Sie ein Feld, das die Datensätze einer Tabelle eindeutig kennzeichnet.

Ein Primärschlüsselfeld kann Teil einer Beziehung sein, wenn es in einer weiteren Tabelle, beispielsweise **tblProjekte**, ein Feld gibt, das die Inhalte der Tabelle mit dem Primärschlüssel, beispielsweise **tblKunden**, aufnimmt und über das Beziehungen-Fenster eine Beziehung zwischen diesem Primärschlüsselfeld und dem Fremdschlüsselfeld festgelegt wurde.




Ein Fremdschlüsselfeld besitzt nicht automatisch einen Index. Unter bestimmten Umständen müssen Sie diesen manuell festlegen. Dies geschieht grundsätzlich so:

- Öffnen Sie die betroffene Tabelle in der Entwurfsansicht.
- Markieren Sie das Feld, dem Sie einen Index zuweisen möchten.
- Wählen Sie den Befehl **Ansicht|Indizes** aus (Access 2003 und älter) beziehungsweise klicken Sie auf die Ribbon-Schaltfläche **Entwurf|Einblenden/Ausblenden|Indizes** (Access 2007 und jünger).
- Es erscheint der Dialog **Indizes: <Tabellenname>**. Hier legen Sie den neuen Index an, indem Sie einen eindeutigen Namen für den Index eingeben (beispielsweise **idxTblProjekteKundeID**), das Feld auswählen, für das Sie den Index anlegen möchten und eine Sortierreihenfolge festlegen, für die der Index optimiert werden soll.
- Schließlich legen Sie mit den drei Eigenschaften **Primärschlüssel**, **Eindeutig** und **Nullwerte ignorieren** die Art des Index fest.




Stellen Sie **Primärschlüssel** auf **Ja** ein, setzt Access gleichzeitig die Option **Eindeutig** auf **Ja**. Sie können aber auch einen Index als eindeutigen Index markieren, der kein Primärschlüssel ist.

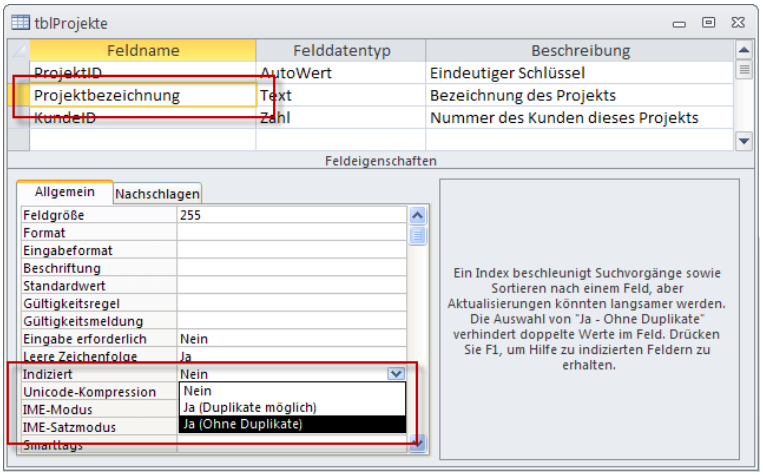
Der Unterschied ist, dass das Primärschlüsselfeld immer einen Wert enthalten muss. Ein Feld, für dass Sie einen eindeutigen Index festgelegt haben, kann einen eindeutigen Wert enthalten, aber auch den Wert **Null**. **Null** bedeutet, dass das Feld gar keinen Wert enthält.

## Eindeutigen Index anlegen

Ein Index mit dem Wert **Ja** für die Eigenschaft **Primärschlüssel** ist automatisch ein eindeutiger Index. Sie können aber auch Feldern, die kein Primärschlüsselfeld sind, mit einem eindeutigen Index versehen. Dazu stellen Sie einfach nur die Eigenschaft **Eindeutig** dieses Feldes im Dialog **Indizes: <Tabellenname>** auf **Ja** ein. Wie oben erwähnt, können Sie dieses Feld zwar standardmäßig leer lassen. Wenn Sie aber dort Daten eingeben,

dann müssen diese auch eindeutig sein; das heißt, dass das indizierte Feld keinen Wert in mehr als einem Datensatz enthalten darf.

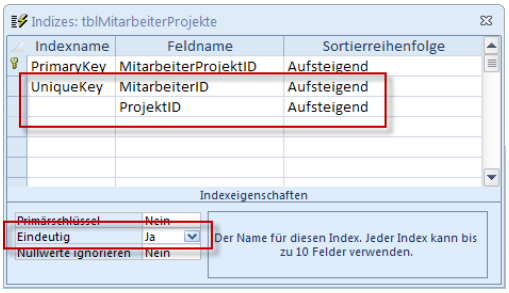
Es gibt übrigens noch eine zweite Möglichkeit, einen eindeutigen Index festzulegen: Dazu stellen Sie einfach die Eigenschaft **Indiziert** auf der Registerseite **Allgemein** des Tabellenentwurfs auf den Wert **Ja (Ohne Duplikate)** ein. Mit der Einstellung **Ja(Duplikate möglich)** fügen Sie einen einfachen Index ohne Anspruch auf eindeutige Werte hinzu.




Wenn Sie für ein Feld einen eindeutigen Index festlegen, bedeutet dies automatisch eine Kontrolle bei der Eingabe der Werte in dieses Feld durch Access. Wenn der Benutzer einen Wert eingibt, der bereits im gleichen Feld eines anderen Datensatzes vorliegt, erscheint eine entsprechende Meldung.

## Zusammengesetzten eindeutigen Index festlegen

Ein eindeutiger Index kann sich auch auf mehrer Felder beziehen. Zusammengesetzte eindeutige Indizes werden meist bei Verknüpfungstabellen zur Herstellung von m:n-Beziehungen verwendet. Dort gehen Sie genauso vor wie beim Anlegen eines normalen eindeutigen Index über den Dialog **Indizes**. Der einzige Unterschied ist, dass Sie gleich unterhalb der Zeile mit dem Index eine weitere Zeilen füllen, allerdings nur mit dem Namen der weiteren Felder, die Sie zum Index hinzufügen möchten.




## Nullwerte ignorieren

Die dritte Eigenschaft des Dialogs **Indizes: <Tabellenname>** namens **Nullwerte ignorieren** wird interessant, wenn ein Feld, für das Sie einen Index definieren, in vielen Fällen leer sein wird. Stellen Sie dann **Nullwerte ignorieren** auf **Ja** ein, damit Access solche Datensätze beim Indizieren nicht berücksichtigt.

Natürlich können Sie auch einen Index für ein Feld definieren, der es weder zum Primärschlüsselfeld macht, noch

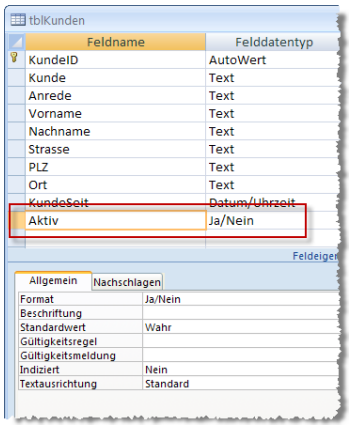
## Folgen der Indizierung

Die direkten Folgen der Definition eines Indizes, unabhängig davon, ob es sich um einen Primärindex, einen eindeutigen oder einen Index ohne Prüfung auf Duplikate handelt, sind folgende:

- Access muss den Index beim Ändern der indizierten Felder, also beim Anlegen, Ändern oder Löschen betroffener Datensätze, aktualisieren. Dies kostet erstmal Zeit.
- Andererseits wird die Geschwindigkeit beim Sortieren von Datensätzen nach dem Inhalt indizierter Felder sowie die Suche nach Werten indizierter Felder dramatisch erhöht.

Solange Sie nicht mit tausenden Datensätzen arbeiten, wird die Aktualisierung des Index eines indizierten Feldes kaum ins Gewicht fallen. Andererseits sollten Sie zumindest Fremdschlüsselfelder indizieren.

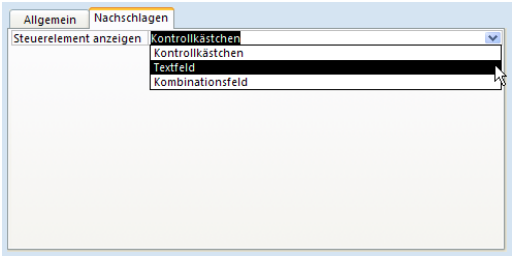
# Ja/Nein-Feld anlegen



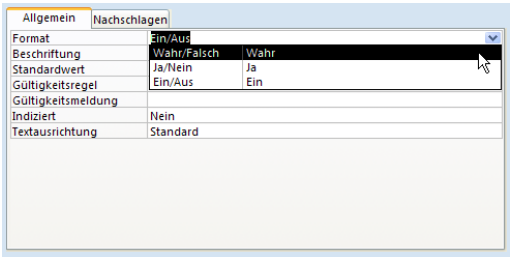

Ein **Ja/Nein**-Feld legen Sie an, indem Sie in die Spalte **Feldname** einer leeren Zeile in der Entwurfsansicht einer Tabelle den Feldnamen eintragen und den Datentyp auf **Ja/Nein** festlegen.

Intern speichert ein **Ja/Nein**-Feld immer einen Zahlenwert, und zwar entweder **0** (**False**) oder -1 (**True**).

Die Abbildung rechts zeigt ein Beispiel für ein **Ja/Nein**-Feld mit seinen Eigenschaften. Üblicherweise soll der Benutzer den Inhalt des **Ja/Nein**-Feldes durch ein Kontrollkästchen eingeben können. Dazu ist im Bereich **Nachschlagen** der Feldeigenschaften in der Entwurfsansicht der Wert **Kontrollkästchen** eingestellt:



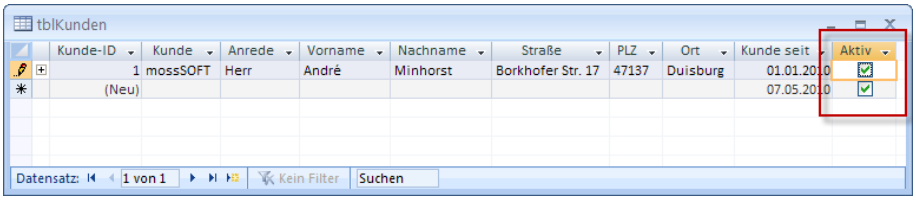

Erst wenn Sie dies in **Textfeld** ändern, wird die Eigenschaft **Format** interessant. Je nachdem, ob der Wert **0** oder **-1** beträgt, zeigt das Feld dann **Wahr/Falsch**, **Ja/Nein** oder **Ein/Aus** an.




Sie können den Wert eines **Ja/Nein**-Feldes auch über ein Kombinationsfeld auswählen lassen. Dann stellen Sie die Eigenschaft **Steuerelement anzeigen** auf **Kombinationsfeld** ein und legen mit den übrigen Eigenschaften fest, welche Werte das Kombinationsfeld zur Auswahl bereitstellen soll (→*[basics]* **Nachschlagefeld für Wertliste einrichten**).

In der Regel verwendet man jedoch ein einfaches Kontrollkästchen für die Eingabe von Daten in ein **Ja/Nein**-Feld – simpler geht es nicht.

Die Abbildung unten ein **Ja/Nein**-Feld in der Datenblattansicht.



# Keine Dateneingabe via Tabelle oder Abfrage

Gerade für Einsteiger ist dies verlockend: Die Tabelle ist erstellt, die Abfrage liefert die richtigen Datensätze, also nutzen wir doch die praktische Datenblattansicht zur Eingabe von Daten direkt in die Tabelle oder über den Umweg einer Abfrage.

Dies ist für den Einsatz auf dem Entwicklerrechner sicher in Ordnung, aber spätestens wenn der Benutzer erstmals mit der Anwendung arbeitet und Daten eingeben, bearbeiten oder löschen möchte, sollte dies ausschließlich über Formulare geschehen.

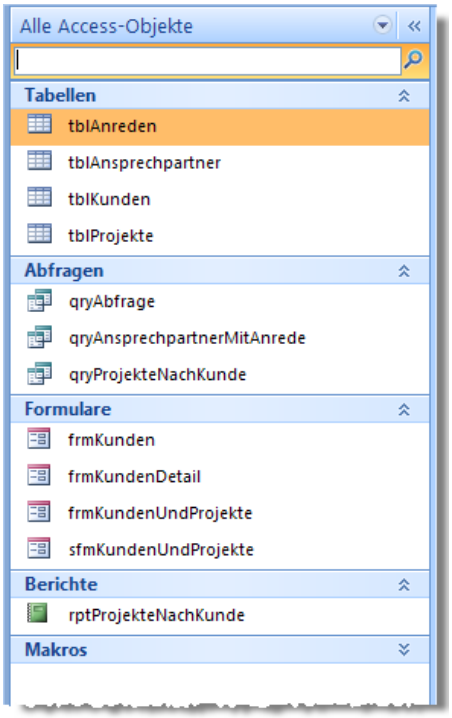
Dafür gibt es unter anderem die folgenden Gründe:

- In einem Formular können Sie eine Eingabemaske erstellen, die eine wesentlich ergonomischere Eingabe von Daten erlaubt als es bei der Datenblattansicht der meisten Tabellen oder Abfragen der Fall ist. Dies beginnt spätestens, wenn der Benutzer in der Datenblattansicht scrollen muss, weil nicht alle Felder gleichzeitig dargestellt werden können.
- Sie können dem Benutzer Meldungen anzeigen, die ihn darüber informieren, dass Änderungs- oder Löschvorgänge nicht mehr rückgängig gemacht werden können und deshalb nicht unbedacht angestoßen werden sollten.
- Sie können ein Formular um Funktionen zur Validierung erweitern. Dies ist zwar eingeschränkt auch in der Datenblattansicht von Tabellen und Abfragen möglich, jedoch können Sie die im Falle einer Fehleingabe erscheinende Meldung nur unzureichend beeinflussen.
- Sie können Formulare um Eingabehilfen erweitern. So bringen Sie dort beispielsweise erklärende Texte für die Eingabe in bestimmte Felder unter oder Sie verwenden spezielle Steuerelemente wie Kombinationsfelder, Listenfelder, Kontrollkästchen oder Optionsgruppen zur Eingabe der Daten.

# Konventionen

**Es gibt verschiedene Bereiche innerhalb von Access, bei denen eine einheitliche Benennung sinnvoll ist. An dieser Stelle sammeln wir alle in Access [basics] verwendeten Konventionen. Dieser Artikel wird ständig erweitert.**

## Datenbankobjekte




Für Datenbankobjekte verwenden wir folgende Präfixe:

- **tbl**: Tabellen (Beispiel: **tblKunden**, **tblProjekte**)
- **qry**: Abfragen (Beispiel: **qryKundenNachFirma**)
- **frm**: Formulare (Beispiel: **frmKundenDetail**)
- **sfm**: Formulare, die als Unterformular in ein anderes Formular eingebettet werden (**sfmKundenUebersicht**)
- **rpt**: Berichte (Beispiel: **rptKundenDetail**)
- **srp**: Berichte, die als Unterbericht in einen anderen Bericht eingebettet werden (**srpKundenProjekte**)
- **mac**: Makros (Beispiel: **macKundeLoeschen**)
- **mdl**: Standardmodule (Beispiel: **mdlTools**)
- **cls**: Klassenmodule (Beispiel: **clsFehlerbehandlung**)

Außerdem gelten folgende Regeln:

- Tabellen erhalten einen Namen, der aus dem Präfix und der Bezeichnung der enthaltenen Objekte in der Pluralform besteht (also beispielsweise **tblProjekte** oder **tblKunden**)
- Tabellen, die der Herstellung einer m:n-Beziehung dienen, erhalten einen Namen, der aus dem Präfix und der Bezeichnungen der verknüpften Tabellen im Plural besteht (etwa **tblMitarbeiterProjekte**). Manchmal gibt es sinnvollere Bezeichnungen. Wenn die Verknüpfungstabelle beispielsweise Bestellungen und Artikel verknüpft, könnte diese durchaus **tblBestellpositionen** statt **tblBestellungenArtikel** heißen.
- Wir verwenden ausschließlich Buchstaben, wobei neue Wörter mit einem großen Buchstaben beginnen (Beispiele: **tblMitarbeiterProjekte**, **qryProjektNachStartdatum**, **frmProjekteDetail**)

## Feldnamen

Feldnamen in Tabellen sollten nach Möglichkeit ausschließlich aus kleinen oder großen Buchstaben bestehen. Wenn ein Wort als Feldname nicht ausreicht, gibt es verschiedene Varianten: Entweder Sie schreiben beide Wörter hintereinander und beginnen jedes Wort mit einem großen Buchstaben (**TelefonPrivat**) oder Sie trennen beide Elemente des Feldnamens durch einen Unterstrich (**Telefon_Privat**). In **Access [basics]** werden wir mit der ersten Variante arbeiten. Zahlen verwenden Sie nur, wenn unbedingt nötig.

Der Name des Primärschlüsselfeldes einer Tabelle wird aus dem Objekt, dessen Eigenschaften die Tabelle speichert und dem Anhängsel **ID** zusammengesetzt, zum Beispiel **KundeID**, **MitarbeiterID** oder **ProjektID**.

Die Namen von Fremdschlüsselfeldern lauten genauso wie die Namen der Primärschlüsselfelder der Tabelle, zu der das Fremdschlüsselfeld eine Beziehung herstellt. Ein Feld zum Referenzieren eines Kunden in einer Projekte-Tabelle würde also genau wie das Primärschlüsselfeld **KundeID** der Tabelle **tblKunden** ebenfalls **KundeID** heißen.

Verwenden Sie auf keinen Fall Leerzeichen oder Sonderzeichen in den Feldnamen.

## Steuerelementnamen in Formularen und Berichten

Für Steuerelemente verwenden wir die folgenden Präfixe:

- **txt**: Textfeld (**TextBox**)
- **lbl**: Bezeichnungsfeld (**Label**)
- **cbo**: Kombinationsfeld (**ComboBox**)
- **chk**: Kontrollkästchen (**CheckBox**)
- **cmd**: Schaltfläche (**CommandButton**)

Außerdem gelten für die Benennung gebundener Steuerelemente (→*[basics]*Gebundene Steuerelemente zu Formularen und Berichten hinzufügen) folgende Regeln:

- Die Bezeichnung gebundener Steuerelemente besteht aus dem Präfix und dem Namen des Feldes, an welches das Steuerelement gebunden ist.

## VBA-Variablen

VBA-Variablennamen verwenden ein Präfix, das einem Kürzel des Datentyps entspricht. Danach folgt eine genaue Beschreibung des Variableninhalts. Die Präfixe für die Basisdatentypen von VBA lauten:

- **str**: **String**
- **byt**: **Byte**
- **bol**: **Boolean**
- **int**: **Integer**
- **lng**: **Long**
- **sng**: **Single**
- **dbl**: **Double**
- **cur**: **Currency**
- **dat**: **Date**
- **obj**: **Object**
- **var**: **Variant**

Der Variablenname darf nur aus Buchstaben, Zahlen und dem Unterstrich (_) bestehen und müssen mit einem Buchstaben beginnen. Bei Variablennamen, die aus mehreren Wörtern bestehen, wird der erste Buchstabe eines jeden Worts groß geschrieben. Beispiel: **lngNeuerKunde**.

## Komfortable Umbenennung gebundener Steuerelemente

Gelegentlich werden Sie keine Lust haben, beim Erstellen eines gebundenen Formulars gleich noch alle Steuerelemente mit den entsprechenden Präfixen zu versehen. Kein Problem: Wir haben ein Tool für diesen Zweck programmiert. Sie finden es samt einer ausführlichen Anleitung unter **diesem Link**.

# Löschweitergabe festlegen

Diese Eigenschaft stellen Sie wie die Aktualisierungsweitergabe im gleichen Dialog ein, in dem Sie auch referentielle Integrität zwischen zwei Feldern verknüpfter Tabellen festlegen (→*[basics]* **Referentielle Integrität festlegen**). Die Löschweitergabe erfährt jedoch ungleich mehr Aufmerksamkeit in der Datenmodellierung. Sie sorgt dafür, dass beim Löschen eines Datensatzes der Mastertabelle auch alle verknüpften Datensätze der Detailtabelle gelöscht werden.

# Nachschlagefeld für eine 1:n-Beziehung einrichten

Wenn zwei Tabellen in einer 1:n-Beziehung stehen, macht es üblicherweise Sinn, wenn Sie in der Detailtabelle (also der Tabelle mit dem Fremdschlüsselfeld der Beziehung) eine Möglichkeit einrichten, einen Datensatz der Mastertabelle (also der Tabelle mit dem Primärschlüsselfeld der Beziehung) auszuwählen.

Dies ist in der Regel ein Nachschlagefeld (im Fachjargon auch »Kombinationsfeld« genannt).

## Nachschlagefeld per Assistent

Ein Nachschlagefeld legen Sie am schnellsten mit dem entsprechenden Assistenten an. Öffnen Sie die Tabelle mit dem betroffenen Feld in der Entwurfsansicht und wählen Sie für den Datentyp des Felds den Eintrag **Nachschlage-Assistent** aus. Die Screenshots zum Ablauf sparen wir uns. Gehen Sie einfach so vor:

- Die Daten sollen aus einer Tabelle oder Abfrage stammen.
- Wählen Sie die Quelltabelle oder -Abfrage aus.
- Fügen Sie die Felder für das Nachschlagefeld hinzu. Normalerweise sind dies das Primärschlüsselfeld und ein Textfeld, das die anzuzeigenden Inhalte enthält.
- Legen Sie fest, ob das anzuzeigende Feld sortiert werden soll.
- Wenn das Primärschlüsselfeld nicht angezeigt werden soll, blenden Sie dieses aus.

Dies hat folgendes mit Ihrem Feld angestellt:

- Anlegen einer Beziehung vom ausgewählten Feld (Fremdschlüsselfeld) zum Primärschlüsselfeld von Quelltabelle oder -abfrage
- Ändern der Eigenschaft **Steuerelement anzeigen** auf **Kombinationsfeld**.
- Erstellen einer SQL-Abfrage, welche die ausgewählten Felder der Datensatzherkunft enthält, für die Eigenschaft **Datensatzherkunft**.
- Einstellen der Eigenschaft **Gebundene Spalte** auf den Wert **1**.
- Einstellen der Eigenschaft **Spaltenanzahl** auf **2** und der Eigenschaft **Spaltenbreiten** auf einen Wert wie **0cm;2,54cm**.
- Einstellen der übrigen Eigenschaften auf der Registerseite **Nachschlagen** des Feldes in der Entwurfsansicht der Tabelle.

Statt eines Textfeldes zur Eingabe von Daten finden Sie in der Datenblattansicht der Tabelle und aller darauf aufbauenden Abfragen und Formulare ein Kombinationsfeld vor, das in der Regel zwei Felder enthält. Das erste liefert den Primärschlüssel, das zweite das anzuzeigende Feld. Durch die Einstellung **Spaltenanzahl = 2** und **Spaltenbreiten = 0cm;2,54cm** zeigt das Kombinationsfeld zwei Spalten an, die erste davon allerdings mit der Breite **0cm**, also quasi unsichtbar. Das ist normalerweise das Ziel: Die erste Spalte enthält als gebundene Spalte das eigentliche Datum des Felds, die zweite bietet die gewünschten Daten zur Auswahl an.

Außerdem legt der Nachschlageassistent eine Beziehung zwischen den Tabellen an, die Sie im Beziehungen-Fenster einsehen können.

**2003 und älter:** Öffnen Sie das Beziehungen-Fenster mit dem Menüeintrag **Extras|Beziehungen**.

**2007 und jünger: Öffnen Sie das Beziehungen-Fenster mit dem Ribbon-Eintrag Datenbanktools|Einblenden/Ausblenden|Beziehungen.**

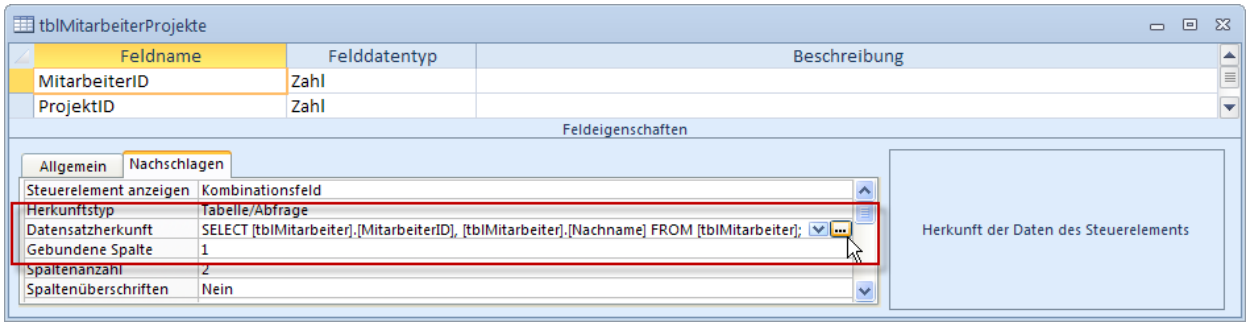
Bei Bedarf definieren Sie hier noch referentielle Integrität oder andere Eigenschaften für die Beziehung (→*[basics]* **Referentielle Integrität festlegen**).

Das Anlegen eines Nachschlagefelds in einer Tabelle führt dazu, dass auch an dieses Feld gebundene Steuerelemente in Formularen als Kombinationsfeld statt als Textfeld ausgeführt werden, was eine Menge Arbeit sparen kann.

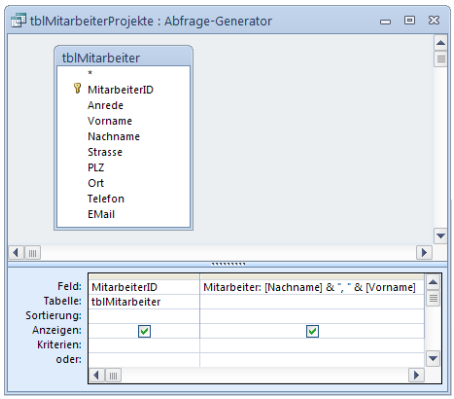
## Inhalt mehrerer Felder zur Anzeige im Nachschlagefeld zusammensetzen

Nachschlagefelder zeigen normalerweise nur den Inhalt eines einzigen Feldes an, zum Beispiel den Firmennamen oder eine Anrede. Manchmal reicht dies nicht aus und das Nachschlagefeld soll beispielsweise Vor- und Nachname einer Person anzeigen. Dazu brauchen Sie einfach nur die Datensatzherkunft des Formulars zu ändern. Gehen Sie wie folgt vor:

- Öffnen Sie die betroffene Tabelle in der Entwurfsansicht.
- Wählen Sie das Nachschlagefeld aus.
- Wechseln Sie in den Eigenschaften zur Registerseite **Nachschlagen**.
- Klicken Sie auf die Eigenschaft **Datensatzherkunft** und dann auf die Schaltfläche mit den drei Punkten.




- In der nun erscheinenden Entwurfsansicht der als Datensatzherkunft verwendeten Abfrage ändern Sie den Inhalt der zweiten Spalte auf einen Ausdruck, der aus einem Bezeichner und dem anzuzeigenden Inhalt besteht. In diesem Fall: **Mitarbeiter: [Nachname] & ", " & [Vorname]**. Dies gibt einen Ausdruck wie **Müller, Klaus**.




- Nach dem Schließen der Abfrage wird ihr SQL-Text in die Eigenschaft **Datensatzherkunft** des Nachschlagefelds übernommen.

# Nachschlagefeld für Wertliste einrichten

Manche Felder erhalten nur wenige mögliche Werte. Diese können Sie vorbereiten und dem Benutzer zur Auswahl über ein Kombinationsfeld bereitstellen. Angenommen, Sie haben bereits ein Textfeld namens Anrede eingerichtet und möchten dem Benutzer die beiden Einträge **Herr** und **Frau** zur Auswahl anbieten. Dann führen Sie folgende Schritte durch:

- Öffnen Sie die betroffene Tabelle in der Entwurfsansicht.
- Markieren Sie das Textfeld.
- Zeigen Sie im unteren Bereich der Entwurfsansicht die Feldeigenschaften auf der Registerseite **Nachschlagen** an.
- Stellen Sie den Wert von Steuerelement anzeigen auf **Kombinationsfeld** ein.
- Wählen Sie für **Herkunftstyp** den Eintrag **Wertliste** aus.
- Geben Sie für **Datensatzherkunft** die in Anführungszeichen geschriebenen Werte nacheinander durch Semikola getrennt ein, also etwa **“Herr“;“Frau“**.
- Wenn nur diese Werte gültig sind, stellen Sie außerdem **Nur Listeneinträge** auf **Ja** ein. Access liefert dann eine Warnmeldung, wenn der Benutzer einen anderen Wert eingibt.

Fertig – nach einem Wechsel in die Datenblattansicht können Sie nun die angegebenen Werte per Kombinationsfeld auswählen.

# OK- und Abbrechen-Schaltfläche hinzufügen

Normalerweise sollen in ein Detailformular eingegebene Änderungen beim Schließen des Formulars entweder akzeptiert oder verworfen werden. Daher sehen wir zwei verschiedene Schaltflächen dafür vor. Führen Sie die folgenden Schritte durch, um diese Schaltflächen anzulegen:

- Fügen Sie zwei Schaltflächen zum Bereich **Formularfuß** hinzu, indem Sie dieses Steuerelement in der Toolbox anklicken und dann auf die Position im Formular klicken, an der Sie das Steuerelement einfügen möchten.
- Gegebenenfalls erscheint nun der **Befehlsschaltflächen-Assistent**. Wir brauchen ihn nicht, daher können Sie gleich alle Steuerelementassistenten deaktivieren. Dazu klicken Sie einfach auf die Schaltfläche **Steuerelementassistenten** in der Toolbox.
- Stellen Sie die Eigenschaft **Andere|Name** der ersten Schaltfläche auf **cmdOK** und die der zweiten auf **cmdAbbrechen** ein. Wenn Sie englische Steuerelementbezeichnungen verwenden möchten, können Sie dies tun (hier etwa mit der Bezeichnung **cmdCancel** für die zweite Schaltfläche) – im Rahmen dieses Buchs verwenden wir der besseren Verständlichkeit halber deutsche Bezeichnungen.
- Nun sollen die Schaltflächen auch noch entsprechende Texte anzeigen. Dazu klicken Sie entweder direkt auf die Texte der Schaltflächen und ändern die Beschriftungen an Ort und Stelle oder Sie legen diese über die Eigenschaft **Format|Beschriftung** fest – etwa auf **OK** und **Abbrechen**.

Nun müssen Sie die Schaltflächen nur noch mit Leben füllen. Wie das geht, lesen Sie unter →[basics] **OK-Schaltfläche erstellen** und →[basics] **Abbrechen-Schaltfläche erstellen**.

# OK-Schaltfläche erstellen

Einfache **OK**-Schaltflächen führen wir in zwei Versionen: Die erste soll das Formular einfach nur schließen, die zweite soll es unsichtbar machen. Warum das so ist und welcher Code dafür nötig ist, zeigen die folgenden Abschnitte.

### Variante 1: OK-Schaltfläche zum Schließen eines Formulars

Eine **OK**-Schaltfläche soll ein Formular normalerweise einfach nur schließen, was bei gebundenen Formularen üblicherweise dazu führt, dass neu angelegte oder geänderte Daten automatisch gespeichert werden. Dazu fügen Sie ihm eine Ereignisprozedur für die Ereigniseigenschaft **Beim Klicken** hinzu (→**Ereignisprozedur hinzufügen**), die wie folgt aussieht:

```
Private Sub cmdOK_Click()
    DoCmd.Close acForm, Me.Name
End Sub
```

Die einzigen Anweisung dieser Routine sorgt für das Schließen des Formulars und für das Speichern der enthaltenen Daten.

### Variante 2: OK-Schaltfläche zum Verbergen eines Formulars

Es kann auch sein, dass die **OK**-Schaltfläche das Formular nur unsichtbar machen, aber nicht schließen soll. Das ist beispielsweise der Fall, wenn das Formular als modaler Dialog aufgerufen wurde. Die bedeutet, dass der Benutzer nur auf dieses Formular zugreifen kann, nicht aber auf andere Elemente der Access-Benutzeroberfläche – genau bis zu dem Zeitpunkt, zu dem das Formular den Fokus verliert. Und das geschieht genau dann, wenn das Formular geschlossen oder unsichtbar gemacht wird. Ein weiterer Vorteil von so geöffneten Formularen ist, dass auch der aufrufende Code angehalten wird, bis das Formular den Fokus verliert. Wird das Formular hier nur unsichtbar gemacht und nicht geschlossen, kann der aufrufende Code im Anschluss noch auf das Formular zugreifen und beispielsweise seine Steuerelemente auslesen. In diesem Fall verwenden Sie die folgende Routine:

```
Private Sub cmdOK_Click()
    Me.Visible = False
End Sub
```

### OK-Klick per Eingabetaste

Statt eines Mausklicks auf die **OK**-Schaltfläche können Sie auch die Betätigung der Eingabetaste verwenden, um die oben beschriebene Ereignisprozedur auszulösen. Stellen Sie dazu die Eigenschaft **Standard** der Schaltfläche **cmdOK** auf **Ja** ein. Wenn der Benutzer nun nach der Dateneingabe auf die Eingabetaste drückt, fasst Access dies wie einen Klick auf die Schaltfläche **cmdOK** auf und führt die entsprechende Ereignisprozedur aus.

# Optionen einstellen

Access bietet eine ganze Reihe von Optionen an, mit denen Sie Verhaltensweisen einer Access-Datenbank und auch von Access selbst steuern können. In diesem Beitrag zeigen wir Ihnen, wie Sie Optionen allgemein einstellen können. In weiteren Beiträgen erfahren Sie, welche Optionen es gibt und wie Sie diese anpassen. Die Optionen stellen wir jeweils vor, wenn diese in einem der Artikel von **Access [basics]** zum Einsatz kommen.

## Access-Optionen einstellen

**2003 und älter:** Wählen Sie den Menüeintrag **Extras|Optionen** aus, um den Dialog zum Bearbeiten der Access-Optionen zu öffnen.

**2007 und jünger:** Klicken Sie im Office-Menü auf die Schaltfläche **Access-Optionen** (2007) beziehungsweise den Ribbon-Eintrag **Datei|Optionen** (2010). Die Access-Optionen finden Sie in allen Bereich mit Ausnahme von **Aktuelle Datenbank** – dies sind die spezifischen Optionen für die aktuell geöffnete Datenbankdatei.

## Datenbank-Optionen einstellen

**2003 und älter:** Wählen Sie den Menüeintrag **Extras|Start** aus, um den Dialog Start mit den spezifischen Optionen für die aktuell geöffnete Datenbankdatei zu öffnen.

**2007 und jünger:** Öffnen Sie die Access-Optionen wie oben unter **Access-Optionen einstellen** beschrieben. Die Optionen der aktuell geöffneten Datenbank liegen im Bereich **Aktuelle Datenbank**.

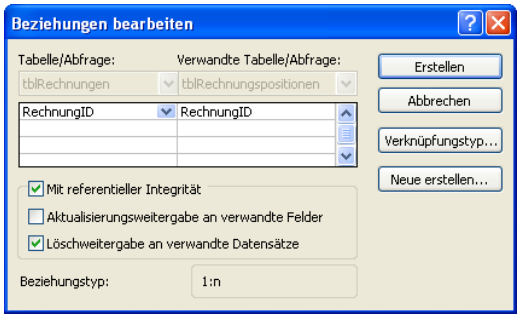
# Primärschlüsselfeld festlegen

Als Primärschlüsselfeld legen Sie das Feld einer Tabelle fest, dass als eindeutiger Index dienen soll. In der Regel ist das ein Autowertfeld (→*[basics]* **Autowertfeld anlegen**). Die Eigenschaft des Feldes als Primärschlüsselfeld stellen Sie wie folgt ein:

- Markieren Sie das Feld, das als Primärschlüsselfeld definiert werden soll.
- entweder über die Symbolleistenschaltfläche **Primärschlüssel** oder über den gleichnamigen Eintrag des Kontextmenüs eines Feldes ein.

# Referentielle Integrität festlegen

Referentielle Integrität für die Beziehung zwischen zwei Tabellen legen Sie im Beziehungen-Fenster fest, und zwar im Dialog **Beziehungen bearbeiten**. Diesen öffnen Sie entweder, indem Sie doppelt auf eine bestehende Beziehungslinie klicken oder indem Sie durch Ziehen eines Feldes auf das zu verknüpfende Feld eine neue Beziehung anlegen.




Seitenränder eines Berichtes einstellen

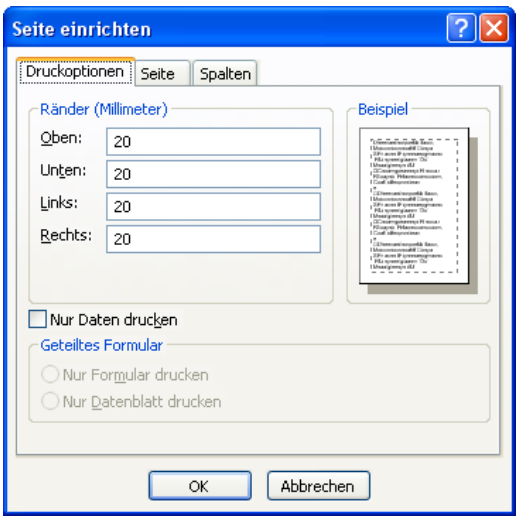
Die Seitenränder eines Berichts werden in den verschiedenen Access-Versionen auf unterschiedliche Arten eingestellt beziehungsweise der Aufruf des dafür zuständigen Dialogs geschieht unterschiedlich.

**2003 und älter:** Wählen Sie den Menübefehl **Datei|Seite einrichten** aus, um den Dialog **Seite einrichten** zu aktivieren.

**2007 und jünger: Betätigen Sie den Ribbon-Eintrag Seite einrichten|Seitenlayout|Seite einrichten, um den Dialog Seite einrichten zu öffnen.**

Der Dialog sieht in allen Versionen gleich aus und dient im Wesentlichen dazu, die Seitenränder des Berichts festzulegen. Diese Einstellung wird in Millimetern angegeben. Sie legt fest, in welchem Abstand vom jeweiligen Seitenrand sich der in der Entwurfsansicht definierte Bereich tatsächlich befindet.

Das bedeutet auch, dass die Summe des Wertes der Eigenschaft **Breite** des Berichts und dem linken und rechten Seitenrand bei Verwendung des DIN-A4-Formats im Hochformat nicht größer als 217 mm sein darf.

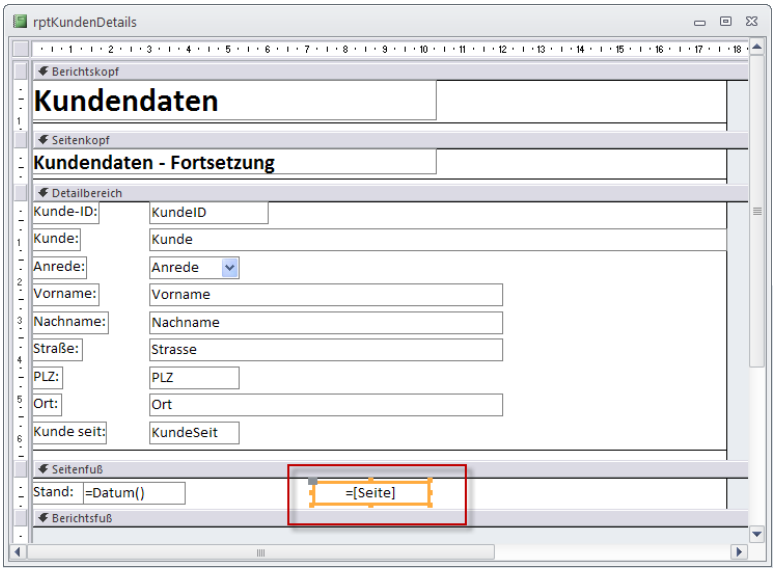


# Seitenzahl zu Bericht hinzufügen

Für die Anzeige der Seitenzahl in einem Bericht gibt es verschiedene Möglichkeiten: Entweder man zeigt nur die aktuelle Seitennummer an, oder die aktuelle Seitennummer und die Gesamtzahl der Seiten (getrennt etwa durch einen Schrägstrich, oder Sie fügen noch Text hinzu, sodass die Angabe der Seitenzahl zum Beispiel **Seite 1** oder **Seite 5 von 7** lautet.

Die Seitenanzahl geschieht meist im Seitenfußbereich, und zwar durch ein entsprechendes Textfeld. Nachdem Sie dieses an der gewünschten Stelle hinzugefügt haben, stellen Sie die Eigenschaft **Steuerelementinhalt** auf den entsprechenden Wert ein. Hier sind einige Varianten:

- **1**: **=Seite**
- **Seite 1**: **=“Seite “ & Seite**
- **1/6**: **=Seite & “/“ & Seiten**
- **Seite 1 von 6**: **=“Seite “ & Seite & “von“ & Seiten**




Platzieren Sie das Textfeld an der gewünschten Stelle, beispielsweise zentriert im Seitenfußbereich und stellen Sie seine Eigenschaft **Name** auf einen Wert wie **txtSeite** ein. Dies kann beispielswiese wie in der Abbildung rechts aussehen.

# Standardwerte für Tabellenfelder einstellen

Mit der Eigenschaft **Standardwert** können Sie den Wert festlegen, den Access für ein Feld beim Anlegen eines neuen Datensatzes automatisch einstellt. Diese Eigenschaft stellen Sie in der Entwurfsansicht einer Tabelle ein, indem Sie das betroffene Feld markieren und den Standardwert in die entsprechende Eigenschaft eintragen. Beispiele für Standardwerte sind die folgenden:

- Datumsfelder: **Now()** – aktuelles Datum/Uhrzeit, **Date()** – aktuelles Datum
- Zahlenfelder: konkrete Zahlenwerte wie **0**
- Textfelder: beliebige Texte
- **Ja/Nein**-Felder: **True**/**False**, **-1**/**0**

# Startformular einer Anwendung festlegen

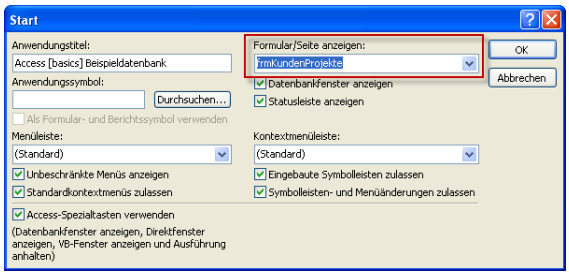
Access-Anwendungen sollen meist gleich beim Anzeigen ein Formular anzeigen. Um dies zu bewerkstelligen, gibt es mehrere Möglichkeiten:

- Sie legen das zu öffnende Formular in den Optionen der jeweiligen Datenbank fest.
- Sie erstellen ein AutoExec-Makro, das einen Befehl zum Öffnen des Formulars enthält.

In diesem Artikel wird die erste Möglichkeit vorgestellt. Die Einstellung der entsprechenden Option verläuft je nach der verwendeten Access-Version unterschiedlich:

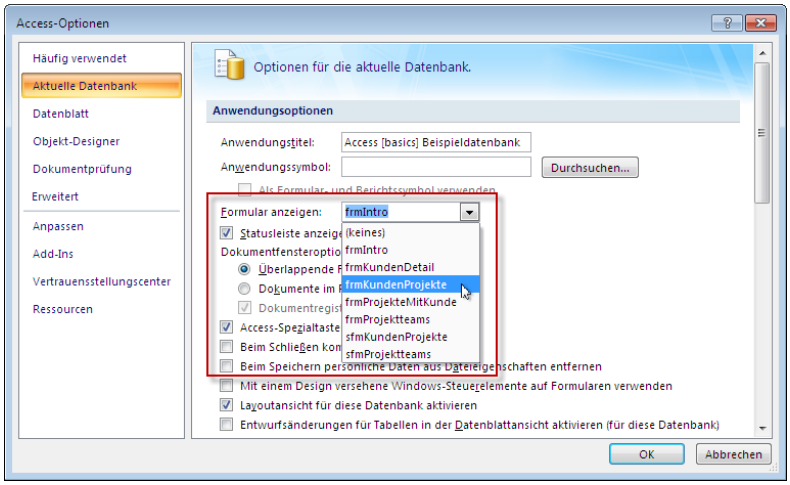
## Access 2003 und älter

- Öffnen Sie die **Start**-Optionen über den Menüeintrag **Extras|Start**.
- Wählen Sie für die Option **Formular/Seite anzeigen** das zu öffnende Formular aus:




## Access 2007

- Klicken Sie auf den Office-Button und dann auf die Schaltfläche **Access-Optionen**.
- Wechseln Sie dort zum Bereich **Aktuelle Datenbank** und wählen Sie das gewünschte Formular für die Option **Formular anzeigen** aus:



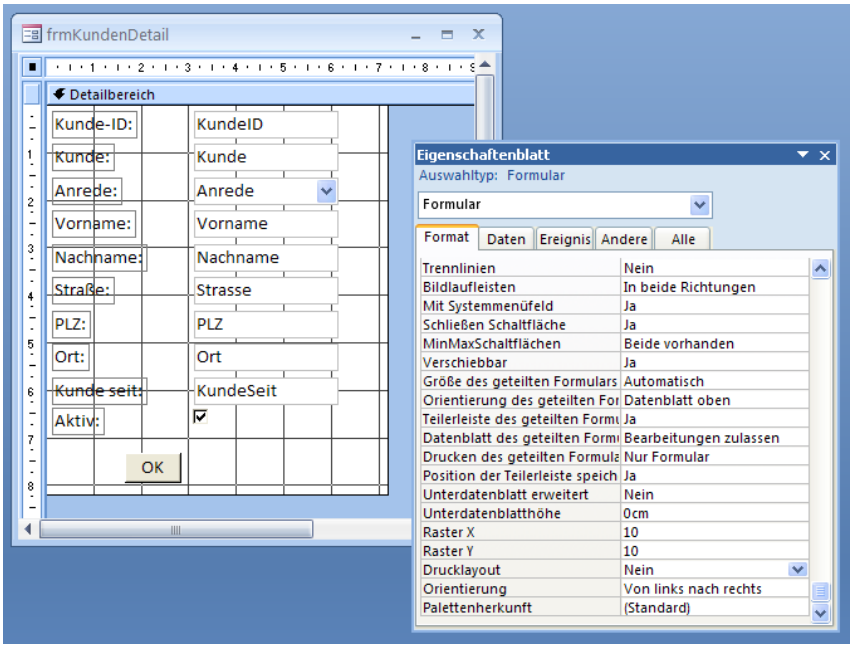

## Access 2010

- Betätigen Sie den Ribbon-Eintrag **Datei|Optionen**.
- Wechseln Sie wie unter Access 2007 zum Bereich **Aktuelle Datenbank** und wählen Sie das gewünschte Formular für die Option **Formular anzeigen** aus.

# Steuerelemente ausrichten und skalieren

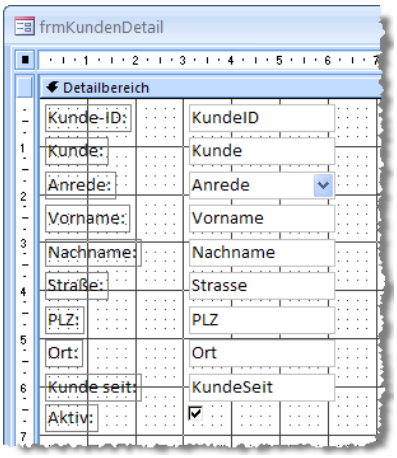
Steuerelemente ausrichten – was soll daran kompliziert sein? Einfach mit der Maus hin- und herschieben, und schon ist man fertig.

Wenn Sie allerdings sehr genau arbeiten möchten und die Größe und Ausrichtungen formularweit übereinstimmen soll, braucht es ein wenig mehr als ein wenig Mausschubserei.




Grundsätzlich sollten Sie wissen, dass es für Formulare und Berichte ein Raster gibt, dass Sie mit dem Menübefehl **Ansicht|Raster** (Access 2003 und älter) beziehungsweise **Anordnen|Einblenden/Ausblenden|Raster** (Access 2007 und jünger) ein- beziehungsweise ausblenden können. Die Abbildung zeigt ein Formular in der Entwurfsansicht mit aktiviertem Raster.

Wenn Sie Steuerelemente langsam mit der Maus bewegen, fällt Ihnen sicher auf, dass diese kleine Sprünge machen – das liegt daran, dass sie an der nächstmöglichen Rasterposition einrasten. Das geschieht aber nicht nur an den Rändern des Rasters, sondern es gibt dazwischen auch noch unsichtbare Rasterlinien. Wieviele Linien zwischen den sichtbaren Linien stecken, stellen Sie mit den Eigenschaften **Raster X** und **Raster Y** ein (das heißt: je größer der Wert für diese Eigenschaften, desto feiner ist das Raster).




Bei kleinen Werten zeigt Access Punkte an, um die Rasterlinien anzudeuten. Die Abbildung zeigt beispielsweise die Rasterpunkte für eine Einstellung von **5** für **Raster X** und **Raster Y**. Wenn Sie nun ein Steuerelement auf dem Formular hin- und herschieben, können Sie deutlicher erkennen, an welchen Stellen dieses einrastet – nämlich immer genau mit der linken oberen Ecke an einem Rasterpunkt.

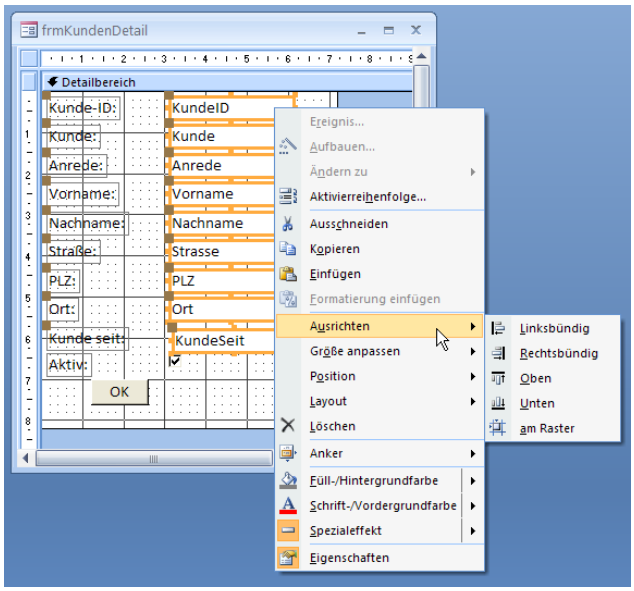
Dieses Verhalten können Sie sich auf mehrere Arten zu Nutze machen:

- Sie können so gut Steuerelemente mit der Maus verschieben und diese auf gleichem Abstand von oben oder links positionieren.
- Sie können Steuerelemente vergrößern und verkleinern und dabei ebenfalls das Raster als Maßeinheit verwenden.
- Sie können das aktuell markierte Steuerelement mit den Cursor-Tasten (**Nach oben**, **Nach unten**, **Nach links** und **Nach rechts**) jeweils um eine Rasterlinie nach oben, unten, links oder rechts verschieben.
- Sie können für das aktuell markierte Steuerelement bei gedrückter **Umschalt**-Taste mit den Cursor-Tasten so die Größe anpassen, dass rechter und unterer Rand jeweils auf einer Rasterlinie landen.

Wenn Sie eine noch feinere Einstellung benötigen, können Sie die Position und die Größe von Steuerelementen sogar pixelweise einstellen:

- Betätigen Sie die Cursor-Tasten bei gedrückter **Strg**-Taste, um die Position um jeweils ein Pixel nach oben, unten, links oder rechts zu verändern.
- Betätigen Sie die Cursor-Taste bei gedrückter **Strg**- und **Umschalt**-Taste, um die Größe pixelweise anzupassen.

## Steuerelemente am Raster ausrichten




Wenn Sie die Steuerelemente einfach in den Entwurf eingefügt haben, ohne besonders auf ihre Position zu achten, können Sie diese automatisch an der nächsten Rasterlinie ausrichten. Dazu markieren Sie die auszurichtenden Steuerelemente, klicken mit der rechten Maustaste darauf und wählen aus dem nun erscheinenden Kontextmenü den Eintrag **Am Raster ausrichten** aus. Die Steuerelemente werden nun am nächsten Rasterpunkt von ihrer linken oberen Ecke aus ausgerichtet.

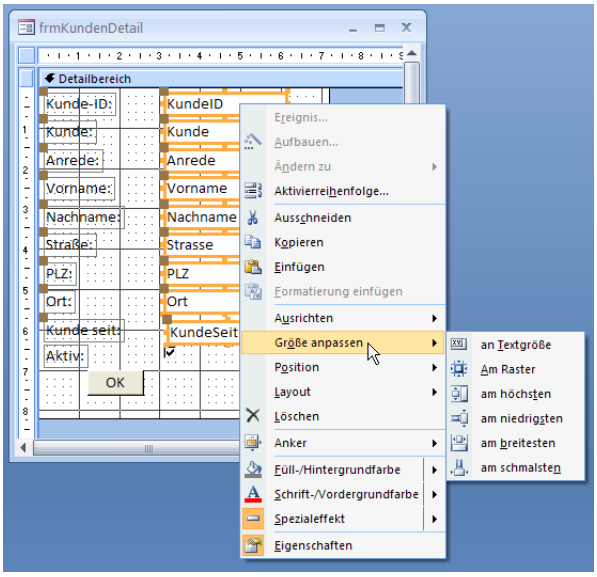
## Steuerelemente aneinander ausrichten

Wenn Steuerelemente gleiche Eigenschaften bezüglich Position oder Größe erhalten sollen, markieren Sie einfach die Steuerelemente, die Sie angleichen möchten, und verwenden die folgenden Möglichkeiten, die sich sämtlich im Kontextmenü (rechte Maustaste) der markierten Steuerelemente befinden:

- **Ausrichten|Linksbündig**: Die Steuerelemente werden an ihrer linken Seite ausgerichtet, wobei das am weitesten links befindliche die Position vorgibt.
- **Ausrichten|Rechtsbündig**, **Ausrichten|Oben**, **Ausrichten|Unten**: wie **Ausrichten|Linksbündig**, nur bezogen auf eine andere Seite

Achtung: Wenn zwei Steuerelemente beispielsweise links ausgerichtet werden sollen und die linke Kante des rechten Steuerelements liegt rechts von der rechten Kante des linken Steuerelements, dann wird das rechte Steuerelement mit der linken Kante genau an die rechte Kante des linken Steuerelements herangeschoben. Einfach, oder?

## Größe von Steuerelementen anpassen



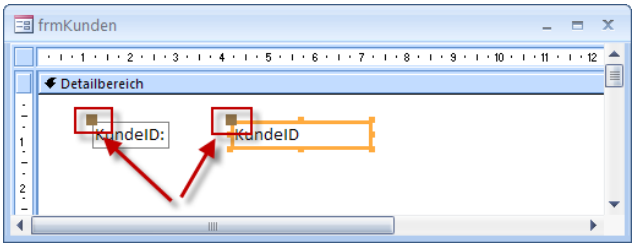

Das Kontextmenü hält auch einige Einträge zum Anpassen der Größe der markierten Steuerelemente bereit. Der Befehl **Größe anpassen|Am Raster** beispielsweise richtet das Steuerelement gleichzeitig mit der linken oberen Ecke am nächsten Rasterpunkt aus und vergrößert es bis zum nächsten rechts unten liegenden Rasterpunkt. Der Befehl **Größe anpassen|An Textgröße** passt die Größe von Steuerelementen mit einer Beschriftung an der Beschriftung an. Bei Steuerelementen wie Textfeldern bezieht sich dies nur auf die Steuerelementhöhe – den zur Laufzeit angezeigten Inhalt kann Access nicht auswerten.

Schließlich gibt es vier Befehle wie **Größe anpassen|Am höchsten**, bei denen die Größe aller markierten Steuerelemente jeweils am höchsten, niedrigsten, breitesten und schmalsten Steuerelement ausgerichtet wird.

**2007 und jünger: Hier gibt es noch weitere Werkzeuge, um Steuerelemente auszurichten. Sie können dort sogenannte gestapelte und tabellarische Layouts für Gruppen von Steuerelementen vorsehen. An dieser Stelle gehen wir jedoch zunächst nicht auf dieses neue Feature ein.**

## Unabhängiges Positionieren von Steuerelement und Bezeichnungsfeld

Wenn Sie ein Steuerelement wie ein Textfeld, ein Kombinationsfeld, ein Kontrollkästchen anlegen, für das automatisch ein Bezeichnungsfeld hinzugefügt wird, möchten Sie beide gegebenenfalls unabhängig voneinander positionieren.




Mit der Maus können Sie dies erledigen, indem Sie auf das etwas größere schwarze Kästchen oben links des markierten Steuerelement klicken und dieses dann an die gewünschte Position verschieben.

Wenn Sie Bezeichnungsfeld oder das »Eltern-Steuerelement« per Tastatur unabhängig voneinander bewegen möchten, müssen Sie deren Beziehung löschen. Dies ist nur möglich, indem Sie das Bezeichnungsfeld ausschneiden (am einfachsten durch markieren des Bezeichnungsfeldes und der Tastenkombination **Strg + X**) und dieses anschließend wieder einfügen (**Strg + V**).

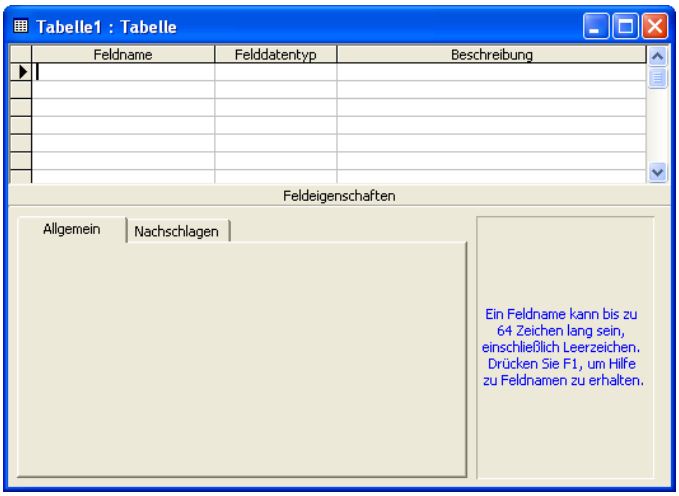
# Tabelle erstellen

Eine Tabelle erstellen Sie am einfachsten auf folgende Art:

**2003 und älter:** Verwenden Sie entweder den Menüeintrag **Einfügen|Tabelle** und wählen Sie im folgenden Dialog **Neue Tabelle** den Eintrag **Entwurfsansicht** aus oder klicken Sie im Bereich **Tabellen** des Datenbankfensters auf den Eintrag **Erstellt eine Tabelle in der Entwurfsansicht**.

**2007 und jünger: Klicken Sie auf den Ribbon-Eintrag Erstellen|Tabellen|Tabellenentwurf.**

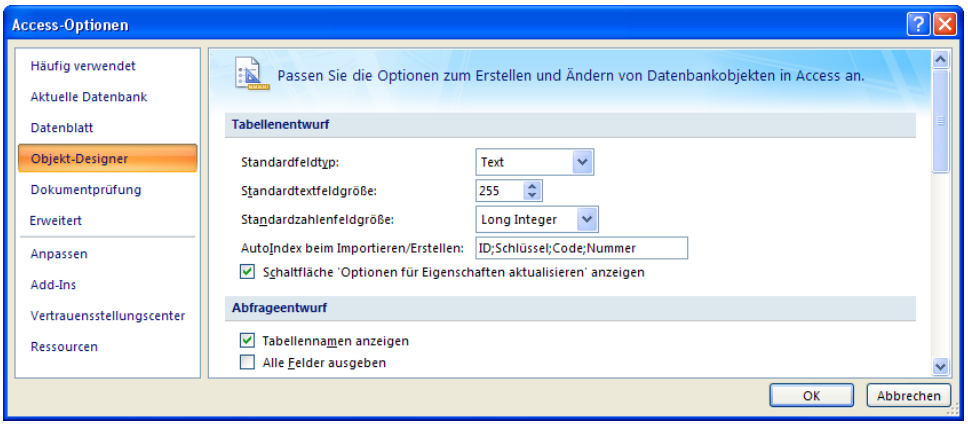
In allen Access-Versionen finden Sie nun die leere Entwurfsansicht vor (Details können je nach Version variieren).




Hier legen Sie die gewünschten Felder an (→*[basics]* **Optionen einstellen**, →*[basics]* **Textfeld anlegen**, →*[basics]* **Zahlenfeld anlegen**, →*[basics]* **Datum/Zeit-Feld anlegen**, →*[basics]* **Ja/Nein-Feld anlegen**)

Als Tabellennamen verwenden Sie das Präfix **tbl** zusammen mit der Mehrzahl der Bezeichnung der in der Tabelle gespeicherten Objekte, also beispielsweise **tblKunden**. Durch die Verwendung des Präfixes können Sie leicht anhand der Bezeichnung zwischen den verschiedenen Objekttypen unterscheiden.

# Tabellenentwurf-Einstellungen anpassen




Wenn Sie eine neue Tabelle anlegen, geschieht das teilweise unter Verwendung von Standardwerten. Dazu gehören die folgenden:

- **Standardfeldtyp**: Feldtyp, der beim Anlegen eines neuen Feldes ausgewählt ist
- **Standardfeldtextgröße**: Anzahl der maximalen Zeichen für neue Textfelder
- **Standardzahlenfeldgröße**: Datentyp für neue Zahlenfelder
- **AutoIndex beim Importieren/Erstellen**: Zeichenketten in Feldnamen, bei denen Access automatisch einen einfachen Index erstellt

Bei älteren Access-Versionen ist **Standardtextfeldgröße** möglicherweise auf einen Wert kleiner **255** eingestellt (meist auf **50**). Bevor Sie irgendwann einmal eine Datenbank nachträglich ändern und gegebenenfalls neu ausliefern müssen, weil ein Benutzer an eine zu klein eingestellte Textfeldgröße gestoßen ist, sollten Sie diese lieber gleich auf den maximalen Wert von **255** einstellen. Dies wirkt sich nicht auf den Speicherbedarf eines solchen Feldes aus: Es wird nur der Speicherplatz für die tatsächlich enthaltenen Zeichen beansprucht.

Auch die Eigenschaft **Autoindex beim Importieren/Erstellen** sollten Sie komplett leeren: Als verantwortungsbewusster Entwickler stellen Sie selbstverständlich alle Indizes selbst ein, die Sie benötigen.

# Tastenkombinationen

Access bietet eine Menge nützlicher Tastenkombinationen. Wir listen diese in diesem Abschnitt auf. Diese Liste wird ständig um die in den Artikeln erwähnten Tastenkombinationen erweitert.

## Allgemeine Tastenkombinationen im Access-Fenster

- **F11**: Datenbankfenster (Access 2003 und älter) beziehungsweise Navigationsbereich (Access 2007 und jünger) ein- und ausblenden
- **Strg + G**: Direktfenster im VBA-Editor öffnen

## Ansichten von Objekten wechseln

- **Strg + Punkt**: Wechselt zur nächsten Ansicht
- **Strg + Komma**: Wechselt zur vorherigen Ansicht

## Tastenkombinationen im Tabellenentwurf

- **F8**: Ausgehend von blinkender Einfügemarke im Entwurfsraster markiert F8 nach einmaliger Betätigung den kompletten Feldinhalt, dann die komplette Zeile und schließlich alle Zeilen.
- **Umschalt + F8**: Wie **F8**, nur umgekehrt.

## Tastenkombinationen im Datenblatt

- **F8**: Ausgehend von blinkender Einfügemarke im Entwurfsraster markiert F8 nach einmaliger Betätigung den kompletten Feldinhalt, dann das komplette Feld, dann die komplette Zeile und schließlich alle Zeilen.
- **Umschalt + F8**: Wie **F8**, nur umgekehrt.

## Tastenkombinationen in Formularen

- **F4**: Eigenschaftsfenster ein-/ausblenden
- **F8**: Datensatzmarkierer aktivieren
- **Strg + S**: Speichern des aktuellen Objekts
- **Alt + F5**: Anzeigen der in der Eigenschaft Standardansicht angegebenen Ansicht
- **Strg + Komma**: Wechseln zur vorherigen Ansicht
- **Strg + Punkt**: Wechseln zur nächsten Ansicht

## Tastenkombinationen für die Zwischenablage

- **Strg + C**: Kopiert den aktuell markierten Text oder das aktuell markierte Objekte
- **Strg + X**: Schneidet den aktuell markierten Text oder das aktuell markierte Objekt aus.
- **Strg + V**: Fügt den Inhalt der Zwischenablage am aktuell markierten Ort ein.

## Tastenkombinationen für Textfelder

- **Umschalt + F2**: Öffnet das Zoomfenster
- **Strg + F2**: Öffnet einen Assistenten, falls verfügbar

## VBA-Editor

- **Strg + G**: Direktfenster aktivieren

- **Alt + F11**: Zwischen VBA-Editor und Access-Fenster wechseln
- **F5**: Aktuell markierte Prozedur starten

# Textfeld anlegen

Ein Textfeld legen Sie an, indem Sie den Namen des Feldes in die Spalte **Feldname** einer leere Zeile einer Tabelle in der Entwurfsansicht schreiben. Der Felddatentyp erhält den Wert **Text**. Die Eigenschaft **Feldgröße** des Textfeldes legt fest, wieviele Zeichen das Feld maximal aufnehmen kann. Da Access für die angegebenen Zeichen keinen Speicherplatz reserviert, sollten Sie immer den maximalen Wert **255** wählen. Wenn diese Anzahl Zeichen nicht ausreicht, verwenden Sie ein Memofeld (→**Memofeld anlegen**).

Sollte ein neues Textfeld in der Entwurfsansicht einer Tabelle nicht gleich auf eine Länge von 255 Zeichen voreingestellt sein, können Sie diese Voreinstellung anpassen (→**Tabellenentwurf-Einstellungen anpassen**).

# Unterformular zu Formular hinzufügen

Es gibt verschiedene Wege, um ein Unterformular in ein Hauptformular zu integrieren. Der einfachste sieht so aus: Erstellen Sie, soweit noch nicht vorhanden, ein Haupt- und ein Unterformular. Wir nennen diese beiden nachfolgend der Einfachheit halber **frm** und **sfm**.

Öffnen Sie das Hauptformular **frm** in der Entwurfsansicht. Platzieren Sie das Hauptformular und das Datenbankfenster nebeneinander. Wechseln Sie im Datenbankfenster zur Registerseite **Formulare**. Ziehen Sie dann per Drag and Drop das Formular **sfm** aus dem Datenbankfenster in den Detailbereich des Hauptformulars **frm** (so geht es unter Access 2003 und älter – ab Access 2007 ist es noch einfacher: Da der Navigationsbereich niemals Formulare überdeckt, brauchen Sie die beiden noch nicht einmal entsprechend zu positionieren).

Die zweite Variante funktioniert, ausgehend von den vorhandenen Formularen **frm** und **sfm**, wie nachfolgend beschrieben. Zum besseren Verständnis dieser Variante sei darauf hingewiesen, dass die Konstruktion aus Haupt- und Unterformular mehr benötigt als nur die beiden Formulare: Das Unterformular gelangt nämlich nicht selbst als Steuerelement in das Formular, sondern es kommt noch ein Unterformularsteuerelement hinzu, das quasi als Container fungiert und das eigentliche Unterformular aufnimmt. Dieses Wissen wird wichtig, wenn Sie einmal per VBA auf ein Unterformular oder die darin enthaltenen Steuerelemente zugreifen möchten.

Hier aber nun die Beschreibung der zweiten Variante zum Hinzufügen eines Unterformulars zu einem Formular:

- Öffnen Sie das Hauptformular **frm** in der Entwurfsansicht.
- Fügen Sie ein Steuerelement des Typs **Unterformular/-bericht** zum Detailbereich hinzu. Vorteil hier: Sie können das Unterformular gleich in der richtigen Größe aufziehen.
- Stellen Sie die Eigenschaft **Herkunftsobjekt** auf den Namen des Unterformulars ein, hier **sfm**.

## Name des Unterformularsteuerelements

Wenn Sie das Unterformular aus dem Datenbankfenster beziehungsweise dem Navigationsbereich per Drag and Drop zum Hauptformular hinzufügen, erhält das dabei erstellte Unterformularsteuerelement den gleichen Namen wie das als Unterformular verwendete Formular. Dies führt in der Praxis oft zu Verwechslungen, da Sie nun das Unterformularsteuerelement und das darin angezeigte Unterformular nicht mehr aufgrund des Namens unterscheiden können!

Sie sollten also, egal auf welche Weise Sie das Unterformular zum Hauptformular hinzufügen, dem Unterformularsteuerelement einen individuellen Steuerelementnamen geben. Wenn das Formular nur ein Unterformular enthält, würde sich ganz einfach **sub** anbieten. Anderenfalls können Sie das Präfix um eine entsprechende Bezeichnung erweitern, beispielswiese **subProjekte** und **subKontakte**.

# VBA-Befehle: MsgBox

Die Funktion zeigt einen Dialog mit einer Meldung an. Der Dialog kann mit unterschiedlichen Schaltflächen und Symbolen ausgestattet werden.

## Syntax

```
MsgBox(<Prompt>, [<Buttons>], [<Title>], [<HelpFile>], [<Context>])
```

## Parameter

- **Prompt** (Datentyp **Variant**): Erwartet die im Dialog anzuzeigende Meldung.
- **Buttons** (Datentyp **VbMsgBoxStyle**): Erwartet eine Konstante der Enumeration **vbMsgBoxStyle**, die Informationen über die anzuzeigenden Schaltflächen liefert und das anzuzeigende Symbol festlegt. Mehrere dieser Konstanten können auch mit dem Bool-Operator **Or** verknüpft werden.
- **Title** (Datentyp **Variant**): Erwartet den im Dialog anzuzeigenden Titel.
- **HelpFile** (Datentyp **Variant**): Erwartet den Namen einer Hilfedatei, die kontextsensitive Hilfe für das Meldungsfenster bereitstellt.
- **Context** (Datentyp **Variant**): Erwartet die Kontextnummer einer Hilfeseite in der unter Helpfile angegebenen Datei.

## Beispiele

Die folgende Routine fragt zunächst das Ergebnis eines ersten Meldungsfensters ab und zeigt abhängig vom Ergebnis unterschiedliche Meldungsfenster an.

```
Public Sub MsgBoxBeispiel()
    Dim lng As VbMsgBoxResult
    lng = MsgBox("Klicken Sie auf eine der Schalt ächen.", _
        vbYesNoCancel + vbCritical + vbDefaultButton1, "Beispielmeldungsfenster")
    Select Case lng
        Case vbYes
            MsgBox "Sie haben 'Ja' angeklickt!", vbOKOnly Or vbExclamation
        Case vbNo
            MsgBox "Sie haben 'Nein' angeklickt.", vbOKOnly Or vbInformation
        Case vbCancel
            MsgBox "Haben Sie 'Abbrechen' angeklickt?", vbOKOnly Or vbQuestion
    End Select
End Sub
```

# VBA-Editor öffnen

Den VBA-Editor können Sie versionsunabhängig auf die folgenden beiden Arten öffnen:

- Tastenkombination **Alt + F11**: Öffnet den VBA-Editor.
- Tastenkombination **Strg + G**: Öffnet den VBA-Editor und setzt den Fokus auf das Direktfenster

Mit der Tastenkombination **Alt + F11** gelangen Sie übrigens vom VBA-Editor wieder zurück zum Access-Fenster. Die gleiche Tastenkombination können Sie auch in den übrigen Office-Anwendungen einsetzen, um zum VBA-Fenster der jeweiligen Anwendung zu gelangen.

**2003 und älter:** Wählen Sie den Menüeintrag **Extras|Makro|Visual Basic Editor** aus.

**2007 und jünger:** Wählen Sie den Ribbon-Eintrag **Datenbanktools|Makro|Visual Basic** aus.

# VBA-Modul anlegen

Es gibt verschiedene Modultypen unter Access: Standardmodule und Klassenmodule. Klassenmodule können wiederum allein stehen oder zu einem Formular oder Bericht gehören.

## Module vom Access-Fenster aus anlegen

Alleinstehende Standard- und Klassenmodule legen Sie versionsabhängig auf eine der folgenden Arten an:

**2003 und älter:** Standardmodule legen Sie vom Bereich **Module** des Datenbankfensters aus durch Betätigen der Schaltfläche **Neu** oder über den Menübefehl **Einfügen|Modul** an.

Klassenmodule legen Sie über den Menüeintrag **Einfügen|Klassenmodul** an.

**2007:** Standardmodule legen Sie über den Ribbon-Eintrag **Erstellen|Andere|Makro|Modul** an, Klassenmodule über **Erstellen|Andere|Makro|Klassenmodul**.

**2010:** Standardmodule legen Sie über den Ribbon-Eintrag **Erstellen|Makros und Code|Modul** und Klassenmodule über **Erstellen|Makros und Code|Klassenmodul** an (hier wurde das Ribbon gegenüber 2007 etwas aufgeräumt).

## Module vom VBA-Editor aus anlegen

Vom VBA-Editor aus geschieht das Anlegen eines Moduls versionsübergreifend mit einer der folgenden Methoden:

- Standardmodule: Menüeintrag **Einfügen|Modul**
- Klassenmodule: Menüeintrag **Einfügen|Klassenmodul**

Alternativ aktivieren Sie den Projekt-Explorer über den Menüeintrag **Ansicht|Projekt-Explorer** oder über die Tastenkombination **Strg + R** und wählen aus dem Kontextmenü einen der Einträge **Einfügen|Modul** oder **Einfügen|Klassenmodul** aus.

# VBA-Routine schreiben

Zum Schreiben eines VBA-Programms öffnen Sie zunächst den VBA-Editor (→[basics] **VBA-Editor öffnen**) und wählen das Modul aus, dem Sie das VBA-Programm hinzufügen möchten. Gegebenenfalls wollen Sie zunächst ein neues Modul anlegen, was Sie wie in →[basics] **VBA-Modul anlegen** erledigen.

# VBA-Prozedur ausführen

Je nachdem, ob eine VBA-Prozedur als **Private** oder **Public** deklariert ist und in was für einem Modultyp Sie diese erstellt haben, gibt es verschiedene Möglichkeiten für die Ausführung.

## VBA-Prozedur aus Standardmodul direkt ausführen

Eine einfache VBA-Prozedur ohne Parameter in einem Standardmodul können Sie grundsätzlich durch Platzieren der Einfügemarke irgendwo innerhalb der VBA-Prozedur und anschließendes Durchführen einer der folgenden Varianten ausführen:

- Tastenkombination **F5**
- Symbolleistenschaltfläche **Sub/Userform** ausführen
- Menüeintrags **Ausführen|Sub/Userform ausführen**

Diese Vorgehensweise gelingt nicht, wenn die VBA-Prozedur einen oder mehrere Parameter verwendet.

## VBA-Prozedur im Standardmodul über das Direktfenster ausführen

Wenn die VBA-Prozedur mindestens einen Parameter enthält, können Sie diese nicht mehr direkt starten, sondern müssen den Aufruf irgendwo niederschreiben. Am einfachsten gelingt dies über das Direktfenster, dass Sie mit **Strg + G** aktivieren. Angenommen, Sie möchten die folgende Prozedur starten, die schlicht den Wert des Parameters in einem Meldungsfenster ausgeben soll:

```
Public Sub MeldungsfensterIndividuell(strText As String)
    MsgBox strText
End Sub
```

Dann schreiben Sie beispielsweise folgenden Aufruf in das Direktfenster und betätigen die Eingabetaste:

```
MeldungsfensterIndividuell "Beispieltext"
```

Sie können auch die **Call**-Anweisung voranstellen, müssen die Parameter dann aber in Klammern einfassen:

```
Call MeldungsfensterIndividuell("Beispieltext")
```

Natürlich können Sie auch VBA-Prozeduren ohne Parameter auf diese Weise über das Direktfenster starten.

Beachten Sie, dass die Prozedur mit dem Schlüsselwort **Public** versehen sein. Als **Private** deklarierte Prozeduren können nicht über das Direktfenster aufgerufen werden.

## VBA-Prozedur in Klassenmodul ausführen

Informationen zu diesem Thema folgen.

# Zahlenfeld anlegen

Ein Zahlenfeld legen Sie an, indem Sie in die Spalte **Feldname** einer leeren Zeile in der Entwurfsansicht einer Tabelle den Feldnamen eintragen und den **Datentyp** auf **Zahl** festlegen.

Entscheidend ist hier der Wert der Eigenschaft **Feldgröße** des Feldes. Er legt fest, wie groß die eingetragenen Zahlenwerte werden dürfen und ob das Feld nur ganze Zahlen oder auch Dezimalanteile erfassen kann. Hier die wichtigsten Feldgrößen für Zahlenfelder im Überblick:

- **Zahl (Byte)**: Zahlen von 0 bis 255, 1 Byte
- **Zahl (Integer)**: Zahlen von -32.768 bis 32.767, 2 Byte
- **Zahl (Long Integer)**: Zahlen von -2.147.483.648 bis 2.147.483.647, 4 Byte. Einsatzbereich: Vorrangig Fremd- und Primärschlüsselfelder, andere Felder, die Ganzzahlen in dieser Größenordnung benötigen
- **Zahl (Single)**: Gleitkommazahlen einfacher Genauigkeit von -3,402823E38 bis -1,401298E-45 für negative Zahlen, 1,401298E-45 bis 3,402823E38 für positive Zahlen und 0, 4 Byte. Einsatzbereich: Zahlen mit Nachkommastellen, beispielsweise Prozentzahlen oder Währungen.
- **Zahl (Double)**: Gleitkommazahlen doppelter Genauigkeit von -1,79769313486232E308 bis -4,94065645841247E-324 für negative Zahlen, 4,94065645841247E-324 bis 1,79769313486232E308 für positive Werte und 0, 8 Byte
- **Zahl (Dezimal)**: Zahlen als Potenzen zur Basis 10. Für Zahlen ohne Nachkommastellen +/-79.228.162.514.264.337.593.543.950.335, für Zahlen mit 28 Nachkommastellen +/-7,9228162514264337593543950335

Für Primärschlüsselfelder und die entsprechenden Fremdschlüsselfelder verwendet man normalerweise die Feldgröße **Zahl (Long Integer)**. In anderen Fällen hängt es von den Anforderungen ab (Größe der Zahlen, Dezimalstellen oder keine).

Wenn ein Zahlenfeld später als Fremdschlüsselfeld verwendet werden soll, stellen Sie seine Eigenschaft **Standardwert** auf eine leere Zeichenfolge ein, was dem Wert **Null** entspricht. Anderenfalls kann es zu Problemen bei der Einrichtung referentieller Integrität kommen.

Außerdem muss ein Fremdschlüsselfeld genau den gleichen Datentyp wie das Primärschlüsselfeld aufweisen, aus dem es seine Werte bezieht. Beim Zahlenfeld bedeutet dies, dass auch die Feldgröße identisch sein muss.

Leeren Sie die Eigenschaft **Standardwert** grundsätzlich immer, wenn in der Beschreibung des Tabellenentwurfs nichts anderes angegeben ist.

Hinzugefügt: Ausgabe 3/2010 – Zuletzt geändert: 6.6.2010