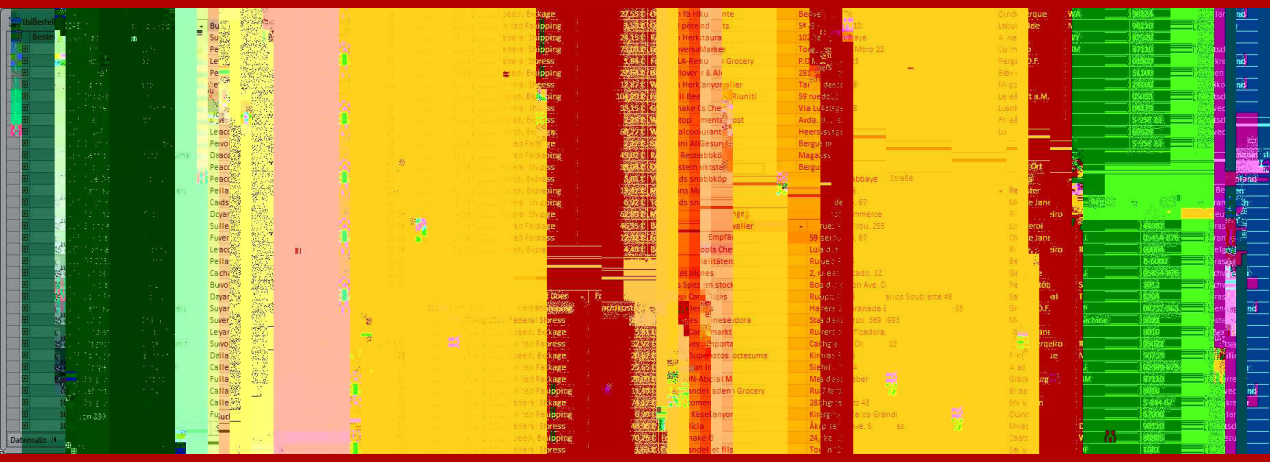
DAS ACCESS-ENTWICKLERBUCH



ACCESS 2010/2013 FÜR FORTGESCHRITTENE

ANDRÉ MINHORST

André Minhorst Verlag

André Minhorst

Access 2010

Das Grundlagenbuch für Entwickler

André Minhorst: Access 2010 – Das Grundlagenbuch für Entwickler

ISBN 978-3-944216-04-1

Borkhofer Straße 17, 47137 Duisburg/Deutschland

1. Auflage 2015

Lektorat André Minhorst
Korrektur Rita Klingenstein
Cover/Titelbild André Minhorst
Typographie, Layout und Satz André Minhorst
Herstellung André Minhorst
Druck und Bindung Kösel, Krugzell (www.koeselbuch.de)

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie. Detaillierte bibliografische Daten finden Sie im Internet unter http://dnb.d-nb.de.

Inhaltsverzeichnis

Vorwort

Ein Buch mit einem Umfang von mehr als 1.000 Seiten ist immer eine Menge Arbeit. Ich weiß nicht, was mehr Zeit kostet: ein solches Buch komplett neu zu schreiben oder, wie in diesem Fall, ein bestehendes Buch an die neue Version einer Software anzupassen. Überarbeitet man ein Buch, muss man wirklich jeden Satz unter die Lupe nehmen und jeden Screenshot neu erstellen, jeder Codeschnipsel muss getestet werden und außerdem muss man die neue Version der Software auch noch genau auf die neuen Features hin untersuchen. Im Falle von Access bietet der Hersteller nicht besonders viel Unterstützung: Microsoft verteilt zwar großzügig Informationen über die Vorteile der wichtigsten neuen Funktionen. Was sich aber im Detail und vor allem hinter den Kulissen getan hat, verrät man nicht. Das gilt insbesondere, wenn der Anwender fortgeschritten ist und sich nicht lange mit neuen Templates oder Wizards aufhalten will. Diese bieten

vielleicht für Einsteiger eine willkommene Basis für die Erstellung einer Anwendung. Fortgeschrittene Benutzer und professionelle Entwickler wollen aber vielleicht eher wissen, wie man solche Templates und Wizards selbst programmiert oder wie man die neuen Funktionen per VBA steuern kann. Also bleibt eine Menge Forschungsarbeit übrig: Für die Entwickler, die schnell Herr der neuen Version werden wollen, um die Softwareentwicklung auf Basis von Access 2010 in ihr Portfolio aufnehmen zu können. Und für die Autoren, die in Büchern, Magazinen und im Web über die neuen Funktionen berichten wollen.

Zum Zeitpunkt der Fertigstellung dieses Buches gibt es Access 2010 bereits über ein Jahr. Microsoft hat vor allem ein Feature angepriesen, und zwar die Webdatenbanken. Access-Anwendungen, die gegenüber herkömmlichen Datenbanken mit eingeschränktem Funktionsumfang erstellt werden und Makros statt VBA zur Automatisierung verwenden, sollen komplett ins Web gestellt werden können. Die Veröffentlichung einer solchen Anwendung ist jedoch nicht ganz trivial: Voraussetzung ist ein Webserver mit SharePoint 2010 und entsprechenden Benutzerlizenzen – nicht gerade eine kostengünstige Kombination.

Es gibt aktuell nur einen Dienstleister, bei dem Sie Ihre Webdatenbank hosten können (mehr dazu im Kapitel »Webdatenbanken«). Nach einer kostenlosen Testphase von 30 Tagen werden hier je nach Anzahl der Datenbanken und Benutzer zwischen 20,- und 100,- EUR fällig. Das Einrichten eines eigenen Servers mit SharePoint 2010 und den notwendigen Access Services ist so kostspielig, dass wohl nur größere Unternehmen diese Möglichkeit haben. Microsoft startet auch erst während der Veröffentlichung dieses Buchs die Betaphase von Office 365, einem Dienst, der neben den Onlineversionen von Word, Excel und Co. auch Access Services anbietet. Ein Test im Rahmen dieses Buchs war leider nicht mehr möglich.

Ob die Webdatenbanken nun einschlagen, weiß ich nicht: Die erste Version macht schon einen ganz guten Eindruck, obwohl man als Access-Entwickler natürlich sehr stark eingeschränkt wird. Es wird sich zeigen, ob Microsoft mit der nächsten Access-Version nachlegt oder ob die Webdatenbanken genau wie die mit Access 2000 eingeführten Datenzugriffsseiten bald wieder eingestampft werden. Tatsache ist, dass Sie wie mit Access gewohnt schnell erste Ergebnisse liefern und Daten über das Web bearbeiten können – viel schneller wahrscheinlich, als es mit den anderen Web-Programmiersprachen möglich ist. Wenn es bald mehr Anbieter gibt, die Webdatenbanken hosten, und Microsoft die technischen Möglichkeiten der Webdatenbanken noch erweitert, könnte es eine Erfolgsgeschichte werden.

Access 2003, 2007 und 2010

Mit Access 2007 hat Microsoft gegenüber Access 2003 einige auffällige Änderungen eingeführt, die in Access 2010 beibehalten oder sogar erweitert wurden – beispielsweise

das Ribbon. Es scheint so, als ob viele Access-Entwickler einen Bogen um Access 2007 gemacht haben, weil sie sich schlicht nicht von den liebgewonnenen Features älterer Access-Versionen trennen wollen. Das Buch weist daher nicht nur auf die Neuheiten von Access 2010, sondern auch auf wichtige Neuerungen von Access 2007 gegenüber Access 2003 hin (zum Beispiel das Ribbon, Anlagefelder et cetera).

Hinweis zu Client-Datenbanken/Webdatenbanken

Das Buch geht im Kapitel »Webdatenbanken« speziell auf die Eigenschaften von Webdatenbanken ein und behandelt Besonderheiten, die bei ihrer Erstellung zu beachten sind, das Kapitel »Makros« ist auch speziell auf Webdatenbanken zugeschnitten.

Nur die Abhandlungen zu Datenmakros sind auch für professionelle Client-Datenbanken interessant. Das bedeutet im Umkehrschluss, dass die übrigen Kapitel etwa zu Tabellen, Abfragen oder Formularen sich ausschließlich mit Client-Datenbanken beschäftigen.

So brauchen Sie sich die Informationen zu Webdatenbanken nicht aus mehreren Kapiteln zusammenzusuchen, sondern finden alles in den zwei genannten Kapiteln.

Neues in Access 2010

Sollten Sie vorrangig nach Neuigkeiten unter Access 2010 suchen, werden Sie hier fündig:

» Webdatenbanken: Lesen Sie die beiden Kapitel »Makros« ab Seite 591 und »Webdatenbanken« ab Seite 637.

» Ribbon: Hier gibt es kleinere Änderungen, vor allem das Aktivieren von Tabs per VBA. Alles weitere unter »Ribbon-Tab per VBA einstellen« ab Seite 739.

» Backstage-Bereich: Die neuen Möglichkeiten werden Sie selbst erkunden wollen, alles zur Anpassung finden Sie unter »Backstage« ab Seite 777.

» IntelliSense: Es gibt nun IntelliSense auch für Ausdrücke, also etwa in Eigenschaften oder beim Entwurf von Abfragen. Tipp: *Strg + Leertaste* öffnet die IntelliSense-Liste, wenn aktuell verfügbar.

» Neuer Ausdrucks-Generator: Der Ausdrucks-Generator wurde überarbeitet und erweitert. Auch hier wurde die IntelliSense-Funktion hinzugefügt.

» Makros: Alles, was Sie zu diesem Objekttyp in Zusammenhang mit Webdatenbanken wissen müssen, erfahren Sie unter »Makros« ab Seite 591.

» Dort lernen Sie auch die neuen Datenmakros kennen, die das Access-Pendant zu den Triggern unter SQL Server darstellen (»Einführung in Datenmakros« ab Seite 612).

» Datentyp *Berechnet*: In Tabellen können Sie nun berechnete Felder unterbringen. Alles hierzu unter »Der Felddatentyp Berechnet« ab Seite 43.

» Navigationssteuerelement: Mit diesem Steuerelement organisieren Sie verschiedene Elemente einer Anwendung in einem Formular. Mehr dazu unter »NavigationControl und NavigationButton« ab Seite 279.

» Webbrowser-Steuerelement: Dieses Steuerelement gibt es erstmalig als eingebautes Steuerelement, das Sie auch an ein Feld einer Tabelle binden können (siehe »Das WebbrowserControl-Steuerelement« ab Seite 286).

» Designs: Sie können nun Designs aus Farben und Schriftarten definieren, die Sie per Knopfdruck Formularen, Berichten und Steuerelementen zuordnen können (»Farben und Schriftarten per Design festlegen« ab Seite 176).

» Bedingte Formatierung: Die bedingte Formatierung wurde stark überarbeitet. Sie können nun 50 Formate je Steuerelement festlegen und Werte als Balkendiagramme darstellen (»Bedingte Formatierung« ab Seite 298).

Dankeschön

Vielen Dank an:

» meine Lektorinnen bei Addison-Wesley, Sylvia Hasselbach und Brigitte Alexandra Bauer-Schiewek,

» Martha Kürzl-Harrison für die Feinabstimmung von Satz und Layout,

» Sascha Trowitzsch für das penibelste und beste Fachlektorat,

» Rita Klingenstein für den Kampf gegen die ewig gleichen Fehler,

» Tchibo »Milde Sorte« für den Kaffee-Genuss,

» And One, Nine Inch Nails und speziell Trent Reznor für den richtigen Arbeitstakt,

» Anja, Maja und Lena für die ständige Erinnerung daran, dass es Wichtigeres gibt als Bücher schreiben, und

» an Sie, lieber Leser, weil ich ohne Sie etwas anderes machen müsste.

Und jetzt: Viel Spaß beim Lesen!

André Minhorst

1

Tabellen und Datenmodellierung

Dieses Kapitel zeigt zunächst die grundlegenden Techniken für das Anlegen von Tabellen. Erwarten Sie aber nicht, dass Sie alle x Varianten vorgesetzt bekommen, mit denen Sie Tabellen, Felder und Indizes in Form bringen – es gibt wesentlich bedeutendere Informationen, die Sie dringend kennen müssen, wenn Sie eine Datenbank mit den enthaltenen Tabellen aufsetzen und auch lange Spaß daran haben wollen. Das Konzipieren des richtigen Datenmodells einer Datenbank wird Sie weit länger aufhalten, als es das Erlernen der grundlegenden Techniken erfordert. Das Buch geht daher in den ersten Abschnitten so kurz wie möglich darauf ein, wie Sie Tabellen, Felder, Indizes und Verknüpfungen anlegen, und vermittelt anschließend das notwendige Hintergrundwissen zu diesen Techniken, wobei der Fokus auf den neuen Features von Access 2007 und Access 2010 liegt. Grundlegendere Informationen erhalten Sie etwa in der Onlinehilfe unter Suchbegriffen wie »Einführung in Tabellen«.

Wenn Sie sich schon mit der Benutzeroberfläche vertraut gemacht haben oder sich intuitiv darin zurechtfinden, können Sie die folgenden Abschnitte überspringen und sich direkt mit 1.2 beschäftigen. Wenn Sie die mit Access 2007 eingeführten Features jedoch noch nicht kennen, sollten Sie die ersten Abschnitte dieses Kapitels auf keinen Fall auslassen: Es gibt nämlich einige Neuheiten seit Access 2007, die Sie kennen sollten, gerade wenn Sie direkt von Access 2003 zu Access 2010 wechseln.

Im mittleren Teil des Kapitels erfahren Sie, wie Sie die Anforderungen an die geplante Anwendung in ein adäquates Datenmodell umsetzen. Dabei ist es wichtig, dass die Daten den Objekten der realen Welt entsprechend abgebildet und dazu in Relation gesetzt werden. Dabei helfen die Normalisierungsregeln, die manch einer vielleicht schon intuitiv einsetzt, und eine konsistente Benennung von Tabellen und Tabellenfeldern.

Damit Sie nicht nur mit trockener Theorie abgespeist werden, finden Sie im hinteren Teil des Kapitels eine Menge Beispiele für Datenmodelle. Diese können Sie als Basis für eigene Anwendungen oder auch nur als Anregung verwenden.

BEISPIELDATENBANKEN

Den Download mit dem Beispielen finden Sie unter *www.acciu.de/aeb2010*. Die Datenbank zu diesem Kapitel heißt *TabellenUndDatenmodellierung.accdb*.

Hintergrund dieses Kapitels ist die Tatsache, dass viele (angehende) Entwickler ins kalte Access-Wasser geworfen werden und keine oder wenig Erfahrung in der Datenmodellierung haben.

Zudem findet sich in den meisten Grundlagen-Büchern zu Access meist nur eine Beispielanwendung mit Datenmodell – und das hält dann im ganzen restlichen Teil des Buches als Grundlage für Abfrage-, Formular-, Berichts- und VBA-Beispiele her.

Zielsetzung

Dieses Kapitel hat verschiedene Ziele:

- Für Einsteiger und Access 2007/2010-Neulinge: Kurze Einführung in die Techniken zur Datenmodellierung (Tabellen, Felder, Indizes, Verknüpfungen)
- Einführung einer Konvention für die Namen von Tabellen und Feldern
- Normalformen: Wozu dienen sie und wie normalisiert man ein Datenmodell?
- Begriffsklärung (Beziehungsarten, referentielle Integrität, Primärschlüssel, Fremdschlüssel)
- Erläuterung der Beziehungsarten anhand praktischer Beispiele

» Vermittlung eines Gefühls für die jeweils richtige Beziehungsart anhand einiger Datenmodelle verschiedener Anwendungen

1.1 Techniken zur Datenmodellierung

Eines vorneweg: Sie sollten in diesem Kapitel eigentlich nicht erfahren, wie Sie Tabellen in der Tabellenansicht von Access erstellen – die ist, so dachte der Autor dieser Zeilen zunächst, nur für Anwender, die nicht wissen, welchen Datentyp die von ihnen eingegebenen Daten haben, und Access die Festlegung des Datentyps überlassen möchten. Genau genommen ist diese Ansicht die erweiterte Datenblattansicht, die Sie beim Anlegen einer neuen Tabelle über den Ribbon-Eintrag *Erstellen|Tabellen|Tabelle* erwartet (siehe Abbildung 1.1). Wenn Sie möchten, können Sie diese Funktion in den Access-Optionen unter *Aktuelle Datenbank|Anwendungsoptionen|Entwurfsänderungen für Tabellen in der Datenblattansicht aktivieren (für diese Datenbank)* deaktivieren. Das wirkt sich übrigens erst nach dem nächsten Öffnen der Datenbank aus. Vor dem Hintergrund, dass es in den neuen Webdatenbanken keine Entwurfsansicht für Tabellen gibt, sollten Sie aber vielleicht ruhig einmal die eine oder andere Tabelle in der Datenblattansicht erstellen. In diesem Fall liefern die beiden Ribbon-Tabs Felder und Tabelle alle notwendigen Funktionen.

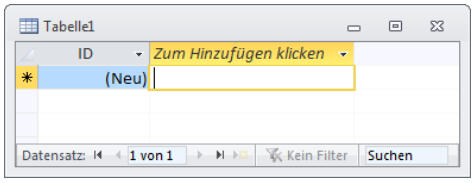


Abbildung 1.1: Access bietet die Möglichkeit, neue Felder ohne Angabe eines Datentyps anzulegen – das ist natürlich nichts für professionelle Entwickler

Keine Assistenten

In diesem Buch werden Sie es mit keinem einzigen Assistenten zu tun bekommen – den Umgang mit Assistenten können Sie sich, wenn Sie möchten, selbst aneignen, indem Sie diese einfach ausprobieren. Wenn Sie die in diesem Buch vorgestellten manuellen Vorgehensweisen kennen, wissen Sie auch, was die Assistenten beherrschen, und können diese gefahrlos einsetzen.

1.1.1 Tabellen anlegen

Zum Anlegen einer neuen Tabelle betätigen Sie den Ribbon-Eintrag *Erstellen|Tabellen|Tabellenentwurf* oder die Tastenkombination *Alt, L, T, W* (was bedeutet, dass Sie nacheinan-

der die Tasten *Alt*, *L*, *T* und *W* betätigen müssen – nach dem Drücken von *Alt* erscheinen im Ribbon die Buchstabenkürzel zu jedem Befehl, aber das werden Sie dann selbst sehen ...). Es erscheint eine neue, leere Tabelle in der Entwurfsansicht, in deren Zeilen Sie die Definition je eines Feldes eintragen. Anschließend führen Sie die folgenden Schritte durch:

» Fügen Sie Namen und Datentypen der Felder sowie gegebenenfalls ihre Beschreibung ein.

» Legen Sie die Schlüssel und Indizes für die Tabelle fest.

» Weisen Sie der Tabelle weitere Eigenschaften zu.

» Speichern Sie die Tabelle unter einem sinnvollen Namen.

Die folgenden Abschnitte beschreiben diese Aufgaben im Detail.

1.1.2 Felder hinzufügen

Tabellen bestehen nicht wie bei Excel aus Zeilen und Spalten, sondern aus Feldern und Datensätzen. Dabei unterscheiden sich nicht nur die Bezeichnungen, sondern auch die Funktionen: In Access enthält ein Datensatz ein Objekt eines bestimmten Typs und die Felder enthalten Werte für dessen Eigenschaften. Für jedes Feld müssen Sie einen Feldnamen mit maximal 64 Zeichen sowie einen Felddatentyp festlegen. Informationen über die richtige Wahl von Feldnamen finden Sie unter »Feldnamen« auf Seite 67, die Felddatentypen stellt Tabelle 1.1 vor.

Sie sollten auch für jedes Feld eine Beschreibung eingeben. Diese wird später, wenn Sie die Daten der Tabelle in einem Formular anzeigen, in der Statusleiste eingeblendet. Außerdem können Sie damit anderen Entwicklern die Arbeit erleichtern, wenn diese Ihre Anwendung verstehen oder anpassen wollen. Bei der Auswahl des Felddatentyps sollten Sie folgende Faktoren berücksichtigen:

» Schätzen Sie ab, welche Länge Texte haben. Haben Sie kurze (bis 255 Zeichen) oder lange Texte? Verwenden Sie je nach Antwort entweder ein Feld mit dem Datentyp *Text* oder *Memo*.

» Welchen Wertebereich nehmen Zahlenfelder ein? Rechnen Sie mit den größten zu erwartenden Bereichen und wählen Sie einen der Zahl-Datentypen aus. Beachten Sie auch, dass manche Datentypen nur ganze Zahlen und manche auch Dezimalzahlen akzeptieren. Wichtig ist der Unterschied zwischen Gleitkommazahlen und Festkommazahlen. Die Nachkommastellen werden bei Gleitkommazahlen durch Brüche von Zweierpotenzen dargestellt, wodurch diese oft ungenau sind. Festkommazahlen werden schlicht als Ganzzahlen gespeichert und durch eine Zehnerpotenz geteilt – für vier Stellen hinter dem Komma also beispielsweise durch 10^4.

» Bei Datums- und Währungsangaben fällt die Auswahl nicht schwer: Die Datentypen *Datum/Zeit* und *Währung* helfen weiter. Verwenden Sie für Währungsangaben niemals den Datentyp *Double*, da dies in Berechnungen zu Rundungsfehlern führen kann.

» Für Primärschlüssel verwenden Sie in der Regel einen *Autowert* mit dem Untertyp *Long Integer*. Nur, wenn Sie mit verteilten Anwendungen arbeiten und die Daten zusammenführen müssen, macht der Untertyp *Replikations-ID* Sinn.

» Binäre Daten wie Dateien (insbesondere Bilder) können Sie seit Access 2007 nicht nur in *OLE-Objekt*-Feldern, sondern auch in Feldern des Datentyps *Anlage* speichern. Die Vor- und Nachteile erfahren Sie unter »Bilder und binäre Dateien« ab Seite 675.

» Wenn Sie für ein Feld ein Nachschlagefeld bereitstellen wollen, mit dem der Benutzer Daten aus einer vorgegebenen Liste oder einer anderen Tabelle auswählen soll, können Sie den Nachschlage-Assistenten für die Einrichtung verwenden. Im Grunde ist *Nachschlage-Assistent* aber kein Datentyp, als Datentyp richtet Access den zuvor ausgewählten Datentyp ein. Sie erfahren aber weiter unten, wie Sie die notwendigen Eigenschaften für den Einsatz von Nachschlagefeldern (auch mehrwertiger) von Hand einrichten können.

» Es gibt Sonderfälle, bei denen Sie einen anderen Datentyp als den offensichtlichen verwenden: Wenn Sie etwa Postleitzahlen eingeben und nur die reinen Zahlenwerte ohne das Länderkürzel (also 47137 statt D-47137) verwenden, sollten Sie ein Textfeld und nicht etwa den Felddatentyp *Zahl (Long)* auswählen. Der Grund: Manche Postleitzahlen enthalten eine führende 0, und die fällt in Zahlenfeldern natürlich unter den Tisch. Außerdem können Sie, wenn Sie etwa deutsche und österreichische Postleitzahlen gemischt verwenden, immer noch nach der Anfangszahl sortieren, während in einem Zahlenfeld nach dem Wert der Zahl sortiert würde.

Felddatentyp	VBA-Datentyp	Beschreibung
Text	String	Kurze Texte, max. 255 Zeichen, 2 Byte je Zeichen
Memo	String	Lange Texte, max. 65.536 Zeichen bei Eingabe über die Benutzeroberfläche, 2 GB bei Eingabe per Code, 2 Byte je Zeichen
Zahl (Byte)	Byte	Zahlen von 0 bis 255, 1 Byte
Zahl (Integer)	Integer	Zahlen von -32768 bis 32.767, 2 Byte
Zahl (Long Integer)	Long	Zahlen von -2.147.483.648 bis 2.147.483.647, 4 Byte
Zahl (Single)	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

Tabelle 1.1: Datentypen beim Anlegen von Tabellen in der Entwurfsansicht

Felddatentyp	VBA-Datentyp	Beschreibung
Zahl (Double)	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 (Replikations-ID)	Benutzerdefinierter Typ	16 Byte
Zahl (Dezimal)	Variant	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
Datum/Uhrzeit	Date	Datums- oder Zeitangaben, 1.1.100 bis 31.12.9999, 8 Byte
Währung	Currency	Skalierte Ganzzahl von -922.337.203.685.477,5808 bis 922.337.203.685.477,5807, 8 Byte
Autowert (Long Integer)	Long	Zahlen von -2.147.483.648 bis 2.147.483.647, 4 Byte
Autowert (Replikations-ID)	Benutzerdefinierter Typ	16 Byte
OLE-Objekt	Byte-Array	Enthält OLE-Objekte und binäre Daten, 1 Gigabyte
Hyperlink	String	Enthält vier Zeichenketten mit maximal 2.048 Zeichen (Anzeigetext, Hyperlink, Unterhyperlink, Hilfetext). Die einzelnen Bestandteile trennt man durch das Raute-Zeichen (#).
Anlage	Byte-Array	Speichert Dateien im binären Format. Genaue Informationen finden Sie in Kapitel 12, »Bilder und binäre Daten«. Maximale Größe: keine Informationen verfügbar, Experimente ergaben kein klares Ergebnis.
Nachschlageassistent	Verschiedene	Stellt die Feldeigenschaften für Felder so ein, dass diese als Nachschlagefeld für Wertlisten, Tabellen/Abfragen oder komplexe Datentypen verwendet werden können.
Berechnet	Verschiedene	Erlaubt das Speichern berechneter Werte auf Basis der übrigen Felder der Tabelle (siehe »Der Felddatentyp Berechnet« auf Seite 43).

Tabelle 1.1: Datentypen beim Anlegen von Tabellen in der Entwurfsansicht (Fortsetzung)

Wenn Sie die Feldnamen und Felddatentypen ermittelt haben, können Sie diese in den Tabellenentwurf eintragen (siehe Abbildung 1.2). Fügen Sie die Felder einfach von oben nach unten ein. Wenn Sie die Reihenfolge der Felder ändern möchten, markieren Sie einfach das zu verschiebende Feld durch einen Klick auf das graue Kästchen links daneben

und ziehen es dann bei gedrückter linker Maustaste an die gewünschte Stelle. Das Löschen und Einfügen von Zeilen erledigen Sie am einfachsten mit den passenden Einträgen des Kontextmenüs der Entwurfsansicht – experimentieren Sie halt einfach mal. Aber Achtung: Das Löschen eines Feldes löscht unwiderruflich auch die enthaltenen Daten. Mit den im unteren Bereich angezeigten Eigenschaften können Sie die einzelnen Felder weiter anpassen. Weitere Informationen zu den Eigenschaften finden Sie weiter unten.

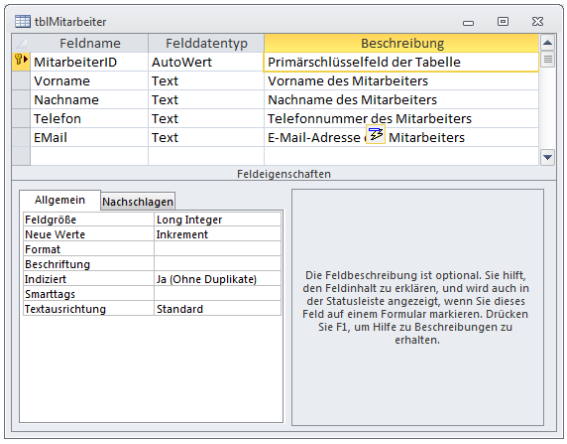


Abbildung 1.2: Eintragen von Feldnamen, Felddatentypen und Beschreibungen in der Tabellenentwurfsansicht

Besonderheiten des Felddatentyps Memo

Seit Access 2007 können Memofelder Rich-Text aufnehmen. Um ein Memo-Feld mit Rich-Text zu füllen, stellen Sie vorab die Eigenschaft *Textformat* auf den Wert *Rich-Text* ein (siehe Abbildung 1.3).

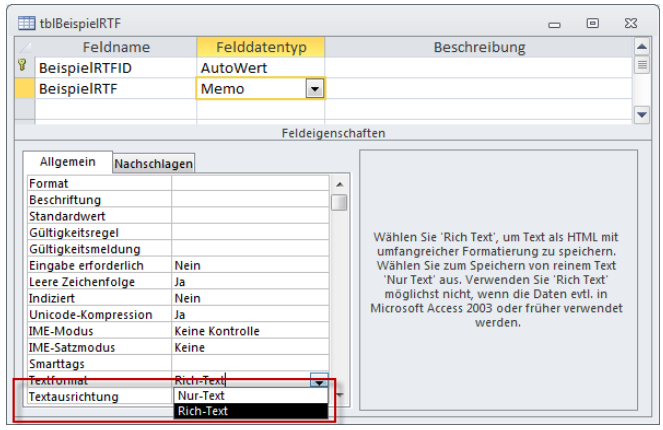


Abbildung 1.3: Vorbereiten eines Memo-Felds auf die Eingabe von Text im Rich-Text-Format

Direkt im Anschluss können Sie in der Datenblattansicht der Tabelle auf die Rich-Text-Formatierung zugreifen. Dazu müssen Sie nur mit der Maus den zu formatierenden Text markieren und die passende Formatierung aus dem nun erscheinenden Popup-Menü auswählen (siehe Abbildung 1.4). Dieses Menü erscheint übrigens nicht, wenn Sie Text mit der Tastatur markieren; in dem Fall müssen Sie auf die passenden Formatierungs-Schaltflächen im Ribbon unter *Start|Schriftart* und *Start|Rich-Text* zurückgreifen.

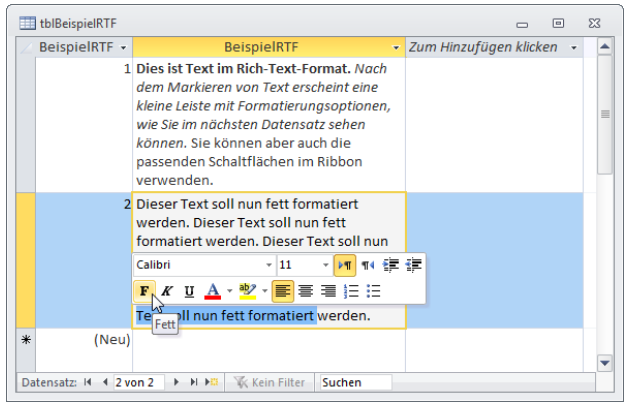


Abbildung 1.4: Formatieren von Rich-Text-Format direkt in der Datenblattansicht einer Tabelle

Access stellt die Rich-Text-Formatierungen übrigens nicht über das etwa in RTF-Dateien verwendete Format, sondern über HTML her. Den verwendeten HTML-Code können Sie nicht direkt über die Benutzeroberfläche, aber über eine kleine VBA-Anweisung ermitteln. Diese setzen Sie im Direktfenster des VBA-Editors ab, den Sie mit *Strg + G* öffnen:

```
Debug.Print DLookup("BeispielRTF", "tblBeispielRTF")
```

Für das erste Rich-Text-Feld aus der Tabelle in Abbildung 1.4 sieht der HTML-Code etwa so aus:

```
<div><strong>Dies ist Text im Rich-Text-Format. </strong><em>Nach dem Markieren von Text
erscheint eine kleine Leiste mit Formatierungsoptionen, wie Sie im nächsten Datensatz
sehen können.</em> Sie können aber auch die passenden Schaltflächen im Ribbon verwen-
den.</div>
```

Den Inhalt eines Memofeldes im Rich-Text-Format können Sie auch per VBA einfügen; zu beachten ist hier, dass nicht alle HTML-Elemente, sondern nur eine Auswahl verfügbar ist. Um welche es sich genau handelt, ist zum Zeitpunkt der Erstellung dieses Buches nicht dokumentiert; es sind aber mehr, als das zum Formatieren verfügbare Popup-Menü anbietet. Weitere Informationen über den Einsatz von Rich-Text-Feldern finden Sie unter »Rich-Text in Textfeldern« auf Seite 264.

Besonderheiten des Felddatentyps Zahl mit Format Prozent

Access 2003 und ältere Versionen von Access brachten eine etwas gewöhnungsbedürftige Behandlung von Prozentzahlen mit sich: Zwar gaben sie für den Felddatentyp *Zahl* mit der Feldgröße *Double* und der Formateinstellung *Prozent* entsprechend formatierte Werte aus, jedoch interpretierten sie die Eingabe von *1* als *100%* und *100* als *10.000%*, was zu vielen Eigenentwicklungen zur Behandlung von Prozentzahlen führte. Seit der Version 2007 zeigt Access sich da wesentlich entwicklerfreundlicher, da es die Eingabe von *1* auch als *1%* interpretiert und von *100* als *100%*.

Beachten Sie, dass Sie die Feldgröße des Zahlenfelds vom standardmäßig vergebenen Wert *Long Integer* auf eine Feldgröße mit Nachkommastellen ändern, also beispielsweise auf *Double*. Für eine größere Genauigkeit verwenden Sie den Datentyp *Währung* und legen dort als *Format* den Eintrag *Prozentzahl* fest. Dies ist sinnvoll, wenn Sie die Prozentzahlen zur Berechnung von Währungsbeträgen verwenden möchten.

Besonderheiten des Felddatentyps Datum/Uhrzeit

Eingaben in Felder des Felddatentyps *Datum/Uhrzeit* speichert Access intern als *Double*-Werte. Dabei entspricht der ganzzahlige Anteil dem Datum (dabei steht 1 für den 31.12.1899) und der Teil hinter dem Komma der Uhrzeit (in Bruchteilen eines Tages). Bei der Eingabe zweistelliger Jahreszahlen interpretiert Access alle Jahre von 00 bis 29 als Jahre des 21. Jahrhunderts (*15* entspricht also dem Jahr *2015*) und alle Jahre von 30 bis 99 als Jahre des 20. Jahrhunderts (*71* entspricht somit dem Jahr *1971*). Ab Access 2007 steht für Datumsfelder eine Eigenschaft zur Verfügung, mit der Sie einen Dialog zur Auswahl von Datumsangaben aktivieren können. Verantwortlich hierfür ist

die Eigenschaft *Datumsauswahl anzeigen*, für die Sie den Wert *Für Datumsangaben* einstellen müssen (siehe Abbildung 1.5).

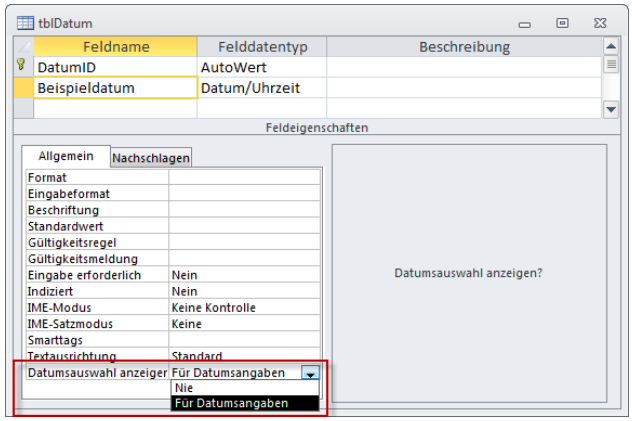


Abbildung 1.5: Für Datumsfelder steht ein Dialog zur Datumsauswahl bereit

Abbildung 1.6 zeigt diesen Dialog im Einsatz. Beim Klick auf ein Datumsfeld erscheint neben dem Feld eine Schaltfläche, mit der Sie den Dialog zur Datumsauswahl öffnen können. Leider ist diese Lösung etwas halbherzig; wer damit etwa Termine auswählen möchte, die weit zurückliegen, muss endlos klicken, bis er zum gewünschten Jahr gelangt.

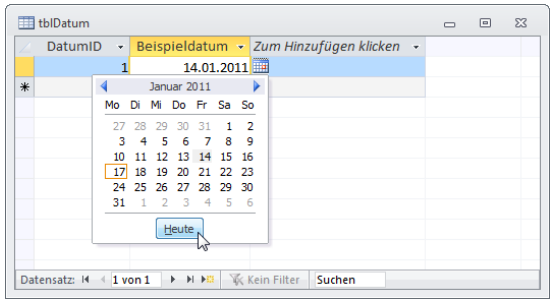


Abbildung 1.6: Einblenden des Dialogs zur Auswahl von Datumsangaben

Besonderheiten des Felddatentyps Anlage

Das Feld *Anlage* gibt es erst seit Access 2007. Es kann eines oder mehrere Dateien speichern und bringt einen Dialog zu ihrer Auswahl mit. Abbildung 1.7 zeigt eine solche Tabelle mit einem leeren Feld. Wie in der Abbildung zu sehen ist, zeigt Access für *Anlage*-Felder keinen Feldnamen in der Kopfzeile an; wenn Sie hier einen Feldnamen sehen möchten, müssen Sie diesen für die Eigenschaft *Beschriftung* des Feldes eintragen.

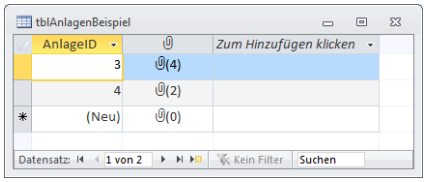


Abbildung 1.7: Eine Tabelle mit einem leeren Feld des Datentyps *Anlage*

Ein Doppelklick auf ein *Anlage*-Feld aktiviert den Dialog aus Abbildung 1.8, mit dem Sie die in diesem Feld enthaltenen Dateien verwalten können. Der mit der Schaltfläche *Hinzufügen* geöffnete Dialog ist ein gewöhnlicher Dateiauswahl-Dialog, mit dem Sie eine oder mehrere Dateien auswählen können.

Sie können als Dateiname auch Links zu Daten im Internet angeben, Access lädt diese dann herunter. Die übrigen Schaltfächen des *Anlagen*-Dialogs sind selbsterklärend; mit ihnen können Sie die enthaltenen Dateien wieder entfernen, diese öffnen oder auf der Festplatte speichern.

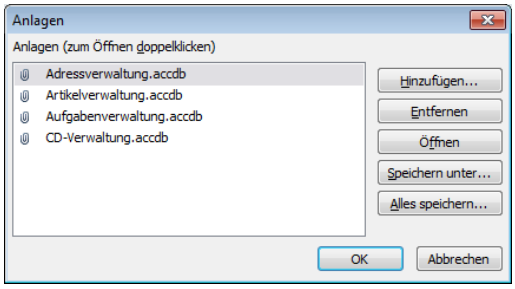


Abbildung 1.8: Mit diesem Dialog verwalten Sie die in einem Anlage-Feld enthaltenen Dateien

Was sich hinter einem *Anlage*-Feld verbirgt, können Sie im *Beziehungen*-Fenster oder im Abfrageentwurf entdecken, wenn Sie dort eine Tabelle mit einem solchen Feld einfügen (siehe Abbildung 1.9).

Es enthält intern drei weitere Felder, die mit den Binärdaten, dem Dateinamen (ohne Verzeichnis) und dem Datentyp beziehungsweise der Dateiendung gefüllt werden.

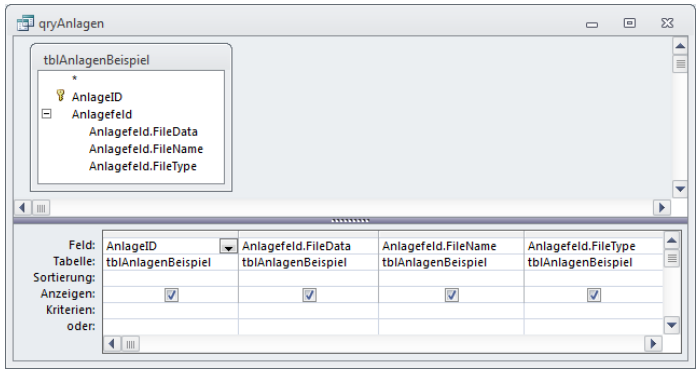


Abbildung 1.9: Abfrage einer Tabelle mit einem Anlage-Feld

Das Ergebnis einer solchen Abfrage sieht wie das einer Abfrage mit zwei per 1:n-Beziehungen verknüpften Tabellen aus. Die Daten der übergeordneten Tabelle (*tblAnlagen-Beispiel*) werden dabei für jeden Datensatz der untergeordneten Tabelle (hier die interne Tabelle mit den in jedem *Anlage*-Feld gespeicherten Daten) wiederholt ausgegeben (siehe Abbildung 1.10).

Eine Besonderheit ergibt sich bei Bilddateien: Für diese können Sie in den Access-Optionen unter *Aktuelle Datenbank|Anwendungsoptionen|Bildeigenschaften-Speicherformat* einstellen, ob sie im Quellbildformat oder im Bitmap-Format gespeichert werden sollen (siehe Abbildung 1.11). In älteren Versionen war nur Letzteres möglich, was zu vielen Workarounds führte.

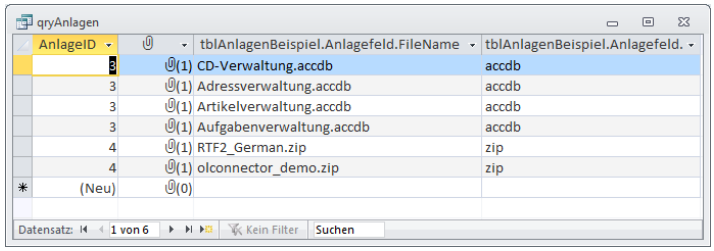


Abbildung 1.10: Abfrage über die Datensätze einer Tabelle mit einem Anlage-Feld mit jeweils mehreren gespeicherten Dateien

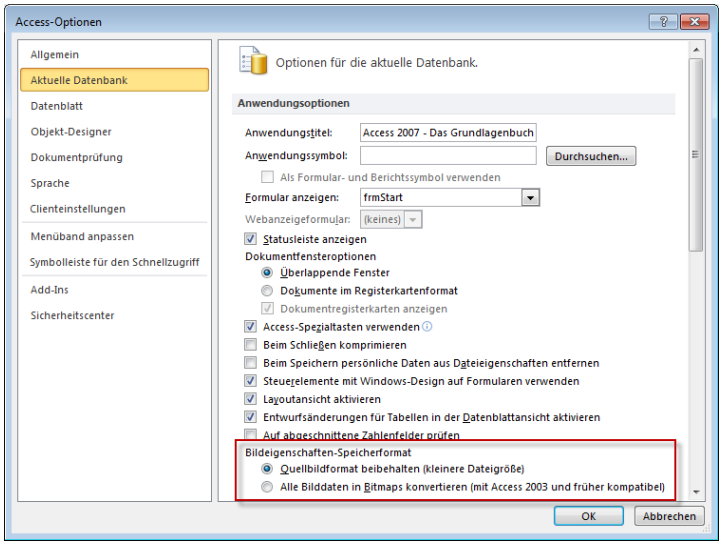


Abbildung 1.11: Eigenschaft zum Einstellen des Formats von in Anlage-Feldern gespeicherten Bilddateien

Anlage-Felder nehmen möglicherweise nicht alle Dateien auf, die Sie darin speichern möchten. Dabei entscheidet sich anhand der Dateiendung, ob eine Datei aufgenommen werden kann oder nicht. Welche Dateiendungen gesperrt sind, erfahren Sie in der Onlinehilfe des VBA-Editors unter »Attachment-Objekt«. Interessanterweise prüft Access nur die Dateiendungen.

Wenn Sie eine Datei mit einer nicht erlaubten Endung in eine mit einer erlaubten Endung umbenennen, akzeptiert Access diese ohne Probleme. Generell stellt sich die Frage, warum man nicht standardmäßig beliebige Dateien in *Anlage*-Feldern speichern kann.

Weitere Informationen zum Thema Anlage-Felder finden Sie unter »Rich-Text in Textfeldern« ab Seite 264. Dort erfahren Sie etwa, wie Sie Dateien per VBA in Anlage-Feldern speichern und wie Sie darin enthaltene Bilder in Formularen und Berichten anzeigen.

Der Felddatentyp Berechnet

Eine Neuheit ab Access 2010 ist der Felddatentyp *Berechnet*. Er entspricht prinzipiell den berechneten Feldern in Abfragen, wo Sie beispielsweise zwei Felder *Nachname* und *Vorname* durch einen Ausdruck wie *[Nachname] & ", " [Vorname]* in der Form *Minhorst, André* zusammenführen können. Auch mathematische Berechnungen wie etwa das Ermitteln der Mehrwertsteuer aus Betrag und Mehrwertsteuersatz sind so möglich.

Dies gelingt nun auch direkt in Tabellen. Um dort ein berechnetes Feld zu erstellen, wählen Sie als Datentyp den Eintrag *Berechnet* aus. Es erscheint sogleich der Ausdruckseditor von Access, der im Übrigen stark überarbeitet wurde. Eine der wichtigsten Neuerungen ist IntelliSense. Damit können Sie wie in Abbildung 1.12 Ausdrücke zusammensetzen, um beispielsweise die Felder der aktuellen Tabelle zu referenzieren.

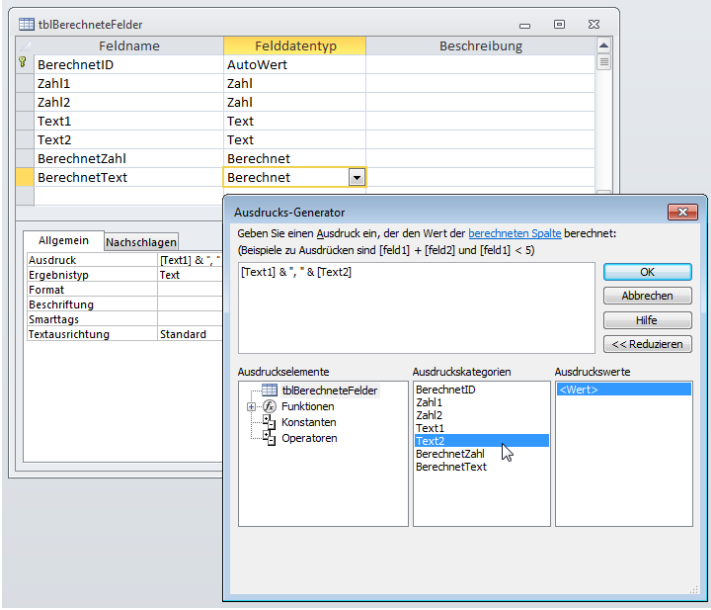


Abbildung 1.12: Zusammenstellen eines berechneten Feldes mit dem Ausdruckseditor

Wenn Sie ein berechnetes Feld erstellen, legt Access zunächst noch keinen Wert für die Eigenschaft *Ergebnistyp* des Feldes fest. Dies geschieht erst nach dem Wechseln in die Datenblattansicht und somit nach dem ersten Berechnen des Inhalts des neuen Feldes. Je nach Datentypen, Inhalten und verwendeten Operatoren entscheidet sich Access für einen Datentyp. Sollen beispielsweise die Inhalte zweier Zahlenfelder addiert werden, kommt ein Zahlendatentyp heraus. Wenn Sie hingegen den Inhalt zweier Textfelder verketten, sollte der Datentyp *Text* lauten. Es gibt aber auch Ausnahmen: Wenn Sie zum Beispiel zwei Textfelder multiplizieren und die Textfelder Zahlen enthalten, legt Access für die Eigenschaft *Ergebnistyp* den Wert *Double* fest.

Sie sollten die Eigenschaft *Ergebnistyp* nach dem Anlegen eines zu berechnenden Feldes also auf jeden Fall nach der ersten Berechnung prüfen. Vereinzelt weisen Access-Entwickler darauf hin, dass die Daten berechneter Felder nicht korrekt aktualisiert würden. Dieses Verhalten war jedoch nicht reproduzierbar. Berechnete Felder in Tabellen erlauben im Gegensatz zu denen in Abfragen keine Verwendung benutzerdefinierter VBA-Funktionen.

Data Type Parts

Die in Access 2007 eingeführten Tabellenvorlagen sind mit Access 2010 verschwunden. Dafür gibt es nun die *Data Type Parts*, eine Möglichkeit, ein oder mehrere Felder als Vorlage zu speichern und in anderen Tabellen wiederzuverwenden. Voraussetzung für den Einsatz der *Data Type Parts* ist die Bearbeitung der Tabelle in der Datenblattansicht. Das Ribbon-Tab *Felder* liefert dann alle notwendigen Funktionen (siehe Abbildung 1.13). In der Entwurfsansicht stehen die benötigten Befehle nicht zur Verfügung.

Klicken Sie im Ribbon-Tab *Felder* auf die Schaltfläche *Weitere Felder*, erscheint eine Liste einiger vordefinierter Felder. Einige davon enthaltenen Felder der Basisdatentypen, andere liefern gleich entsprechende Formatierungen mit, wie beispielsweise die Datumsfelder. Klicken Sie auf einen der oberen Einträge, legt Access Felder des entsprechenden Datentyps an – ohne weitere Einstellungen an den Feldern vorzunehmen.

Ganz unten unter *Schnellstart* finden Sie hingegen einige Einträge, mit denen Sie nicht nur Felder mit weiteren Eigenschaften wie Name, Standardwert et cetera anlegen, sondern auch ganze Sätze zusammenhängender Felder zur Tabelle hinzufügen. So legt *Name* gleich zwei Textfelder namens *Vorname* und *Nachname* an. *Zahlungsart* erstellt gar ein Kombinationsfeld mit vorgefertigtem Nachschlagefeld.

Üblicherweise werden Sie mit den von Microsoft vorgeschlagenen Feldern nicht viel anfangen können. Da passt es ganz gut, dass Sie auch eigene Felder oder komplette Sätze von Feldern als Vorlage speichern können.

Um einen solchen Eintrag hinzuzufügen, erstellen Sie zunächst in der Entwurfsansicht ein Feld mit den gewünschten Eigenschaften. In diesem Fall soll das Feld *Anrede* mit einem Nachschlagefeld zur Auswahl der beiden Werte *Herr* und *Frau* als *Data Type Part* angelegt werden. Nach dem Erstellen des Feldes wechseln Sie in die Datenblattansicht der Tabelle. Aktivieren Sie das Ribbon-Tab *Felder* und betätigen Sie den Eintrag *Hinzufügen und Löschen|Weitere Felder|Auswahl als neuen Datentyp speichern...*, um den Dialog aus Abbildung 1.14 anzuzeigen.

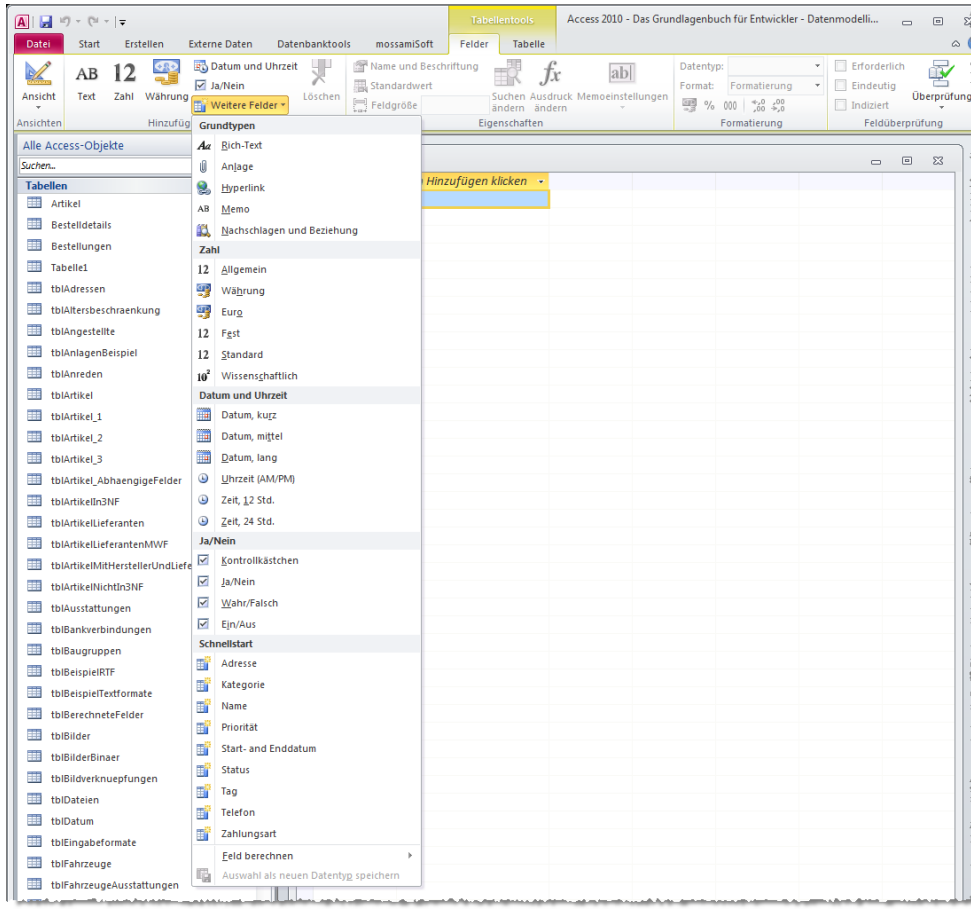


Abbildung 1.13: Einfügen vordefinierter Felder

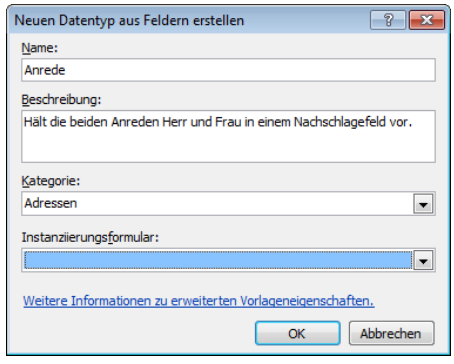


Abbildung 1.14: Anlegen eines neuen Data Type Parts

Hier tragen Sie zunächst den Namen ein, unter dem das neue *Data Type Part* in der Liste zu finden sein soll. Der unter *Beschreibung* angegebene Text wird eingeblendet, wenn Sie mit der Maus über den entsprechenden Eintrag in der Liste der *Data Type Parts* fahren. Wählen Sie außerdem eine der bestehenden Kategorien aus oder legen Sie eine neue Kategorie an, indem Sie einfach den gewünschten Namen eintragen.

Unter *Instanzierungsformular* können Sie ein Formular der aktuellen Datenbank auswählen, das einmalig beim Anlegen des Feldes auf Basis des *Data Type Parts* in der Datenbank gespeichert und geöffnet wird. Nach dem Schließen wird es automatisch wieder gelöscht.

Dieses Formular können Sie nutzen, um weitere Einstellungen zum *Data Type Part* vorzunehmen. Besonders komfortabel ist dies nicht, weil das Formular beispielsweise keine Informationen zum neuen Feld liefert. Sie können das Anlegen des Feldes auch nicht einfach in Abhängigkeit von Benutzereingaben in dieses Formular abbrechen, da das Feld zum Zeitpunkt des Öffnens des Formulars bereits angelegt ist.

Außerdem speichert Access die Tabelle gleich nach dem Anlegen des Feldes. Dies können Sie feststellen, indem Sie die Tabelle nach Hinzufügen des Feldes schließen: Access fragt nicht nach, ob es die Änderungen speichern soll, also wurden diese bereits gespeichert.

Testen Sie nun Ihr erstes selbst angelegtes *Data Type Part*. Dazu legen Sie eine neue Tabelle an und fügen mit dem entsprechenden Eintrag der *Data Type Parts*-Liste das Feld *Anrede* zu dieser Tabelle hinzu. Es funktioniert!

Im Hintergrund speichert Access die Informationen über das Feld beziehungsweise die Felder in einer Datei mit der Endung *.accft* (hier *Anreden.accft*). Das Verzeichnis entspricht einem Unterordner namens *Access* im Vorlagenverzeichnis für Office-Dokumente. Normalerweise heißt dieses Verzeichnis *C:\Users\<Aktueller Benutzer>\AppData\Roaming\Microsoft\Templates\Access*.

Wenn ein *Data Type Part* für alle Benutzer verfügbar sein soll, verschieben Sie ihn etwa unter Windows 7 64-bit in das Verzeichnis *C:\Program Files (x86)\Microsoft Office\Templates\1031\Access\DataType*. Dort finden Sie auch die bereits vorhandenen Data Type Parts von Access.

Informationen über den Speicherort aller benutzerdefinierten *Data Type Parts* sowie der unter *Schnellstart* befindlichen *Data Type Parts* finden Sie in der Tabelle *Templates* der Datenbankdatei *C:\Users\Andre Minhorst\AppData\Roaming\Microsoft\Access\AccessCache.accdb*.

Sie löschen ein *Data Type Part*, indem Sie in der Liste mit der rechten Maustaste auf seinen Eintrag klicken und den Kontextmenübefehl *Datentyp aus Katalog löschen* auswählen.

Application Parts

Application Parts wurden bereits unter »Anwendungsparts« auf Seite 104 vorgestellt. Mit diesem Feature können Sie in Access komplette Tabellen oder gar mehrere, miteinander verknüpfte Tabellen als Vorlage speichern. In der Zieldatenbank können Sie anschließend neue Tabellen auf Basis dieser Tabellen anlegen. Da *Application Parts* zum Anlegen von Tabellen und Beziehungen auf Basis benutzerdefinierter Vorlagen verwendet werden können, scheint ein Verweis auf das Thema an dieser Stelle angebracht.

1.1.3 Schlüssel festlegen

Access bietet verschiedene Schlüsselarten an. Schlüssel haben gemeinsam, dass sie das Anlegen eines Indexes nach sich ziehen. Einen Index können Sie sich als eine Tabelle vorstellen, die alle Daten eines Schlüsselfeldes in der angegebenen Reihenfolge sowie den Primärschlüssel der Tabelle enthält. Beim Sortieren der Tabelle oder beim Suchen nach einem bestimmten Wert aus dem Schlüsselfeld zieht Access dann die Index-Tabelle zu Rate und kommt somit schneller zum Ziel. Im Gegenzug verursachen Indizes beim Anlegen von Datensätzen zusätzliche Arbeit für den Rechner, da der Feldinhalt ja nicht nur in der eigentlichen Tabelle, sondern auch noch in der Index-Tabelle gespeichert werden muss (oder auch in mehreren). Genaue Informationen über die Auswirkungen und den Einsatz von Indizes finden Sie unter »Indizes« ab Seite 837.

In den folgenden Abschnitten erfahren Sie zunächst einmal, wie Sie diese festlegen. Daneben haben Schlüssel noch weitere Eigenschaften, die sie von normalen Feldern abheben – auch diese beschreiben die folgenden Abschnitte.

Primärschlüssel

Sie sollten für jede Tabelle einen Primärschlüssel festlegen. Ein Primärschlüssel ist ein eindeutiger Index, mit dem Sie – wie der Name schon sagt – die Eindeutigkeit der Datensätze einer Tabelle sicherstellen. So können Sie zwei ansonsten völlig identische Datensätze immer noch aufgrund des Feldes mit dem Primärschlüssel unterscheiden. Den Primärschlüssel legen Sie fest, indem Sie im Datenbankentwurf das passende Feld markieren und den Ribbon-Eintrag *Entwurf|Tools|Primärschlüssel* betätigen. Access markiert das Feld dann etwa wie das Feld *MitarbeiterID* in Abbildung 1.2.

Meist hat das Primärschlüsselfeld den Datentyp *Autowert*; damit stellen Sie automatisch sicher, dass niemals ein doppelter Wert für dieses Feld angelegt wird – zumindest nicht, wenn Sie die Daten immer in diese Tabelle eingeben. Alternativ können Sie hier den Unterdatentyp *Replikations-ID* verwenden; das macht beispielsweise Sinn, wenn Mitarbeiter unabhängig voneinander neue Datensätze in verschiedenen Kopien der Datenbank anlegen und diese zu einem bestimmten Zeitpunkt zusammengeführt wer-

den sollen. Bis Access 2003 war diese Vorgehensweise unter dem Stichwort *Replikation* bekannt. Datenbanken, die im Format von Access 2007/2010 erstellt wurden, können Sie nicht replizieren. Wenn Sie Tabellen in der Datenblattansicht entwerfen, was beispielsweise in Webdatenbanken die einzige Möglichkeit ist, legt Access beim Erstellen einer Tabelle automatisch ein Primärschlüsselfeld namens *ID* mit Autowert an.

Warum ist es nun so wichtig, dass jede Tabelle ein Feld mit eindeutigen Werten enthält? Eine der wichtigsten Eigenschaften relationaler Datenbanksysteme wie Microsoft Access ist die Möglichkeit, Beziehungen zwischen Tabellen herzustellen. So kann man etwa einen Kunden einem Projekt zuordnen.Und diese Zuordnung erfolgt über die Zuweisung des Primärschlüsselfeldes der Projektetabelle zu einem Fremdschlüsselfeld der Kundentabelle; das heißt, man trägt den Primärschlüsselwert in ein passendes Feld der Kundentabelle ein. Und wenn das Primärschlüsselfeld keine eindeutigen Werte enthält, wäre auch keine eindeutige Zuordnung der Datensätze dieser beiden Tabellen möglich.

Sie können nur einen Primärschlüssel je Tabelle anlegen, der allerdings nicht nur eines, sondern auch mehrere Felder enthalten kann. Dies macht etwa Sinn, wenn Sie eine Verknüpfungstabelle zur Herstellung einer m:n-Beziehung verwenden und die darin enthaltenen Verknüpfungsfelder zu einem Primärschlüssel zusammenfassen.

Weitere Informationen zu diesem Thema haben Sie bereits unter »Data Type Parts« auf Seite 45 erfahren.

Primärschlüssel dürfen keine *Null*-Werte enthalten, nur für Felder mit den Datentypen *Autowert*, *Zahl*, *Text*, *Datum/Uhrzeit* und *Währung* angelegt werden und nur für Felder, die noch keine doppelten Werte enthalten. Theoretisch ginge das auch für *Ja/Nein*-Felder, aber das macht selbstverständlich keinen Sinn.

Sekundärschlüssel

Neben dem Primärschlüssel können Sie weitere Schlüssel anlegen, die zumindest dafür sorgen, dass Sie schneller nach Daten in den mit einem Schlüssel versehenen Feld suchen oder diese abfragen können. Sie definieren einen Sekundärschlüssel für ein Feld, indem Sie dessen Eigenschaft *Indiziert* entweder auf *Ja (Duplikate möglich)* oder auf *Ja (Ohne Duplikate)* einstellen. Im letzteren Fall darf dieses Feld wie ein Primärschlüsselfeld nur eindeutige Werte enthalten. Kandidaten für Sekundärschlüsselfelder sind etwa Fremdschlüsselfelder in Beziehungen oder solche Felder, in denen Sie vermutlich oft nach Daten suchen.

Anzeigen aller Schlüssel

Das Bearbeiten des Primärschlüssels und der nachfolgend vorgestellten Sekundärschlüssel ist auch in einem speziellen Dialog möglich, den Sie in der Entwurfsansicht einer Tabelle mit dem Ribbon-Eintrag *Entwurf|Einblenden/Ausblenden|Indizes* öffnen. Der

nun erscheinende Dialog *Indizes* zeigt eine Liste mit allen Schlüsselfeldern der Tabelle an (siehe Abbildung 1.15).

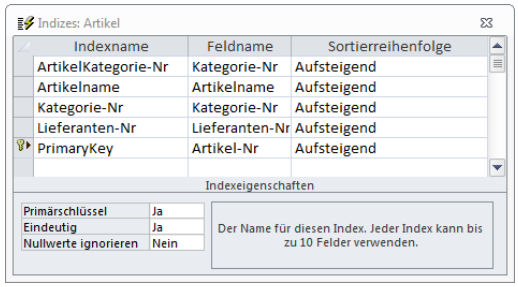


Abbildung 1.15: Dieser Dialog zeigt sämtliche Indizes einer Tabelle

Hier finden Sie zum jeweils markierten Index die drei Eigenschaften *Primärschlüssel*, *Eindeutig* und *Nullwerte ignorieren*. Die Eigenschaft *Primärschlüssel* können Sie wie oben erwähnt nur für einen Schlüssel auf *Ja* einstellen. Wenn ein Schlüssel aus mehreren Feldern zusammengesetzt sein soll, fügen Sie unterhalb der Zeile mit der Schlüsselbezeichnung eine weitere Zeile hinzu, deren Feld *Indexname* leer bleibt und die im Feld *Feldname* den Namen des weiteren zum Schlüssel gehörenden Feldes enthält (siehe Abbildung 1.16).

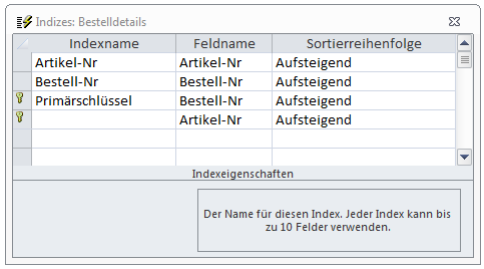


Abbildung 1.16: Beispiel für einen zusammengesetzten Schlüssel im Indizes-Dialog

Wenn Sie die Eigenschaft *Nullwerte ignorieren* auf *Ja* einstellen, ignoriert Access Datensätze mit Nullwerten in dem angegebenen Feld bei der Bildung des Indexes, was bei Feldern mit vielen Nullwerten im Schlüsselfeld Speicherplatz spart.

Automatisch angelegte Indizes

Access legt Felder, deren Feldnamen bestimmte Zeichenfolgen enthalten, automatisch als Index fest. Welche dies sind, können Sie im Dialog *Access-Optionen* unter *Objekt-Designer|Entwurfsansicht für Tabellen|AutoIndex beim Importieren/Erstellen* festlegen (siehe Abbildung 1.17).

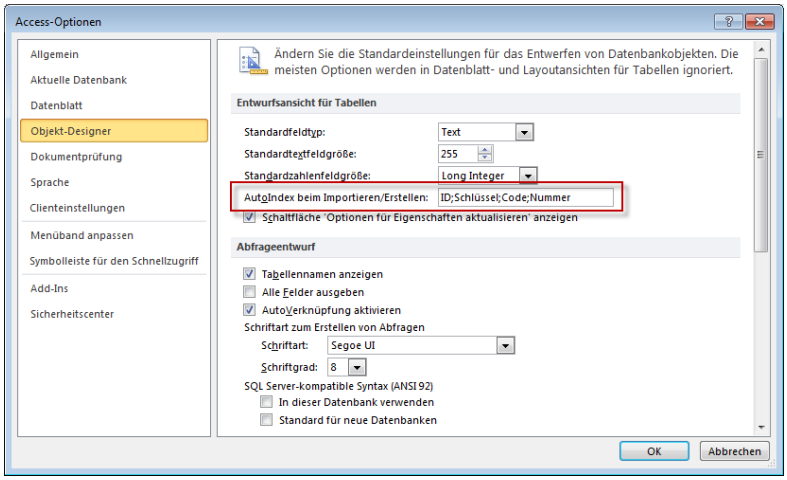


Abbildung 1.17: Für Felder, deren Feldname eine der unter *AutoIndex* beim Importieren/Erstellen angegebenen Zeichenfolgen enthält, legt Access automatisch einen Index an

1.1.4 Eigenschaften festlegen

Mit den Eigenschaften der Tabelle und der Felder können Sie beispielsweise die Optik und die Ergonomie der Tabelle auch im Hinblick auf die spätere Verwendung als Datensatzquelle für Formulare und Berichte verbessern. Die Bedeutung und die Werte der einzelnen Eigenschaften können Sie der Onlinehilfe entnehmen, die folgenden Abschnitte stellen deren Anwendung für spezielle Fälle vor.

Autowertfelder anpassen

Mit der Eigenschaft *Neue Werte* können Sie festlegen, wie Access Autowerte für neue Datensätze bestimmt: Entweder ermittelt Access den aktuell größten Wert und zählt *1* dazu (*Inkrement*) oder wählt einen zufälligen Wert aus (*Zufall*).

Feldgrößen voreinstellen

Die Feldgröße legt fest, welche maximale Größe (Textfelder) oder welchen Wertebereich (Zahlenfelder) ein Feld hat. Beachten Sie, dass Sie damit immer den maximalen Wert festlegen – das birgt einige Gefahren: Bei Textfeldern sollten Sie etwa immer die maximale Größe 255 verwenden, außer Sie möchten die Anzahl der verwendeten Zeichen einschränken (etwa auf 7 für Postleitzahlen wie *D-47137*).

Das ist insofern wichtig, als der Irrglaube herrscht, dass die Feldgröße bei Textfeldern die Anzahl der reservierten Zeichen repräsentiert, was nicht richtig ist. Bei Zahlen ist das anders: Diese belegen immer die in der Tabelle 2.1 angegebene Anzahl Bytes; die

Feldgröße von Zahlenfeldern sollten Sie daher immer so auslegen, dass die größten zu erwartenden Werte soeben hineinpassen. In dem Zusammenhang ist interessant, dass Sie in den Access-Optionen unter *Objekt-Designer|Tabellenentwurf* den *Standardfeldtyp*, die *Standardtextfeldgröße* und die *Standardzahlenfeldgröße* festlegen können.

Wenn Sie hier etwa für die Standardtextfeldgröße den Wert *255* einstellen, kann zumindest kein Benutzer zu Ihnen kommen und sich beschweren, dass mal wieder nur 50 Zeichen in ein Feld zur Eingabe von Internetadressen passen – nur, weil Sie vielleicht einmal zu sparsam waren (50 Zeichen war übrigens in älteren Access-Versionen mal die voreingestellte Standardeinstellung, jetzt sind es 255 Zeichen).

Richtiges Format anzeigen

In Formularen und Berichten können Sie für Steuerelemente mit der *Format*-Eigenschaft festlegen, in welchem Format ihr Inhalt angezeigt werden soll. Da der Inhalt in der Regel immer im gleichen Format erscheinen soll, können Sie eine Menge Arbeit sparen, indem Sie das Format des Feldes direkt im Tabellenentwurf festlegen. Dazu verwenden Sie ebenfalls die *Format*-Eigenschaft.

Je nach dem gewählten Felddatentyp stehen unterschiedliche vordefinierte Formate zur Verfügung, zu denen Sie in der Onlinehilfe ausgiebige Informationen erhalten (etwa über *F1|Suche nach: Format Datum*).

Beschriftungen in Formularen und Berichten vorbereiten

Wenn Sie später Tabellenfelder in Formulare und Berichte einfügen, verwendet Access automatisch den Tabellennamen als Beschriftungsfeld des jeweiligen Steuerelements. In manchen Fällen ist es nicht gewünscht, dass Sie beispielsweise in Feldnamen keine Umlaute verwenden oder dort Unterstriche und Abkürzungen einsetzen (mehr über optimale Feldnamen unter »Feldnamen« auf Seite 67.

Wenn Sie später immer wieder die Feldnamen in lesbare Steuerelementbeschriftungen ändern müssen, kann das auf Dauer nerven. Mit der Eigenschaft *Beschriftung* können Sie jedoch global den Text festlegen, der statt des Feldnamens als Beschriftungstext zum Einsatz kommt. Geben Sie hier also etwa für das Feld *GeaendertAm* den Wert *Geändert am* an.

Globale Validierung

Für die Validierung von Daten können Sie durch passend programmierte Formulare und Steuerelemente sorgen oder Sie legen einige dafür geeignete Eigenschaften von Tabellen und Feldern fest. Diese werden bei jeder Änderung von Daten über die Benutzeroberfläche herangezogen, also nicht nur beim Ändern der Daten direkt in der Tabelle.

Die Eigenschaften *Gültigkeitsregel* und *Gültigkeitsmeldung* dienen dazu, Regeln für den Wertebereich der einzugebenen Daten aufzustellen. Mehr zu diesem Thema erfahren Sie unter »Integrität der Werte (Wertbereichsintegrität)« auf Seite 80 und in der Onlinehilfe unter den entsprechenden Begriffen. Eine weitere Möglichkeit der globalen Validierung liefert die Eigenschaft *Eingabeformat*. Damit geben Sie an, wie der einzugebende Ausdruck im Detail aufgebaut sein muss.

Standardwerte von Tabellenfeldern

Eine weitere Vereinfachung bei der Eingabe von Daten ist das Bereitstellen von Standardwerten. So können Sie etwa einen statischen Wert vergeben oder eine Funktion zur Ermittlung eines Standardwertes heranziehen. Benutzerdefinierte VBA-Funktionen sind dabei allerdings nicht erlaubt. In beiden Fällen tragen Sie den passenden Ausdruck in das Feld für die Eigenschaft *Standardwert* ein. Beispiele:

» *0*: füllt das Feld mit dem Wert *0*.

» *Datum()*: füllt das Feld eines neuen Datensatzes mit dem aktuellen Datum.

Die Einstellungen für die Eigenschaft *Standardwert* wirken sich auch auf den Einsatz des Feldes in Formularen aus. Access 2007 und Access 2010 stellen im Gegensatz zu den Vorgängerversionen den Standardwert von *Long*-Feldern nicht mehr automatisch auf 0 ein.

Unterdatenblätter

Unterdatenblätter sind unter Umständen ein performance-fressendes Feature, das mit Access 2000 eingeführt wurde. Diese bieten die Möglichkeit, innerhalb der Datenblattansicht einer Tabelle alle zu einem Datensatz gehörenden Datensätze aus einer verknüpften Tabelle anzuzeigen. Da Sie Daten mit Formularen/Unterformularen meist wesentlich übersichtlicher darstellen können, sollten Sie diese Option meist abschalten. Dazu stellen Sie die Eigenschaft *Unterdatenblattname* einer jeden Tabelle in der Entwurfsansicht auf *[Keines]* ein. Die Funktion hat aber auch Vorteile: Wenn Sie beispielsweise die Unterdatensätze zu mehreren Datensätzen gleichzeitig anzeigen möchten, ist dies ein sehr einfacher Weg.

1.1.5 Beziehungen herstellen

Das Herstellen einer Beziehung zwischen zwei Tabellen setzt voraus, dass die Datenbank eine Tabelle enthält, die auf die Daten einer anderen Tabelle verweisen soll. Erstere enthält dazu ein Fremdschlüsselfeld und heißt Detailtabelle.

Das Fremdschlüsselfeld nimmt später Daten auf, die aus dem Primärschlüsselfeld der anderen Tabelle stammen, der sogenannten Mastertabelle.

Nun können Sie ohne irgendwelche weiteren Maßnahmen Daten aus dem Primärschlüsselfeld der Mastertabelle in das Fremdschlüsselfeld der Detailtabelle eintragen und stellen damit schon eine Beziehung zwischen den beiden Tabellen her. Dies von Hand zu erledigen, ist jedoch ziemlich aufwendig, also bietet Access einige Hilfsmittel, die sowohl die Auswahl der verknüpften Datensätze als auch die Prüfung der Konsistenz übernehmen.

Gleiche Datentypen

Grundvoraussetzung für das Herstellen einer Beziehung mit referentieller Integrität (was das ist, erfahren Sie weiter unten) zwischen zwei Tabellen ist, dass das Fremdschlüsselfeld der Detailtabelle und das Primärschlüsselfeld der Mastertabelle den gleichen Datentyp haben. In der Regel ist dies der Datentyp *Zahl (Long)*.

Wenn Sie keine referentielle Integrität benötigen, können Sie auch andere Kombinationen einsetzen.

Nachschlage-Assistent

Die einfachste Möglichkeit besteht darin, dem Verknüpfungsfeld den Wert *Nachschlage-Assistent* aus der Liste der Datentypen zuzuweisen. Dies startet einen Assistenten, der alle notwendigen Informationen abfragt. Dieses Buch will Ihnen aber nicht den Umgang mit Assistenten, sondern Techniken und Hintergründe beibringen, und deshalb folgt nun eine Beschreibung, wie Sie das, was der Nachschlage-Assistent so produziert, selbst auf die Beine stellen.

Dennoch sei hier auf eine Neuerung verwiesen: Seit Access 2010 bietet der Nachschlage-Assistent im letzten Schritt die Möglichkeit, direkt referentielle Integrität und Löschweitergabe für die automatisch erstellte Beziehung zu definieren (siehe Abbildung 1.18).

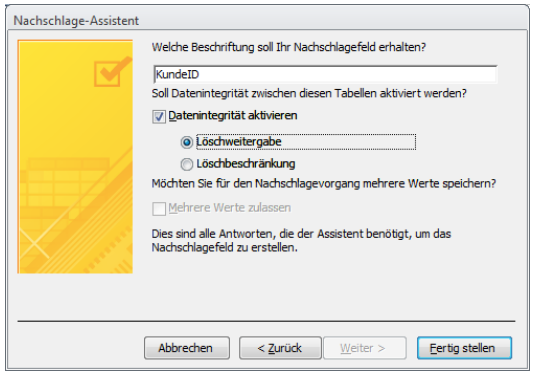


Abbildung 1.18: Festlegen referentieller Integrität im Nachschlage-Assistenten

Beziehung herstellen

Das Herstellen der Beziehung erledigen Sie im Dialog *Beziehungen* (auch *Beziehungen*-Fenster genannt). Sie öffnen dieses Fenster mit dem Ribbon-Eintrag *Datenbanktools|Einblenden/Ausblenden|Beziehungen* (siehe Abbildung 1.19). In dieser Abbildung übernimmt die Tabelle *tblKunden* die Rolle der Mastertabelle und *tblProjekte* die der Detailtabelle. Das Feld *KundeID* der Tabelle *tblProjekte* soll einmal die Inhalte des Feldes *KundeID* der Tabelle *tblKunden* aufnehmen – und das auf möglichst einfache Weise.

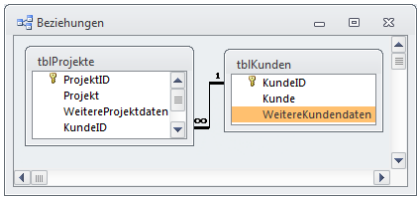


Abbildung 1.19: Das Beziehungen-Fenster mit zwei verknüpften Tabellen

Der Kontextmenüeintrag *Tabelle anzeigen* öffnet einen Dialog mit allen in der Datenbank enthaltenen Tabellen. Wählen Sie hier die Tabellen aus, zwischen denen Sie eine Beziehung herstellen möchten (siehe Abbildung 1.20). Wenn Sie alle Tabellen im Beziehungen-Fenster anzeigen möchten, die bereits Teil einer Beziehung sind, können Sie dies auch über den Eintrag *Alle anzeigen* aus dem Kontextmenü erledigen. Sie können die gewünschten Tabellen auch direkt aus dem Navigationsbereich in das Beziehungen-Fenster ziehen, wobei auch eine Mehrfachauswahl möglich ist (das Datenbankfenster älterer Access-Versionen bot diese Möglichkeit nicht).

Abbildung 1.20: In diesem Dialog wählen Sie die im Beziehungen-Fenster anzuzeigenden Tabellen aus.

Zum Herstellen der Beziehung ziehen Sie nun das Primärschlüsselfeld der Mastertabelle auf das Fremdschlüsselfeld der Detailtabelle und lassen es dort fallen. Access legt eine passende Verbindungslinie zum Kennzeichnen der Beziehung an. Wenn Sie nun mit der

rechten Maustaste auf diese Linie klicken und den Kontextmenüeintrag *Beziehung bearbeiten* auswählen, zeigt Access einen Dialog an, mit dem Sie referentielle Integrität, Lösch- und Aktualisierungsweitergabe einstellen können. Was das alles ist, erfahren Sie unter »Format der Werte (semantische Integrität)« auf Seite 81.

Auswahl von per 1:n-Beziehung verknüpften Daten

Das Herstellen einer Beziehung ist nur ein Teil der Arbeit, die Ihnen normalerweise der Nachschlage-Assistent abnehmen würde.

Der Großteil besteht im Anlegen eines Nachschlagefeldes, mit dem Sie die Daten aus der verknüpften Tabelle bequem auswählen können. Alles, was der Nachschlage-Assistent dazu beiträgt, ist das Anpassen weniger Eigenschaften – und das können Sie auch von Hand erledigen.

Im obigen Beispiel mit der Kunden- und der Projektetabelle legen Sie die erforderlichen Eigenschaften für das Feld *KundeID* der Tabelle *tblProjekte* fest. Öffnen Sie diese Tabelle in der Entwurfsansicht (vom Beziehungen-Fenster aus geht dies schnell über das Kontextmenü des jeweiligen Tabelleneintrags) und markieren Sie das Feld *KundeID*.

Im unteren Bereich wechseln Sie nun zur Registerseite *Nachschlagen*. Dort finden Sie zunächst nur eine einzige Eigenschaft namens *Steuerelement anzeigen* vor, was sich aber ändert, wenn Sie deren Wert auf *Kombinationsfeld* einstellen (siehe Abbildung 1.21).

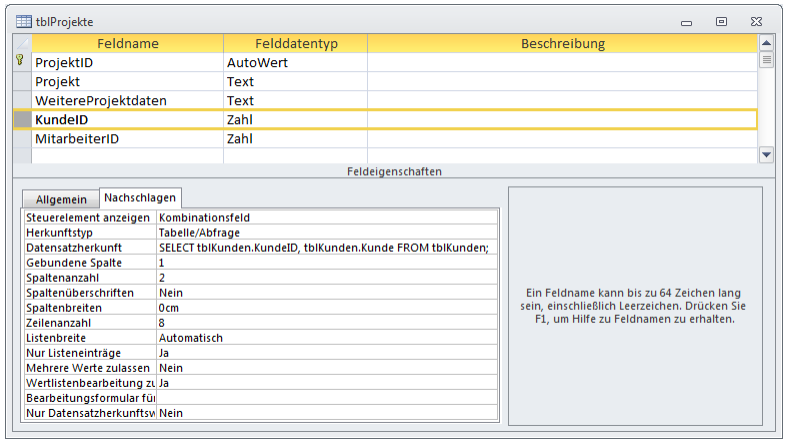


Abbildung 1.21: Die Eigenschaften eines Verknüpfungsfeldes, das als Nachschlagefeld ausgelegt werden soll

Mit einigen der anderen Eigenschaften sorgen Sie nun für die Bevölkerung Ihres Nachschlagefeldes:

» *Herkunftstyp*: Stellen Sie hier den Wert *Tabelle/Abfrage* ein. Wozu der Wert *Wertliste* dient, erfahren Sie weiter unten.

» *Datensatzherkunft:* Legt fest, aus welcher Tabelle oder Abfrage die Daten stammen. Üblicherweise gibt man hier einen SQL-Ausdruck an beziehungsweise generiert diesen über die Abfrage-Entwurfsansicht, die Sie mit der Schaltfläche mit den drei Punkten (...) öffnen. Prinzipiell muss die Abfrage zunächst das Primärschlüsselfeld der zu verknüpfenden Tabelle enthalten, weil dessen Inhalt gespeichert und darüber die Beziehung hergestellt wird. Access erlaubt zusätzlich die Anzeige weiterer Felder und das Ausblenden des Primärschlüsselwertes. Sie brauchen also den Primärschlüssel zum Speichern und ein oder mehrere weitere Felder zum Anzeigen, also sieht die Datensatzherkunft in diesem Beispiel so aus:

```
SELECT KundenID, Kunde FROM tblKunden;
```

» Gegebenenfalls sortieren Sie die Daten alphabetisch nach den Kunden, was die folgende Datensatzherkunft nach sich ziehen würde:

```
SELECT KundenID, Kunde FROM tblKunde ORDER BY Kunde;
```

» *Gebundene Spalte*: Diese Eigenschaft legt fest, welche Spalte der Datensatzherkunft gespeichert werden soll. Standardmäßig ist dies die erste Spalte, was auch in diesem Beispiel passt.

» *Spaltenanzahl*: Gibt an, wie viele Spalten der Datensatzherkunft von links aus gesehen angezeigt werden sollen.

» *Spaltenbreiten*: Legt die Breite der angezeigten Spalten fest. Wenn die erste die gebundene Spalte ist und diese nicht angezeigt werden soll (was üblicherweise so ist) und es nur noch eine weitere Spalte gibt, tragen Sie hier den Wert *0cm* ein. Damit wird die erste Spalte mit dem Primärschlüssel quasi ausgeblendet und die nächste Spalte nimmt den kompletten verfügbaren Platz ein.

» *Zeilenanzahl*: Hiermit legen Sie fest, wie viele Zeilen beim Ausklappen angezeigt werden sollen (Standardwert *16*).

» *Listenbreite*: Legt die Gesamtbreite der Liste fest.

Es gibt eine Reihe Varianten bei der Gestaltung von Nachschlagefeldern auf Basis einer Tabelle oder Abfrage, die mit der Vorgehensweise beim Erstellen von Kombinationsfeldern identisch sind. Weitere Informationen finden Sie unter »Kombinationsfelder« ab Seite 268. Wenn Sie ein Nachschlagefeld definiert haben, können Sie damit eine Menge anfangen: Access zeigt es nicht nur in der Datenblattansicht der jeweiligen Tabelle oder auf dieser Tabelle aufbauenden Abfragen an (aber dort soll der Anwender ja ohnehin keine Daten ändern), sondern übernimmt die Nachschlagefelder auch standardmäßig in Formulare, wenn Sie dort das passende Feld anlegen.

Dazu richtet Access einfach ein Kombinationsfeld mit den passenden Eigenschaften ein. Manchmal hat dies aber auch Nachteile: Wenn Sie einmal die tatsächliche ID des verknüpften Datensatzes ermitteln möchten, müssen Sie die Einstellungen für das Nachschlagefeld entsprechend ändern. Wenn Sie dies verhindern möchten und die Fremdschlüsselfelder in der Tabellendefinition als einfache Textfelder festlegen, müssen Sie diese in Formularen bei Bedarf manuell in Kombinationsfelder umwandeln. Außerdem erfordert das Füllen der Nachschlagefelder natürlich auch Zeit.

Bearbeiten der Einträge eines Nachschlagefeldes

Bisher mussten Sie, wenn Sie die Einträge eines Nachschlagefeldes für die Auswahl von per 1:n-Beziehung verknüpften Daten anpassen wollten, eigene Funktionen bereitstellen. Access versucht, dies zu vereinfachen, indem es die Möglichkeit zum direkten Öffnen eines geeigneten Formulars bietet. Alles, was Sie dazu veranlassen müssen, ist das Erstellen eines geeigneten Formulars (am besten mit dem Wert *Wahr* für die Eigenschaft *Popup*, damit dieses Formular vor dem Weiterarbeiten mit anderen Formularen wieder geschlossen werden muss), hier auf Basis der Tabelle *tblKunden*, sowie das Einstellen zweier Eigenschaften:

Die Eigenschaft *Wertlistenbearbeitung zulassen* erhält den Wert *Ja* und die Eigenschaft *Bearbeitungsformular für Listenelemente* den Namen des dafür vorgesehenen Formulars (hier: *frmKunden*). Wenn Sie das Nachschlagefeld nun in der Datenblattansicht öffnen, erscheint eine Schaltfläche wie in Abbildung 1.22, die per Mausklick das angegebene Formular öffnet. Diese Funktion ist auch in Formularen verfügbar.

Für das schnelle Ändern der zugrunde liegenden Daten ist diese Funktion ausreichend, aber nicht für eine professionelle Anwendung: Das angezeigte Formular zeigt nämlich weder direkt den aktuell im Nachschlagefeld ausgewählten Datensatz an (außer, es ist zufällig gerade der erste Datensatz ausgewählt), noch aktualisiert es das ausgewählte Feld auf den zuletzt im Formular angezeigten Datensatz. Falls Sie mehr benötigen, sollten Sie also doch lieber manuell eine passende Funktion erstellen – und wie das funktioniert, erfahren Sie unter »Von Formular zu Formular« ab Seite 233.

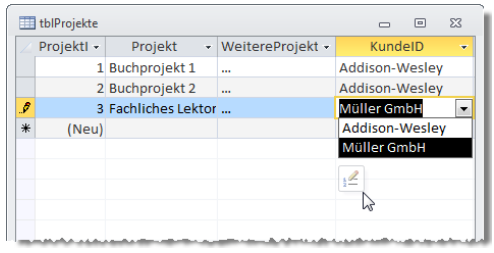


Abbildung 1.22: Beim Öffnen eines Nachschlagefeldes erscheint ein Symbol zum Anzeigen eines Formulars zum Bearbeiten der zugrunde liegenden Daten

Auswahl von per m:n-Beziehung verknüpfte Daten

Die hier vorgestellte Vorgehensweise zum Vereinfachen der Auswahl der Daten verknüpfter Tabellen ist nicht neu.

Anders verhält es sich mit der Tatsache, dass Sie hier eine Mehrfachauswahl vornehmen können, ohne eine Verknüpfungstabelle zu erstellen, wie es sonst bei der Umsetzung von m:n-Beziehungen erforderlich ist.

Dazu stellen Sie einfach die Eigenschaft *Mehrere Werte zulassen* auf den Wert *Ja* ein. Access zeigt dann zusätzlich zu den Listeneinträgen jeweils ein Kontrollkästchen an, mit dem Sie die enthaltenen Einträge auswählen können. So brauchen Sie nun etwa für die Auswahl der Mitarbeiter eines Projekts theoretisch keine m:n-Beziehung mehr, wie folgendes Beispiel zeigt.

Die Tabelle *tblProjekte* enthält dabei ein Feld namens *MitarbeiterID*, mit dem sich mehrere Mitarbeiter auswählen lassen. Dazu stellen Sie die Feldeigenschaften wie in Abbildung 1.23 ein. Der nicht komplett sichtbare Inhalt der Eigenschaft *Datensatzherkunft* sieht so aus:

```
SELECT tblMitarbeiter.MitarbeiterID, [Nachname] & ", " & [Vorname] AS Mitarbeiter FROM
tblMitarbeiter ORDER BY [Nachname] & ", " & [Vorname];
```

Damit legen Sie fest, dass *Nachname* und *Vorname* als ein Feld in die Datensatzquelle aufgenommen werden.

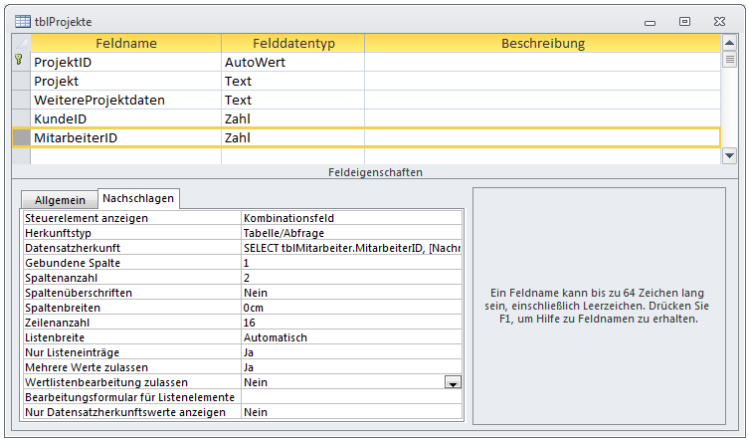


Abbildung 1.23: Einstellungen für ein Nachschlagefeld, mit dem Sie mehrere Datensätze gleichzeitig auswählen können

In der Datenblattansicht ergibt sich dann das Bild aus Abbildung 1.24. Das Nachschlagefeld zeigt beim Aufklappen Kombinationsfelder an, mit denen Sie einen oder mehrere Datensätze auswählen können.

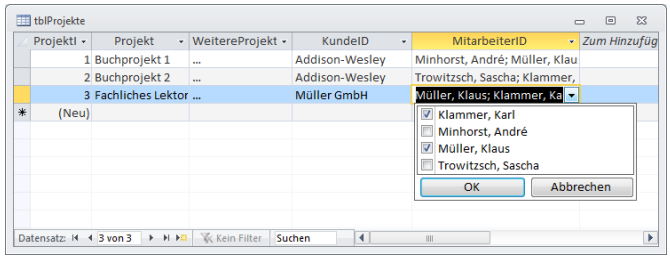


Abbildung 1.24: Auswahl mehrerer Mitarbeiter zu einem Projekt

Nach der Auswahl werden die angezeigten Werte (nicht die gebundenen) in einer semikola-separierten Liste angezeigt. Natürlich gibt es einige Besonderheiten bei der Abfrage von Tabellen mit mehrwertigen Nachschlagefeldern. Mehr darüber erfahren Sie unter »Abfragen mit Anlage-Feldern und mehrwertigen Feldern« ab Seite 126.

Bearbeiten der Daten in mehrwertigen Feldern

Die in einem solchen mehrwertigen Feld angezeigten Daten können Sie genau wie bei üblichen Nachschlagefeldern mit einem Formular bearbeiten, wenn Sie die Eigenschaft *Wertlistenbearbeiten zulassen* auf den Wert *Ja* einstellen und für die Eigenschaft *Bearbeitungsformular für Listenelemente* den Namen eines geeigneten Formulars angeben – alles Weitere wurde bereits weiter oben erläutert. Bei ungebundenen Datensatzherkünften, bei denen Sie die Eigenschaft *Herkunftstyp* auf *Wertliste* einstellen und unter *Datensatzherkunft* eine semikolonseparierte Liste der anzuzeigenden Daten anlegen, können Sie auch einen eingebauten Dialog zur Bearbeitung der Daten verwenden; mehr dazu im folgenden Abschnitt.

1.1.6 Nachschlagefelder mit Wertliste

Die oben vorgestellten Nachschlagefelder können Sie auch ohne den Einsatz einer Tabelle realisieren. Dazu speichern Sie die zu verknüpfenden Daten in einer sogenannten Wertliste, indem Sie diese durch Semikola voneinander getrennt auflisten. Sinnvoll ist dies, wenn die mit dem Nachschlagefeld auszuwählenden Daten überschaubar sind und der Bestand sich nicht oder nur selten ändert (warum dies so ist, erfahren Sie weiter unten). In allen anderen Fällen verwenden Sie besser eine Tabelle als Datensatzherkunft. Ein Beispiel ist etwa das Geschlecht von Personen. Diese können Sie in einer eigenen Tabelle speichern, aber da voraussichtlich in nächster Zeit zu *männlich* und *weiblich* kaum zusätzliche Einträge hinzukommen werden, können Sie hierfür auch eine Wertliste verwenden.

Die Eigenschaften für ein Nachschlagefeld namens *Geschlecht* sehen so aus wie in Abbildung 1.25. Im Vergleich zu den vorgegebenen Werten brauchen Sie nur die Eigen-

schaft *Steuerelement anzeigen* auf *Kombinationsfeld* und dann *Herkunftstyp* auf *Wertliste* und *Datensatzherkunft* auf *"männlich";"weiblich"* einzustellen; die übrigen Werte können Sie beibehalten.

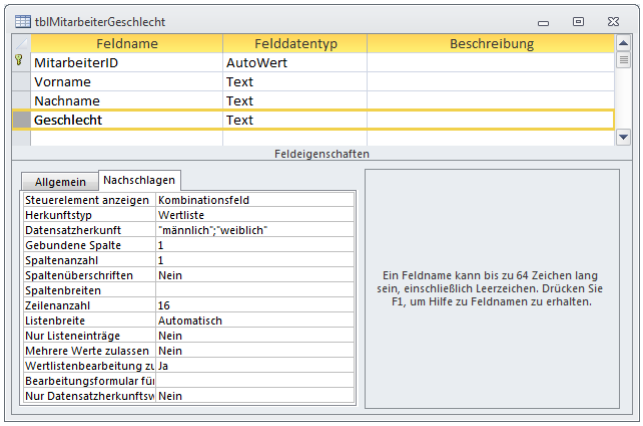


Abbildung 1.25: Einstellungen für ein Nachschlagefeld mit Wertliste

Für eine vereinfachte Eingabe der Wertlisten-Elemente aktivieren Sie die Eigenschaft *Wertlistenbearbeitung zulassen*. Außerdem muss die Wertliste einspaltig sein. Danach klicken Sie nur noch auf die beim Aktivieren des Kombinationsfeldes erscheinende Schaltfläche *Listenelemente bearbeiten* (auch per Kontextmenü verfügbar) und schreiben im folgenden Dialog die einzelnen Einträge in jeweils eine eigene Zeile (siehe Abbildung 1.26). Auch einen Standardwert können Sie dort auswählen; dieser wird direkt in die passende Eigenschaft *Standardwert* auf der Registerseite *Allgemein* eingetragen.

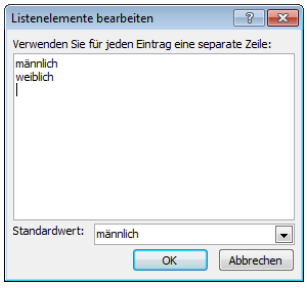


Abbildung 1.26: Eingabe von Listenelementen einer Wertliste per Dialog

Dieses Feature ist insofern nett, als es die Eingabe der zur Verfügung stehenden Werte erleichtert. Sie können es aber auch deaktivieren, indem Sie die Eigenschaft *Wertlistenbearbeitung zulassen* auf *Nein* einstellen. Und das ist – sobald Sie die gewünschten Werte eingetragen haben – dringend anzuraten: Auch, wenn Sie dem An-

wender keine Möglichkeit geben, direkt auf die Datenblattansicht von Tabellen oder Abfragen zuzugreifen: Er erhält Zugriff auf diesen Dialog, wenn er in einem Formular ein Kombinationsfeld aufklappt, das an ein Feld mit den oben beschriebenen Nachschlagefeld-Eigenschaften gebunden ist. Es erscheint dann ein kleines Symbol unterhalb der Liste, mit dem der Anwender diesen Dialog öffnen kann.

Prinzipiell nimmt Ihnen Access damit eine Menge Arbeit ab, aber das Hinzufügen von Einträgen auf diese Weise hat einen entscheidenden Nachteil: Sie haben keine Möglichkeit, die hinzugefügten Werte zu validieren – so kann der Benutzer dort etwa mehrere leere Zeilen einfügen oder bereits verwendete Listeneinträge aus der Liste entfernen. Wenn der Benutzer Daten hinzufügen können soll, verwenden Sie besser ein Kombinationsfeld, das beim Eintragen eines nicht vorhandenen Datensatzes eine entsprechende Prozedur aufruft, mit der Sie die Daten validieren können (weitere Informationen unter »Eingabevalidierung« ab Seite 251). Der zweite Nachteil ist, dass Sie keine mehrspaltigen Wertlisten bearbeiten können, die in der ersten Spalte etwa eine nicht angezeigte ID und in der zweiten Spalte den tatsächlichen Wert anzeigen.

m:n-Beziehung für Wertlisten

Auch die Auswahl mehrerer Einträge einer Wertliste als Inhalt eines einzigen Feldes ist möglich. Im Grunde sind diese Felder die m:n-Beziehungs-Variante der oben vorgestellten 1:n-Beziehungs-Wertlisten. Der einzige Unterschied ist, dass Sie hiermit nicht nur einen, sondern mehrere Einträge der angegebenen Wertliste auswählen können. Das ist ganz einfach: Sie müssen lediglich einige Eigenschaften auf der *Nachschlagen*-Registerseite der Feldeigenschaften des betroffenen Feldes ändern.

Die wichtigste Eigenschaft lautet *Mehrere Werte zulassen*: Stellen Sie diese auf *Ja* ein. Als Beispiel dient die Tabelle aus Abbildung 1.27, mit der man zu jedem Artikel mehrere Lieferanten auswählen können soll.

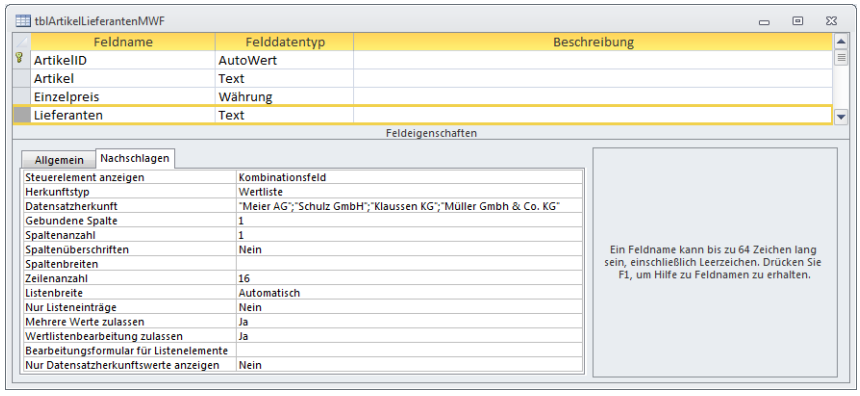


Abbildung 1.27: Einstellungen für ein Nachschlagefeld mit Mehrfachauswahl

Anschließend können Sie – vorausgesetzt, es sind Daten vorhanden – mit der Auswahl mehrerer Datensätze für nur ein einziges Feld beginnen (siehe Abbildung 1.28). Abbildung 1.29 zeigt, wie Access den Feldinhalt nach dem Bearbeiten anzeigt.

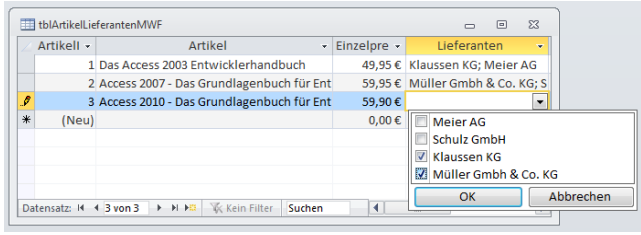


Abbildung 1.28: Ein mehrwertiges Nachschlagefeld

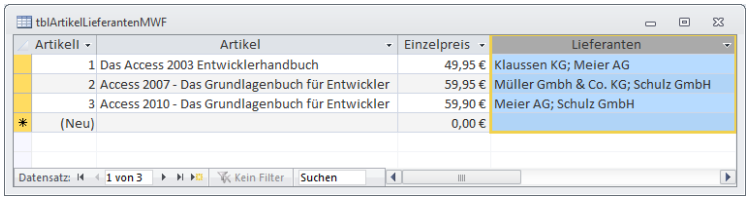


Abbildung 1.29: Anzeige der in einem mehrwertigen Feld ausgewählten Daten

Im Prinzip ist dies eine vereinfachte Form der m:n-Beziehung, mit der Ausnahme, dass Sie nicht den üblichen Zugriff auf die verknüpften Daten in einer weiteren Tabelle haben. Laut Microsoft speichert Access diese Daten aber dennoch in passenden, versteckten Tabellen, auf die der Anwender aber über die Benutzeroberfläche keinen direkten Zugriff wie auf die anderen Tabellen erhält.

1.1.7 Tabelleninformationen im Überblick

Wenn Sie einmal einen Überblick über die Tabellen, Felder und ihre Eigenschaften benötigen, um sich etwa in eine Tabelle eines anderen Entwicklers einzuarbeiten, können Sie sich eine solche Übersicht mit dem Datenbankdokumentierer erstellen lassen. Diesen starten Sie mit dem Ribbon-Eintrag *Datenbanktools|Analysieren|Datenbankdokumentierer*.

1.2 Namenskonventionen für Tabellen und Felder

Die Namenskonventionen für Tabellen enthalten Vorschläge für die Benennung von Tabellen und der enthaltenen Felder. Die nachfolgenden Abschnitte fassen dabei Elemente der ungarischen Notation (*http://www.xoc.net/standards/rvbanc.asp*) und eini-

ge weitere Regeln zusammen, die sich bewährt haben und von vielen Entwicklern so oder ähnlich berücksichtigt werden. Details zur ungarischen Notation finden Sie unter dem oben genannten Link; ein Abdruck der recht umfangreichen Tabellen ist aus Platzgründen leider nicht möglich.

Grundsätzlich gelten für die Vergabe von Namen an Access-Objekte, Feldnamen und Steuerelemente folgende Regeln:

» Der Name darf aus maximal 64 Zeichen bestehen.

» Der Name kann aus beliebigen Zeichen mit Ausnahme des Punktes (.), Ausrufezeichens (!), Gravis-Akzents (`), einfachen (') und doppelten Anführungszeichens (") und eckigen Klammern ([]) bestehen.

» Leerzeichen dürfen nicht am Anfang oder am Ende des Namens stehen.

Zu empfehlen ist darüber hinaus, dass Namen von Access-Objekten, Feldern und Steuerelementen gar keine Sonderzeichen enthalten, da Namen mit Sonderzeichen an manchen Stellen eine Spezialbehandlung erfordern – beispielsweise müssen Sie diese in SQL-Ausdrücken und in VBA in eckige Klammern einfassen. Verwenden Sie außerdem besser keine Namen, die bereits für ein reserviertes Wort innerhalb von Access, VBA oder referenzierten Objektbibliotheken benutzt werden.

Abschreckendes Beispiel ist die Verwendung von »Name« als Feldname in Tabellen mit Kontaktdaten – diese Bezeichnung ist bereits als Eigenschaft diverser Objekte vergeben. Access 2010 zeigt allerdings direkt beim Anlegen eines Tabellenfeldes oder von Objekten wie Tabellen, Abfragen, Formularen oder Berichten eine Warnmeldung an, wenn die Bezeichnung bereits reserviert ist. Natürlich können Sie auch Sonderzeichen, Leerzeichen oder bereits als Schlüsselwort verwendete Namen aufgreifen. Sie müssen dann allerdings einige Regeln beachten – etwa, dass Sie die Bezeichnungen von Feldern in SQL-Ausdrücken und in VBA in eckige Klammern setzen, wenn diese nicht ohnehin in Anführungszeichen stehen.

Warum sollen Sie eine Namenskonvention verwenden?

Grundsätzlich können Sie Tabellen und die enthaltenen Felder nennen, wie Sie möchten. Abhängig davon, ob Sie eine Anwendung nur für das stille Kämmerlein oder für jemand anderen entwickeln, sollten Sie jedoch folgende Punkte beachten:

» Eine feste Systematik bei der Vergabe von Namen für Tabellen und Felder erleichtert Ihnen beim Programmieren das Leben. Wenn Ihre Tabellennamen etwa grundsätzlich mit »tbl« beginnen und der Plural der darin beschriebenen Objekte angefügt ist, müssen Sie sich beim Referenzieren dieser Tabelle aus Formularen, Berichten oder aus VBA-Modulen niemals Gedanken machen, wie Sie die Tabelle genannt haben, wenn Sie nur wissen, welches Objekt darin beschrieben wird.

» Präfixe bei Objektnamen erlauben Ihnen, auf einfache Weise ein Objekt am Namen zu erkennen und gleiche Basisnamen für mehrere unterschiedliche Objekttypen zu verwenden. Wenn Sie eine Tabelle beispielsweise nur *Bestellungen* nennen, können Sie schon keine Abfrage gleichen Namens mehr verwenden. Daher wählen Sie für die Tabelle den Namen *tblBestellungen* und für die Abfrage den Namen *qryBestellungen*.

» Wenn mehrere Entwickler die gleiche Konvention verwenden, erleichtert dies anderen Entwicklern das Analysieren und Bearbeiten der Anwendung und des enthaltenen Codes. Sie würden wahrscheinlich verrückt werden, wenn Sie eine Anwendung weiterentwickeln müssten, die eine völlig andere Konvention als die Ihnen vertraute verwendet – oder auch gar keine.

Verwenden alle Access-Entwickler die gleiche Konvention?

Wenn Sie eine Anwendung oder eine Beispieldatenbank eines halbwegs professionellen Access-Entwicklers in die Hände bekommen, werden Sie feststellen, dass sich dort die ungarische Notation durchgesetzt hat. Natürlich gibt es hier und da Abweichungen, die persönlichen Vorlieben oder einfach der Macht der Gewohnheit unterliegen, aber mit diesen kann man in der Regel gut leben. Sicher kommt ein Entwickler damit klar, wenn ein Bericht nicht das Präfix »rpt« hat, sondern mit »rep« beginnt. Auch nennen manche Entwickler eine Datensatzgruppe »rs...« und nicht »rst...«, aber es versteht trotzdem jeder, was gemeint ist.

Microsoft selbst geht leider mit schlechtem Beispiel voran: Erstellen Sie mal eine Datenbank auf Basis der Vorlage *Marketingprojekte*. Da fehlen jegliche Präfixe und Objekt- und Feldnamen enthalten Leerzeichen und Sonderzeichen wie Minus, Klammern und Prozent.

1.2.1 Tabellennamen

Tabellen haben genau wie alle anderen Objekte unter Access ein bestimmtes Präfix, damit man sie von anderen Objekten unterscheiden kann. Dieses Präfix lautet *tbl*. Der Rest des Tabellennamens soll möglichst gut beschreiben, was die Tabelle enthält. Dabei wählt man für diesen Teil des Tabellennamens in der Regel die Pluralform. Der Grund dafür ist, dass die meisten Tabellen auch mehrere Datensätze und damit Informationen über das betroffene Objekt enthalten. Man kann nicht oft genug erwähnen, dass jede »normale« Tabelle Informationen über ein reales Objekt enthalten sollte, wie beispielsweise Personen, Fahrzeuge, Projekte, Termine, Gebäude oder Artikel.

Tabellen mit realen Objekten

Tabellen, die reale Objekte beschreiben, heißen beispielsweise folgendermaßen:

» *tblArtikel*

» *tblProjekte*

» *tblPersonen*

» *tblFahrzeuge*

Verknüpfungstabellen

Wenn schon von »normalen« Tabellen die Rede ist, sollen auch die anderen Typen vorgestellt werden: m:n-Beziehungen erfordern die Verwendung von Verknüpfungstabellen, die mindestens jeweils das Primärschlüsselfeld der zu verknüpfenden Tabellen enthalten. In manchen Fällen lässt sich für solche Tabellen ein erstklassiger Name finden, doch eine Kombination der Namen der beteiligten Tabellen macht meist schon deutlich, was die Tabelle für Informationen enthält. Beispiele:

» *tblFahrzeugeSonderausstattungen*

» *tblBestellungenArtikel* (besser: *tblBestelldetails*)

» *tblArtikelFirmen* (besser: *tblLieferanten*)

Detailtabellen in 1:1-Beziehungen

Und da wären noch die Tabellen, die zusätzliche Daten zu anderen Tabellen enthalten und per 1:1-Beziehung mit diesen verknüpft sind. Mit ein wenig objektorientierter Sichtweise bilden die »Basistabelle« und die »Erweiterungstabelle« eine neue Tabelle, die quasi von der »Basistabelle« erbt.

Wenn Sie beispielsweise eine Tabelle *tblPersonen* haben, die Kunden und Mitarbeiter zusammenfassen soll, können Sie die spezifischen Daten der beiden Personenarten in weiteren Tabellen speichern, die Sie 1:1 mit der »Basistabelle« verknüpfen.

Diese Tabellen könnten Sie *tblKunden* oder *tblMitarbeiter* nennen, aber daraus würde nicht deutlich, dass diese Tabellen lediglich Detaildaten zu einer weiteren Tabelle enthalten. Besser wären Bezeichnungen wie *tblPersonenKunden* und *tblPersonenMitarbeiter*.

Lookup-Tabellen

Die letzte Gruppe Tabellen, die nicht reale Objekte aus dem wirklichen Leben nachbilden, sind sogenannte »Lookup-Tabellen«. Diese Tabellen enthalten Informationen, die eigentlich zu den »normalen« Tabellen gehören, aber im Zuge der Normalisierung in eigene Tabellen ausgegrenzt wurden.

Beispiele dafür sind Anrede, Geschlecht oder Kontaktart. Für diese Tabellen gelten die gleichen Regeln wie für »normale« Tabellen, also beispielsweise folgende:

» *tblAnreden*

» *tblGeschlechter* (der Plural liest sich merkwürdig, ist aber hier konsequent)

» *tblKontaktarten*

Temporäre Tabellen

Gelegentlich benötigen Sie eine Tabelle nur kurze Zeit und löschen sie nach der Verwendung wieder. Um diese Tabellen von anderen unterscheiden zu können, vor allem aber, damit Sie einen Überblick behalten, welche Tabellen nur vorübergehend benötigt werden, können Sie diese Tabellen kennzeichnen, indem Sie ihnen das Suffix *Tmp* oder *Temp* anhängen, etwa *tblImportTemp*. Eine Einsatzmöglichkeit für solche Tabellen ist etwa das Speichern des Ergebnisses von aufwändigen Abfragen (gegebenenfalls mit eingebauten und benutzerdefinierten Funktionen). Wenn die Ermittlung des Abfrageergebnisses lange dauert und das Ergebnis beziehungsweise die zugrunde liegenden Daten sich selten ändern, macht das Speichern des Ergebnisses in einer temporären Tabelle sehr viel Sinn.

1.2.2 Feldnamen

Entgegen der bei Variablen üblichen Verwendung von Präfixen, durch die sich eine Aussage über den Datentyp der Variablen treffen lässt, verwendet man für Feldnamen im Allgemeinen kein Präfix. Das ist natürlich Geschmackssache; man findet jedoch kaum Datenbanken, in denen auch die Feldnamen mit einem entsprechenden Präfix versehen sind.

Primärschlüsselfelder

Der Name des Primärschlüsselfeldes setzt sich aus der Bezeichnung des durch die Tabelle abgebildeten Objekts im Singular und dem Suffix *ID* zusammen. Beispiele:

» *tblArtikel*: *ArtikelID*

» *tblPersonen*: *PersonID*

In Verknüpfungstabellen kommt es darauf an, ob die Tabelle einen eigenen Primärschlüssel hat oder ob sie aus den Fremdschlüsselfeldern zur Verknüpfung mit den jeweiligen Tabellen zusammengesetzt wird. Im ersteren Fall gibt es vermutlich auch einen sinnvollen Tabellennamen, der nicht aus den Namen der beiden verknüpften Tabellen besteht, wie etwa *tblBestelldetails*. Hier würde der Primärschlüssel (soweit vorhanden) *BestelldetailID* heißen.

In Tabellen, die zusätzliche Daten zu einer anderen Tabelle enthalten und per 1:1-Beziehung mit dieser verknüpft sind, verwendet man normalerweise den gleichen Namen für

das Primärschlüsselfeld wie in der Basistabelle. Gegebenenfalls ist das Verknüpfungsfeld der Erweiterungstabelle nicht das Primärschlüsselfeld der Erweiterungstabelle, sondern lediglich ein eindeutiges Feld. In diesem Fall greift wieder die erstgenannte Regel: Primärschlüsselname = enthaltenes Objekt im Singular + *ID*.

Fremdschlüsselfelder

Für die Benennung von Fremdschlüsselfeldern gibt es in der Praxis zwei Ansätze: Der einfachere verwendet einfach den Namen des Primärschlüsselfeldes der verknüpften Tabelle. Der zweite Ansatz macht noch ein wenig deutlicher, dass es sich bei diesem Feld definitiv um ein Fremdschlüsselfeld handelt, indem er dem Namen dieses Feldes noch das Präfix *ref* verpasst. Die zweite Variante hat Vorteile, wenn es um m:n-Beziehungen und 1:1-Beziehungen geht: Eine Tabelle mit zwei Feldern, die beide das Präfix *ref* enthalten, lässt sich zweifelsfrei als Verknüpfungstabelle identifizieren; und auch eine Tabelle, deren Primärschlüsselfeld das Präfix *ref* enthält, ist schnell als Erweiterungsteil einer 1:1-Beziehung enttarnt.

Allgemein

Sowohl für Feldnamen von Primärschlüsselfeldern, Fremdschlüsselfeldern als auch für alle anderen Felder gilt, dass der Name des Feldes sorgfältig gewählt sein will.

Am besten ist es, wenn er grobe Informationen über den Datentyp enthält:

» Text: *Vorname*, *Nachname*

» Datum: *Geburtsdatum*, *Einstellungsdatum*, *Erscheinungstermin*

» Zahlen: *Anzahl*, *Meldebestand*, *Lagerbestand*

» Währung: *Einzelpreis*

» Boolean: *MehrwertsteuerAusweisbar*, *Aktiv*, *InReihenfolge*

Unterstrich — ja oder nein?

Ob Sie in Tabellen- und Feldnamen den Unterstrich als Trennung zwischen einzelnen Wörtern verwenden oder die einzelnen Wörter einfach groß beginnen, ist reine Geschmackssache (in diesem Buch finden Sie übrigens ausschließlich letztere Variante). Wichtig ist, dass Sie es überhaupt hervorheben, wenn ein Tabellen- oder Variablenname aus mehr als einem Wort besteht.

Beispiele:

» *Anzahl_Datensaetze* oder *AnzahlDatensaetze*

» *Fahrzeug_ID* oder *FahrzeugID*

Man sollte es aber nicht übertreiben: Dem Autor sind schon Varianten untergekommen, in denen nicht nur jedes einzelne Wort, sondern fast jede Silbe großgeschrieben wurde (etwa *MehrwertSteuersatz*) – wenn ein Wort im Deutschen zusammengeschrieben wird, sind auch keine Großbuchstaben erforderlich.

Einen deutlichen Vorteil hat die Verwendung von Großbuchstaben zur Unterteilung mehrerer Wörter: Sie müssen sich nicht merken, was Sie groß- und was Sie kleingeschrieben haben. Access ist äußerst nachsichtig, was die Groß-/Kleinschreibung von Objektnamen angeht. Einen Unterstrich zu viel oder zu wenig wird Access Ihnen hingegen nicht verzeihen.

Lang oder kurz — mit oder ohne Abkürzung?

Die Zeiten, in denen die beschränkte Länge von Variablennamen die Fantasie der Programmierer beflügelte, sind zum Glück vorbei – und Gleiches gilt für Tabellen- und Feldnamen. Die in der Zwischenüberschrift gestellte Frage nach der Länge von Feldnamen ist leicht zu beantworten: So lang wie nötig und so kurz wie möglich. Der Feldname sollte nicht in die Irre führen, nur um ein paar Zeichen zu sparen, andererseits lassen sich zu lange Feldnamen nicht gut merken. Wenn Sie für die Zusammenstellung jeder einzelnen SQL-Anweisung erst das Beziehungsfenster öffnen müssen, wissen Sie, dass Sie an den Tabellen- und Feldnamen arbeiten müssen. Für Tabellen- wie Feldnamen gilt außerdem: Die Länge kann sich negativ auf die Performance auswirken. Wann immer Zeichenketten im Spiel sind, sollten Sie sich auf das Notwendigste beschränken.

1.3 Normalisierung

Unter Normalisierung versteht man die Überführung des Datenmodells (also der Tabellen und ihrer Beziehungen) in einen bestimmten Zustand. Dieser Zustand wird durch die Nummer der sogenannten Normalform unterschieden. Meistens genügt das Erreichen der dritten Normalform, um Redundanzen und Inkonsistenzen vorzubeugen und dadurch die Wartung der enthaltenen Daten zu vereinfachen.

Die Möglichkeit, Redundanzen und Inkonsistenzen zu vermeiden, ist eine der Haupteigenschaften von relationalen Datenbanksystemen. Jeder, der schon einmal eine Datenbank für einen Kunden entwickeln sollte, der die betroffenen Daten zuvor mit Excel verwaltet hat, und das Vergnügen hatte, auch den Import dieser Daten vorzunehmen, weiß, was Redundanzen und Inkonsistenzen sind (das soll keine Verunglimpfung der Möglichkeiten von Excel sein – aber dessen Stärken liegen eher woanders).

Ein gern gesehenes Beispiel ist die Verwaltung von Rechnungen in einer einzigen Tabelle. Dort finden sich unter Umständen alle Rechnungs- und Kundendaten zu ei-

ner Rechnung in einer Zeile. Sobald zwei Rechnungen für den gleichen Kunden gespeichert werden, gibt es auch zwei Kopien der Kundendaten in der Tabelle. Ändern sich die Kundendaten, werden diese Änderungen möglicherweise nur in einer neuen Zeile vorgenommen. Sobald ein anderer Mitarbeiter eine Rechnung für diesen Kunden stellen soll, steht er vor mindestens zwei verschiedenen Kunden-Datensätzen und das Unheil nimmt seinen Lauf. Um solches Ungemach zu verhindern, gibt es relationale Datenbanken, die Normalformen und die referentielle Integrität.

Halb automatisches Normalisieren

In den folgenden Abschnitten lernen Sie die unterschiedlichen Normalformen kennen und finden Beispiele für das Umwandeln nicht normalisierter Tabellen in die jeweilige Normalform.

Die Zwischenüberschrift bezieht sich darauf, dass jeder Normalisierungsschritt nicht vollautomatisch abläuft. Genau genommen besteht jeder Schritt aus drei Teilen: Der erste passt das Datenmodell an und der zweite sorgt für die Umorganisierung der vorhandenen Daten.

Der dritte Schritt räumt auf und löscht eventuell nicht mehr benötigte Felder. Das Anpassen des Datenmodells und damit des Tabellenentwurfs erfolgt manuell. Das Umorganisieren der Daten kann – bei kleinen Datenmengen – ebenfalls manuell erfolgen, aber das ist sicher keine Arbeit für einen Entwickler: Der baut sich eine kleine Routine, die diesen Vorgang mit ein paar Aktionsabfragen oder im schlimmsten Fall ein paar ADO- oder DAO-Anweisungen automatisch durchführt.

Warum nicht direkt normalisieren?

Der Grund für den Einsatz der Normalisierung liegt meist in Altlasten bezogen auf die Organisation der Daten vor der Erstellung einer Datenbankanwendung. Viele Datenbanken werden neu erstellt, weil bestehende Daten auf die bisherige Art und Weise nicht mehr verwaltet werden können – entweder es sind keine Erweiterungen mehr möglich, es wurden immer wieder neue Tabellen und Felder an das bestehende Datenmodell angestückelt und die Anwendung läuft nicht mehr schnell genug oder die Daten liegen in einem nicht für diesen Zweck geeigneten Format vor – etwa in Form von Excel-Tabellen.

In diesen Fällen müssen Sie ein bestehendes Datenmodell normalisieren. Das bedeutet nicht, dass Sie mit spitzen Fingern an der Originaldatenbank herumschrauben müssen.

Meist werden Sie eher eine neue Datenbank erstellen und ein neues Datenmodell aufsetzen, das alle in dem vorhandenen Datenmodell enthaltenen Informationen beinhaltet. Anschließend werden Sie die Daten in die neue Datenbank importieren – natürlich entsprechend umorganisiert.

2

Abfragen

Abfragen sind die Schnittstelle zwischen den in den Tabellen enthaltenen Daten und den für die Bearbeitung und Anzeige zuständigen Formularen und Berichten sowie für die Weiterverarbeitung per VBA. Genau wie Tabellen sollten die Benutzer einer professionellen Datenbank auch eine Abfrage nie zu Gesicht bekommen. Sie dient lediglich dazu, die Daten für die eigentliche Bearbeitung aufzubereiten. Dabei gibt es verschiedene Möglichkeiten:

» Einschränken der Felder einer Tabelle: Mit einer Abfrage können Sie die Felder auswählen, deren Inhalte als Abfrageergebnis angezeigt werden sollen.

» Einschränken der Daten einer Tabelle: Genau wie die Felder können Sie auch die anzuzeigenden Datensätze einer Tabelle per Abfrage einschränken. Dazu verwenden Sie ein geeignetes Kriterium für ein oder mehrere Felder.

» Zusammenführen der Daten verschiedener Tabellen: Nach der Normalisierung liegen Daten in vielen Fällen in mehreren, miteinander verknüpften Tabellen vor. In einer Abfrage können Sie die Felder der verknüpften Tabellen wieder zusammenführen und – in den meisten Fällen – wie eine einzige Tabelle verwenden.

» Zusammenführen der Daten gleichartiger Tabellen: Auch wenn man gleichartige Daten aus verschiedenen Tabellen benötigt – etwa die Namen der Mitarbeiter aus der Mitarbeitertabelle und diejenigen aus der Kunden-Tabelle –, hilft eine Abfrage weiter (siehe »UNION-Abfragen« ab Seite 141).

» Spezielle Aufbereitung von Daten: Mit Hilfe von Kreuztabellenabfragen lassen sich Daten wie beispielsweise die Verkaufszahlen von Produkten in bestimmten Zeiträumen in einer Art Matrix ausgeben.

» Berechnungen auf Basis der Felder der zugrunde liegenden Tabellen ausführen: Berechnete Werte in Tabellenfeldern sind bekanntlich tabu, da diese zu redundanten Daten führen. Daher gehören Berechnungen in Abfragen.

Abfragen werden Sie überall antreffen: Als Datensatzquelle von Formularen und Berichten, als Datensatzherkunft von Kombinations- und Listenfeldern, als Datenquelle von Recordset-Objekten in VBA-Code und vielleicht auch als Bestandteil einer weiteren Abfrage.

BEISPIELDATENBANK

Die Beispieldatei zu diesem Kapitel finden Sie unter dem Namen *Abfragen.accdb* auf *www.acciu.de/aeb2010*.

Weil Abfragen so wichtig sind, stellt Access eine mächtige Entwurfsansicht dafür bereit, die nur wenig Wünsche offen lässt. Zwar lassen sich dort keine UNION-Abfragen oder PassThrough-Abfragen eingeben; dafür ist aber eine zusätzliche SQL-Ansicht vorhanden, in die man nicht nur diese Abfragen eingeben, sondern mit der man auch den SQL-Text der anderen Abfragen ausgeben kann.

Wie Sie vielleicht zu Beginn des Buches gelesen haben, setzt dies grundlegende Kenntnisse im Umgang mit Access voraus.

Daher finden Sie im Folgenden auch keine detaillierte Beschreibung für die Anwendung der Abfrage-Entwurfsansicht, sondern lediglich eine kurze Einführung in das Anlegen von Abfragen mit Access 2010 und – wesentlich umfangreicher als die Einführung – Informationen zu oft benötigten Vorgehensweisen im Zusammenhang mit Abfragen.

Dabei findet gelegentlich ein Vorgriff auf die im Kapiel »Access-SQL« ab Seite 451 enthaltene umfassende Beschreibung der Abfragesprache SQL statt.

2.1 Anlegen von Abfragen mit Access 2010

Prinzipiell gibt es nur zwei mögliche Startpunkte für das Anlegen von Abfragen in Access 2010: den Assistenten, den Sie im Ribbon mit *Erstellen|Andere|Abfrage-Assistent* aufrufen, sowie den direkten Sprung zur Entwurfsansicht, zu erreichen mit *Erstellen|Andere|Abfrageentwurf*. Da dieses Buch keine Assistenten behandelt (die meisten davon sind selbsterklärend; probieren Sie sie ruhig aus!), bleibt nur noch die Beschreibung der Entwurfsansicht (siehe Abbildung 2.1).

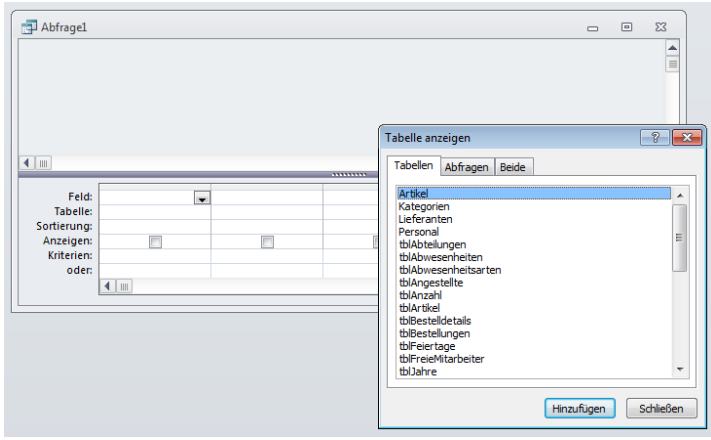


Abbildung 2.1: Eine leere Abfrage mit dem Dialog zum Auswählen der gewünschten Tabellen

Da die Chancen gut stehen, dass Sie schon wissen, wie Sie eine Auswahlabfrage erstellen, was Kriterien sind, wie Sie Aktionsabfragen wie Aktualisierungs-, Anfüge-, Lösch- und Tabellenerstellungsabfragen erstellen und dass Sie für manche Abfragetypen wie etwa UNION-Abfragen die SQL-Ansicht heranziehen müssen, spart dieses Buch die diesbezüglichen Grundlagen aus und verweist auf die Onlinehilfe.

IntelliSense im Abfrageentwurf

Es gibt jedoch eine Neuerung unter Access 2010, die den Entwurf von Abfragen betrifft. IntelliSense ist nun auch hier verfügbar, zum Beispiel für die Definition der im Abfrageergebnis anzuzeigenden Feldinhalte. Klicken Sie einmal in die Zeile *Feld* des Abfrageentwurfs und betätigen Sie die Tastenkombination *Strg + F2*. Es erscheint der Ausdrucks-Generator, der vor allem beim Zusammenstellen berechneter Ausdrücke hervorragende Dienste leistet (siehe Abbildung 2.2). Und auch beim Festlegen der Kriterien einer Abfrage können Sie sich auf IntelliSense verlassen, wie Abbildung 2.3 zeigt. Geben Sie einfach den ersten Buchstaben des gewünschten Schlüsselworts oder Objektnamens ein und IntelliSense bietet alle infrage kommenden Ausdrücke an.

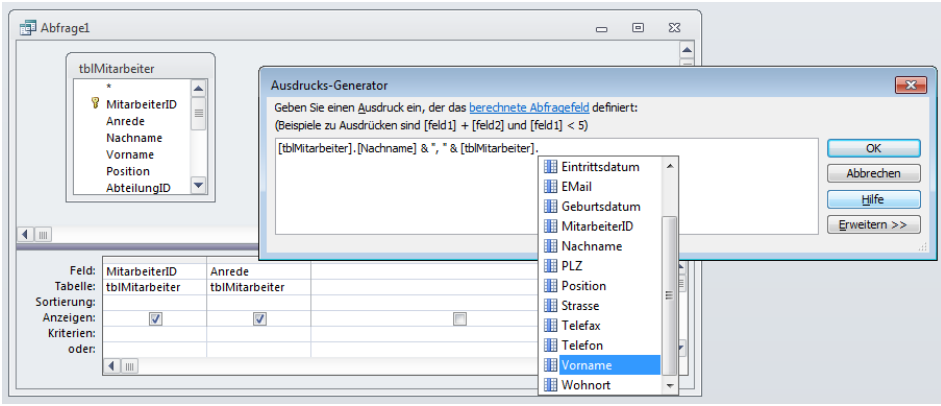


Abbildung 2.2: Eingabe von Ausdrücken für die Feld-Eigenschaft einer Abfrage

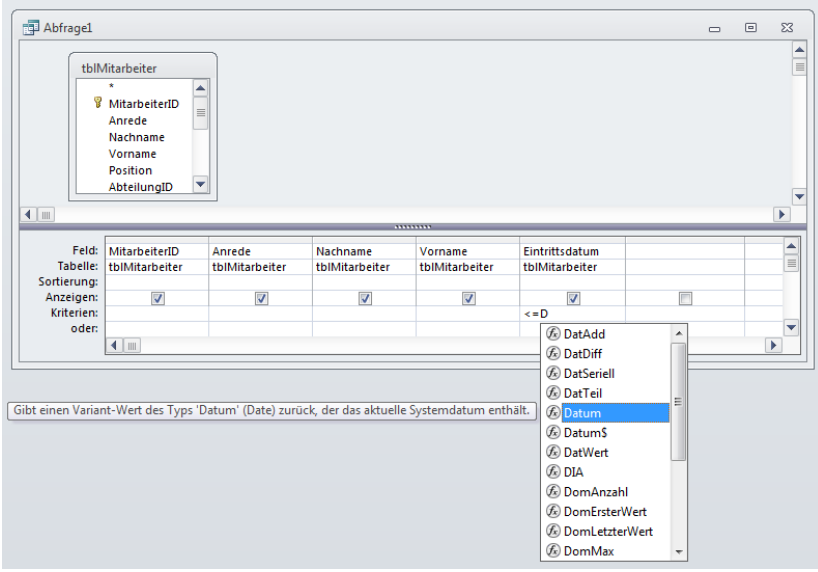


Abbildung 2.3: Auswahl von Kriterien per IntelliSense

2.2 Abfragen mit Anlage-Feldern und mehrwertigen Feldern

Für Versionsspringer, die von Access 2003 gleich zu Access 2010 wechseln, ist außerdem der Umgang mit mehrwertigen Feldern in Abfragen wichtig. Abfragen bieten nämlich eine Möglichkeit, auf die in mehrwertigen Feldern und in Anlage-Feldern gespei-

cherten Daten zuzugreifen. Dazu fügen Sie einfach der Abfrage eine Tabelle mit einem solchen Feld hinzu und erkennen bereits in der Entwurfsansicht, dass nicht nur das Feld selbst, sondern auch die in der damit verknüpften, verborgenen Tabelle enthaltenen Felder angezeigt werden (siehe Abbildung 2.4).

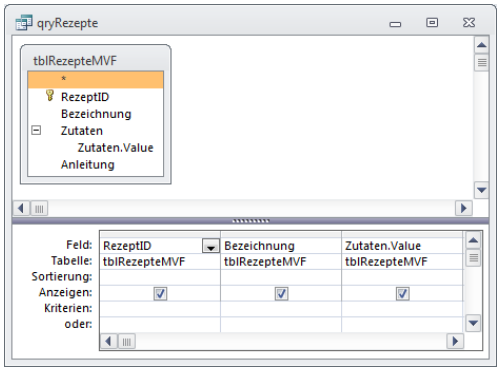


Abbildung 2.4: Abfrage auf eine Tabelle mit einem mehrwertigen Feld

Wenn Sie die obige Abfrage auf die Tabelle aus Abbildung 2.5 ausführen, würden Sie normalerweise vermutlich zwei Datensätze als Ergebnis erwarten. Aber weit gefehlt: Wie Abbildung 2.6 zeigt, enthält die Abfrage für jede Kombination aus Rezeptbezeichnung und Zutat einen einzelnen Datensatz. Im Prinzip ist dies das Gleiche, als wenn Sie die entsprechenden Felder der drei an einer m:n-Beziehung beteiligten Tabellen abfragen.

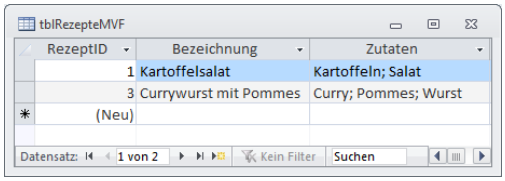


Abbildung 2.5: Eine Abfrage auf eine Tabelle mit einem mehrwertigen Feld wie diese ...

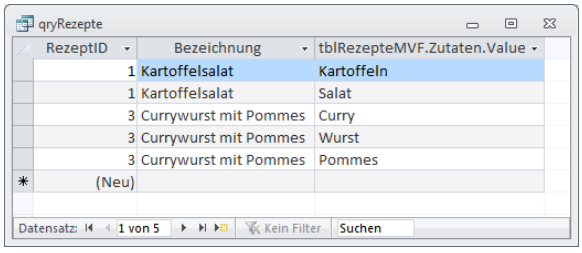


Abbildung 2.6: ... zeigt mitnichten die gleiche Anzahl Datensätze an

2.3 Verwendung von Abfragen als Datensatzquelle oder Datensatzherkunft

Formulare und Berichte beziehen ihre Daten aus der unter der Eigenschaft *Datensatzquelle* angegebenen Tabelle oder Abfrage, bei Kombinations- und Listenfeldern heißt die entsprechende Eigenschaft *Datensatzherkunft*. Zum Füllen dieser beiden Eigenschaften gibt es verschiedene Techniken, die nachfolgend erläutert werden. Die einfachste ist das Setzen der entsprechenden Eigenschaft auf eine bestehende Tabelle oder Abfrage. Der Einsatz einer Tabelle ist dabei nur sinnvoll, wenn alle Felder und alle Datensätze der Tabelle benötigt werden.

Wenn nicht alle Felder der Tabelle Verwendung finden oder nicht alle Datensätze angezeigt werden sollen, verwenden Sie eine Abfrage. Das kommt auch der Performance Ihrer Anwendung zu Gute. Die Herkunft der Daten heißt in Formularen und Berichten *Datensatzquelle* und in Steuerelementen wie dem Kombinationsfeld oder Listenfeld *Datensatzherkunft*. Wenn es im Folgenden nicht explizit um die Datensatzherkunft solcher Steuerelemente geht, wird verallgemeinernd der Begriff *Datensatzquelle* verwendet.

2.3.1 Tabelle als Datensatzquelle

Die einfachste Art der Datensatzquelle ist eine Tabelle. In vielen Fällen haben Tabellen aber mehr Felder oder enthalten mehr Datensätze als tatsächlich angezeigt werden sollen. Lookup-Tabellen, die nur aus einem Primärschlüsselfeld und einem weiteren Feld bestehen, können aber durchaus ohne Verwendung einer Abfrage eingesetzt werden – etwa als Datensatzherkunft von Kombinationsfeldern.

Erst wenn die enthaltenen Daten auch noch sortiert werden sollen, ist eine Abfrage erforderlich. Beispiel für diese und die folgenden Möglichkeiten zur Bestückung von Formularen und Steuerelementen mit Datenherkünften ist ein Formular, das die Abwesenheit von Mitarbeitern nach Kalenderjahren filtert (siehe Abbildung 2.7).

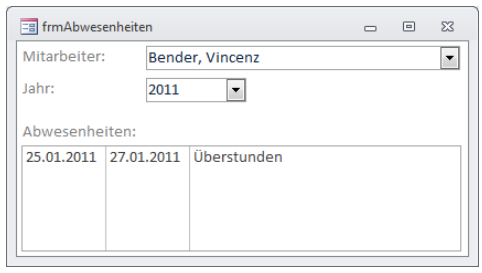


Abbildung 2.7: Beispiel für das Zuweisen der Datensatzherkunft

Ein gutes Beispiel für die Verwendung einer Tabelle als Datensatzherkunft ist das Kombinationsfeld *cboMonat*. Wenn die Daten in der gewünschten Reihenfolge vorliegen, können Sie guten Gewissens eine Tabelle statt einer Abfrage als Datensatzherkunft verwenden.

Das ist hier der Fall: Das einzige Feld Jahr dient der Auswahl des Jahres, für das die Abwesenheiten angezeigt werden sollen (siehe Abbildung 2.8). Eine Tabelle oder Abfrage, die als Datensatzherkunft von Kombinationsfeldern oder Listenfeldern dient, kann auch mehr Felder besitzen, als das Steuerelement anzeigt. Das Steuerelement fragt automatisch nur die benötigten Felder ab.

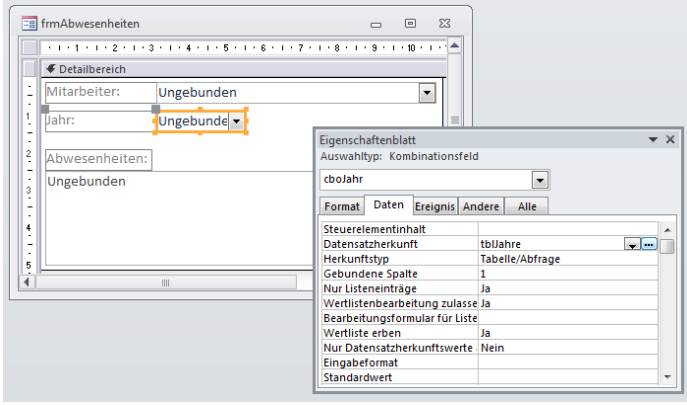


Abbildung 2.8: Tabelle als Datensatzherkunft

2.3.2 SQL-Ausdruck als Datensatzquelle

Die zweite Möglichkeit sind reine SQL-Ausdrücke. Diese können statt des Namens der Tabelle oder der Abfrage für die *Datensatzquelle*- oder *Datensatzherkunft*-Eigenschaft angegeben werden. In vielen Fällen geht es einfach schneller, wenn man mal eben eine kurze SQL-Anweisung für die entsprechende Eigenschaft einträgt, als wenn man zunächst eine Abfrage erstellt, diese speichert und dann die Abfrage als Wert der jeweiligen Eigenschaft einträgt.

Im Beispielformular ist das Kombinationsfeld *cboMitarbeiter* mit einem solchen SQL-Ausdruck ausgestattet. Dieser lautet folgendermaßen:

```
SELECT tblMitarbeiter.MitarbeiterID, [Nachname] & ", " & [Vorname] AS Mitarbeiter FROM
tblMitarbeiter;
```

Dieser Ausdruck wurde über die Entwurfsansicht für Abfragen erstellt, aber nicht sichtbar als Abfrage gespeichert (die Abfrage wird nicht im Navigationsfenster angezeigt, ist aber in der Systemtabelle *MSysObjects* unter einem Namen wie *~sq_cfrmAbwesenheiten~sq_*

ccboMitarbeiter aufgeführt). In diesem Fall wird der reine SQL-Ausdruck in die Eigenschaft *Datensatzherkunft* des Kombinationsfeldes eingetragen.

2.3.3 Gespeicherte Abfrage als Datensatzquelle

Die einfachste, weil ohne VBA- und SQL-Kenntnisse zu bewältigende und daher auch für Einsteiger geeignete Möglichkeit zur Erstellung einer Abfrage bietet die dafür vorgesehene Entwurfsansicht für Abfragen. Damit ist die Einschränkung der Datensatzquelle sowohl bezüglich der Felder als auch der Datensätze möglich und auch die Verwendung von Parametern ist relativ einfach.

Die Datensatzherkunft des Kombinationsfeldes *cboMitarbeiter* lässt sich natürlich ebenso mit einer gespeicherten Abfrage wie mit einem SQL-Ausdruck füllen. Um den SQL-Ausdruck in eine im Navigationsbereich sichtbar gespeicherte Abfrage zu überführen, klicken Sie einfach auf die Schaltfläche mit den drei Punkten (...) und speichern die nun in der Entwurfsansicht angezeigte Abfrage ab – beispielsweise unter dem Namen *qryFrmAbwesenheitenCbo-Mitarbeiter*. Auf diese Weise erkennen Sie später schnell, wofür Sie diese Abfrage benötigen.

Verwenden Sie nun besser einen SQL-Ausdruck oder eine gespeicherte Abfrage als Datensatzquelle? Performancetechnisch betrachtet besteht kein großer Unterschied – in beiden Fällen wird die Abfrage beim ersten Aufruf kompiliert und damit optimiert (siehe auch »Abfragen und die ACE-Engine« ab Seite 840). Bleiben zwei Gründe, die für das Speichern der Abfrage sprechen: Entweder benötigen Sie die Abfrage an mehreren Stellen oder Sie möchten die Abfrage testen beziehungsweise optimieren, während das Formular in der Formularansicht angezeigt wird. In allen anderen Fällen scheint die Verwendung eines SQL-Ausdrucks naheliegender, zumal der ohnehin spärliche Platz im Navigationsbereich sonst ziemlich schnell überfüllt sein dürfte und seine Suchfunktion zum Einsatz käme.

2.3.4 Datensatzquelle per VBA zuweisen

Die Eigenschaften *Datensatzquelle* und *Datensatzherkunft* stehen auch unter VBA zur Verfügung – dort verwendet man die Eigenschaftsnamen *RecordSource* (Formular) und *RowSource* (Kombinations- und Listenfelder). Sie können diesen Eigenschaften von Formularen sowie Kombinations- und Listenfeldern per VBA den Namen einer Tabelle oder Abfrage oder auch einen SQL-Ausdruck zuweisen. Was macht das für einen Sinn? Manchmal weiß man noch nicht genau, wie die anzuzeigenden Daten gestaltet sind. Das ist meist bei Suchformularen der Fall: Ein Formular bietet mehrere Such- und Sortierkriterien an, die der Benutzer mit den gewünschten Werten füllen kann. Man könnte die Kriterien einfach in Form von Parametern an die der Suche zugrunde liegende Abfrage übergeben, aber je nach Komplexität und Anzahl der ent-

haltenen Tabellen gerät die Abfrage recht komplex. Eine Abfrage über die drei per m:n-Beziehung verknüpften Tabellen *tblBestellungen*, *tblBestelldetails* und *tblArtikel*, deren Suchkriterien sich über die äußeren Tabellen erstrecken, benötigt auch alle Tabellen der Abfrage. Soll eine Suche allerdings alle Artikel liefern, deren Suchkriterien sich lediglich auf die Tabellen *tblBestelldetails* und *tblArtikel* erstrecken, kann man die Tabelle *tblBestellungen* und die notwendige Verknüpfung und damit Zeit und Ressourcen sparen – vorausgesetzt, das Abfrageergebnis gibt keine darin enthaltenen Felder zurück. Hier würde dann eine VBA-Routine zum Einsatz kommen, die einen SQL-Ausdruck mit den benötigten Tabellen zusammensetzt.

2.3.5 Parameter statt Zusammensetzen von SQL-Ausdrücken

Auch in anderen, einfacheren Fällen, in denen eine Abfrage lediglich aus einer einzigen Tabelle besteht und nur ein Parameter eingesetzt werden muss, verwenden viele Entwickler VBA, um einen SQL-String zusammenzusetzen und diesen als Datensatzquelle zu verwenden. Das sieht dann beispielsweise so aus:

```
Private Sub ListenfeldAktualisieren()
    Dim strSQL As String
    Dim strSQLSelect As String
    Dim strSQLWhere As String
    'Basisabfrage (SELECT-Teil)
    strSQLSelect = "SELECT AbwesenheitID, StartDatum, EndDatum, " _
        & " Abwesenheitsart FROM tblAbwesenheitsarten " _
        & "INNER JOIN tblAbwesenheiten " _
        & "ON tblAbwesenheitsarten.AbwesenheitsartID = " _
        & "tblAbwesenheiten.Abwesenheitart "
    'Erstes Kombinationsfeld auswerten
    If Not Nz(Me!cboMitarbeiter, 0) = 0 Then
        strSQLWhere = "MitarbeiterID = " & Me!cboMitarbeiter
    End If
    'Zweites Kombinationsfeld auswerten
    If Not Nz(Me!cboJahr, 0) = 0 Then
        If Len(strSQLWhere) > 0 Then
            strSQLWhere = strSQLWhere & " AND "
        End If
        strSQLWhere = strSQLWhere _
            & "Year(tblAbwesenheiten.StartDatum) = " _
            & Me!cboJahr.Column(1)
    End If
```

```
    'SELECT-Teil zum SQL-Ausdruck hinzufügen
    strSQL = strSQLSelect
    'Falls WHERE-Bedingung vorhanden,
    'WHERE-Teil zum SQL-Ausdruck hinzufügen
    If Len(strSQLWhere) > 0 Then
        strSQL = strSQL & " WHERE " & strSQLWhere
    End If
    'Neue Datensatzherkunft zuweisen und Listenfeld aktualisieren
    Me!lstAbwesenheiten.RowSource = strSQL
    Me!lstAbwesenheiten.Requery
End Sub
```

Listing 2.1: Datensatzherkunft für ein Listenfeld per zusammengesetzten SQL-Ausdruck ermitteln

Die Prozedur wird von den beiden Prozeduren aufgerufen, die durch die Ereigniseigenschaft *Nach Aktualisierung* der beiden Kombinationsfelder *cboMitarbeiter* und *cboJahr* ausgelöst werden. Die hier ermittelte SQL-Anweisung ist bei keiner Ausführung kompiliert. Eine Alternative ist die Verwendung einer gespeicherten Abfrage mit Parametern. Die Parameter, die normalerweise die Anzeige eines Dialogs zum Eingeben des Parameters hervorrufen, füllen Sie ebenfalls per VBA.

Beim späteren Aufruf weisen Sie dem Listenfeld ein Recordset zu, das auf der kompilierten Abfrage inklusive Parametern basiert. Dies funktioniert übrigens erst ab Access XP. Unter Access 2000 und älteren Versionen von Access haben Kombinations- und Listenfelder noch keine Recordset-Eigenschaft.

Die Abfrage sieht wie in Abbildung 2.9 aus. Die ersten vier Felder der Abfrage werden angezeigt, die letzten beiden sind lediglich Kriterienfelder. Als Kriterien dienen die per VBA zu füllenden Parameter *[cboMitarbeiter]* und *[cboJahr]*. Ersterer wird direkt mit dem Inhalt des Feldes *MitarbeiterID* verglichen, Letzterer mit dem Ausdruck, der durch die Anwendung der *Jahr*-Funktion auf dem Inhalt des Feldes *StartDatum* erzeugt wird. Dabei handelt es sich um die dem Datum entsprechende Jahreszahl.

Nun fehlt noch die Prozedur, mit der die Parameter per Code gefüllt werden und das Ergebnis der Abfrage dem Listenfeld zugewiesen wird. Diese Prozedur erstellt ein *QueryDef*-Objekt auf Basis der Abfrage *qryFrmAbwesenheitenLstAbwesenheitParameter*. Dieses Objekt enthält eine Auflistung namens *Parameters*, mit der Sie die in der Abfrage gespeicherten Parameter referenzieren und die gewünschten Werte zuweisen können. Der Parameter *[cboMitarbeiter]* soll mit dem gebundenen Feld der Datensatzherkunft des Kombinationsfeldes *cboMitarbeiter* gefüllt werden, der Parameter *[cboJahr]* wird mit dem im Kombinationsfeld *cboJahr* angezeigten Wert bestückt.

Beachten Sie, dass der angezeigte Wert nicht mit dem Wert des gebundenen Feldes übereinstimmt, sondern das Jahr und nicht dessen ID enthält!

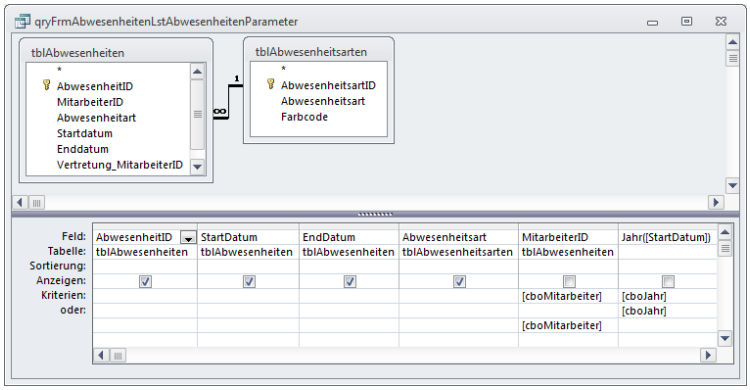


Abbildung 2.9: Abfrage mit Parametern

Nach dem Füllen der Parameter wird die Abfrage mit der *OpenRecordset*-Methode ausgeführt und das Ergebnis in ein *Recordset*-Objekt geschrieben, das schließlich der entsprechenden Eigenschaft des Listenfeldes zugewiesen wird. Die Prozedur verwendet einige Objekte, Methoden und Eigenschaften der DAO-Bibliothek von Access. Detaillierte Informationen finden Sie im Kapitel »DAO« ab Seite 527.

```
Private Sub ListenfeldAktualisierenParameter()
    Dim db As DAO.Database
    Dim qdf As DAO.QueryDef
    Dim rst As DAO.Recordset
    'Database- und Querydef-Objekt festlegen
    Set db = CurrentDb
    Set qdf = db.QueryDefs("qryFrmAbwesenheitenLstAbwesenheitenParameter")
    'Parameter [cboMitarbeiter] mit der im Kombinationsfeld
    'cboMitarbeiter ausgewählten MitarbeiterID füllen
    qdf.Parameters("cboMitarbeiter").Value = Me!cboMitarbeiter
    'Parameter [cboJahr] mit dem im Kombinationsfeld angezeigten Jahr füllen
    qdf.Parameters("cboJahr").Value = Me!cboJahr.Column(1)
    'Abfrage ausführen und Ergebnis in Recordset-Objekt ablegen
    Set rst = qdf.OpenRecordset
    'Recordset der gleichnamigen Eigenschaft des Listenfelds zuweisen
    Set Me!lstAbwesenheiten.Recordset = rst
    Set rst = Nothing
    Set qdf = Nothing
    Set db = Nothing
End Sub
```

Listing 2.2: Listenfeld mit Parameterabfrage füllen

Der erste Test mit dieser Routine läuft nur zufrieden stellend, wenn in beiden Kombinationsfeldern ein Wert ausgewählt ist. Eigentlich soll das Listenfeld beim Öffnen des Formulars alle Abwesenheiten anzeigen, beim Auswählen eines Mitarbeiters ohne Jahr alle Abwesenheiten dieses Mitarbeiters für alle Jahre, um beim Auswählen lediglich eines Jahres alle Abwesenheiten dieses Jahres für alle Mitarbeiter zu berücksichtigen.

Das funktioniert deshalb nicht, weil die Abfrage beispielsweise bei fehlender Auswahl des Mitarbeiters den Wert *Null* als Parameter übergibt. Und da es keine Mitarbeiter mit der *MitarbeiterID Null* gibt, werden auch keine Abwesenheiten angezeigt. Das Gleiche gilt für die Auswahl des Jahres.

Die Verwendung von *Null* als Standardwert bei fehlender Eingabe eines Parameters ist übrigens nicht zwingend, sondern in diesem Fall durch die Verwendung des Standarddatentyps *Variant* bedingt. Sie können für einen Parameter in der Abfragedefinition durchaus andere Datentypen angeben; folglich werden dann auch die entsprechenden Standardwerte bei fehlendem Wert verwendet (etwa *0* bei Zahlentypen oder eine leere Zeichenkette bei *String*-Variablen). Sie müssen also dafür sorgen, dass die Parameter im Falle einer fehlenden Auswahl neutralisiert werden. Folgender Trick hilft dabei weiter: Fügen Sie in die Kriterienspalte der betroffenen Felder die folgenden abgewandelten Ausdrücke ein. Ein Datensatz wird angezeigt, wenn das Kriterium wahr ist – und das ist entweder bei passendem Parameterwert oder bei der Übergabe des Wertes *Null* der Fall:

```
[cboMitarbeiter] Oder [cboMitarbeiter] Ist Null
[cboJahr] Oder [cboJahr] Ist Null
```

Interessant ist, was Access nach dem Schließen und erneuten Öffnen aus den Kriterien macht (siehe Abbildung 2.10).

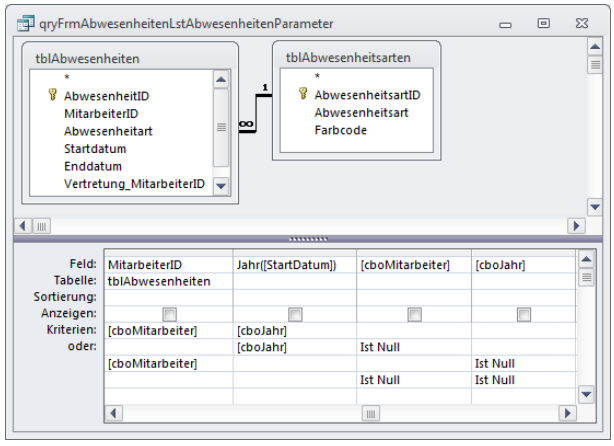


Abbildung 2.10: Zwei harmlose Kriterienausdrücke nach der Überarbeitung durch Access

2.3.6 Abfragen mit Parameter oder zusammengesetzte SQL-Ausdrücke?

Welche der beiden Varianten Sie verwenden, hängt von der Menge der Parameter ab. Je mehr Parameter vorkommen, desto langsamer wird die Abfrage und umso komplizierter wird der Abfrageentwurf.

Wenn Sie sich den Abfrageentwurf aus Abbildung 3.10 ansehen und sich vorstellen, wie eine Abfrage mit vier oder mehr Parametern aussehen wird, können Sie sich vermutlich ausmalen, wie viel Spaß eine nachträgliche Änderung am Abfrageentwurf machen wird. Für Abfragen mit mehreren Parametern empfiehlt sich daher eher die Verwendung eines per VBA zusammengesetzten SQL-Ausdrucks.

2.3.7 Probleme mit Kriterienausdrücken bei SQL-Ausdrücken in VBA

Viele Fehler bei der Verwendung von SQL-Ausdrücken unter VBA passieren im Zusammenhang mit den Kriterien. Mal meldet Access das Problem, dass zu wenig Parameter übergeben wurden (siehe Abbildung 2.11), ein anderes Mal funktionieren die Vergleiche mit übergebenen Datumsangaben nicht.

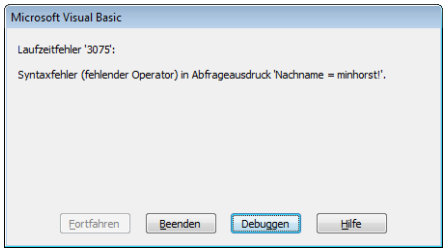


Abbildung 2.11: Fehlermeldung beim Verwenden einer SQL-Anweisung per VBA

2.3.8 Zeichenkette oder Zahlenwert?

Der Fehler aus Abbildung 2.11 resultiert fast immer aus dem Fehlen von Anführungszeichen im SQL-Ausdruck beim Verwenden von Zeichenketten als Kriterium.

Das folgende Listing zeigt einen solchen Fehler:

```
Public Function MitarbeiterSuchen(strNachname As String)
    …
    Set rst = CurrentDB.OpenRecordset("SELECT MitarbeiterID, " _
        & "Vorname, Nachname FROM tblMitarbeiter WHERE Nachname = " _
```

```
        & strNachname)
    …
End Function
```

Listing 2.3: Falsche Verwendung einer Zeichenkette als Kriterium

In dieses Listing wurden keine Anführungszeichen für die Zeichenkette integriert. Der SQL-Ausdruck sieht für den Aufruf *MitarbeiterSuchen "Müller"* wie folgt aus:

```
SELECT MitarbeiterID, Vorname, Nachname
FROM tblMitarbeiter
WHERE Nachname = Müller
```

»Müller« wird hierbei nicht als Zeichenkette, sondern als Parameter ausgelegt. Da für diesen kein Wert vorliegt, erscheint obige Fehlermeldung. Die Lösung des Problems ist einfach. Fassen Sie den Parameter einfach in Anführungszeichen oder Hochkommata ein (in einer Zeile):

```
Set rst = db.OpenRecordset("SELECT MitarbeiterID, Vorname, Nachname FROM tblMitarbeiter
WHERE Nachname = '" & strNachname & "'")
```

oder besser

```
Set rst = db.OpenRecordset("SELECT MitarbeiterID, Vorname, Nachname FROM tblMitarbeiter
WHERE Nachname = """ & strNachname & """")
```

2.3.9 Probleme mit Datumsangaben

Auch Datumsangaben führen immer wieder zu Problemen. Die folgende Routine soll beispielsweise Informationen über Abwesenheiten ausgeben, deren Beginn in einem bestimmten Zeitraum liegt, der durch die Parameter *datStart* und *datEnde* angegeben werden kann.

```
Public Function AbwesenheitenZeitraum(datStart As Date, datEnde As Date)
    …
    Set rst = db.OpenRecordset("SELECT * FROM tblAbwesenheiten " _
        & "WHERE Startdatum BETWEEN " & datStart & " AND " & datEnde, _
        dbOpenDynaset)
    …
End Function
```

Listing 2.4: Ermitteln von Abwesenheiten in einem bestimmten Zeitraum

Wenn Sie die Routine mit folgendem Aufruf starten, erscheint die Fehlermeldung aus Abbildung 2.12. Auf den ersten Blick scheinen hier die Anführungszeichen im resultierenden SQL-Ausdruck zu fehlen:

```
AbwesenheitenZeitraum "1.1.2007", "31.1.2007"
```

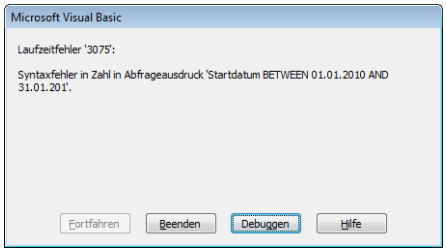


Abbildung 2.12: Fehlermeldung bei der Verwendung von Datumsangaben in SQL-Ausdrücken

Ändern Sie den Aufruf der SQL-Anweisung wie folgt um, gibt es allerdings eine andere Fehlermeldung (siehe Abbildung 2.13):

```
Set rst = db.OpenRecordset("SELECT * FROM tblAbwesenheiten WHERE Startdatum BETWEEN '"
& datStart & "' AND '" & datEnde & "'", dbOpenDynaset)
```

Diesmal hat Access Probleme mit dem Datentyp – ein String scheint das Bedürfnis nach einem Wert des Typs *DATETIME* nicht zu befriedigen. Mit dem Wissen, dass Access Datumsangaben intern als Zahlenwerte behandelt, erscheint dies schnell logisch. Dabei gibt es zwei Möglichkeiten: Verwenden Sie den Datentyp *Double*, um Datumsangaben inklusive Uhrzeit zu verwalten, dann entspricht die Zahl vor dem Komma der Anzahl Tage seit dem 31.12.1899 und die Zahl nach dem Komma der Anzahl Sekunden, die am angegebenen Tag verstrichen sind. Für Datumsangaben ohne Uhrzeit reicht dementsprechend der Datentyp *Long* aus.

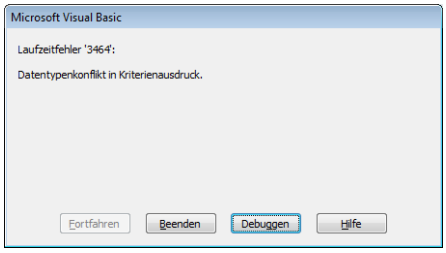


Abbildung 2.13: Auch die Zeichenkette taugt nicht als Datumskriterium

Nun müssen Sie aber nicht alle Datumsangaben explizit in einen Zahlen-Datentyp umwandeln. Es reicht, wenn Sie ein standardisiertes Format verwenden. Dieses hat die Form *yyyy-mm-dd*. Zusätzlich fassen Sie diesen Ausdruck in der Abfrage in Raute-Zeichen (#) ein.

Im obigen Code sieht das Ganze dann wie folgt aus (in einer Zeile):

```
Set rst = db.OpenRecordset("SELECT * FROM tblAbwesenheiten WHERE Startdatum BETWEEN
#" & Format(strStart, "yyyy-mm-dd") & "# AND #" & Format(strEnde, "yyyy-mm-dd") & "#",
dbOpenDynaset)
```

Das Formatieren des Datums und das Einfassen in Rauten lässt sich auch per Funktion erledigen:

```
Public Function SQLDatum(strDatum As String) As String
    SQLDatum = Format(strDatum, "\#yyyy-mm-dd\#")
End Function
```

Listing 2.5: Funktion zum Standardisieren von Datumsangaben

Die Zeile aus obiger Routine sähe dann so aus (in einer Zeile):

```
Set rst = db.OpenRecordset("SELECT * FROM tblAbwesenheiten WHERE Startdatum BETWEEN " &
SQLDatum(strStart) & " AND " & SQLDatum(strEnde),
dbOpenDynaset)
```

2.3.10 Verweis auf Steuerelemente

Access-SQL erlaubt direkte Verweise auf Formulare und Steuerelemente. Das ist hilfreich, wenn Sie etwa den Feldinhalt eines Formulars als Abfragekriterium verwenden möchten.

Eine solche Abfrage sieht beispielsweise wie in Abbildung 2.14 aus.

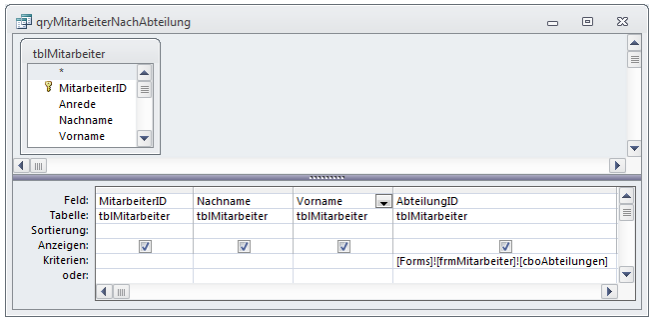


Abbildung 2.14: Abfrage mit einem Verweis auf ein Formularsteuerelement

Ein Blick in die entsprechende SQL-Anweisung zeigt, dass der Ausdruck tatsächlich im gleichen Format wie in VBA in die Abfrage integriert wurde:

```
SELECT MitarbeiterID, Nachname, Vorname, AbteilungID
FROM tblMitarbeiter
WHERE AbteilungID=[Forms]![frmMitarbeiter]![cboAbteilungen];
```

Dies funktioniert sogar, wenn Sie mit anderen Backends als etwa MySQL arbeiten. Beim Zusammensetzen von SQL-Abfragen via VBA bietet es sich allerdings an, die entsprechenden Ausdrücke direkt auszulesen und als festen Parameterwert in die Abfrage zu integrieren. Die Verwendung einer fest in eine Abfrage eingebauten Referenz bringt den Nachteil mit sich, dass Sie diese Abfrage nicht einsetzen können, wenn Sie einen anderen als diesen Parameter verwenden möchten. Die Variante, eine Abfrage ohne Formularreferenzen zu erstellen und derartige Kriterien erst per VBA oder direkt in einer *Datensatzquelle*- oder *Datensatzherkunft*-Eigenschaft zuzuweisen, ist flexibler.

2.4 Aktualisierbarkeit von Abfragen

Je nachdem, wie die in einer Abfrage enthaltenen Tabellen beschaffen sind und wie Sie diese zusammensetzen, sind Abfragen nicht aktualisierbar, das heißt, dass Sie über die Datenblattansicht und damit auch über die Anzeige in Formularen keine Datensätze bearbeiten oder hinzufügen können.

Es ist sicher jedem Access-Entwickler schon einmal passiert, dass er per VBA einen Datensatz einer Abfrage ändern oder hinzufügen wollte und eine entsprechende Fehlermeldung erschien, für die es scheinbar keine Erklärung gab.

2.4.1 Wie erkennen Sie, ob das Abfrageergebnis aktualisierbar ist?

Wenn Sie eine Abfrage in der Datenblattansicht öffnen, erkennen Sie recht schnell, ob diese aktualisierbar ist – die Schaltfläche zum Springen auf einen neuen Datensatz ist ausgeblendet und unter dem letzten Datensatz befindet sich keine leere Zeile zum Anlegen eines neuen Datensatzes (siehe Abbildung 2.16).

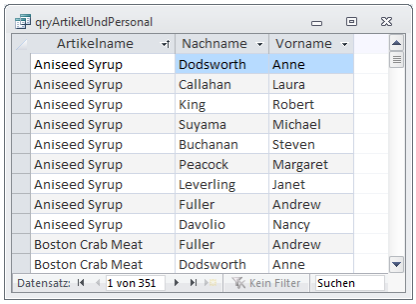


Abbildung 2.15: Eine nicht aktualisierbare Abfrage erkennt man an der deaktivierten Schaltfläche »Neuer Datensatz« und an der fehlenden Zeile mit einem leeren neuen Datensatz

2.4.2 Nicht aktualisierbare Abfragen

Nachfolgend finden Sie Beispiele für Abfragen, die niemals aktualisierbar sind:

» Abfragen mit zwei nicht verknüpften Tabellen. Beispiele sind alle Abfragen, die alle Kombinationen aus den Datensätzen zweier Tabellen anzeigen sollen (siehe Abbildung 2.16).

» Abfragen mit drei oder mehr Tabellen, deren innere Tabelle die 1-Seite für die beiden äußeren Tabellen stellt. Beispiel: Sie möchten zu einem Mitarbeiter gleichzeitig die Abwesenheiten und den Urlaubsanspruch ausgeben (siehe Abbildung 2.17).

» Abfragen mit Gruppierungen und Aggregatfunktionen

» Abfragen, bei denen die Eigenschaft *Keine Duplikate* auf *Ja* eingestellt ist

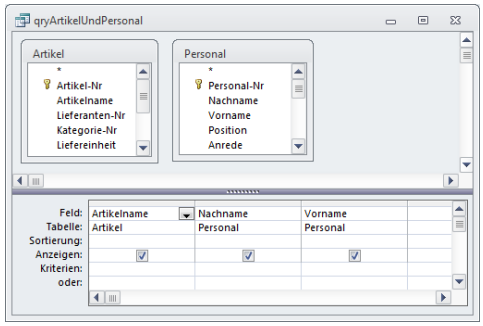


Abbildung 2.16: Diese Abfrage liefert kein aktualisierbares Ergebnis

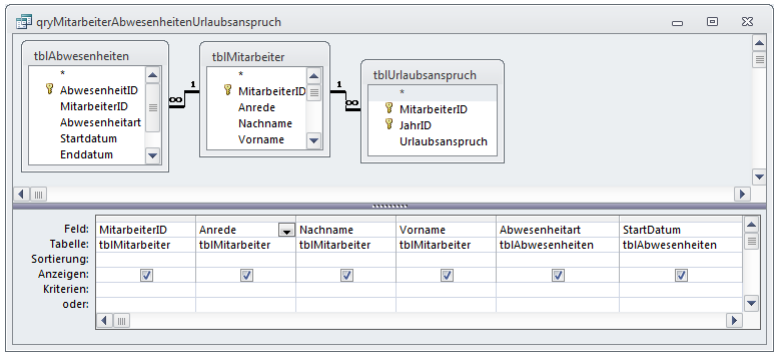


Abbildung 2.17: Auch diese Abfrage ist nicht aktualisierbar

Weitere Möglichkeiten sind folgende:

» Abfragen, deren Eigenschaft *Recordsettyp* den Wert *Snapshot* besitzt

3

Formulare

Formulare sind das wichtigste Element der Benutzeroberfläche von Access-Anwendungen. Sie sorgen dafür, dass der Benutzer bei der Dateneingabe nicht mit ellenlangen Tabellen oder Abfragen mit Tausenden von Datensätzen hantieren muss, sondern die Daten übersichtlich dargestellt bekommt – je nach Anforderung als Liste oder Detailansicht.

Mit Kombinationsfeldern, Listenfeldern und Unterformularen lassen sich auch die Daten verknüpfter Tabellen problemlos darstellen.

Der große Vorteil von Access-Formularen gegenüber denen von Visual Basic oder den .NET-Programmiersprachen liegt in den speziellen Funktionen für die Verwendung mit Tabellen und Abfragen als Datensatzquelle. Diese Techniken sind mit dafür verantwort-

lich, dass Access heute das am weitesten verbreitete Datenbank-Managementsystem für Windows-Rechner ist. Abgerundet wird dies durch die Möglichkeit, Formulare und die enthaltenen Steuerelemente zu programmieren und dabei die zahlreichen Ereignisse zu nutzen.

In diesem Kapitel erfahren Sie zunächst, welche Neuerungen Access 2010 in Zusammenhang mit Formularen mitbringt, und erhalten eine kleine Einführung in Formulare. Dann geht es ans Eingemachte: Sie lernen, wie Sie Daten auf Basis einzelner oder verknüpfter Tabellen in Abfragen darstellen können und wie Sie VBA zur Realisierung wichtiger Funktionen wie etwa zur Suche von Datensätzen verwenden.

Das folgende Kapitel geht dann genauer auf die verschiedenen Steuerelemente ein, die Sie in Formularen einsetzen können. Dort geht es nicht nur um die Standardsteuerelemente von Access, sondern auch um Steuerelemente wie das *ListView*-, *TreeView*- oder *ImageList*-Steuerelement.

Da alle in diesem Kapitel beschriebenen Funktionen mehr oder weniger den Einsatz von VBA und Ereigniseigenschaften von Formularen und Steuerelementen benötigen, finden Sie zunächst eine Beschreibung der wichtigsten Ereignisse und ihre Abfolge für das Formular selbst und einige Steuerelemente.

BEISPIELDATENBANK

Die Beispieldatei zu diesem Kapitel finden Sie unter dem Namen *Formulare.accdb* auf *www.acciu.de/aeb2010*.

3.1 Formulare in Access 2010

Microsoft hat sich eine Menge Features einfallen lassen, die das Erstellen von Formularen für Einsteiger vereinfachen. Es gibt aber auch Neuerungen, die dem alteingesessenen Access-Entwickler das Leben erleichtern. Auch hier haben wir die Neuerungen seit Access 2007 mit berücksichtigt, damit Umsteiger von Access 2003 und älter sich direkt zurechtfinden. Beim Gang durch das Entwerfen von Formularen lernen Sie alles Neue kennen.

3.1.1 Anlegen eines Formulars

Das schnelle Anlegen von Formularen erfolgt in Access 2010 mit dem passenden Ribbon-Befehl auf Basis des aktuell markierten Tabellen- oder Abfrage-Objekts. Die Ribbon-Einträge befinden sich sämtlich im Bereich *Erstellen|Formulare*. Die einfachste Variante mit der Bezeichnung *Formular* erstellt automatisch ein einfaches Formular

in der Ansicht *Einzelnes Formular*. Dabei berücksichtigt diese Funktion eventuelle Unterdatenblätter der zugrunde liegenden Tabelle/Abfrage und legt dafür passende Unterformulare auf Basis der verknüpften Tabellen an.

Mit dem Eintrag *Erstellen|Formulare|Geteiltes Formular* legen Sie ein geteiltes Formular an, das gleichzeitig eine Liste der Datensätze der zugrunde liegenden Tabelle oder Abfrage und die Detailansicht des aktuell ausgewählten Datensatzes anzeigt. Mehr zu geteilten Formularen erfahren Sie weiter unten in »Geteilte Formulare« ab Seite 170.

Weitere Ribbon-Schaltflächen verheißen das schnelle Anlegen der gängigen Ansichten. Wer's manuell mag, der klickt direkt auf den Ribbon-Eintrag *Erstellen|Formulare|Formularentwurf* und legt selbst Hand an.

Gebundene und ungebundene Formulare

Üblicherweise verwendet man in Access Formulare, um Daten aus den zugrunde liegenden Tabellen anzuzeigen, und das bedeutet, dass man das Formular an eine Tabelle oder Abfrage bindet. Dazu stellt man die Eigenschaft *Datensatzquelle* entweder auf den Namen einer Tabelle, den Namen einer Abfrage oder einen passenden SQL-Ausdruck ein.

Natürlich kann es auch ungebundene Formulare geben – etwa, um den Benutzer mit einem Startdialog einige nützliche Hinweise mit auf den Weg zu geben, dem Benutzer statt eines Ribbons ein Übersichtsformular zum Steuern der Anwendung zu bieten oder Informationen abzufragen – beispielsweise Such- oder Filterkriterien für die Anzeige von Daten in Formularen oder Berichten. Solche Formulare unterscheiden sich lediglich durch das Leerlassen der Eigenschaft *Datensatzquelle* von gebundenen Formularen.

Gebundene Felder über die Feldliste anlegen

Wenn Sie sich entschieden haben, ein Formular von Grund auf selbst anzulegen (also ohne eine der Funktionen zum automatischen Einfügen etwa von Feldern oder der Datensatzquelle), gehen Sie folgendermaßen vor: Nach dem Auswählen der Datensatzquelle können Sie die darin enthaltenen Felder auf dem Formular platzieren. Wie Sie diese anordnen, hängt im Wesentlichen davon ab, wie Sie die Daten darstellen möchten und welchen Zweck das Formular hat. Grundsätzlich geschieht das aber immer gleich:

» Aktivieren Sie die Entwurfs- oder die Layoutansicht des Formulars.

» Klicken Sie auf den Ribbon-Eintrag *Entwurf|Tools|Vorhandene Felder hinzufügen*, um die Feldliste anzuzeigen.

» Ziehen Sie die benötigten Felder aus der Feldliste (auch per Mehrfachauswahl) auf den gewünschten Bereich des Formulars. Welche Bereiche es gibt und wozu diese dienen, erfahren Sie weiter unten.

Die Feldliste zur Auswahl der anzuzeigenden Felder hat sich im Vergleich zu älteren Access-Versionen stark verändert. Zwar zeigt sie noch die Felder der ausgewählten Datensatzquelle an, aber anders als gewohnt (siehe Abbildung 3.1): Die eigentliche Feldliste im oberen Bereich enthält nicht nur die Felder der Datensatzquelle, sondern auch noch die Angabe der passenden Tabelle. Im mittleren Bereich zeigt Access direkt mit der Datensatzquelle verknüpfte Tabellen und ihre Felder an (falls Sie im Beziehungen-Fenster entsprechende Beziehungen angelegt oder den Nachschlageassistenten damit betraut haben) und im unteren Bereich die übrigen Tabellen. Falls die Feldliste nur die Felder der aktuellen Datensatzquelle anzeigt, können Sie die Ansicht mit einem Klick auf die Schaltfläche *Alle Tabellen anzeigen* erweitern.

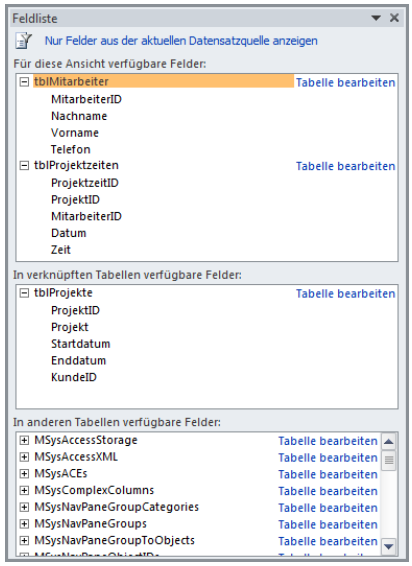


Abbildung 3.1: Auswahl der im Formular anzuzeigenden Felder

Bei Feldern aus verknüpften Tabellen fügt Access diese Tabellen einfach der Datensatzquelle hinzu, indem es den passenden SQL-Ausdruck in der Eigenschaft *Datensatzquelle* anpasst, bei nicht verknüpften Tabellen ist noch ein wenig Handarbeit notwendig: Sie müssen dann in einem Dialog angeben, über welches Feld ein Bezug hergestellt werden soll (siehe Abbildung 3.2).

Prinzipiell stellen Sie mit diesem Dialog nachträglich eine Verknüpfung her (wenn auch nur auf Datensatzquellen-Ebene), die in der zugrunde liegenden Abfrage gespeichert wird. Das ist ein interessantes Werkzeug, wenn man mal eben ein Feld nachreichen möchte, das man beim Anlegen der Datensatzquelle vergessen hat. Die Frage ist, ob dies bei Tabellen, die noch nicht verknüpft sind, sinnvoll ist – und gerade dazu ist der Dialog wohl vorgesehen.

Das Hinzufügen von Feldern aus bereits verknüpften Tabellen hingegen kann eine echte Erleichterung sein: Wie oft passiert es, dass man Felder etwa aus Lookup-Tabellen wie einer Anrede-Tabelle beim Zusammenstellen der Datensatzquelle außer Acht lässt und diese sonst nachträglich anpassen müsste. Das geht nun schon wesentlich schneller, indem man einfach das passende Feld aus dem mittleren Bereich der Feldliste in den Entwurf zieht.

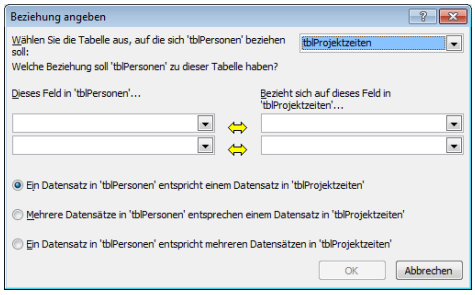


Abbildung 3.2: Nachträgliches Hinzufügen von Feldern aus nicht verknüpften Tabellen

Felder über Steuerelemente anlegen

Der zweite Weg, ein gebundenes Feld anzulegen, ist ein wenig aufwändiger. Er ist sinnvoll, wenn Sie kein Textfeld für die Anzeige des Feldinhaltes benötigen, sondern auf ein anderes Steuerelement zurückgreifen möchten. Bei Kombinations- und Listenfeldern kann man dem vorbeugen, indem man die Nachschlage-Eigenschaften des zugrunde liegenden Feldes entsprechend vorbereitet (mehr dazu unter »Beziehungen herstellen« ab Seite 53 und folgende).

Auch für Boolean-Felder legt Access automatisch ein Kontrollkästchen an. Vielleicht möchten Sie aber auch einmal eine Optionsgruppe anlegen, um die Werte eines Zahlenfeldes abzubilden, oder Sie benötigen ein Kombinations- oder Listenfeld, ohne dass Sie die Nachschlage-Eigenschaften des Tabellenfelds entsprechend eingestellt haben. Dann klicken Sie einfach auf die passende Schaltfläche im Ribbon-Bereich *Entwurf|Steuerelemente* und ziehen dann das Feld, dessen Inhalt das Steuerelement anzeigen soll, in den Formularentwurf. Die Eigenschaft *Steuerelementinhalt* wird dann gleich auf das entsprechende Feld eingestellt.

Formularbereiche

Der meistverwendete Formularbereich ist vermutlich der Detailbereich. Er enthält in der Regel die an die Datensatzquelle gebundenen Steuerelemente – eben jene Felder, die Sie aus der Feldliste auf das Formular ziehen können. Die hier enthaltenen Felder zeigt Access in allen Ansichten an – in manchen sogar nur diese Felder und keine Steuerelemente der anderen Bereiche.

Formulare können außerdem einen Formularkopf und einen Formularfuß aufweisen. Der Formularkopf ist beispielsweise gut für die Anzeige von Überschriften, berechneten Werten wie dem Systemdatum oder Steuerelementen wie Schaltflächen geeignet.

Der Formularfuß enthält oft Ergebnisse von Berechnungen wie etwa der Anzahl der im Detailbereich angezeigten Datensätze oder einer Rechnungssumme. Dort können Sie auch – wie in vielen Windows-Anwendungen – die Schaltflächen zum Steuern des Formulars unterbringen. Viele Benutzer sind es gewohnt, dort Schaltflächen mit Beschriftungen wie *OK* oder *Abbrechen* zu finden.

Die Bereiche *Seitenkopf* und *Seitenfuß* sind nur interessant, wenn Sie Formulare ausdrucken wollen. Für das Ausdrucken von Daten sind unter Access aber Berichte verantwortlich; sie bieten auch wesentlich bessere Möglichkeiten zur optimalen Darstellung der Daten. Daher behandelt dieses Buch diese Bereiche an dieser Stelle nicht weiter.

Das Ein- und Ausblenden der verschiedenen Formularbereiche erledigen Sie mit den Kontextmenüeinträgen *Seitenkopf/-fuß* und *Formularkopf/-fuß*.

3.1.2 Formularansichten

Bei einem Formular, das der Eingabe von Daten dient, werden Sie vermutlich nur jeweils einen einzigen Datensatz anzeigen. Sie stellen dann die Eigenschaft *Standardansicht* auf *Einzelnes Formular* ein.

Das ist nicht nur die Basis zur Eingabe, sondern auch zur Anzeige von Daten im Detail sowie zum Bearbeiten von Datensätzen. Generell können Sie die Ansicht eines Formulars im Ribbon unter *Start|Ansichten* oder über das Kontextmenü der Titelleiste eines Formulars oder des Formularhintergrunds einstellen.

Entwurfsansicht

In der Entwurfsansicht eines Formulars passen Sie das Formular an: Sie fügen Steuerelemente hinzu, stellen Eigenschaften wie Höhe, Breite oder Farben ein und ordnen die Elemente wie gewünscht an. Es würde den Rahmen dieses Kapitels sprengen, wenn es alle Einzelheiten in Zusammenhang mit der Entwurfsansicht beschreiben sollte. Deshalb beschränkt es sich auf die Neuheiten seit Access 2007 (für alle, die von Access 2003 oder älter zu Access 2010 wechseln), die sich vor allem auf die verbesserte Ausrichtung von Steuerelementen beziehen (siehe »Hilfreiche Funktionen für den Formularentwurf« ab Seite 171).

Wenn Ihnen die Grundlagen beim Erstellen von Formularen mit der Entwurfsansicht nicht geläufig sind, liefert die Onlinehilfe von Access allerdings gutes Material: Geben Sie dort als Suchbegriff einfach »Formulare« ein und Sie erhalten eine umfangreiche Liste mit allen möglichen Themen rund um die Erstellung von Formularen.

Layoutansicht

Die Layoutansicht ist ein neues Feature, das mit Access 2007 eingeführt wurde. Es stellt eine Art Zwischenstufe zwischen der Entwurfsansicht und der Formularansicht dar. In dieser Ansicht zeigt ein Formular die zugrunde liegenden Daten an und lässt sich gleichzeitig wie in der Entwurfsansicht bearbeiten. Der wichtigste Vorteil der Layoutansicht ist, dass Sie Größe, Position und Formatierung der Steuerelemente auf die enthaltenen Daten abstimmen können. Bisher war dazu ein stetiges Wechseln zwischen Entwurfs- und Formularansicht notwendig.

Der Wert der Eigenschaft *Layoutansicht zulassen* (VBA: *LayoutView*) entscheidet, ob Sie ein Formular grundsätzlich in der Layoutansicht anzeigen können. Wenn diese Voraussetzung geschaffen ist, können Sie sie problemlos einsetzen, um wie in Abbildung 3.3 die Breite von Steuerelementen in der Endlosansicht eines Formulars den vorhandenen Inhalten anzupassen.

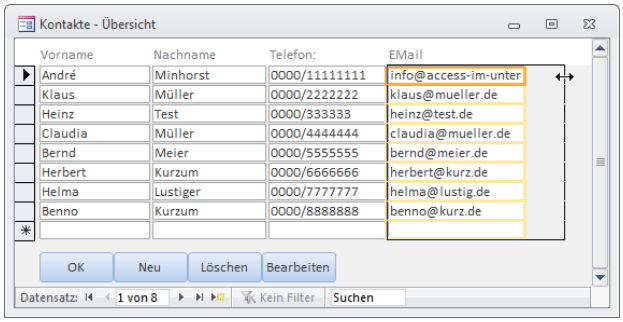


Abbildung 3.3: Anpassen der Breite von Steuerelementen in der Layoutansicht

Im Gegensatz zu Access 2007 können Sie nun einige Entwurfsaufgaben mehr in der Layoutansicht erledigen – so lassen sich etwa Steuerelemente über die Steuerelement-Schaltflächen im Ribbon-Bereich *Entwurf|Steuerelemente* hinzufügen. Sie können nun auch Steuerelemente duplizieren oder mit den Cursor-Tasten in Kombination mit der *Umschalt*- und der *Strg*-Taste fein positionieren beziehungsweise ihre Größe anpassen. Dies ist nun auch nötig, denn in Webdatenbanken steht nur noch die Layoutansicht für den Entwurf von Formularen und Berichten zur Verfügung (mehr dazu unter »Webdatenbanken« ab Seite 637).

Datenblattansicht

Die Datenblattansicht zeigt die im Detailbereich befindlichen gebundenen Felder genau wie in der von Tabellen und Abfragen bekannten Datenblattansicht an. Steuerelemente im Kopf- und Fußbereich stellt die Datenblattansicht nicht dar. Aus dem Detailbereich fallen auch einige Steuerelementtypen weg. Angezeigt werden nur Text-

felder, Kombinationsfelder, Listenfelder (als Kombinationsfeld), Optionsfelder und Kontrollkästchen, die nicht in Optionsgruppen organisiert sind, Optionsgruppen (allerdings nur als Textfeld, nicht mit Optionsfeldern), Umschaltflächen (als Wert) und einige andere Steuerelemente. Bezeichnungsfelder, die an eines der genannten Steuerelemente gebunden sind, zeigt die Datenblattansicht als Spaltenkopf an.

In der Praxis werden Sie die Datenblattansicht wohl nie für eigenständige Formulare einsetzen, sondern nur als Unterformular – auch, wenn Sie eigentlich nur Daten in der Datenblattansicht anzeigen möchten. Der Grund ist ganz einfach: Die Datenblattansicht ist sehr egoistisch und lässt keinen Platz für andere Steuerelemente. Wenn Sie eine solche Ansicht etwa mit einem kleinen Suchfeld oder mit Schaltflächen zum Schließen oder Abbrechen versehen möchten, müssen Sie eine Art »Rahmen-Formular« verwenden und das Formular in der Datenblattansicht dort als Unterformular einbetten. Wie das aussieht, sehen Sie in Abbildung 3.4. Weitere Informationen zu dieser Ansicht finden Sie unter »Einfache Daten in der Übersicht als Datenblatt« ab Seite 203.

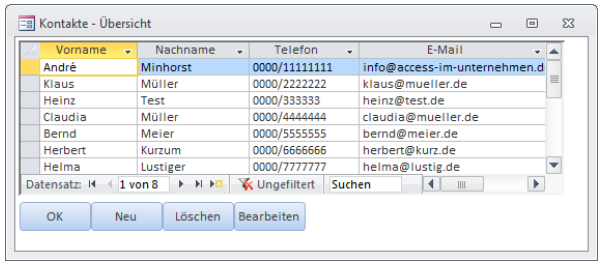


Abbildung 3.4: Ein Formular in der Datenblattansicht mit Formular-Rahmen zum Hinzufügen von Schaltflächen

Die Datenblattansicht kommt auch oft zum Einsatz, wenn es um die Darstellung von 1:n- oder m:n-Beziehungen geht. Dabei zeigt dann ein Hauptformular die Daten der Detailtabelle und das Unterformular die Daten der Mastertabelle in der Datenblattansicht an. Ein Beispiel sehen Sie in Abbildung 3.5, weitere Informationen zu dieser Darstellung finden Sie in »1:n-Beziehung per Unterformular und Datenblattansicht« ab Seite 212.

Einfaches Formular

Diese Ansicht ist für die Anzeige der Details eines Datensatzes geeignet. Sie zeigt jeweils einen einzigen Datensatz an. Im Gegensatz zur Datenblattansicht können Sie die Steuerelemente ganz nach Ihren Wünschen auf die Bereiche des Formulars aufteilen und – noch wichtiger – Sie können alle Steuerelemente einsetzen, die Access hergibt. Besonders interessant sind dabei Listenfelder und Unterformulare, mit denen Sie die strukturierte Anzeige von Daten aus mehreren verknüpften Tabellen anzeigen können.

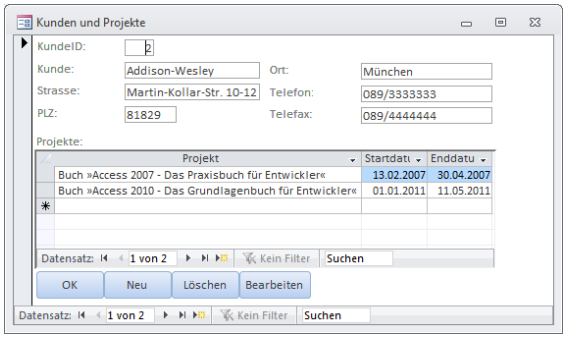


Abbildung 3.5: Für Unterformulare bietet sich oft die Datenblattansicht an

Endlosformular

Endlosformulare sind prinzipiell einfache Formulare, von denen Sie aber mehrere gleichzeitig (und zwar untereinander) anzeigen können. Damit können Sie einige Nachteile der Datenblattansicht ausmerzen: So lassen sich auch hier beliebige Steuerelemente anzeigen. Eine Ausnahme sind Unterformulare: Diese können Sie nicht in der Endlosansicht eines Formulars einsetzen. Um mit einem Endlosformular ein Formular in der Datenblattansicht nachzubauen, brauchen Sie nur die Steuerelemente genauso wie in der Datenblattansicht anzuordnen, passende Beschriftungsfelder im Formularkopf zu platzieren und die Höhe des Detailbereichs entsprechend zu verkleinern. Das kann etwa so wie in Abbildung 3.6 aussehen (weitere Informationen zu dieser Ansicht finden Sie unter »Einfache Daten in der Übersicht mit Endlosformularen« ab Seite 199). Eine solche Endlosansicht können Sie natürlich auch als Unterformular einsetzen, um eine 1:n- oder m:n-Beziehung darzustellen.

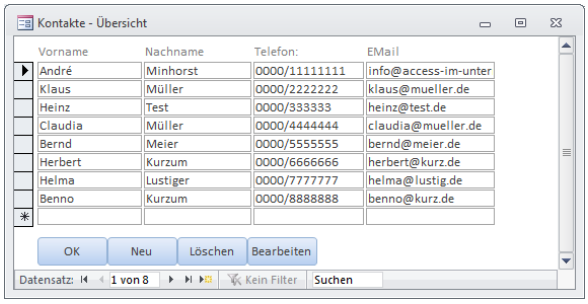


Abbildung 3.6: Ein Endlosformular als Ersatz für die Datenblattansicht

PivotTable- und PivotChart-Ansicht

Diese beiden Ansichten bietet Access seit der Version XP an. Sie bieten eine recht gute Entwurfsansicht, mit der sich schnell ansprechende Ergebnisse erzielen lassen. Aller-

dings sind diese beiden Ansichten in Zusammenhang mit größeren Datenmengen recht langsam. Da das Interesse seitens der Access-Entwickler an diesen Ansichten relativ gering ist, geht dieses Buch nicht näher darauf ein.

3.1.3 Geteilte Formulare

Seit Access 2007 gibt es einige Eigenschaften, mit denen Sie Formulare zur Anzeige von Daten in Listen- und Detailansicht aufteilen können. Sicher haben Sie schon einmal ein Unterformular oder ein Listenfeld in einem Formular verwendet, um eine Übersicht über Datensätze – etwa von Mitarbeiterdaten – anzuzeigen und darüber die Auswahl eines Datensatzes zur Anzeige der Detaildaten in einem anderen Bereich des Formulars einzublenden.

Hierzu ist VBA-Programmierung notwendig, und Microsoft hat mit den sogenannten »Split Forms« eine Möglichkeit geschaffen, dies durch einfaches Anpassen einiger Eigenschaften zu erreichen. (Natürlich können Sie dies auch per Assistenten durchführen, aber diese kleinen Helfershelfer will dieses Buch ja beiseite lassen ...)

Grundvoraussetzung für das Teilen eines Formulars ist das Einstellen der Eigenschaft *Standardansicht* auf den neuen Wert *Geteiltes Formular* (siehe Abbildung 3.7).

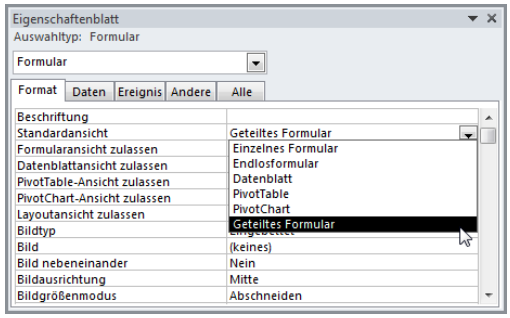


Abbildung 3.7: Mit der richtigen Einstellung für die Eigenschaft Standardansicht erreichen Sie die Anzeige eines geteilten Formulars

Haben Sie diese Eigenschaft eingestellt, brauchen Sie eigentlich nur noch eine passende Datensatzquelle auszuwählen und die Felder in der gewünschten Reihenfolge anzuordnen. Das kann dann, in einem sehr einfachen Fall, wie in Abbildung 3.8 aussehen.

Ein Wechseln in die Formularansicht sorgt schließlich für die Anzeige des geteilten Formulars (siehe Abbildung 3.9). Mit einem Klick auf einen der Datensätze in der Hälfte mit der Datenblattansicht zeigen Sie den jeweils angeklickten Datensatz im Detailbereich des Formulars an.

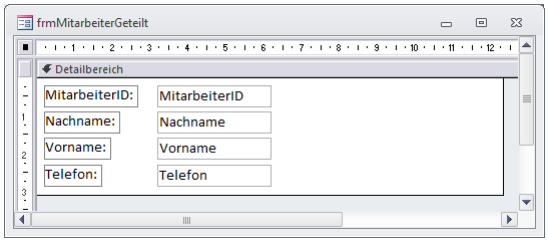


Abbildung 3.8: Dieser Entwurf soll den Detailbereich eines geteilten Formulars ausmachen

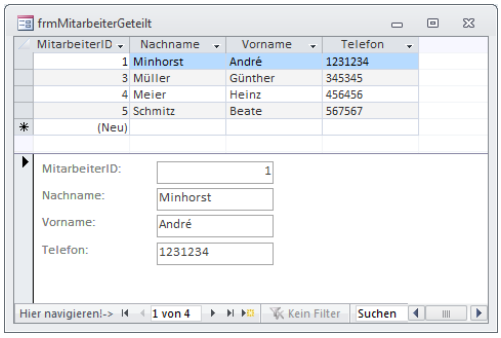


Abbildung 3.9: Die geteilte Ansicht eines Formulars mit Datenblatt und Detailansicht

Natürlich können Sie für geteilte Formulare einige Einstellungen vornehmen. Welche dies sind, zeigt Tabelle 3.4.

Weitere Möglichkeiten bietet das Kontextmenü des Datenblatt-Bereichs des geteilten Formulars: Hier haben Sie alle Möglichkeiten, die Sie auch sonst in der Datenblattansicht haben.

Die Formatierung des Datenblattbereichs können Sie mit den Steuerelementen im Ribbon unter *Start|Schriftart* anpassen – vorher müssen Sie allerdings den Fokus auf das Datenblatt setzen.

Die passenden Eigenschaften unter VBA heißen *DatasheetAlternateBackColor*, *DatasheetBackColor*, *DatasheetBorderLineStyle*, *DatasheetCellsEffect*, *DatasheetColumnHeaderUnderlineStyle*, *DatasheetFontHeight*, *DatasheetFontItalic*, *DatasheetFontName*, *DatasheetFontUnderline*, *DatasheetFontWeight*, *DatasheetForeColor*, *DatasheetGridlinesBehaviour* und *DatasheetGridlinesColor* – für weitere Informationen siehe Onlinehilfe.

3.1.4 Hilfreiche Funktionen für den Formularentwurf

Seit Access 2007 gibt es einige neue Funktionen zum Entwerfen von Formularen, die mit Access 2010 noch erweitert wurden.

Eigenschaft (deutsch)	Eigenschaft (VBA)	Beschreibung
Größe des geteilten Formulars	SplitFormSize	Legt die Höhe des Detailbereichs des Formulars fest. Einheit in VBA: Twips (567 Twips entsprechen 1 cm).
Orientierung des geteilten Formulars	SplitFormOrientation	Gibt die Position des Datenblatts an. Mögliche Werte: 0 (acDatasheetOnTop, oben, Standard) 1 (acDatasheetOnBottom, unten) 2 (acDatasheetOnLeft, links) 3 (acDatasheetOnRight, rechts)
Teilerleiste des geteilten Formulars	SplitFormSplitterBar	Gibt an, ob der Benutzer die Größe der Bereiche mit einer Teilerleiste einstellen kann. Boolean.
Datenblatt des geteilten Formulars	SplitFormDatasheet	Gibt an, ob der Benutzer Daten im Datenblatt bearbeiten darf. Mögliche Werte: 0 (acDatasheetAllowEdits, editieren erlaubt) 1 (acDatasheetReadOnly, schreibgeschützt)
Drucken des geteilten Formulars	SplitFormPrinting	Gibt an, welcher Bereich des Formulars gedruckt wird. Mögliche Werte: 0 (acFormOnly, Detailansicht) 1 (acGridOnly, Datenblatt)
Position der Teilerleiste speichern	SplitFormSplitterBarSave	Gibt an, ob die Position der Teilerleiste beim Schließen gespeichert wird. Boolean.

Tabelle 3.4: Eigenschaften zum Anpassen der geteilten Formularansicht

Automatische Layouts

Wenn Sie Steuerelemente und ihre Beschriftungsfelder in Tabellenform anordnen möchten, können Sie einfach alle betroffenen Elemente markieren und dann einen der Ribbon-Einträge *Anordnen|Tabelle|Tabelle* oder *Anordnen|Tabelle|Gestapelt* auswählen. Diese Funktionen stehen in der Entwurfs- und der Layoutansicht zur Verfügung. Access ordnet die Steuerelemente dann wie in Abbildung 3.10 (Ausrichtung: *Tabelle*) oder wie in Abbildung 3.11 (Ausrichtung: *Gestapelt*) an.

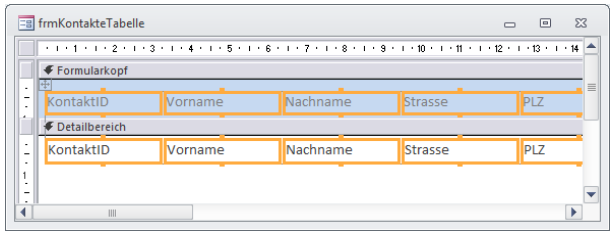


Abbildung 3.10: Felder mit tabellarischer Ausrichtung

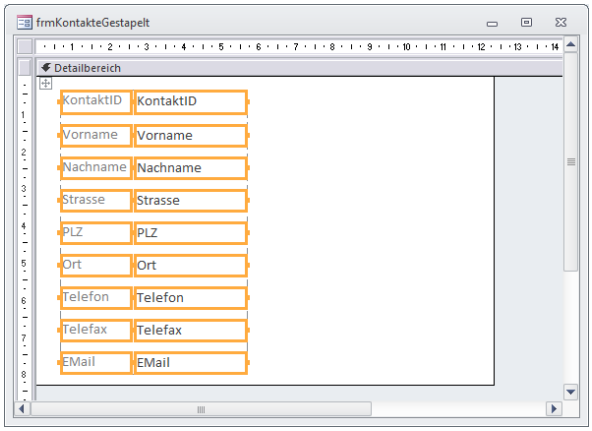


Abbildung 3.11: Felder in der »gestapelten« Ausrichtung

Es handelt sich hierbei aber nicht nur um das automatische Anordnen, denn Access fügt gleichzeitig eine Art Rahmen hinzu, der die Positionen der enthaltenen Steuerelemente wahrt. Außerdem bietet dieser Rahmen die Möglichkeit, dass Sie die Reihenfolge der Steuerelemente neu sortieren, ohne sich um die Wahrung der Abstände und der Position kümmern zu müssen. Markieren Sie dazu einfach das zu verschiebende Feld und ziehen Sie es an die gewünschte Stelle. Eine orange Linie zeigt dabei an, wo Access das Steuerelement platziert.

Bei der tabellarischen Ausrichtung übernimmt Access automatisch die Aufgabe, die Feldbezeichnungen als Spaltenköpfe in den Formularkopf zu kopieren. Wenn Sie einzelne Elemente einer Gruppe von gemeinsam ausgerichteten Steuerelementen individuell anpassen möchten, müssen Sie mit Einschränkungen leben: In der tabellarischen Ausrichtung können Sie nur die Breite und in der gestapelten Ausrichtung nur die Höhe der einzelnen Steuerelemente anpassen.

Das ist aber nicht schlimm: Wenn Sie die Steuerelemente mit einer der Ausrichten-Funktionen positioniert haben, können Sie den Layoutrahmen wieder entfernen, indem Sie diesen markieren und den Ribbon-Eintrag *Anordnen|Tabelle|Entfernen* betätigen. Danach können Sie manuell Hand an die Steuerelemente anlegen, um etwa die Breite des PLZ-Felds in Abbildung 3.11 zu verringern.

Den Layoutrahmen markieren Sie im Übrigen mit einem Klick auf das kleine Kreuz, das nach einem Klick auf eines der enthaltenen Steuerelemente erscheint (siehe Abbildung 3.12).

Sie können dem Layout auch nachträglich einzelne Steuerelemente hinzufügen oder Steuerelemente daraus entfernen. Zum Hinzufügen ziehen Sie einfach das passende Steuerelement an die gewünschte Stelle im Layout, zum Entfernen markieren Sie das

betroffene Steuerelement und klicken anschließend auf die Ribbon-Schaltfläche *Anordnen|Tabelle|Layout entfernen*. Nach dem Entfernen verbleibt das entfernte Steuerelement an Ort und Stelle und die daneben oder darunter befindlichen Steuerelemente rücken an dessen Platz. Sie müssen auf einem Layout entfernte Steuerelemente also noch manuell verschieben, damit diese nicht mit den Steuerelementen des Layouts überlappen.

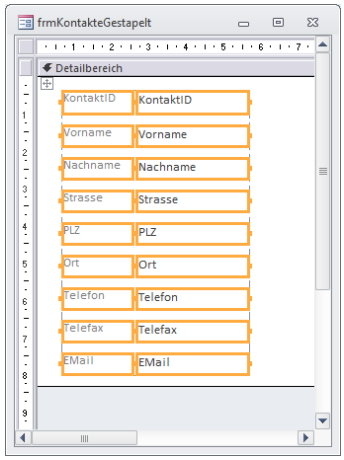


Abbildung 3.12: Mit dem Kreuz links oben markieren Sie das komplette Layout

Die Layoutansicht sollte dem Entwickler einer Datenbank vorbehalten sein. Sie sollten daher entweder die Eigenschaft *Layoutansicht zulassen* für die gewünschten Formulare auf *Nein* einstellen oder die Ansicht für die komplette Datenbank deaktivieren, indem Sie in den Access-Optionen die Eigenschaft *Aktuelle Datenbank|Anwendungsoptionen|Layoutansicht aktivieren* auf *Ja* einstellen. In VBA können Sie dieses Feature mit der Eigenschaft *AllowLayoutView* des Formulars aktivieren und deaktivieren.

Mit Access 2010 gibt es einige Verbesserungen: So können Sie nun das Raster weiter verfeinern. Sie können Zellen horizontal oder vertikal aufteilen, um zusätzliche Steuerelemente einfügen zu können (etwa eine Schaltfläche), und diese wieder verbinden. Jede Zelle darf aber nur ein Steuerelement enthalten.

Wenn Sie Zellen aufteilen, fügt Access in den entstandenen Leerräumen automatisch das neue Steuerelement *EmptyCell* als Platzhalter ein. Dies ist vor allem dem Layout von Webformularen und -berichten geschuldet.

Die notwendigen Befehle finden Sie im Kontextmenü des jeweiligen Elements (siehe Abbildung 3.13). Außerdem finden Sie hier die Möglichkeit, komplette Zeilen und Spalten einzufügen (Kontextmenüeinträge *Einfügen|Links einfügen*, *Einfügen|Rechts einfügen*, *Einfügen|Oben einfügen* und *Einfügen|Unten einfügen*).

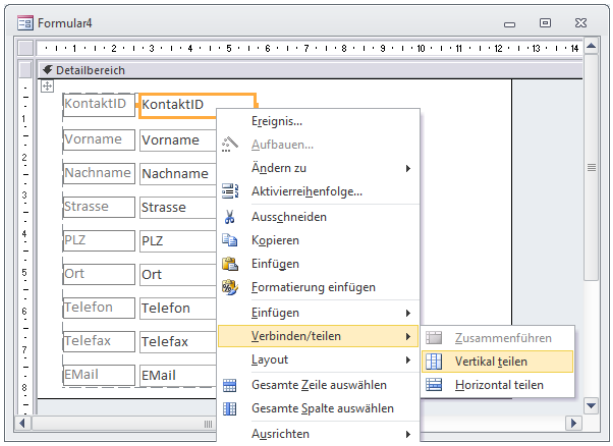


Abbildung 3.13: Aufteilen und verbinden der Zellen eines Rasters

Diese Aktionen können Sie auch mit der *RunCommand*-Methode durchführen, und zwar mit den folgenden Konstanten:

» *acCmdLayoutInsertRowAbove*

» *acCmdLayoutInsertRowBelow*

» *acCmdLayoutInsertColumnLeft*

» *acCmdLayoutInsertColumnRight*

» *acCmdLayoutMergeCells*

» *acCmdLayoutSplitColumnCell*

» *acCmdLayoutSplitRowCell*

Das komplette Layout markieren Sie mit *RunCommand acCmdSelectEntireLayout*.

Berechnungen in der Datenblattansicht

In Tabellen, Abfragen und Formularen steht seit Access 2007 für die Datenblattansicht eine neue Funktion zum Durchführen von Berechnungen und zur Anzeige ihrer Ergebnisse bereit. Da der Benutzer Ihrer Datenbank keinen Kontakt mit der Datenblattansicht der Tabellen und Abfragen haben soll, ist dies ein Fall für das Formulare-Kapitel. Wie weiter oben erwähnt, erscheinen Daten in der Datenblattansicht meist in Form von Unterformularen in der Datenblattansicht – ein Formular in der Datenblattansicht allein macht wenig Sinn, da man beispielsweise keine zusätzlichen Steuerelemente wie Schaltflächen dort unterbringen kann. Ob nun in einem Formular oder in einem Unterformular: Sie können einem Datenblatt eine Zeile mit Berechnungen hinzufü-

gen, indem Sie es in der Datenblattansicht markieren und im Ribbon die Schaltfläche *Start|Datensätze|Summen* anklicken. Schon erscheint die gewünschte Zeile in der Tabelle und Sie müssen nur noch das Feld und die Berechnungsmethode festlegen (siehe Abbildung 3.14). Praktischerweise bleibt die Berechnungszeile am unteren Rand des Datenblatts »kleben« und wird nicht nach unten weggeschoben, wenn das Datenblatt mehr Datensätze enthält, als es gleichzeitig anzeigen kann.

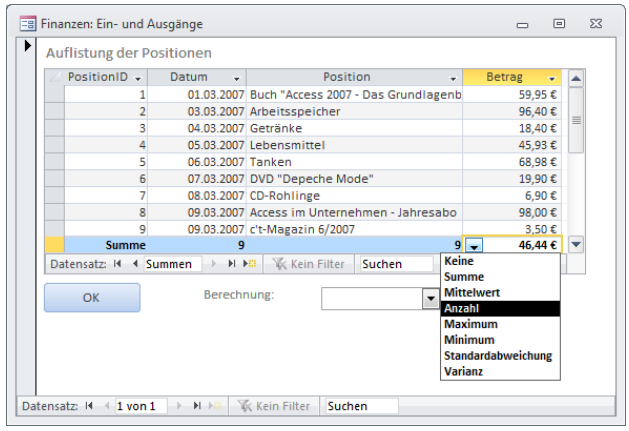


Abbildung 3.14: Auswahl einer Berechnungsart für das Feld *Betrag*

Access stellt je nach dem Datentyp des Feldes unterschiedliche Funktionen zur Verfügung. Bei Zahlen stehen naturgemäß mehr Varianten bereit als für Textfelder – hier gibt es lediglich die Möglichkeit, die Anzahl der Einträge auszugeben. Wie für viele neue Features seit Access 2007 gilt: Probieren Sie es einfach aus, es geht ganz leicht.

3.1.5 Farben und Schriftarten per Design festlegen

Mit den sogenannten Designs können Sie das Aussehen von Formularen und Berichten sowie der enthaltenen Steuerelemente einfach anpassen. Die dafür verantwortlichen Elemente der Benutzeroberfläche finden Sie im Formular- beziehungsweise Berichtsentwurf im Ribbon-Tab *Entwurf* (siehe Abbildung 3.15).

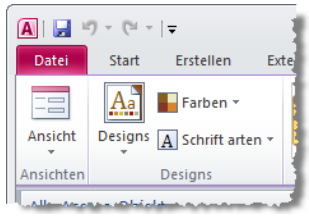


Abbildung 3.15: Aufruf der Funktionen zum Festlegen von Designs

Dies geschieht im Falle der Farben durch folgende Struktur:

» Bei den Farben gibt es eine Zuordnung von Farbcode zu Platzhaltern wie beispielsweise *Text/Hintergrund - dunkel 1* (siehe Abbildung 3.16).

» In den Farbeigenschaften eines Formulars, Berichts oder Steuerelements legen Sie entweder direkt einen Farbcode fest (also ein entsprechender Zahlenwert) oder Sie geben einen der Platzhalter aus Abbildung 3.16 an, aufzurufen über den Ribbon-Eintrag *Designs|Farben|Neue Designfarben erstellen*. Letzteres ist die Voraussetzung für die zentrale Anpassung der Farben über den Dialog aus der Abbildung.

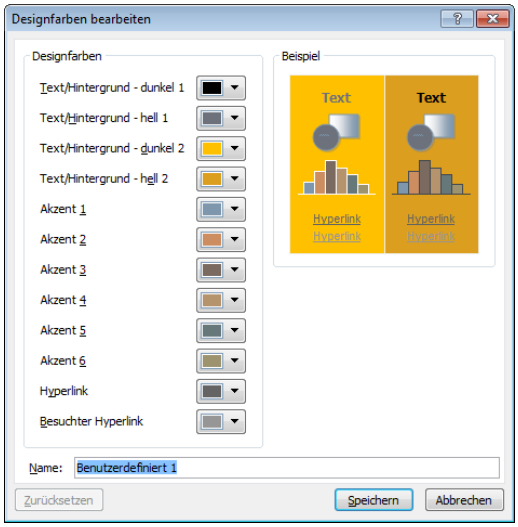


Abbildung 3.16: Zuordnung von Farben zu Platzhaltern

Sie können nun beliebig viele eigene Zusammenstellungen der Designfarben erstellen und speichern. Anschließend klicken Sie mit der rechten Maustaste auf einen der Einträge in der Liste der Farbschemata und legen fest, ob dieser auf alle Formulare beziehungsweise Berichte angewendet werden soll oder nur auf das aktuell geöffnete.

Sie können hier auch eigene Schemata löschen oder bearbeiten (siehe Abbildung 3.17).

Die Platzhalter aus Abbildung 3.16 tauchen im Dialog zum Einstellen der Farbe eines Elements der Benutzeroberfläche in etwas veränderter Form wieder auf (siehe Abbildung 3.18).

Verändert deshalb, weil es beispielsweise statt *Text/Hintergrund - dunkel 1* einen Eintrag *Schwarz, Text 1* gibt. Die einzelnen Schemafarben können Sie noch in helleren oder dunkleren Tönen einsetzen – die entsprechenden Schaltflächen befinden sich unterhalb der Hauptfarben des Schemas.

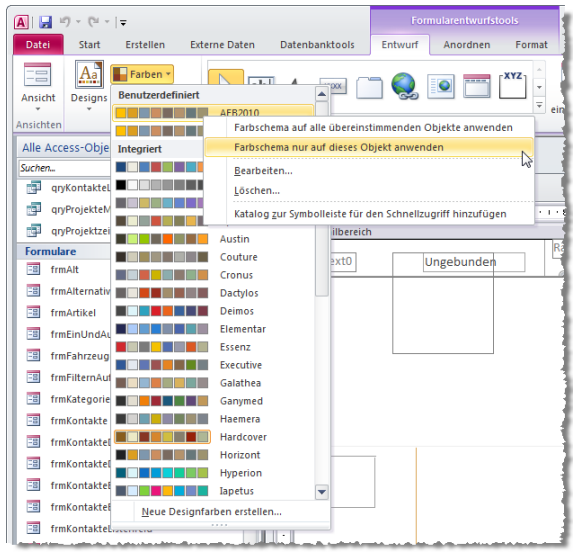


Abbildung 3.17: Festlegen eines Farbschemas

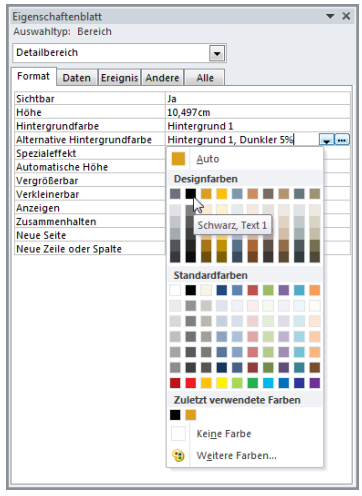


Abbildung 3.18: Auswählen der Platzhalter des Farbschemas für ein Steuerelement

Wenn Sie einem Element eine Designfarbe zuweisen, legen Sie damit drei Eigenschaften fest:

» Index der Designfarbe (*BorderThemeColorIndex*), siehe unten

» Abdunkelung (*BorderShade*): Wert zwischen 1 und 100, der die Abdunkelung des Elements festlegt (100 entspricht der Originalfarbe)

» Aufhellung (*BorderTint*): Wert zwischen 1 und 100, der die Aufhellung des Elements festlegt (100 entspricht der Originalfarbe)

Die folgenden Zahlenwerte entsprechen dem Index für die Designfarben aus Abbildung 3.16:

» *0* (*Text 1*)

» *1* (*Hintergrund 1*)

» *2* (*Text 2*)

» *3* (*Hintergrund 2*)

» *4* (*Akzent 1*)

» *5* (*Akzent 2*)

» *6* (*Akzent 3*)

» *7* (*Akzent 4*)

» *8* (*Akzent 5*)

» *9* (*Akzent 6*)

» *10* (*Hyperlink*)

» *11* (*Besuchter Hyperlink*)

Diese Informationen sind einzeln nur per VBA zugänglich. Im Eigenschaftsfenster wählen Sie nur einen Eintrag des Dialogs wie *Schwarz, Text 1, Heller 25%* aus.

Farben statisch gestalten

Wenn die Farben nicht durch den Wechsel des Designs beeinflusst werden sollen, wählen Sie einfach eine der Standardfarben aus dem Dialog aus Abbildung 3.18 aus oder legen manuell einen entsprechenden Zahlenwert fest.

Design oder nicht Design?

Mit der Eigenschaft *Design verwenden* (VBA: *UseTheme*) können Sie für Navigationsschaltflächen, Navigationssteuerelemente, Schaltflächen, Registerkarten-Steuerelemente und Umschaltflächen festlegen, ob Sie das angegebene Design verwenden wollen oder nicht.

Deshalb enthalten diese Steuerelemente für alle Elemente, deren Farbe eingestellt werden kann, vier Eigenschaften.

Hat *Design verwenden* den Wert *Nein*, wird die in *BackColor* angegebene Farbe verwendet, sonst die in den übrigen drei Eigenschaften festgelegte:

» *...Color*

» *...Shade*

» *...ThemeColorIndex*

» *...Tint*

Die drei Pünktchen werden dabei für die verschiedenen Eigenschaften durch *Back-*, *Border-*, *Fore-*, *Gridline-*, *Hover-*, *HoverFore-*, *Pressed-* und *PressedFore-* ersetzt. Steht *Design verwenden* für ein Steuerelement auf *Nein*, kommen auch die Effekte etwa für den Farbverlauf nicht zum Tragen.

Design-Schriften

Für die Steuerelemente mit Beschriftung wie etwa Bezeichnungsfelder oder Schaltflächen oder mit Texten wie das Textfeld können Sie auch die Schriftart per Design festlegen. Wenn Sie im Ribbon unter *Entwurf|Designs|Schriftarten* einen Eintrag auswählen, wirkt sich dies direkt auf alle Texte im aktuell geöffneten Formular aus. Sie legen damit allerdings nur die Schriftart fest, andere Einstellungen wie Schriftgröße et cetera bleiben unberührt.

Genau genommen können Sie sogar zwei Schriftarten pro Design festlegen. Klicken Sie im Ribbon auf *Entwurf|Designs|Schriftarten* und wählen dann ganz unten den Eintrag *Neue Designschriftart erstellen...* aus, erscheint der Dialog aus Abbildung 3.19.

Hier wählen Sie zwei Schriftarten aus, die Sie später auf die Steuerelemente des Formulars verteilen können. Standardmäßig wird beim Neuanlegen von Steuerelementen oder beim Zuweisen dieses Designs die *Textkörper*-Schriftart verwendet.

Nur wenn Sie aus dem Ribbon den Eintrag *Entwurf|Kopfzeile/Fußzeile|Titel* auswählen, wird ein Bezeichnungsfeld mit der *Überschriften*-Schriftart erstellt.

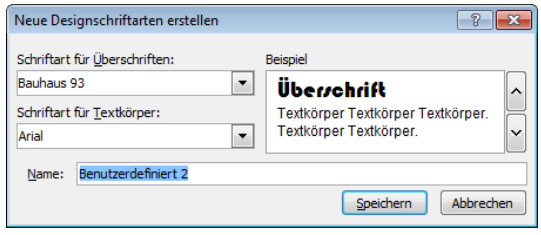


Abbildung 3.19: Neue Design-Schriftarten festlegen

Sie können die Steuerelemente jedoch komfortabel über das Eigenschaftsfenster mit den beiden Design-Schriftarten bestücken (siehe Abbildung 3.20). Beachten Sie, dass nur die mit dem Zusatz *(Kopfbereich)* beziehungsweise *(Detailbereich)* versehenen

Schriftarten die aktuellen Design-Schriftarten sind, die beim Wechseln des Designs durch andere Schriftarten ersetzt werden. Wenn Sie etwa die Schriftart *Arial* ohne Zusatz auswählen, bleibt diese Schriftart auch beim Design-Wechsel erhalten.

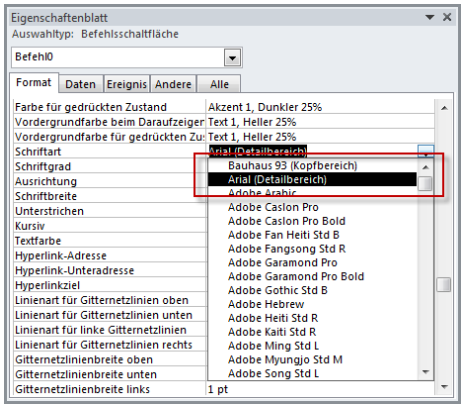


Abbildung 3.20: Einfache Auswahl der beiden Design-Schriftarten

Per VBA können Sie übrigens prüfen, ob ein Element der Benutzeroberfläche eine der beiden Designschriftarten verwendet oder nicht: Die Eigenschaft *ThemeFontIndex* liefert einen der folgenden Werte zurück:

» *0*: Es wird die Designschriftart für den Kopfbereich verwendet.

» *1*: Es wird die Designschriftart für den Detailbereich verwendet.

» *-1*: Es wird keine Designschriftart verwendet.

Farben und Schriften zusammenführen

Wenn Sie mit den aktuell ausgewählten Farben und Schriftarten zufrieden sind, können Sie beide kombiniert speichern. Dies erledigen Sie mit einem Klick auf den Ribbon-Eintrag *Entwurf|Designs|Designs* und durch anschließendes Auswählen des Eintrags *Aktuelles Design speichern*.

Die Informationen werden im Vorlagen-Verzeichnis von Office im Unterordner *Document Themes* als Datei mit der Endung *.thmx* gespeichert (unter Windows 7 beispielsweise im Verzeichnis *C:\Users\Andre Minhorst\AppData\Roaming\Microsoft\Templates\Document Themes*).

Wenn Sie eine mit einem Design ausgestattete Datenbank weitergeben, brauchen Sie die *.thmx*-Datei übrigens nicht mit auf den Zielrechner zu übertragen – alle Eigenschaften sind natürlich in der Datenbank gespeichert. Sie können das Design dann lediglich nicht auf neue Formulare oder Berichte anwenden.

3.1.6 Sonstige Neuerungen

Es gibt noch einige weitere kleine Neuheiten, die eine Erwähnung wert sind.

Alternierender Hintergrund in Datenblättern

Access 2010 bietet die Möglichkeit, die Datenblattansicht von Tabellen, Abfragen und Formularen mit einer alternierenden Hintergrundfarbe zu versehen. Eine Access-weite Option finden Sie in den Access-Optionen unter *Datenblatt|Standardfarben|Alternative Hintergrundfarbe*. Den Effekt können Sie sich in zahlreichen Abbildungen dieses Kapitels ansehen. Wenn Sie eine individuelle Färbung für einzelne Objekte vornehmen möchten, verwenden Sie in der Datenblattansicht den Ribbon-Eintrag *Start|Schriftart|Alternative Füllung/Hintergrundfarbe*. Alternativ können Sie dies auch mit VBA erreichen; die passende Eigenschaft heißt *DatasheetAlternateBackColor*.

Steuerelemente mit Abstand

Mit den folgenden vier Eigenschaften stellen Sie den Abstand von Steuerelementen zur nächsten Gitternetzlinie in einem der Layouts *Tabelle* oder *Geschachtelt* fest. Die Eigenschaften sind für fast alle Steuerelemente verfügbar:

» *Textabstand oben* (*TopPadding*)

» *Textabstand unten* (*BottomPadding*)

» *Textabstand links* (*LeftPadding*)

» *Textabstand rechts* (*RightPadding*)

Steuerelemente verankern

Mit dem Ribbon-Eintrag *Anordnen|Position|Anker* können Sie Steuerelemente mit dem Seitenrand des Formulars verankern. Alternativ stellen Sie die Eigenschaften *Horizontaler Anker* (*HorizonalAnchor*) und *Vertikaler Anker* (*VerticalAnchor*) des betroffenen Steuerelements ein. Die Standardeinstellungen sind *Links* für den horizontalen Anker und *Oben* für den vertikalen Anker. Das bedeutet, dass sich das Steuerelement beim Vergrößern oder Verkleinern des Formulars nicht ändert.

Wie sich die unterschiedlichen Einstellungen auswirken, probieren Sie am besten selbst aus – und zwar in der Layoutansicht, die dafür perfekt geeignet ist. Einen guten Eindruck vermittelt das Beispiel in Abbildung 3.21.

Beschriftung der Navigationsleiste

Links von der Navigationsleiste von Formularen können Sie ab Access 2007 einen Text anzeigen. Diesen legen Sie mit der Eigenschaft *Navigationsbeschriftung* fest (in VBA: *Na-*

vigationCaption). Abbildung 3.22 zeigt ein Beispiel für eine auf diese Weise beschriftete Navigationsleiste. Da die Länge begrenzt und die Anzeige unauffällig ist, scheint der Nutzen dieser Funktion eingeschränkt zu sein.

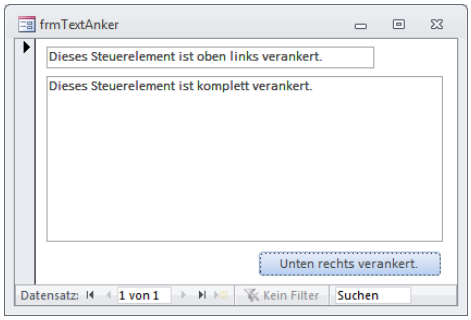


Abbildung 3.21: Verankerte Elemente eines Formulars

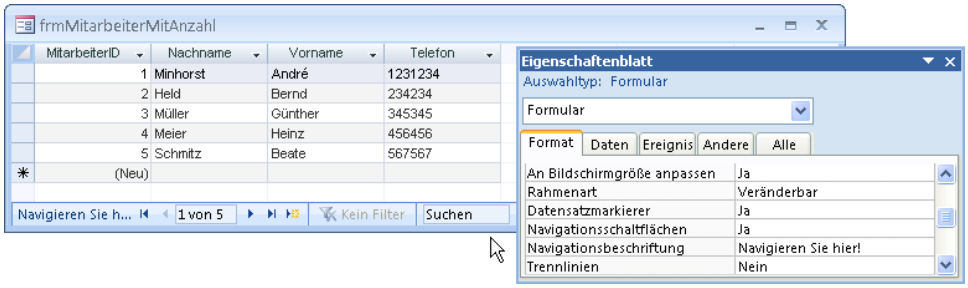


Abbildung 3.22: Die Beschriftung der Navigationsleiste darf nicht sehr lang sein

Wenn Sie den Benutzer mit der Beschriftung unbedingt auf die Navigationselemente aufmerksam machen möchten, stellen Sie für die Eigenschaft *Zeitgeberintervall* des Formulars den Wert *500* ein und hinterlegen für die Ereigniseigenschaft *Bei Zeitgeber* die folgende Routine:

```
Private Sub Form_Timer()
    Static bol As Boolean
    bol = Not bol
    If bol Then
        Me.NavigationCaption = "Hier navigieren!->"
    Else
        Me.NavigationCaption = "->Hier navigieren!"
    End If
End Sub
```

Listing 3.1: Wechselnde Navigationsleistenbeschriftungen per Zeitgeber

Dies lässt die Beschriftung *»Hier navigieren«* hin- und herspringen. Beachten Sie, dass die verwendeten Zeichen immer die gleiche Breite einnehmen, sonst springt auch die eigentliche Navigationsleiste hin und her. (Kleiner Tipp: Nehmen Sie nicht alles ernst, was in diesem Buch geschrieben steht.)

Formular-Ribbon

Sie können für jedes Formular eine der in der Datenbank gespeicherten Ribbon-Definitionen festlegen. Damit können Sie beispielsweise dafür sorgen, dass bei Aktivierung eines Formulars nur formularspezifische Ribbon-Elemente angezeigt werden, oder einfach das Ribbon um ein spezielles Tab-Element für das Formular erweitern. Sie legen das Ribbon mit der Formular-Eigenschaft *Name des Menübands* (unter VBA: *RibbonName*) fest.

Dies funktioniert nicht bei Popup- und geteilten Formularen. Weitere Informationen zum Thema Ribbons finden Sie unter »Ribbon« ab Seite 713.

3.1.7 Formularvorlage

Nicht neu, aber möglicherweise nicht jedem bekannt sind die folgenden beiden Features, die das Erstellen von Formularen vereinfachen.

Mit der Zeit werden Sie sich einen gewissen Stil angewöhnen, was das Aussehen von Formularen angeht. Vielleicht verwenden Sie eine bestimmte Hintergrundfarbe für Kopf-, Fuß- und Detailbereich oder stellen weitere Eigenschaften immer auf die gleichen Werte ein.

Wenn Sie solche Standardeinstellungen öfter benötigen, können Sie ein Formular mit all den gewünschten Eigenschaften anlegen und es unter dem Formularnamen *Normal* speichern. Von nun an besitzen alle neu erstellten Formulare standardmäßig diese Eigenschaften. Erst wenn Sie das Formular *Normal* entfernen oder umbenennen, verwendet Access wieder die ursprünglichen Standardeinstellungen beim Anlegen von Formularen.

Das Gleiche gilt übrigens auch für Berichte, auch hier verwenden Sie standardmäßig den Berichtsnamen *Normal* für die Vorlage. Wenn Sie einen anderen Formular- beziehungsweise Berichtsnamen als *Normal* einsetzen möchten, können Sie dies in den Access-Optionen unter *Objekt-Designer|Formular/Berichte|Formularvorlage/Berichtsvorlage* einstellen.

Beim Öffnen filtern und sortieren

Mit den beiden Eigenschaften *Beim Laden filtern* (VBA: *FilterOnLoad*) und *Beim Laden sortieren* (VBA: *OrderByOnLoad*) legen Sie fest, ob eine in den Eigenschaften *Filter* (*Filter*)

oder *Sortiert nach* (*OrderBy*) angegebene Filterung beziehungsweise Sortierung direkt beim Laden des Formulars ausgeführt werden soll.

An Bildschirmgröße anpassen

Mit der gleichnamigen Eigenschaft bestimmen Sie, ob Formulare, die zu groß für die aktuelle Access-Fensterbreite sind, dem verfügbaren Platz angepasst werden sollen. Unter VBA lautet der Name dieser Eigenschaft *FitToScreen* – obwohl dies streng genommen keine Anpassung an den Bildschirm, sondern an den freien MDI-Bereich von Access ist.

Keine Entwurfsänderungen in der Formularansicht

Die Eigenschaft *Entwurfsänderungen zulassen* (*AllowDesignChanges*) entfällt. Sie bewirkte in älteren Versionen, dass Formulare nur in der Entwurfsansicht das Eigenschaftenfenster angezeigt haben. Seit Access 2007 können Sie das Eigenschaftenfenster grundsätzlich nur noch in der Entwurfs- und der Layoutansicht einblenden.

3.1.8 Berichte in Unterformularen

Mit Access 2010 können Unterformulare erstmals Berichte anzeigen. Dazu legen Sie einfach den Berichtsnamen als Wert für die Eigenschaft *Herkunftsobjekt* fest.

3.2 Formulare öffnen

Formulare öffnen Sie in der Regel mit der *DoCmd.OpenForm*-Anweisung. Es geht zwar auch über das Instanzieren des entsprechenden Formularobjekts und anschließendes Einstellen der Eigenschaft *Visible* auf den Wert *True*, das ist aber eher Spezialfällen wie dem Öffnen mehrerer Instanzen des gleichen Formulars vorbehalten.

Da die *DoCmd.OpenForm*-Anweisung allgemein bekannt sein dürfte (ansonsten liefert die Onlinehilfe einen guten Überblick), soll sie hier nicht im Detail beschrieben werden. Im Hinblick auf die nachfolgenden Beispiele sei nur erwähnt, dass die dortigen Aufrufe »benannte Parameter« verwenden. Das bedeutet, dass die optionalen Parameter nicht in der vorgegebenen Reihenfolge durch Kommata getrennt an die Anweisung angefügt werden, sondern unter Angabe des jeweiligen Parameternamens:

```
DoCmd.OpenForm "frmKontakteDetailansicht", DataMode:=acFormAdd, _
    WindowMode:=acDialog
```

Mit einer herkömmlichen Parameterliste hätte diese Anweisung so ausgesehen:

```
DoCmd.OpenForm "frmKontakteDetailansicht", , , , acFormAdd, acDialog
```

3.3 Ereignisse in Formularen und Steuerelementen

Formulare lassen sich wie auch Berichte bis zu einem gewissen Grad ohne den Einsatz von Ereigniseigenschaften und VBA verwenden. Sobald Sie aber auch nur eine *OK*-Schaltfläche hinzufügen möchten, ist es so weit: Die Programmierung der ersten Ereignisprozedur steht an. Ereignisse stellen die Schnittstelle zwischen Oberflächenelementen und der VBA-Programmierung derselben bereit. In diesem Abschnitt erfahren Sie, welche der zahlreichen Ereignisse von Formularen und der enthaltenen Steuerelemente oft zum Einsatz kommen und wie Sie diese optimal einsetzen – dazu gehört auch, dass Sie sich mit der Reihenfolge der Ereignisse vertraut machen.

3.3.1 Ereignisse in Formularen

Formulare bieten eine unüberschaubare Menge Ereignisse. Seit Access 2007 sind keine mehr hinzugekommen, lediglich die neuen Steuerelemente bringen einige neue Ereignisse mit. Weitere Informationen zu diesen Steuerelementen finden Sie im Kapitel »Steuerelemente« ab Seite 263.

Anlegen eines Ereignisses

Abbildung 3.23 zeigt einen Teil der Ereigniseigenschaften von Formularen.

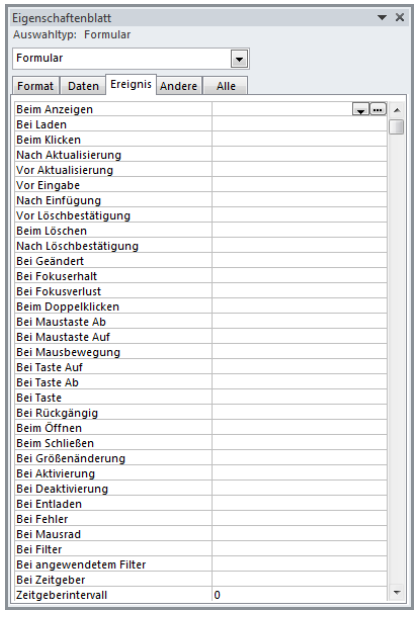


Abbildung 3.23: Übersicht der Ereigniseigenschaften eines Formulars

Zum Anlegen eines Ereignisses klicken Sie doppelt in das entsprechende Textfeld, sodass dieses den Wert *[Ereignisprozedur]* erhält, und betätigen dann die Schaltfläche mit den drei Punkten (...) wie in Abbildung 3.24. Wenn Sie es noch einfacher haben möchten, aktivieren Sie in den Access-Optionen im Bereich *Objekt-Designer|Formulare/Berichte* den Eintrag *Immer Ereignisprozeduren verwenden*.

Dann müssen Sie nur noch auf die beim Aktivieren des Feldes erscheinende Schaltfläche klicken, um die Ereignisprozedur anzulegen. Wenn Sie dies beispielsweise mit der ersten aufgeführten Ereigniseigenschaft *Beim Anzeigen* machen, öffnet sich der VBA-Editor und zeigt direkt den Rumpf der gewünschten Prozedur an:

```
Private Sub Form_Current()


End Sub
```

Listing 3.2: Leerer Rumpf einer frisch angelegten Ereignisprozedur

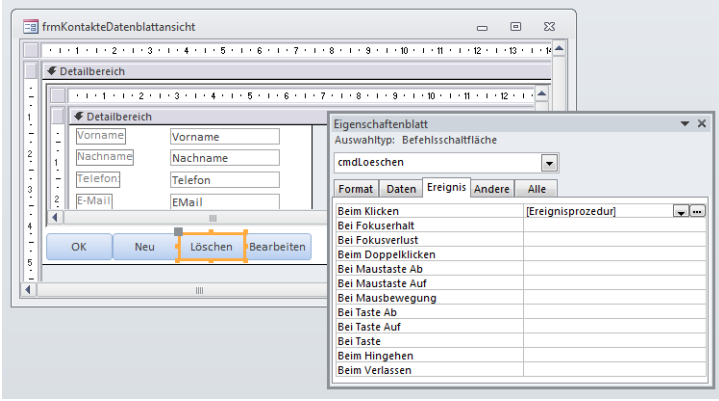


Abbildung 3.24: Anlegen einer Ereignisprozedur

Neben den Ereignisprozeduren können Sie auch Access-Makros und VBA-Funktionen per Ereignis auslösen. Dazu tragen Sie einfach den Namen des Access-Makros oder der Funktion ein. Access-Makros bleiben in diesem Buch weitgehend außen vor; der Einsatz von Funktionen kann jedoch durchaus sinnvoll sein. So können Sie beispielsweise für *OK*-Schaltflächen eine globale Funktion schreiben, die eine Anweisung zum Schließen des Formulars enthält, oder eine Funktion im Formularmodul anlegen, die durch mehrere Ereignisse des gleichen Formulars ausgelöst wird.

Ereigniseigenschaft und Ereignisprozedur — ein unzertrennliches Paar

Damit ein Ereignis durch die entsprechende Aktion – also etwa Betätigen einer Schaltfläche – ausgelöst wird, muss die Ereigniseigenschaft den Eintrag *[Ereignisprozedur]* ent-

halten und eine Ereignisprozedur mit dem für dieses Ereignis vorgesehenen Namen im Klassenmodul des Formulars oder des Berichts vorliegen – beispielsweise *cmdOK_Click*.

3.3.2 Abfolge und Bedeutung der Ereignisse beim Öffnen und Schließen eines Formulars

Besonders wichtig beim Umgang mit Ereigniseigenschaften ist die Reihenfolge ihrer Abarbeitung und der Zusammenhang mit den im Hintergrund ausgelösten internen Prozessen wie Laden der angezeigten Daten, Übergeben von Öffnungsargumenten und dergleichen in gebundenen Formularen. Selbst wenn Sie eine Access-Anwendung völlig ohne Dokumentation programmieren müssen, können Sie sich hier selbst weiterhelfen: Schreiben Sie einfach für alle Ereigniseigenschaften, die Sie interessieren, eine kleine Prozedur, die den jeweiligen Ereignisnamen ausgibt. Wenn Sie dann das Formular öffnen oder die gewünschte Aktion durchführen, können Sie anschließend oder währenddessen beobachten, in welcher Reihenfolge die Ereignisse ablaufen. Wenn Sie beispielsweise herausfinden, welche Sequenz von Ereignissen beim einfachen Öffnen und Schließen eines Formulars abläuft, legen Sie die Ereignisprozeduren aus folgendem Listing an:

```
Private Sub Form_Activate()
    Debug.Print "Beim Aktivieren"
End Sub

Private Sub Form_Close()
    Debug.Print "Beim Schließen"
End Sub

Private Sub Form_Current()
    Debug.Print "Beim Anzeigen"
End Sub

Private Sub Form_Deactivate()
    Debug.Print "Beim Deaktivieren"
End Sub

Private Sub Form_Load()
    Debug.Print "Beim Laden"
End Sub

Private Sub Form_Open(Cancel As Integer)
    Debug.Print "Beim Öffnen"
End Sub
```

```
Private Sub Form_Resize()
    Debug.Print "Bei Größenänderung"
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Debug.Print "Beim Entladen"
End Sub
```

Listing 3.3: Ausgabe von Meldungen beim Auslösen unterschiedlicher Ereignisprozeduren

Als Ergebnis erhalten Sie beim Öffnen des Formulars folgende Abfolge:

» *Beim Öffnen*: Tritt beim Öffnen ein. Bietet die Möglichkeit, das Öffnen abzubrechen, wenn beispielsweise keine Daten vorhanden sind – diese werden dementsprechend bereits vor dem *Beim Öffnen*-Ereignis eingelesen.

» *Beim Laden*: Tritt ein, wenn das Formular einschließlich Steuerelementen geöffnet, aber noch nicht sichtbar ist. Eignet sich für Aktionen wie Setzen von Standardwerten oder Werten von Steuerelementen.

» *Bei Größenänderung*: Wird beim Öffnen und bei der Änderung der Größe eines Formulars ausgelöst. Kann zum Anpassen der Größe von Steuerelementen an die Größe des Formulars verwendet werden – falls Sie nicht zufällig von den Möglichkeiten rund um die Verankerung von Steuerelementen Gebrauch machen.

» *Beim Aktivieren*: Wird ausgelöst, wenn das Formular den Fokus erhält.

» *Beim Anzeigen*: Wird beim Anzeigen, bei jedem Datensatzwechsel und beim Aktualisieren ausgelöst.

Beim Schließen sieht der Ablauf so aus:

» *Beim Entladen*: Wird durch einen Klick auf eine *Schließen*-Schaltfläche oder die *DoCmd.Close*-Anweisung ausgelöst, findet aber vor dem eigentlichen Schließen statt. Der Schließen-Vorgang kann an dieser Stelle durch Setzen des *Cancel*-Parameters auf den Wert *True* unterbrochen werden.

» *Beim Deaktivieren*: Wird ausgelöst, wenn das Formular den Fokus verliert.

» *Beim Schließen*: Wird im Moment des Schließens ausgelöst.

3.3.3 Abfolge und Bedeutung der Ereignisse beim Bearbeiten von Datensätzen

Das Bearbeiten eines Datensatzes beginnt mit der ersten Änderung an einem der Felder und endet mit dem Speichervorgang des Datensatzes.

Ändern von Feldinhalten

Änderungen von Datensätzen beginnen mit dem Ändern des Inhalts mindestens eines Feldes. Dies löst eines oder mehrere der folgenden Ereignisse aus:

» *Vor Eingabe*: Wird vor der Eingabe des ersten Zeichens ausgelöst, aber nur in Verbindung mit neuen Datensätzen. Kann abgebrochen werden.

» *Bei Geändert*: Wird beim ersten Eingeben oder Löschen eines Zeichens in einem der Datensätze ausgelöst. Kann abgebrochen werden.

Speichern der Änderungen

Wird die Änderung gespeichert, werden in der Regel die folgenden drei Ereignisse ausgelöst. Das erste Ereignis *Vor Aktualisierung* kann den Ablauf allerdings abbrechen, etwa wenn eine Validierung fehlschlägt.

» *Vor Aktualisierung*: Wird nach dem Auslösen des Speicherns, aber vor dem eigentlichen Speichervorgang ausgelöst. Ist ein gängiger Platz für die Durchführung von Validierungen. Folgendes Beispiel zeigt, wie Sie die Aktualisierung abbrechen, wenn bestimmte Daten nicht vorhanden sind (weitere Informationen zum Validieren siehe »Validieren vor dem Speichern« ab Seite 252):

```
If IsNull(Me!Telefon) And IsNull(Me!EMail) Then
    MsgBox "Bitte geben Sie eine Telefonnummer oder eine " _
        &  "E-Mail-Adresse ein.", vbExclamation + vbOKOnly, _
        "Fehlende Daten"
    Me!Telefon.SetFocus
    Cancel = True
End If
```

» *Nach Aktualisierung*: Wird nach dem Speichern des Datensatzes ausgelöst.

» *Nach Eingabe*: Wird nach dem Speichern eines neuen Datensatzes ausgelöst; gilt nicht für bestehende Datensätze.

» *Beim Anzeigen*: Wird beim Wechseln beziehungsweise Speichern des Datensatzes aufgerufen.

Abbruch des Änderungsvorgangs

Neben dem Speichern bietet sich nach Änderungen an Feldern eines Datensatzes auch die Möglichkeit, den Vorgang abzubrechen und die vorgenommenen Änderungen am aktuellen Datensatz zu verwerfen. Dies löst das *Bei Rückgängig*-Ereignis des Formulars aus. Das Ereignis kann durch Setzen des *Cancel*-Parameters auf den Wert *True* abgebrochen werden.

Löschen eines Datensatzes

Das Löschen von Datensätzen ist keine triviale Geschichte. Wenn Sie auf eine *Löschen*-Schaltfläche klicken, ist der Datensatz noch lange nicht gelöscht.

Zunächst wird die Anzeige aktualisiert, die auf einem temporären, zwischengespeicherten Datenbestand basiert, und dann noch abgewartet, ob die Ereignisse *Bei Löschbestätigung* oder *Nach Löschbestätigung* Änderungen bringen. Erst dann überträgt Access die Änderungen in die Tabellen der Datenbank.

» *Beim Löschen*: Wird beim Aufrufen des Löschvorgangs und unmittelbar vor dem Löschen ausgelöst. Wenn mehrere Datensätze gleichzeitig gelöscht werden, wird dieses Ereignis für jeden Datensatz einmal aufgerufen.

» *Beim Anzeigen*: Wird nach dem Löschen, aber vor dem Übernehmen der Änderungen in die Tabelle ausgelöst.

» *Bei Löschbestätigung*: Wird nur ausgelöst, wenn Access eine Bestätigung der Datensatzänderung anzeigt – diese Option lässt sich im Dialog aus Abbildung 3.25 einstellen. Die Bestätigungsmeldung kann unterbunden und durch eine eigene Meldung ersetzt werden. Beispiel:

```
If MsgBox("Möchten Sie den Datensatz wirklich löschen?", _
    vbExclamation + vbOKCancel, _
    "Löschbestätigung") = vbCancel Then
    Cancel = True
End If
Response = acDataErrContinue
```

» *Nach Löschbestätigung*: Wird nur ausgelöst, wenn eine angezeigte Bestätigung einer Datensatzänderung auch bestätigt wurde beziehungsweise wenn der Löschvorgang nach einer benutzerdefinierten Meldung im Ereignis *Bei Löschbestätigung* nicht abgebrochen wurde.

3.4 Ereignisse von Steuerelementen

Neben dem Formular selbst lösen auch die Steuerelemente Ereignisse aus. Nachfolgend finden Sie einige wichtige Ereigniseigenschaften von Steuerelementen.

Unterformulare

Wenn Sie mit Unterformularen arbeiten, sollten Sie wissen, in welcher Reihenfolge die Ereignisse beim Öffnen und Schließen von Unterformularen in Bezug auf die Ereignisse des Hauptformulars ausgelöst werden.

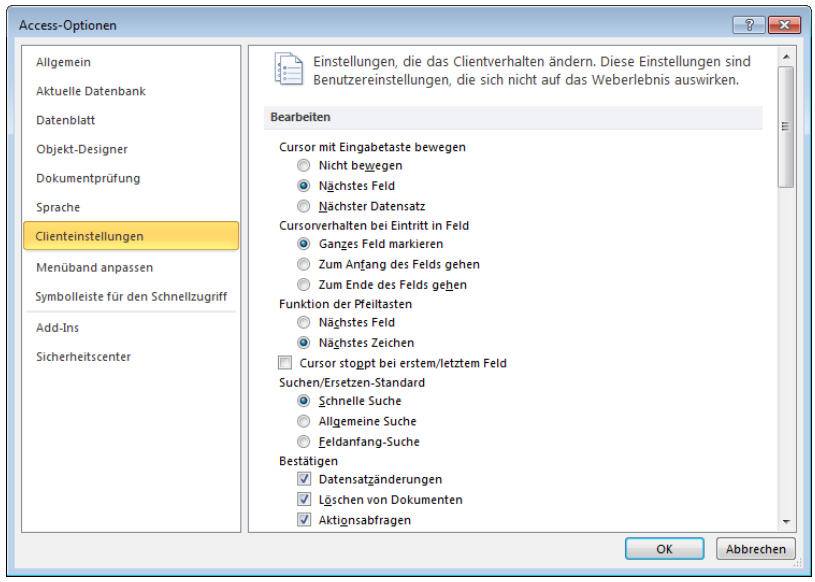


Abbildung 3.25: Aktivieren der Anzeige einer Meldung beim Ändern von Datensätzen

Die Abfolge sieht folgendermaßen aus:

» Unterformular *Beim Öffnen*

» Unterformular *Beim Laden*

» Unterformular *Bei Größenänderung*

» Unterformular *Beim Anzeigen*

» Hauptformular *Beim Öffnen*

» Hauptformular *Beim Laden*

» Hauptformular *Bei Größenänderung*

» Hauptformular *Bei Aktivierung*

» Hauptformular *Beim Anzeigen*

Das Unterformular wird in der Tat komplett vor dem Hauptformular geladen. Lediglich das Filtern der Datensätze des Unterformulars in Abhängigkeit von den für das Hauptformular vorgesehenen Datensätzen erfolgt noch vor dem *Beim Öffnen*-Ereignis des Unterformulars.

Beim Schließen des Formulars wird zuerst das Hauptformular und dann das Unterformular geschlossen:

» Hauptformular *Beim Entladen*

» Hauptformular *Bei Deaktivierung*

» Hauptformular *Beim Schließen*

» Unterformular *Beim Entladen*

» Unterformular *Beim Schließen*

Textfelder

Beim Setzen der Einfügemarke in ein Textfeld, beim Ändern des Inhalts und beim anschließenden Verlassen treten die folgenden Ereignisse auf:

» *Beim Hingehen*: Beim Eintreten in das Feld

» *Bei Fokuserhalt*

» *Bei Geändert*: Beim Ändern des ersten Zeichens. Stellt die Eigenschaft *Dirty* auf den Wert *True* ein.

» *Bei Änderung*: Beim Ändern jedes Zeichens

» *Vor Aktualisierung*: Bei jeder Aktion, die zum Verlassen des Feldes führt. Dieses Ereignis bietet die Möglichkeit, feldbezogene Validierungen durchzuführen und die Aktualisierung abzubrechen. Beispiel:

```
If IsNumeric(Left(Me.Projekt, 1)) Then
    MsgBox "Der Projektname darf nicht mit einer Zahl " _
        & "beginnen.", vbOKOnly + vbExclamation, _
        "Fehlerhafte Eingabe"
    Cancel = True
End If
```

» *Nach Aktualisierung:* Nach dem Ereignis *Vor Aktualsisierung*, wenn dieses nicht abgebrochen wurde. Bietet die Möglichkeit, den eingegebenen Wert weiter zu verwerten oder zu ändern.

» *Beim Verlassen*: Nach Abarbeitung der Ereignisse *Vor Aktualisierung* und *Nach Aktualisierung*

» *Bei Fokusverlust*: Beim Setzen des Fokus auf ein anderes Steuerelement

In Zusammenhang mit diesen Ereignissen sind drei Eigenschaften von Textfeldern wichtig: *Value, OldValue* und *Text. OldValue* enthält während des ganzen Änderungsvorgangs den vorherigen Wert des Feldes. *Text* enthält den aktuell im Textfeld angezeigten Wert und *Value* den alten Wert, bis es mit dem Auslösen des Ereignisses *Vor Aktualisierung*

den aktuellen Inhalt des Feldes beziehungsweise der Eigenschaft *Text* zugewiesen bekommt.

Kombinationsfelder

Das Auswählen eines Eintrags und das anschließende Verlassen eines Kombinationsfeldes löst die folgenden Ereignisse aus (wenn dieses noch nicht den Fokus hat).

In runden Klammern finden Sie Ereignisse, die nur bei der manuellen Eingabe von Zeichen ausgeführt werden, in eckigen Klammern die Ereignisse, die nur bei manueller Eingabe nicht vorhandener Listeneinträge ausgelöst werden.

» *Beim Hingehen*

» *Bei Fokuserhalt*

» *Bei Änderung*: Beim Auswählen eines Eintrags oder bei der ersten Eingabe eines Zeichens

» (*Bei Geändert*: Bei der manuellen Eingabe von Zeichen)

» [*Bei nicht in Liste*: Nach der manuellen Eingabe eines Wertes, der nicht in der Liste enthalten ist. Hier können Sie eine Prozedur hinterlegen, die nicht in der Liste enthaltene Einträge in der zugrunde liegenden Datensatzherkunft anlegt und den neuen Wert im Kombinationsfeld festlegt. In diesem Fall folgen die übrigen Ereignisse, sonst wird der Änderungsvorgang unterbrochen.]

» *Vor Aktualisierung*

» *Nach Aktualisierung*

» *Beim Klicken*

» *Bei Geändert*

» *Beim Verlassen*

» *Bei Fokusverlust*

Weitere Steuerelemente

Die übrigen Steuerelemente wie Listenfelder, Kontrollkästchen oder Optionsgruppen haben je nach Typ unterschiedliche Ereignisse. Nachdem Sie erfahren haben, wie Sie die Abfolge der Ereignisse von Formularen und Steuerelementen ermitteln, soll an dieser Stelle nicht weiter auf die Ereignisse der noch nicht besprochenen Steuerelemente und ihre Abfolge eingegangen werden. Diesbezügliche Unklarheiten lassen sich leicht experimentell beseitigen, indem Sie die betroffenen Ereignisse mit entsprechenden Prozeduren ausstatten.

3.5 Abbildung verschiedener Beziehungsarten

Die vorhandene Fachliteratur beschränkt sich weitgehend auf die Darstellung von Daten aus einzelnen oder aus per 1:n-Beziehung verknüpften Tabellen. Daher erhalten Sie in diesem Abschnitt des Kapitels einen Überblick über die Realisierung der verschiedenen Beziehungsarten in Formularen.

3.5.1 Einfache Daten in der Detailansicht

Die Daten aus einzelnen Tabellen oder einfachen Abfragen lassen sich mit wenigen Schritten in einem Formular darstellen. Üblicherweise sind hier zwei Darstellungen gefragt: Detailansichten oder Übersichtslisten.

Mit »einfachen Abfragen« sind hier Abfragen gemeint, deren Daten quasi wie eine Tabelle gehandhabt werden – also etwa Abfragen, die zwei per 1:1-Beziehung verknüpfte Tabellen darstellen oder die eine einzelne Tabelle mit einem oder mehreren *Lookup*-Feldern beinhalten.

Detailansichten dienen dazu, die sich oft über viele Felder erstreckenden Daten in einer Form anzuzeigen, die die Daten einerseits vernünftig strukturiert und andererseits eine komfortable Bearbeitung ermöglicht.

Übersichtslisten dienen selten dazu, die Daten kompletter Tabellen anzuzeigen – es sei denn, diese enthalten nicht besonders viele Felder.

Wenn der Benutzer scrollen muss, um alle Felder eines Formulars sehen zu können, ist die Menge der angezeigten Felder zu groß.

Meist dienen solche Übersichtslisten dazu, wichtige Informationen zu den angezeigten Datensätzen sowie die Anzeige einer Detailansicht zu bieten.

Übersichtlisten gibt es in drei Formen: als Endlosformular, Datenblattansicht oder in geteilten Formularen.

Detailansicht einfacher Daten in Formularen

Die Erstellung einer Detailansicht einfacher Daten läuft in folgenden Schritten ab:

» Anlegen eines neuen, leeren Formulars

» Festlegen der Datensatzquelle im Eigenschaftsfenster

» Einfügen der benötigten Felder (Ribboneintrag *Entwurf|Tools|Vorhandene Felder hinzufügen*, siehe Abbildung 3.26)

» Anpassen und Ausrichten der Felder und Anpassen der Beschriftungen

Das ist im Prinzip einfach. Wichtig ist lediglich, dass Sie beim Anlegen der Datensatzquelle in Schritt 2 darauf achten, dass die Datensatzquelle nur die Felder enthält, die Sie auch wirklich benötigen.

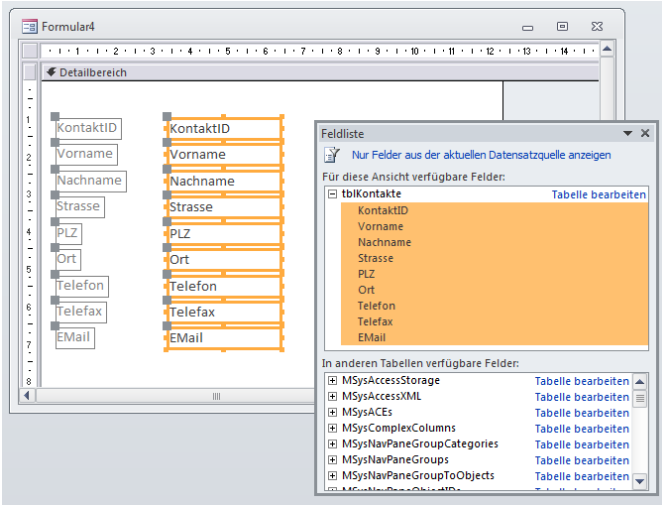


Abbildung 3.26: Das Hinzufügen der Einträge der Feldliste in das Formular kann durch Ziehen mit der Maus erfolgen

Muss man in Detailansichten blättern können?

Die oben erstellte Detailansicht enthält einige Elemente, die die Navigation in den Datensätzen des Formulars erleichtern:

» *Bildlaufleisten*

» *Datensatzmarkierer*

» *Navigationsschaltflächen*

Das Formular aus Abbildung 3.27 enthält all diese Elemente. Die Frage ist nun: Welche davon benötigen Sie in einer Detailansicht?

Was zu einer anderen Frage führt: Was macht der Benutzer eigentlich mit dieser Detailansicht?

Nun, er soll die Details eines Datensatzes betrachten und gegebenenfalls die enthaltenen Daten ändern beziehungsweise neue Datensätze anlegen oder bestehende löschen können.

Soll er mit der Navigationsleiste arbeiten, um innerhalb der Datensätze zu navigieren? Seit Access 2007 vielleicht: Immerhin befindet sich dort ein *Suchen*-Feld.

Optimal wäre es trotzdem, wenn der Benutzer die oben genannten Elemente gar nicht benötigt, sondern mit anderen Mitteln zu einem gesuchten oder einem neuen Datensatz gelangt – immerhin ist nicht jeder Benutzer mit diesen Steuerelementen vertraut und intuitiv zu bedienen sind sie auch nicht unbedingt.

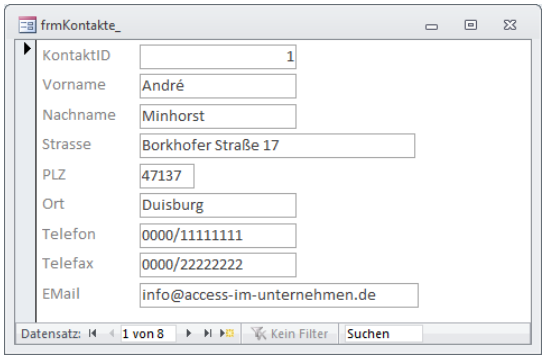


Abbildung 3.27: Die Formularansicht des Detailformulars

Angenommen, der Benutzer benötigt diese zusätzlichen Elemente gar nicht: Dann zeigen Sie diese doch gar nicht an.

Drei Mausklicks im Eigenschaftsfenster der Entwurfsansicht des Formulars, einer noch, wenn das Formular beim Öffnen zentriert angezeigt werden soll, was immer Sinn macht, und noch eine Beschriftung hinzufügen – blitzschnell sieht das Eigenschaftsfenster wie in Abbildung 3.28 und das Formular wie in Abbildung 3.29 aus.

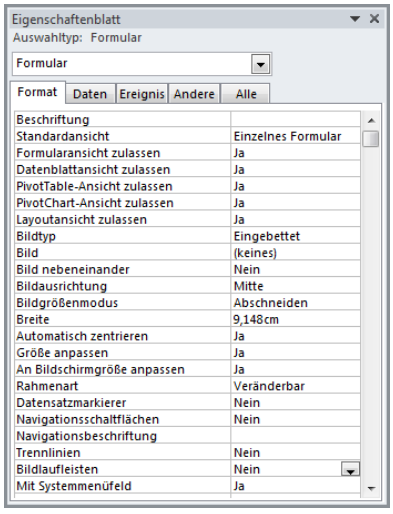


Abbildung 3.28: Eigenschaften eines Formulars ...

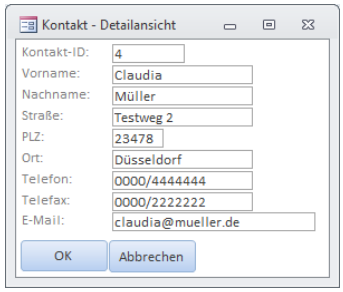


Abbildung 3.29: ... ohne unnötigen Schnickschnack (*frmKontakteDetailansicht*)

Navigation und Aktion in Detailformularen

Nun fehlen noch die Möglichkeiten zum Auswählen eines Datensatzes, zum Anlegen eines neuen und Löschen des bestehenden Datensatzes. Nur: Müssen diese Elemente auf das Detailformular? Das ist sicher Geschmackssache, aber Sie sollten sich für eine der folgenden beiden Möglichkeiten entscheiden: Entweder Sie verwenden nur eine *OK*- und eine *Abbrechen*-Schaltfläche oder Sie fügen neben einer Möglichkeit zur Auswahl von Datensätzen auch noch je eine Schaltfläche zum Löschen und zum Anlegen von Datensätzen hinzu.
Nachfolgend finden Sie eine Beschreibung der ersten Variante. Die Schaltflächen zum Auswählen, Anlegen und Löschen eines Datensatzes werden in Zusammenhang mit den unten beschriebenen Übersichtsformularen in der Endlos- und der Datenblattansicht beschrieben.

OK- und Abbrechen-Schaltflächen

Diese beiden Schaltflächen lösen jeweils Prozeduren mit nur einer Zeile VBA-Code aus. Legen Sie die beiden Schaltflächen an, stellen Sie die Eigenschaft *Beschriftung* auf die Werte *OK* und *Abbrechen* und die Eigenschaft *Name* auf die Werte *cmdOK* und *cmdAbbrechen* ein. Legen Sie dann für beide je eine Prozedur für die Ereigniseigenschaft *Beim Klicken* an (siehe Abbildung 3.30):

```
Private Sub cmdOK_Click()
    DoCmd.Close acForm, Me.Name
End Sub

Private Sub cmdAbbrechen_Click()
    Me.Undo
    DoCmd.Close acForm, Me.Name
End Sub
```

Listing 3.4: Schließen mit und ohne Übernahme der Änderungen

Die *OK*-Schaltfläche sorgt in der Regel nur dafür, dass ein Formular geschlossen wird. Es gibt aber auch Ausnahmen: Wenn ein Formular geöffnet wurde, um Daten zu ermitteln, die in der aufrufenden Routine weiter verarbeitet werden sollen, müssen diese vor dem Schließen natürlich erst abgefragt werden. Wie dies funktioniert, erfahren Sie weiter unten in »Von Formular zu Formular« ab Seite 233.

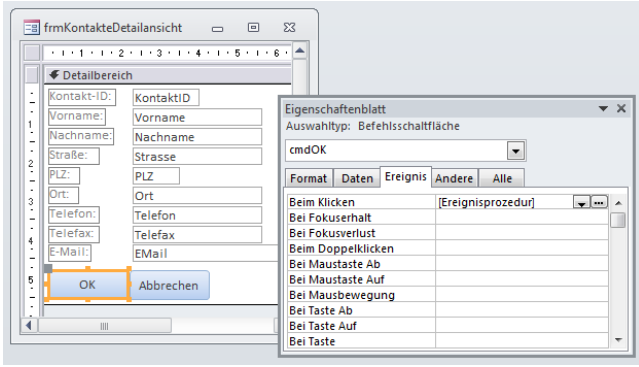


Abbildung 3.30: Anlegen der Beim Klicken-Ereigniseigenschaft für eine Schaltfläche (*frmDetailansicht*)

3.5.2 Einfache Daten in der Übersicht mit Endlosformularen

Ein Formular zur Anzeige der Übersicht von Datensätzen enthält die gleiche Datensatzquelle wie das Formular zur Anzeige der Detailansicht, in der Regel jedoch mit weniger Feldern. Für Tabellen, die so wenige Felder enthalten, dass diese leicht nebeneinander angezeigt werden können, braucht prinzipiell gar keine Detailansicht erstellt zu werden. Die Felder sind meist wie in Abbildung 3.31 nebeneinander angeordnet. In der Abbildung finden Sie direkt die Elemente zum Steuern von Aktionen wie Neuanlegen, Löschen und Bearbeiten von Datensätzen.

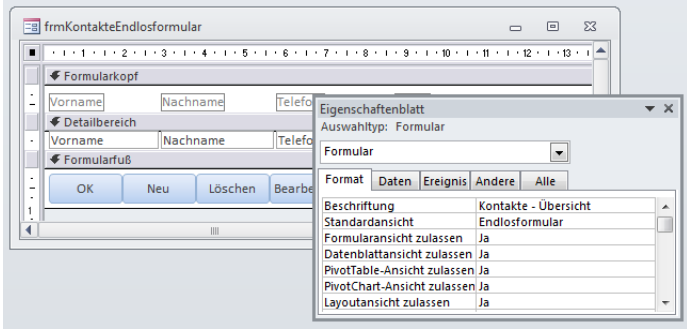


Abbildung 3.31: Entwurfsansicht eines Übersichtsformulars (*frmKontakteEndlosformular*)

Die *OK*-Schaltfläche dient wie die Schaltfläche des zuvor beschriebenen Formulars lediglich dem Schließen des Formulars.

Die *Neu*-Schaltfläche soll das oben beschriebene Formular zur detaillierten Ansicht eines Datensatzes öffnen und einen leeren Datensatz anzeigen. Dafür ist die folgende Ereigniseigenschaft verantwortlich, die durch das *Beim Klicken*-Ereignis der Schaltfläche ausgelöst wird.

Formular mit leerem Datensatz öffnen

Die Routine verwendet die *OpenForm*-Methode des *DoCmd*-Objekts zum Öffnen des Formulars *frmKontakteDetailansicht*. Der Parameter *DataMode* erhält dabei den Wert *acFormAdd*, damit das Formular beim Öffnen direkt einen leeren Datensatz anzeigt.

Formular als modalen Dialog öffnen

Der Wert des Parameters *WindowMode* liefert die Voraussetzung dafür, dass das aufrufende Formular – die Übersicht – den angezeigten Datenbestand direkt nach dem Eingeben des neuen Datensatzes und Schließen des Detailformulars aktualisieren kann (siehe Abbildung 3.32). Durch den Wert *acDialog* wird das Formular *frmKontakteDetailansicht* als modaler Dialog geöffnet, was bedeutet, dass innerhalb der Access-Anwendung nichts mehr geht, solange dieses Formular geöffnet ist – selbst der aufrufende Code wird währenddessen angehalten. Erst wenn das Formular den Fokus verliert, also geschlossen oder unsichtbar gemacht wird, läuft die aufrufende Routine weiter. Dadurch kann die Übersicht direkt nach dem Schließen der Detailansicht aktualisiert werden.

```
Private Sub cmdNeu_Click()
    DoCmd.OpenForm "frmKontakteDetailansicht", DataMode:=acFormAdd, _
        WindowMode:=acDialog
    Me.Requery
End Sub
```

Listing 3.5: Aufrufen des Detailformulars zum Anlegen eines neuen Datensatzes

Löschen von Datensätzen

Die nächste Schaltfläche dient dem Löschen des aktuell markierten Datensatzes. Die einfachste Variante des benötigten Codes sieht wie folgt aus:

```
Private Sub cmdLoeschen_Click()
    On Error Resume Next
    DoCmd.RunCommand acCmdDeleteRecord
End Sub
```

Listing 3.6: Löschen eines Datensatzes

4

Steuerelemente

Das vorhergehende Kapitel, »Formulare«, setzt die Kenntnis der Standardsteuerelemente voraus. So ist es auch in diesem Kapitel. Es erweitert Ihre Grundkenntnisse um interessante Techniken zur optimalen Verwendung von Steuerelementen.

Aber es beschränkt sich nicht auf die Standardsteuerelemente, sondern zeigt auch, was Sie mit ActiveX-Steuerelementen wie dem TreeView-, dem ListView- oder dem ImageList-Steuerelement anfangen können.

Und natürlich stellt es die Neuigkeiten vor, die Access mit den Versionen 2007 und 2010 in Zusammenhang mit Steuerelementen liefert – neben einigen Detailverbesserungen gibt es übrigens sogar ein ganz neues Steuerelement. Um Sie mit den wichtigsten Informationen zu versorgen, lassen wir das eine oder andere Steuerelement aus: Hier

sollten Sie sich die grundlegenden Techniken mit Hilfe der Onlinehilfe oder einem Einsteigerbuch aneignen.

BEISPIELDATENBANKEN

Den Download mit dem Beispielen finden Sie unter *www.acciu.de/aeb2010*. Die Datenbank zu diesem Kapitel heißt *Steuerelemente.accdb*.

4.1 Textfelder

Textfelder als meistgenutztes Steuerelement sind schon in Access 2007 natürlich nicht von Änderungen verschont geblieben. Wie Sie grundsätzlich mit Textfeldern umgehen, beschreibt die Onlinehilfe im Artikel »Hinzufügen eines Textfeld-Steuerelements zu einem Formular oder zu einem Bericht«.

4.1.1 Rich-Text in Textfeldern

In Tabellen können Sie für Memofelder nun das Textformat Rich-Text einstellen. Wenn Sie ein Textfeld auf Basis eines solchen Feldes in einem Formular verwenden, können Sie die Eigenschaft *Textformat* auch im Formular auf *Rich-Text* einstellen (bei »normalen« Textfeldern erzeugt dies eine Meldung, dass die Einstellung ungültig sei).

Textfelder im Rich-Text-Modus zeigen beim Markieren von Textpassagen mit der Maus ein zunächst transparentes Menü an, aus dem Sie mit der Maus die gewünschte Formatierung auswählen können (siehe Abbildung 4.1). Sie können aber auch die Ribbon-Schaltflächen aus dem Bereich *Start|Textformatierung* für einzelne Formatierungsoptionen verwenden.

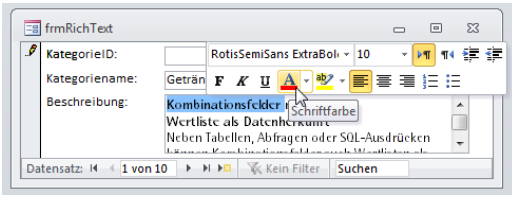


Abbildung 4.1: Ein Textfeld im Rich-Text-Modus

Die formatierten Texte speichert Access im HTML-Format. Den Quelltext können Sie normalerweise nicht einsehen, es sei denn, Sie stellen die Eigenschaft *Textformat* eines darauf basierenden Textfelds auf *Nur Text* ein (erzeugt eine vorherige Warnung). Per VBA legen Sie diese Eigenschaft so fest:

```
Me!Textbox1.TextFormat = acTextFormatPlain
Me!Textbox1.TextFormat = acTextFormatHTMLRichText
```

Sie können auch per VBA auf den Inhalt zugreifen. Dazu reicht eine einfache *DLookup*-Anweisung aus. Alternativ können Sie sich den HTML-Text mit folgenden Zeilen – etwa beim Anzeigen des aktuellen Datensatzes eines Formulars – anzeigen lassen:

```
Private Sub Form_Current()
    Debug.Print Me!Beschreibung
    Debug.Print Application.PlainText(Me!Beschreibung)
End Sub
```

Listing 4.1: Ausgeben des HTML- und des angezeigten Texts

In dem Zusammenhang sind auch zwei neue Methoden des *Application*-Objekts erwähnenswert: *HTMLEncode* und *PlainText*. *HTMLEncode* erzeugt beispielsweise folgende Ausgabe:

```
Debug.Print Application.HtmlEncode ("<p>Dies ist ein Text</p>")
&lt;p&gt;Dies ist ein Text&lt;/p&gt;
```

Den umgekehrten Weg beschreitet *PlainText*:

```
Debug.Print Application.PlainText("&lt;p&gt;Dies ist ein Text&lt;/p&gt;")
<p>Dies ist ein Text</p>
```

4.1.2 Datum auswählen

Eine weitere Neuerung, die speziell das Textfeld betrifft, ist der *DatePicker*. Diesen können Sie für Textfelder aktivieren, die an Datumsfelder gebunden sind, was auch in der Datenblattansicht funktioniert.

Die passende Eigenschaft heißt *Datumsauswahl* (VBA: *ShowDatePicker*) und akzeptiert die beiden Werte *Für Datumsangaben* oder *Nie*. Abbildung 4.2 zeigt das beim Eintritt in ein Datumsfeld mit aktiviertem *DatePicker* angezeigte Symbol zum Öffnen des *DatePickers*. Mit einem Klick auf diese Schaltfläche öffnen Sie den *DatePicker* (siehe Abbildung 4.3).

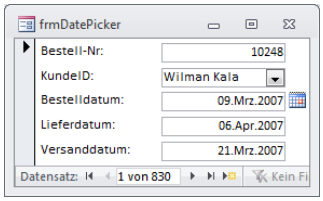


Abbildung 4.2: Die Schaltfläche zum Öffnen des *DatePickers*

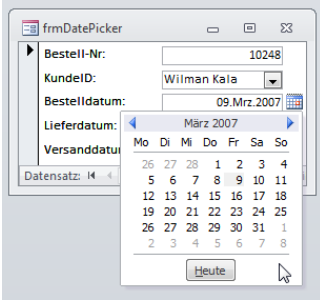


Abbildung 4.3: Der DatePicker in Aktion

Der *DatePicker* hat einen entscheidenden Nachteil: Er bietet keine Möglichkeit zur direkten Auswahl des Monats und des Jahres. Wenn Sie Daten eingeben möchten, die weit in der Vergangenheit liegen, bemühen Sie am besten doch die Tastatur oder eine alternative Lösung eines Drittanbieters. Per VBA zeigen Sie den *DatePicker* so an:

```
RunCommand acCmdShowDatePicker
```

4.1.3 Texte als Hyperlink anzeigen

Mit der Eigenschaft *Als Hyperlink anzeigen* können Sie festlegen, dass auch Daten, die kein Hyperlink sind, als Hyperlink angezeigt werden (siehe Abbildung 4.4). Die VBA-Variante dieser Eigenschaft heißt *DisplayAsHyperlink*.

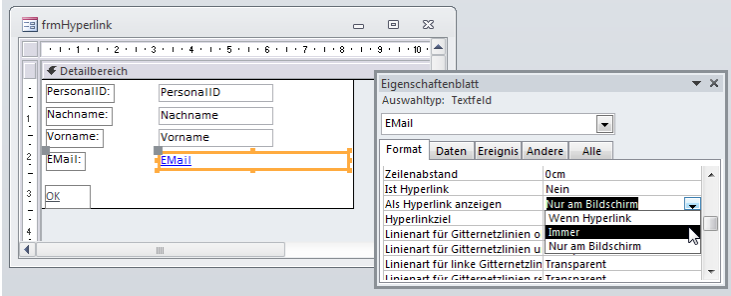


Abbildung 4.4: Anzeigen von Daten als Hyperlink

4.1.4 Abgeschnittene Zahlenfelder

Nicht direkt eine Steuerelementeigenschaft, aber doch in diesem Zusammenhang interessant ist die Eigenschaft *Aktuelle Datenbank|Anwendungsoptionen|Auf abgeschnittene Zahlenfelder prüfen* in den Access-Optionen. Damit legen Sie fest, ob Zahlenwerte, die zu lang für ihr Textfeld sind, wie in Excel durch eine Reihe Rauten (#) ersetzt werden.

4.2 Schaltflächen

Wie Sie Schaltflächen anlegen, diese mit Ereignissen versehen und mehr erfahren Sie in der Onlinehilfe im Artikel »Verwenden einer Befehlsschaltfläche zum Starten einer Aktion oder einer Reihe von Aktionen«, hier vor allem in den hinteren Abschnitten. Neu ist bei den Schaltflächen vor allem das lang ersehnte gleichzeitige Anzeigen eines Symbols und einer Beschriftung. Damit braucht man zu diesem Zweck nicht mehr das etwas unkomfortable *CommandButton*-Steuerelement der Bibliothek *Microsoft Forms 2.0* einzusetzen.

Ein Icon fügen Sie ganz einfach über die Eigenschaft *Bild* hinzu – dort öffnet sich ein Dialog, mit dem Sie eines der eingebauten oder ein benutzerdefiniertes Icon auswählen können. Sie können auch andere Dateiformate als die vom Dialog angezeigten *.bmp* und *.ico* verwenden – nämlich alle von GDI+ verarbeitbaren Formate, also *.jpg*, *.png*, *.gif*, *.tif*, *.emf* und *.wmf*. Abbildung 4.5 zeigt drei Beispiele für die Gestaltungsmöglichkeiten mit den neuen Schaltflächeneigenschaften – Ihrer Fantasie sind hier aber (fast) keine Grenzen gesetzt.

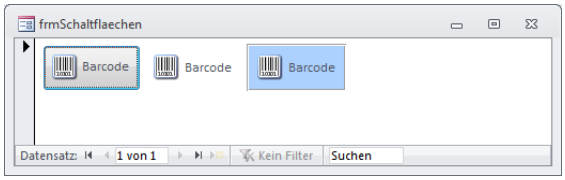


Abbildung 4.5: Verschiedene Schaltflächen-Designs

Ein einmal für eine Schaltfläche ausgewähltes Bild wird unter Access 2010 automatisch der Bildergalerie (*Entwurf|Bild einfügen*) hinzugefügt. Über dieses Ribbon-Element kann dasselbe Bild dann später komfortabel immer wieder in Schaltflächen oder Bildsteuerelemente eingefügt werden, ohne das der Dateiauswahldialog erneut bemüht werden müsste. Außerdem bieten Schaltflächen folgende neue Möglichkeiten:

» Wahlweise normaler oder transparenter Hintergrund (Eigenschaft *Hintergrundart*, VBA: *BackStyle*, Einstellungen: *Normal* (*1*), *Transparent* (*2*)). Bei transparentem Hintergrund können Sie beispielsweise ein andersfarbiges Rechteck hinter die Schaltfläche legen, um eine alternative Hintergrundfarbe zu erzeugen.

» Beschriftung zum Bild anordnen (Eigenschaft *Anordnung der Bildbeschriftung*, VBA: *PictureCaptionArrangement*, Werte: *acNoPictureCaption (0)*, *acGeneral (1)*, *acTop (2)*, *acBottom (3)*, *acLeft (4)*, *acRight (5)*)

» Bild/Beschriftung ausrichten (Eigenschaft *Ausrichtung*, VBA: *Alignment*, Werte: *Allgemeine Ausrichtung* (*0*), *links* (*1*), *zentriert* (*2*, Standardwert), *rechts* (*3*), *verteilt* (*4*))

» Beim Überfahren einer Schaltfläche kann der Mauszeiger in eine Hand umgewandelt werden (Eigenschaft *Cursor beim Bewegen*, VBA: *CursorOnHover*, Werte: *acCursorOnHoverDefault (0)*, *acCursorOnHoverHyperlinkHand (1)*)

Designs

Für Schaltflächen und wenige andere Steuerelemente gibt es mit Access 2010 noch weitere Möglichkeiten zur Anpassung des Designs. Mehr dazu lesen Sie unter »Effekte für Schaltflächen, Umschaltflächen, Registersteuerelement und Navigationsschaltfläche« ab Seite 297.

4.3 Kombinationsfelder

Den Inhalt von Kombinationsfeldern können Sie ab jetzt per eingebautem Dialog editieren, wenn folgende Bedingungen erfüllt sind: Erstens muss das Kombinationsfeld ungebunden sein, zweitens darf die Wertliste nur eine Spalte haben und drittens muss die Eigenschaft *Wertlistenbearbeitung zulassen* (*AllowValueListEdits*) den Wert *Ja* haben. Für ein Standardkombinationsfeld müssen Sie dazu nur die Eigenschaft *Herkunftsart* (*RowSourceType*) auf *Wertliste* einstellen. Abbildung 4.6 zeigt die Schaltfläche, die beim Ausklappen des Kombinationsfeldes erscheint.

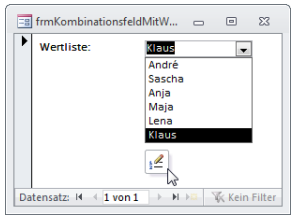


Abbildung 4.6: Ein (noch) leeres Kombinationsfeld

Mit dieser Schaltfläche öffnen Sie den Dialog aus Abbildung 4.7. Dort können Sie Einträge hinzufügen (jeden in eine Zeile) und einen Standardwert festlegen. Damit ändern Sie allerdings eine Eigenschaft und damit auch den Entwurf des Formulars – was bedeutet, dass Sie beim Schließen das Speichern des Formulars bestätigen müssen. Sie können dies umgehen, indem Sie das Formular mit einer eigenen Schaltfläche schließen, deren *Beim Klicken*-Ereignisprozedur die folgende Zeile enthält:

```
Docmd.Close acForm, Me.Name, acSaveYes
```

Das Erweitern einer Wertliste von Kombinationsfeldern und Listenfeldern ist somit in einer *.accde*- oder Runtime-Version der Datenbank nicht verfügbar; hier müssen Sie eine eigene Lösung programmieren.

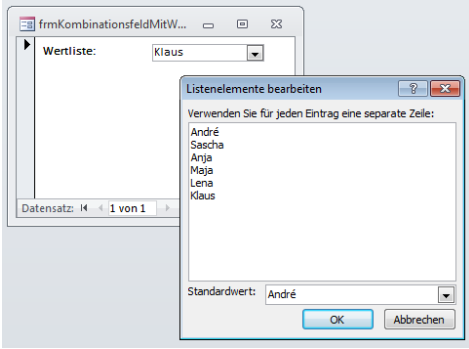


Abbildung 4.7: Bearbeiten der Wertliste eines Formulars

4.3.1 Wertliste erben

Wenn Sie Verknüpfungsfelder mit einem Nachschlagefeld ausstatten, das die schnelle Auswahl der Daten ermöglicht (etwa zur Auswahl der Anrede aus einer Wertliste), übernimmt Access dies beim Einfügen eines solchen Feldes aus der Feldliste in ein Formular.

Mit der Eigenschaft *Wertliste erben* (*InheritValueList*) können Sie festlegen, ob Änderungen, die Sie im Kombinationsfeld an der Wertliste vornehmen, in die Auswahldaten des Feldes übertragen werden. Gleichzeitig muss natürlich die Eigenschaft *Wertlistenbearbeitung zulassen* aktiviert sein.

4.3.2 Formular zum Bearbeiten anzeigen

Für die Eigenschaft *Bearbeitungsformular für Listenelemente* können Sie ein Formular angeben, mit dem man die Datensatzherkunft des Kombinationsfelds bearbeiten kann. Access öffnet das Formular dann auf die gleiche Weise wie den Dialog zum Eingeben von Wertlisteneinträgen.

Leider erkennt Access nicht automatisch, welcher Datensatz gerade im Listenfeld angezeigt wird, und öffnet das Bearbeitungsformular ungefiltert. Nach dem Bearbeiten aktualisiert Access die Datensatzherkunft des Kombinationsfeldes.

Die VBA-Bezeichnung für diese Eigenschaft ist *ListItemsEditForm*.

4.3.3 Wachsen und Schrumpfen

Sie können für Kombinationsfelder in Berichten die Eigenschaften *Vergrößerbar* und *Verkleinerbar* einstellen. Die Eigenschaft wirkt genau wie bei Textfeldern. VBA-Bezeichnungen: *CanGrow* und *CanShrink*.

4.3.4 Hyperlinks

Genau wie bei Textfeldern können Sie auch für Kombinationsfelder entscheiden, ob auch Inhalte, die kein Hyperlink sind, als Hyperlink formatiert werden sollen (*Als Hyperlink anzeigen/DisplayAsHyperlink*).

4.3.5 Mehrwertige Felder

Kombinations- und Listenfelder liefern einige Eigenschaften, die Sie in Zusammenhang mit mehrwertigen Feldern einsetzen können: *ItemsSelected*, *Selected* und *SeparatorCharacters*. Die Technik scheint noch unausgegoren; wenn Sie möchten, experimentieren Sie selbst mit diesen Eigenschaften.

Empfehlung: Verwenden Sie die bekannten Techniken zur Anzeige von Daten in m:n-Beziehungen – speichern Sie alle Daten in Tabellen, verknüpfen Sie diese ebenfalls über eine Tabelle und setzen Sie dann die im Kapitel »Formulare« ab Seite 161 vorgestellten Techniken ein.

4.4 Kombinationsfeld-Techniken

In den folgenden Abschnitten finden Sie einige oft benötigte Techniken für den Einsatz von Kombinationsfeldern.

4.4.1 Kombinationsfeld aufklappen

Wenn Sie ein Kombinationsfeld beim Öffnen des Formulars oder zu einer anderen Gelegenheit per VBA aufklappen möchten, verwenden Sie die *DropDown*-Methode des Kombinationsfeldes (siehe Abbildung 4.8).

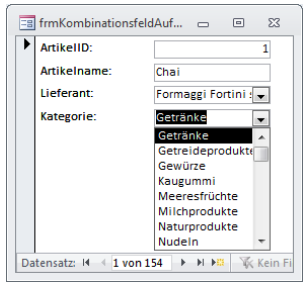


Abbildung 4.8: Automatisch aufgeklapptes Kombinationsfeld

Vorher müssen Sie diesem Steuerelement noch den Fokus zuweisen:

```
Private Sub Form_Current()
    Me!KategorieID.SetFocus
    Me!KategorieID.Dropdown
End Sub
```

Listing 4.2: Aufklappen eines Kombinationsfeldes

4.4.2 Auswählen-Eintrag hinzufügen

Leere Kombinationsfelder sehen nicht schön aus. Mit der folgenden Datensatzherkunft können Sie den enthaltenen Positionen den Eintrag *<Auswählen>* voranstellen:

```
SELECT 0, '<Auswählen>' AS Firma FROM tblLieferanten
UNION
SELECT LieferantID, Firma FROM tblLieferanten ORDER BY Firma;
```

Nun müssen Sie noch dafür sorgen, dass der künstlich hinzugefügte Eintrag automatisch ausgewählt wird, wenn das Feld noch keinen Wert hat. Und das geht ganz einfach: Stellen Sie einfach den Standardwert des zugrunde liegenden Feldes in der Tabelle oder des Steuerelements auf *0* ein (Abbildung 4.9).

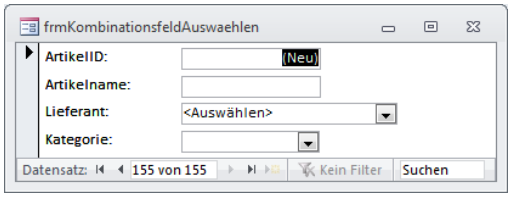


Abbildung 4.9: Bei neuen Datensätzen zeigt das Kombinationsfeld einen passenden Eintrag an

4.4.3 Abhängige Kombinationsfelder

Früher oder später brauchen Sie einmal zwei Kombinationsfelder, bei denen der Wert des zweiten vom aktuellen Wert des ersten Kombinationsfeldes abhängt. So geht's:

» Erstellen Sie zwei Kombinationsfelder auf Basis der Tabellen *tblKategorien* und *tblArtikel* der Beispieldatenbank, die jeweils an die ID gebunden sind und die Kategorie beziehungsweise den Artikel anzeigen (siehe Abbildung 4.10).

» Legen Sie die folgenden beiden Ereignisprozeduren sowie die Prozedur *ArtikelAktualisieren* an.

```
Private Sub cboKategorien_AfterUpdate()
    ArtikelAktualisieren
End Sub
```

```
Private Sub Form_Load()
    ArtikelAktualisieren
End Sub

Private Sub ArtikelAktualisieren()
    Me!cboArtikel.RowSource = "SELECT ArtikelID, Artikelname " _
        & "FROM tblArtikel WHERE KategorieID = " & Nz(Me!cboKategorien,0)
End Sub
```

Listing 4.3: Diese Routinen benötigen Sie für abhängige Kombinationsfelder

Die beiden Ereignisprozeduren sorgen beim Öffnen sowie nach dem Aktualisieren des Kategorien-Kombinationsfelds für das erneute Abfragen der passenden Artikel.

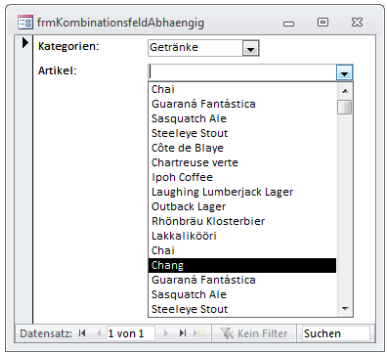


Abbildung 4.10: Abhängige Kombinationsfelder

4.4.4 Bestimmten Eintrag auswählen

Gerade beim vorherigen Beispiel ist es sinnvoll, beim Öffnen einen Eintrag aus dem Listenfeld auszuwählen. Dazu verwenden Sie die Eigenschaft *ItemData*. Passen Sie die folgenden zwei der im vorherigen Beispiel vorgestellten Routinen an: Das Formular aus Abbildung 4.11 zeigt die beim Laden ausgewählten Einträge an. Wichtig ist, dass Sie die Datensatzherkunft des zweiten, abhängigen Kombinationsfeldes zunächst aktualisieren und dann erst den ersten Eintrag auswählen.

```
Private Sub Form_Load()
    Me!cboKategorien = Me!cboKategorien.ItemData(0)
    ArtikelAktualisieren
End Sub

Private Sub ArtikelAktualisieren()
    Me!cboArtikel.RowSource = "SELECT ArtikelID, Artikelname " _
```

```
        & "FROM tblArtikel WHERE KategorieID = " & Me!cboKategorien
    Me!cboArtikel = Me!cboArtikel.ItemData(0)
End Sub
```

Listing 4.4: Diese Routinen sorgen für die Vorauswahl des jeweils ersten Eintrags eines Kombinationsfelds

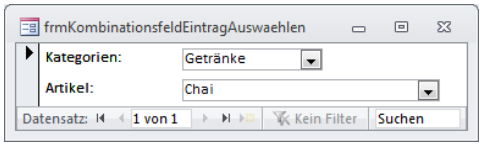


Abbildung 4.11: Formular mit zuvor ausgewählten Datensätzen

4.4.5 Aktuell markierten Eintrag auslesen

Beim Auslesen von Werten aus Kombinationsfeldern besteht das Problem meist im Zugriff auf die einzelnen Spalten.

Hier enthält die erste Spalte die ID und wird nicht angezeigt, die zweite Spalte schließlich liefert den angezeigten Wert (siehe Abbildung 4.12).

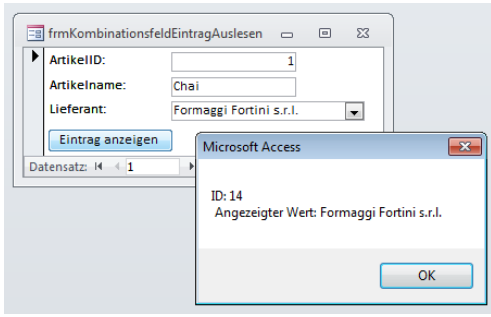


Abbildung 4.12: Auslesen des aktuell markierten Eintrags

Mit der folgenden Ereignisprozedur zeigen Sie per Mausklick die erste und die zweite Spalte des aktuell ausgewählten Eintrags über die *Column*-Eigenschaft des Kombinationsfeldes an:

```
Private Sub cmdEintragAnzeigen_Click()
    MsgBox "ID: " & Me.LieferantID & vbCrLf & " Angezeigter Wert: " _
        & Me.LieferantID.Column(1)
End Sub
```

Listing 4.5: Anzeigen verschiedener Spalten des aktuellen Eintrags eines Kombinationsfelds

4.4.6 Wert zu einem gebundenen Kombinationsfeld hinzufügen

Wenn Sie dem Benutzer erlauben wollen, über ein Kombinationsfeld Werte einzutippen, die dann in der zugrunde liegenden Datenbank gespeichert werden, legen Sie die folgende Prozedur für das Ereignis *Bei Nicht in Liste* an:

```
Private Sub cboKategorien_NotInList(NewData As String, Response As Integer)
    Dim db As DAO.Database
    Dim rst As DAO.Recordset2
    Set db = CurrentDb
    Set rst = db.OpenRecordset("tblKategorien", dbOpenDynaset)
    rst.AddNew
    rst!Kategoriename = NewData
    rst.Update
    Response = acDataErrContinue
    rst.Close
    Set rst = Nothing
    Set db = Nothing
End Sub
```

Listing 4.6: Hinzufügen eines Tabelleneintrags per Kombinationsfeld

Das Ganze können Sie natürlich auch noch ausbauen – etwa, indem Sie eine Rückfrage einbauen, die den Benutzer vor dem Speichern fragt, ob er tatsächlich einen neuen Datensatz anlegen möchte. Außerdem fehlt natürlich noch eine Fehlerbehandlung.

4.4.7 Weitere Techniken

Eine weitere Technik in Zusammenhang mit Kombinationsfeldern finden Sie unter »Ereignisse von Steuerelementen« ab Seite 191. Dort erfahren Sie, wie Sie die von einem Kombinationsfeld angezeigten Daten mit Hilfe eines Formulars bearbeiten können. In der Onlinehilfe finden Sie im Übrigen noch Hinweise zu den interessanten Funktionen zum Hinzufügen und Entfernen von Listeneinträgen: *AddItem* und *RemoveItem.*

4.5 Listenfelder

Listenfelder bieten wesentlich weniger neue Features als Kombinationsfelder – und gar keine, die Sie nicht schon vom Kombinationsfeld her kennen. Also stellen die folgenden Abschnitte einige bewährte Listenfeld-Techniken vor. Dabei ist anzumerken, dass die meisten in Zusammenhang mit Kombinationsfeldern genannten Techniken auch für

Listenfelder gelten (also auf verschiedene Spalten des aktuellen Eintrags zugreifen, abhängige Listenfelder erstellen, einen bestimmten Eintrag festlegen und so weiter).

4.5.1 Mehrfachauswahl auslesen

Im Gegensatz zu Kombinationsfeldern (mit Ausnahme dieser neumodischen mehrwertigen Felder natürlich ...) können Sie mit Listenfeldern einen oder mehrere Einträge auswählen. Dazu stellen Sie die Eigenschaft *Mehrfachauswahl* entweder auf *Einzeln* oder *Erweitert* ein. Die ausgewählten Einträge gibt eine Routine mit identischem Aufbau für beide Varianten aus (siehe Abbildung 4.13):

```
Private Sub cmdAusgabeMehrfachauswahlEinzeln_Click()
    Dim var As Variant
    Dim str As String
    For Each var In Me!lstMehrfachauswahlEinfach.ItemsSelected
        str = str & var & vbCrLf
    Next var
    MsgBox str
End Sub
```

Listing 4.7: Auslesen eines Listenfeldes mit Mehrfachauswahl

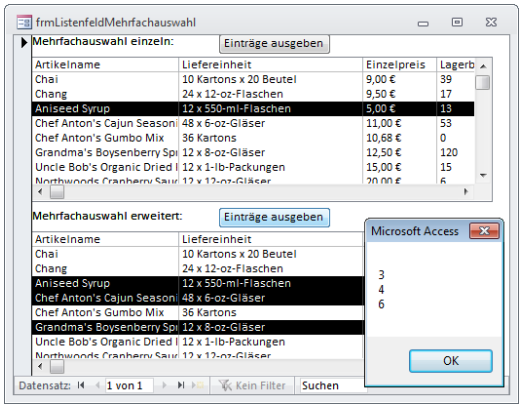


Abbildung 4.13: Mehrfachauswahl einfach und erweitert auslesen

Wenn Sie nicht nur die erste, sondern auch die folgenden Spalten lesen möchten, greifen Sie über die *Column*-Eigenschaft auf die weiteren Spalten zu. Wenn Sie die Eigenschaft *Mehrfachauswahl* auf *Keine* einstellen, können Sie einfach über die *Column*-Eigenschaft auf den ausgewählten Wert zugreifen. Achten Sie darauf, dass kein Wert ausgewählt sein könnte, und prüfen Sie den Wert des Listenfeldes zuvor mit *IsNull* auf einen *Null*-Wert.

4.5.2 Ja/Nein-Felder im Listenfeld anzeigen

Ja/Nein-Felder sehen im Listenfeld immer etwas daneben aus: Immerhin ist die passende Spalte mit den Werten *0* und *-1* gefüllt. Um dies zu ändern, passen Sie die entsprechend an, und zwar so, dass das Ja/Nein-Feld in eine *IIf*-Funktion gepackt wird, die für den Wert *True* die Zeichenfolge *x* und für den Wert *False* eine leere Zeichenfolge ausgibt (siehe Abbildung 4.14):

```
SELECT tblArtikel.ArtikelID, tblArtikel.Artikelname, IIf([Auslaufartikel],
"x","") AS [Läuft aus] FROM tblArtikel;
```

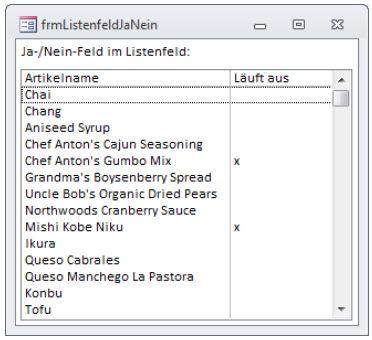


Abbildung 4.14: Hübsches Ja/Nein-Feld im Listenfeld

4.5.3 Weitere Techniken

Weitere Listenfeldtechniken finden Sie unter »Daten in der Übersicht als Listenfeld« ab Seite 207, »1:n-Beziehung per Listenfeld« ab Seite 216 und »m:n-Beziehungen per Listenfeld« ab Seite 223.

4.6 Unterformulare

Neu in Access 2010 ist, dass Sie in Unterformularen auch Berichte anzeigen können. Diese erscheinen allerdings in der Berichtsansicht, also nicht etwa in der mit dem Ausdruck übereinstimmenden Vorschauansicht. Das bedeutet aber auch, dass Sie die interaktiven Möglichkeiten der Berichtsansicht nutzen können.

Ein wichtiger Einsatzzweck ist die Darstellung von Bereichen unterschiedlicher Größe. Wenn Sie in einem Unterformularsteuerelement ein Formular anzeigen, das etwa Rechnungspositionen in der Endlosansicht anzeigt, dann ist die Höhe des Detailbereichs immer gleich groß – auch wenn der Text der Rechnungsposition einmal länger ist und nicht komplett angezeigt werden kann.

Ein Bericht in der Berichtsansicht bietet Einstellungen, mit denen die Höhe des Bereichs an den Inhalt der Textfelder angepasst wird.

Berichte binden Sie genau wie Formulare in ein anderes Formular ein. Interessant wird es, wenn Sie den Bericht oder die enthaltenen Steuerelemente referenzieren möchten.

Auf das in einem Unterformularsteuerelement enthaltene Formular greifen Sie etwa wie folgt zu:

```
Debug.Print Forms!frmHauptformular!sfmUnterformularsteuerelement.Form.Name
```

Statt auf die Eigenschaft *Name* können Sie hier auch etwa auf die enthaltenen Steuerelemente zugreifen. Beim Zugriff auf einen Bericht im Unterformularsteuerelement ersetzen Sie Form durch Report:

```
Debug.Print Forms!frmHauptformular!sfmUnterformularsteuerelement.Report.Name
```

Gegebenenfalls weisen Sie einem Unterformularsteuerelement dynamisch Formulare und Berichte zu. Sie können dann innerhalb des Klassenmoduls des Hauptformulars mit folgender Funktion herausfinden, ob das Unterformularsteuerelement einen Bericht enthält.

Die Funktion liefert eine der Konstanten *acForm* (*2*) oder *acReport* (*3*) beziehungsweise den entsprechenden Zahlenwert zurück. Ist das Unterformular leer, lautet das Ergebnis *-1*:

```
Public Function TypeOfSourceObject(sfm As SubForm) As AcObjectType
    On Error Resume Next
    If sfm.Form Is Nothing Then
        If sfm.Report Is Nothing Then
            TypeOfSourceObject = -1
        Else
            TypeOfSourceObject = acReport
        End If
    Else
        TypeOfSourceObject = acForm
    End If
End Function
```

Einen Beispielaufruf dieser Funktion finden Sie im Formular *frmWithSubreport*.

Leeren Hauptentwurf filtern

Seit Access 2007 bieten Unterformulare die Eigenschaft *Leeren Hauptentwurf filtern* (VBA: *FilterOnEmptyMaster*). Damit legen Sie fest, ob das Unterformular bei einem leeren Datensatz im Hauptformular alle Datensätze (Einstellung: *Nein*) oder keinen (Standard, Einstellung: *Ja*) anzeigen soll (siehe Abbildung 4.15).

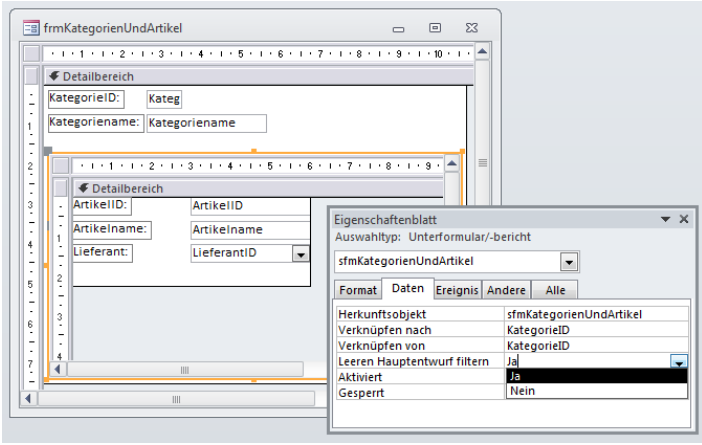


Abbildung 4.15: Die neue Eigenschaft von Unterformular-Steuerelementen

4.7 Das Anlagen-Steuerelement

Seit Access 2007 gibt es das Anlagen-Steuerelement. Die Vorstellung des Anlagen-Steuerelements befindet sich im Kapitel »Bilder und binäre Dateien« ab Seite 675. Der Grund ist, dass dieser Themenkomplex mehrere Objektarten stark ineinander verwickelt – hier gehören Tabellen, Formulare, Berichte und natürlich auch VBA-Module dazu.

4.8 Optionsgruppe, Umschaltfläche, Kontrollkästchen, Bildsteuerelement und Co.

Informationen zu diesen Steuerelementen finden Sie in der Onlinehilfe von Access. Im Vergleich zu den bereits detaillierter beschriebenen Steuerelementen werden diese weniger genutzt und bieten nicht so viele interessante Techniken.

Es gibt übrigens noch eine neue Art, ein Bildsteuerelement zum Formular hinzuzufügen: Wenn Sie im Ribbon unter *Entwurf|Steuerelemente* ganz links auf die Schaltfläche *Logo* klicken, fügt Access dem Formularkopf ein Bildsteuerelement hinzu, für das Sie direkt beim Anlegen ein Bild auswählen müssen. Dieser Vorgang lässt sich nicht wie beim Anlegen eines Bildsteuerelements abbrechen, um so ein leeres Bildsteuerelement zu erzeugen.

Statt des Bildsteuerelements sollten Sie im Übrigen besser das *Anlage*-Feld oder das *Image*-Steuerelement der *MSForms*-Bibliothek verwenden (mehr dazu im Kapitel »Bilder und binäre Dateien« ab Seite 675).

4.9 NavigationControl und NavigationButton

Das *NavigationControl*-Steuerelement erlaubt das Anlegen horizontaler und vertikaler, gegebenenfalls auch verschachtelter Leisten mit Schaltflächen zum Aufrufen verschiedener Formulare. Das Ribbon bietet im Tab *Erstellen* einige Vorlagen an (siehe Abbildung 4.16).

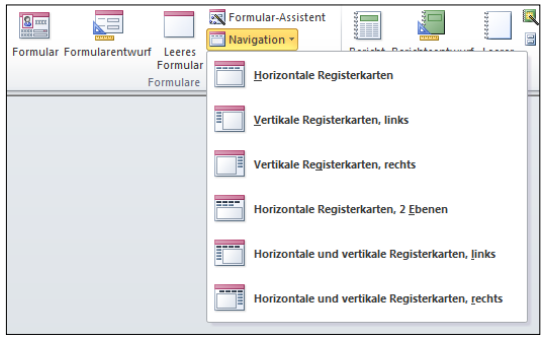


Abbildung 4.16: Hinzufügen verschiedener Varianten des NavigationControl-Steuerelements

Mit dem vierten Ribbon-Eintrag und einigen neu angelegten Schaltflächen erstellen Sie beispielsweise ein Formular wie das aus Abbildung 4.17. Dieses Formular besteht aus den folgenden Steuerelementen:

- *nc1*: Oberes Navigationssteuerelement
- *pge1*: Schaltfläche *Seite 1*
- *pge2*: Schaltfläche *Seite 2*
- *NavigationButton1*: Schaltfläche *[Neues hinzufügen]* – in der Formularansicht nicht mehr sichtbar
- *nsfm1*: Element unter den Navigationsleisten
- *nc2*: Unteres Navigationssteuerelement
- *pge1_1*: Schaltfläche *Seite 1-1*
- *pge1_2*: Schaltfläche *Seite 1-2*
- *pge2_1*: Schaltfläche *Seite 2-1*

» *pge2_2*: Schaltfläche *Seite 2-2*

» *NavigationButton7*: Schaltfläche *[Neues hinzufügen]* im unteren Navigationssteuerelement

Die Steuerelemente werden nicht ineinander verschachtelt, sondern liegen alle direkt auf dem Hauptformular.

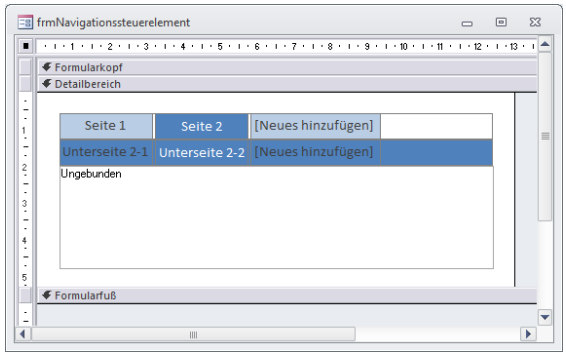


Abbildung 4.17: Navigationsformular mit zwei horizontalen Ebenen

4.9.1 Navigationssteuerelement manuell bestücken

Wenn Sie keines der Templates verwenden möchten, legen Sie ein neues, leeres Formular in der Entwurfsansicht an und fügen das *Navigationssteuerelement* über die Liste der Steuerelemente im Ribbon hinzu.

Das eigentliche Navigationssteuerelement ist der obere Teil, in dem sich der Eintrag *[Neues hinzufügen]* und der leere Bereich befinden. Mit der Eigenschaft *Bereich* legen Sie die Ausrichtung des Navigationssteuerelements fest (*Horizontal* oder *Vertikal*). Unter VBA heißt diese Eigenschaft *Span* und kann die Werte *acHorizontal* oder *acVertical* annehmen.

Navigationsziele hinzufügen

Der leere, große Bereich unter beziehungsweise neben dem Navigationssteuerelement kann ein Formular oder einen Bericht anzeigen. Sie haben richtig gelesen: Sie können innerhalb eines Formulars einen Bericht anzeigen! Allein das ist schon ein sehr sinnvoller Einsatz für ein Navigationssteuerelement.

Ein Formular oder einen Bericht fügen Sie auf die folgenden Arten hinzu:

» Markieren Sie die Navigationsschaltfläche und tragen Sie als Titel den Namen des Formulars oder Berichts ein. Access fügt das entsprechende Element automatisch

hinzu. Den Namen können Sie anschließend wieder ändern, das Navigationsziel bleibt erhalten.

» Wählen Sie das Navigationsziel für die Eigenschaft *Name des Navigationsziels* aus (siehe Abbildung 4.18).

» In der Layoutansicht können Sie das Webformular sogar direkt aus dem Navigationsbereich auf die neue Navigationsschaltfläche ziehen.

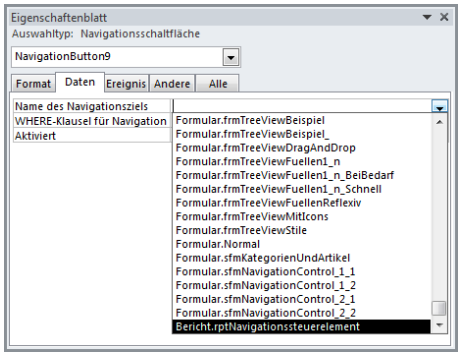


Abbildung 4.18: Hinzufügen eines Navigationsziels

4.9.2 Aufbau des Navigationssteuerelements

Das Navigationssteuerelement als Gesamtpaket besteht aus einem *NavigationControl*, das wiederum ein oder mehrere *NavigationButton*-Steuerelemente enthält. Dazu gehört außerdem ein Unterformular-Steuerelement, das die in der Eigenschaft *Name des Navigationsziels* der einzelnen *NavigationButton*-Elemente angegebenen Formulare anzeigt.

Wenn Sie ein *NavigationControl* zum Formular hinzufügen oder eine der Formularvorlagen mit Navigationssteuerelement verwenden, steht die Zuordnung der einzelnen Steuerelemente bereits fest.

Wenn die durch die Navigationsschaltflächen angewählten Formulare und Berichte in einem Unterformular-Steuerelement angezeigt werden, woher weiß das *NavigationControl* dann, welches Unterformular-Steuerelement es verwenden soll? Das Eigenschaftsfenster gibt jedenfalls keinen Hinweis darauf her. Dafür aber die VBA-Eigenschaft *Subform* des *NavigationControl*-Steuerelements: Es enthält den Namen des Unterformular-Steuerelements, das die mit den Navigationsschaltflächen ausgewählten Unterformulare/-berichte anzeigt.

4.9.3 Layout oder nicht?

Das *NavigationControl*-Steuerelement und sein Unterformular werden automatisch in einem Layout angelegt. Das äußere Layout, welches das *NavigationControl* und die *Navigationbuttons* auf der einen sowie das Unterformular-Steuerelement auf der anderen Seite enthält, können Sie entfernen. Dazu markieren Sie das komplette Layout durch einen Klick auf das Verschieben-Symbol links oben (siehe Abbildung 4.19) und wählen dann den Kontextmenüeintrag *Layout|Layout entfernen* aus.

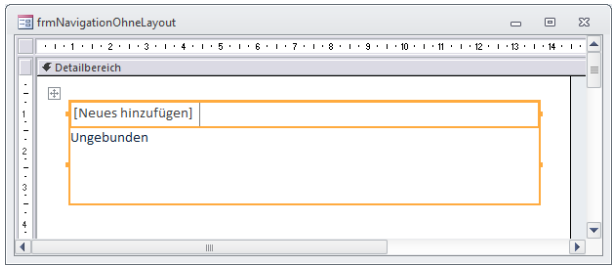


Abbildung 4.19: Markieren des Layouts eines Navigationssteuerelements

Danach können Sie das Navigationssteuerelement und das Unterformular unabhängig voneinander positionieren (siehe Abbildung 4.20).

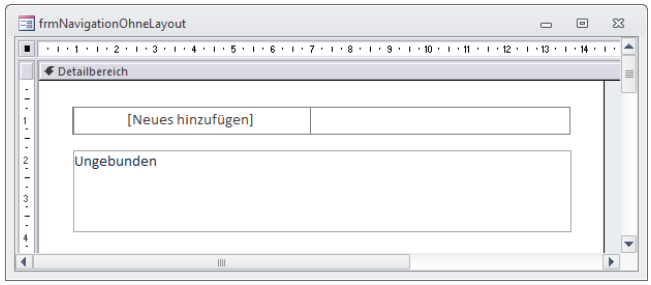


Abbildung 4.20: Elemente des Navigationssteuerelements ohne Layout

Von horizontal nach vertikal und umgekehrt

Wenn Sie ein Navigationssteuerelement zu einem Formular hinzufügen, zeigt dieses die Navigationsschaltflächen zunächst in horizontaler Anordnung an. Das Ändern der Eigenschaft *Bereich* auf den Wert *Vertikal* ändert nur die Anordnung der Schaltflächen, ordnet aber nicht das *Navigationcontrol* und das Unterformular nebeneinander an, wie es eigentlich sein sollte.

Dies können Sie auch unter Beibehaltung des Layouts wie folgt selbst durchführen:

» Stellen Sie die Eigenschaft *Bereich* von *Horizontal* auf *Vertikal* um.

» Markieren Sie die Zelle des Layouts, in der sich das Navigationssteuerelement befindet.

» Klicken Sie mit der rechten Maustaste in die Zelle und wählen Sie aus dem Kontextmenü den Eintrag *Verbinden/Teilen|Horizontal teilen* aus.

» Verschieben Sie das Unterformular-Steuerelement per Drag and Drop von unten in die neue Zelle rechts neben dem *NavigationControl*.

» Klicken Sie in die Zelle, in der sich zuvor das Unterformular-Steuerelement befand, und drücken Sie die *Entfernen*-Taste.

» Nun brauchen Sie nur noch die Höhe des Layouts anzupassen. Dies erledigen Sie, indem Sie das komplette Layout markieren und dann rechts unten anfassen und die Größe nach Wunsch einstellen (siehe Abbildung 4.21).

» Schließlich stellen Sie noch die Breite der linken Zelle des Layouts auf die Breite der Navigationsschaltflächen ein.

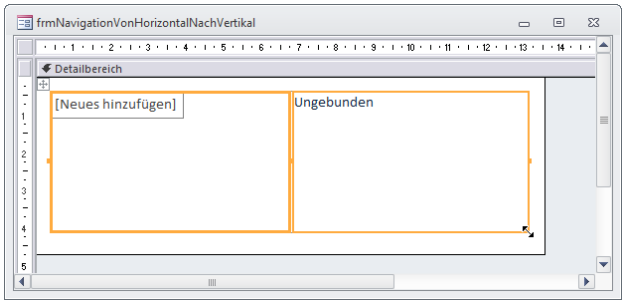


Abbildung 4.21: Anpassen der Größe eines Layouts und der enthaltenen Elemente

4.9.4 Verschachtelte Navigationselemente selbst bauen

Wie aber gestaltet man eine Navigation mit zwei oder mehr Ebenen, ohne die Vorlagen zu verwenden? Immerhin müsste man ja allen Ebenen zumindest das gleiche Unterformular-Steuerelement zuweisen und außerdem dafür sorgen, dass nach dem Anklicken einer Schaltfläche der ersten Ebene die korrespondierenden Schaltflächen der zweiten Ebene angezeigt werden. Wenn man weiß, wie es geht, gelingt dies ganz einfach:

» Legen Sie ein erstes Navigationssteuerelement in einem neuen, leeren Formular an.

» Fügen Sie dann ein zweites Navigationssteuerelement hinzu. Dieses erscheint bereits ohne Unterformular-Steuerelement, also nur mit der Navigationsleiste.

» Verschieben Sie die zuletzt hinzugefügte Navigationsleiste zwischen die erste Navigationsleiste und das Unterformular-Steuerelement – ein horizontaler Balken zeigt an, wenn Sie das Steuerelement fallen lassen können.

Schon fertig! Auf die gleiche Weise können Sie noch weitere Ebenen hinzufügen (siehe Abbildung 4.22).

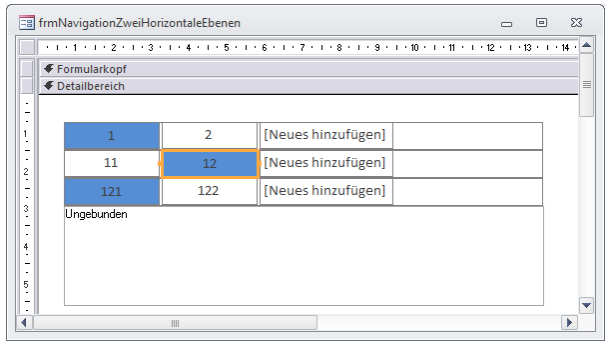


Abbildung 4.22: Navigationssteuerelement mit mehreren manuell hinzugefügten Ebenen

Die Beziehung zwischen den einzelnen Ebenen des Navigationssteuerelements fragen Sie unter VBA übrigens mit den Properties *NavigationParent* und *NavigationChild* ab.

4.9.5 Filtern des Unterformulars/-berichts

Die für ein Navigationssteuerelement verwendeten Unterformular-Steuerelemente besitzen nicht die Eigenschaften *Verknüpfen von* und *Verknüpfen nach*, Sie können also keine Filterung der Daten des Unterformulars in Abhängigkeit von den im Hauptformular angezeigten Daten vornehmen.

Dafür offeriert das *NavigationButton*-Element aber die Eigenschaft *WHERE-Klausel für Navigation* an. Hier tragen Sie nach bewährtem Schema einen Ausdruck wie *Vorname LIKE 'A*'* ein. Das Unterformular zeigt dann nur Datensätze etwa einer Tabelle namens *tblAdressen* an, deren Feld *Vorname* einen Wert mit dem Anfangsbuchstaben *A* enthält.

4.9.6 Beispiel Adressregister

Ein gutes Beispiel für den Einsatz des NavigationControls ist ein Adressregister. Es soll für jeden Buchstaben einen NavigationButton besitzen und die Adressen gefiltert nach dem jeweils ausgewählten Buchstaben anzeigen (siehe Abbildung 4.23).

Die Erstellung eines solchen Formulars sollte nach dem Studium der obigen Erläuterungen kein Problem mehr sein. Da dies jedoch ein Entwickler- und kein Einsteigerbuch ist, erledigen wir diesen mehr als mühseligen Job doch mit einer kleinen VBA-Prozedur.

Voraussetzung ist ein Formular namens *sfmAdressen*, das die Daten der Tabelle *tblMitarbeiter* der Beispieldatenbank in der Datenblattansicht anzeigt und später als Unterformular im Navigationssteuerelement eingesetzt wird.

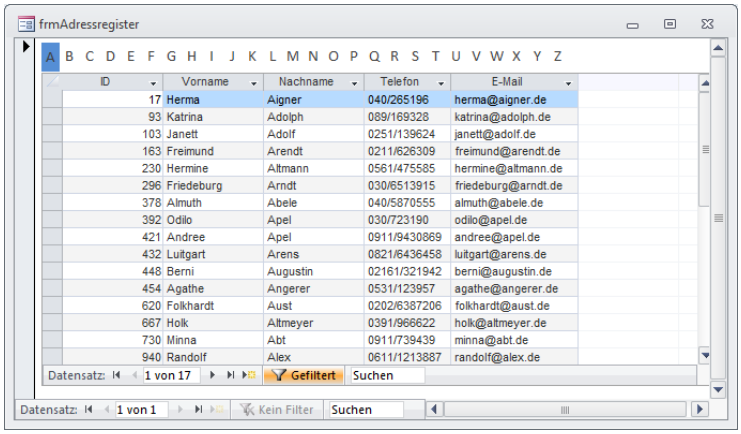


Abbildung 4.23: Adressregister mit *NavigationControl*

Die Prozedur Adressregister schließt und löscht zunächst ein eventuell bestehendes Formular namens *frmAdressregister*. Dieses erstellt sie dann neu und merkt sich den Namen des Formulars (etwa *Formular1*), um es später umbenennen zu können. Dann erstellt es ein Steuerelement des Typs *NavigationControl* und gibt ihm den Namen *navAdressregister*.

Mit dem *NavigationControl* wurde auch ein Unterformular-Steuerelement hinzugefügt, dessen Name auch in der Eigenschaft *SubForm* des *NavigationControl*-Steuerelements angegeben ist. Das Steuerelement dieses Namens erhält als neuen Namen *sfmAdressregister*. In einer *For...Next*-Schleife legt die Prozedur dann für jeden Buchstaben ein neues *NavigationButton*-Element an und gibt den Namen des *NavigationControl*-Steuerelements als *Parent* an.

Jedes dieser Steuerelemente wird mit dem jeweiligen Buchstaben als *Caption* versehen. Außerdem wird für jedes Element das Formular *sfmAdressen* als Unterformular angegeben. Der Filter wird mit *NavigationWhereClause* auf einen Ausdruck eingestellt, der das Feld *Nachname* der Datenherkunft des Unterformulars nach dem jeweiligen Anfangsbuchstaben filtert.

Schließlich schließt die Prozedur das Formular, benennt es um und öffnet es wieder, damit Sie es sich gleich ansehen können.

```
Public Sub Adressregister()
    Dim frm As Form
    Dim ctlNavi As NavigationControl
    Dim ctlButton As NavigationButton
    Dim strForm As String
    Dim i As Integer
    Dim ctl As Control
    On Error Resume Next
    DoCmd.Close acForm, "frmAdressregister"
    DoCmd.DeleteObject acForm, "frmAdressregister"
    On Error GoTo 0
    Set frm = CreateForm
    frm.Visible = False
    strForm = frm.Name
    Set ctlNavi = CreateControl(frm.Name, acNavigationControl)
    ctlNavi.Name = "navAdressregister"
    frm.Controls(ctlNavi.SubForm).Name = "sfmAdressregister"
    For i = 1 To 26
        Set ctlButton = CreateControl(frm.Name, acNavigationButton, , ctlNavi.Name)
        With ctlButton
            .Name = "navbtn_" & Chr(64 + i)
            .Caption = Chr(64 + i)
            .Width = 250
            .NavigationTargetName = "sfmAdressen"
            .NavigationWhereClause = "Nachname LIKE '" & Chr(64 + i) & "*'"
        End With
    Next i
    DoCmd.Close acForm, frm.Name, acSaveYes
    DoCmd.Rename "frmAdressregister", acForm, strForm
    DoCmd.OpenForm "frmAdressregister"
End Sub
```

Listing 4.8: Erstellen eines Adressregisters per VBA

4.10 Das WebbrowserControl-Steuerelement

Bereits in früheren Versionen von Access konnten Sie mit dem gleichnamigen ActiveX-Steuerelement Internetseiten anzeigen und auf ihre Inhalte zugreifen. Jetzt bietet Access

ein eingebautes Steuerelement, das prinzipiell die gleichen Funktionen offeriert. Im Gegensatz zur ActiveX-Variante finden Sie hier jedoch einige weitere Eigenschaften, die etwa die Bindung an ein Tabellenfeld erlauben – so zeigt das *Webbrowser*-Steuerelement beispielsweise beim Datensatzwechseln immer die im aktuellen Datensatz angegebene Internetseite an.

Zusätzlich hat Microsoft einige Ereignisprozeduren im Eigenschaftsfenster zugänglich gemacht – mit dem ActiveX-Steuerelement musste man die Ereignisprozeduren noch über den VBA-Editor anlegen.

Die meisten der folgenden Beispiele verwenden die Tabelle *tblURLs* als Datenherkunft. Diese Tabelle ist wie in Abbildung 4.24 aufgebaut und enthält die Felder *URLID* (Primärschlüsselfeld, Autowert) und *URL* (Text).

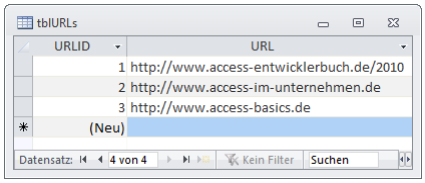


Abbildung 4.24: Datenherkunft für das Webbrowser-Steuerelement

Legen Sie ein neues Formular an und stellen die Datenherkunft des Formulars auf die Tabelle *tblURLs* ein. Fügen Sie das Feld *URL* zum Detailbereich des Formulars hinzu – dieses Textfeld verwenden Sie, um die *URL* während der Anzeige der entsprechenden Internetseite im *Webbrowser*-Steuerelement auszugeben und zu ändern.

Fügen Sie dann mit den folgenden Schritten ein an das Feld *URL* gebundenes *Webbrowser*-Steuerelement hinzu:

» Klicken Sie auf die Schaltfläche für das *Webbrowser*-Steuerelement in der Liste der Steuerelemente im Ribbon (*Entwurf|Steuerelemente*).

» Ziehen Sie dann erneut das Feld *URL* aus der Feldliste in das Formular.

» Ändern Sie den Namen des Steuerelements in *ctlWebbrowser*.

Fertig – der Entwurf sollte nun wie in Abbildung 4.25 aussehen.

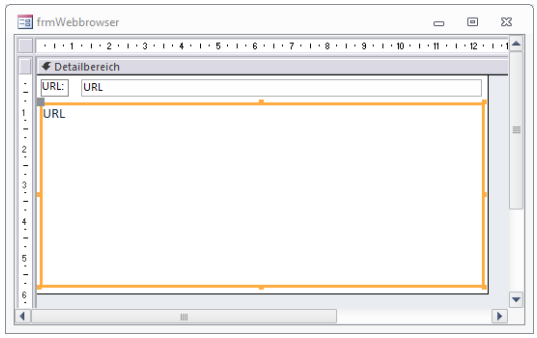


Abbildung 4.25: Das Webbrowser-Steuerelement in der Entwurfsansicht

Nach einem Wechsel in die Formularansicht erscheint bereits die erste Seite, sofern die Tabelle *tblURLs* mindestens einen Datensatz enthält (siehe Abbildung 4.26). Das Textfeld auf Basis des Felds *URL* ist doppelt nützlich: Sie können sowohl erkennen, welche Seite das Webbrowser-Steuerelement gerade anzeigt als auch eine andere URL eingeben. Nach dem Aktualisieren des Textfelds wird auch die angezeigte Webseite aktualisiert.

Abbildung 4.26: Eine Internetseite im Webbrowser-Steuerelement

Dummerweise versucht das Webbrowser-Steuerelement auch eine Seite zu laden, wenn das Feld *URL* keinen Wert enthält. Das Resultat ist eine Meldung wie in Abbildung 4.27.

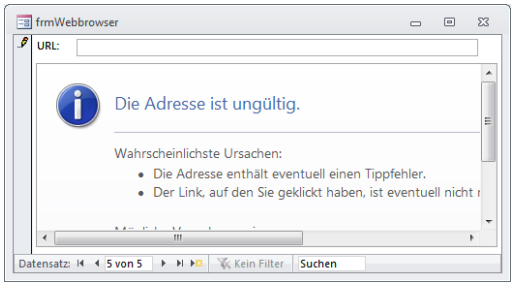


Abbildung 4.27: Anzeige beim Eingeben einer leeren Zeichenkette

Damit dies nicht geschieht, können Sie als Standardwert dieses Feldes in der Tabelle *tblURLS* den Wert *about:blank* angeben. Das Webbrowser-Steuerelement zeigt dann eine weiße Seite an.

Die aktuell angezeigte Webadresse ermitteln Sie mit der Eigenschaft *LocationURL*. Die liefert die komplette URL, während die *Value*-Eigenschaft, also der Wert des Feldes *URL* in der Tabelle *tblURLs*, statt *http://www.xyz.de* auch die Werte *www.xyz.de* oder *xyz.de* enthalten kann.

4.10.1 Fortschritt verfolgen

Das Webbrowser-Steuerelement bietet ein Ereignis und eine Eigenschaft, mit der Sie den Fortschritt beim Laden einer Seite gut nachvollziehen können. Beim Anlegen der Ereignisprozedur werden sich Leser, die bislang mit dem ActiveX-Webbrowser gearbeitet haben, vermutlich freuen, dass Microsoft die Ereignisse nun im Eigenschaftsfenster verfügbar macht (siehe Abbildung 4.28).

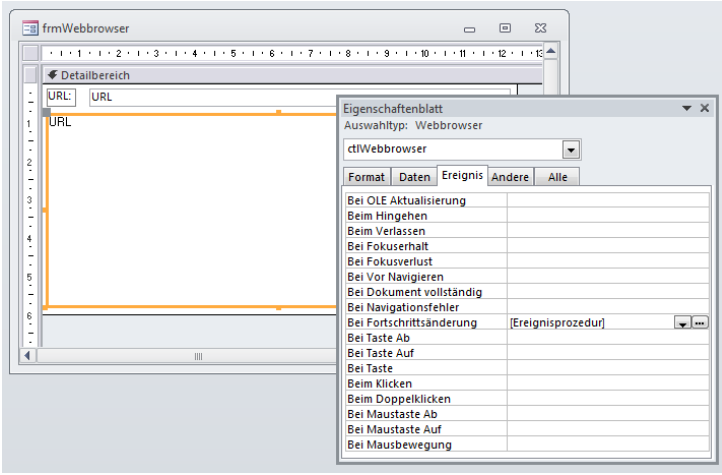


Abbildung 4.28: Anlegen einer Ereignisprozedur für das Webbrowser-Steuerelement

Legen Sie dort eine Ereignisprozedur für das Ereignis *Bei Fortschrittsänderung* an und füllen Sie diese wie folgt:

```
Private Sub ctlWebbrowser_ProgressChange(ByVal Progress As Long,
        ByVal ProgressMax As Long)
    Debug.Print Progress, ProgressMax, ctlWebbrowser.ReadyState
End Sub
```

Diese Prozedur gibt zunächst die Werte der beiden Parameter *Progress* und *ProgressMax* im Direktfenster aus. Zusätzlich lassen wir noch den Wert der Eigenschaft *ReadyState* des Steuerelements ausgeben. Die Werte dieser Eigenschaften entsprechen den folgenden Konstanten:

» *0*: *acUnintialized*

» *1*: *acLoading*

» *2*: *acLoaded*

» *3*: *acInteractive*

» *4*: *acComplete*

Das Ergebnis sieht beim Laden einer einfachen Internetseite noch einfach aus (siehe Abbildung 4.29).

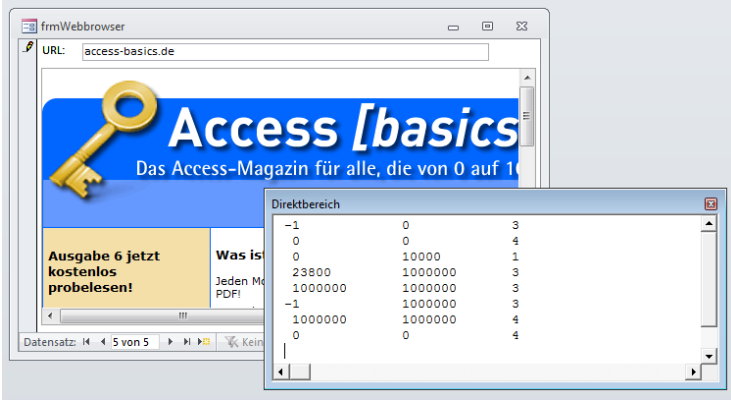


Abbildung 4.29: Fortschritt und Status beim Laden einer einfachen Internetseite

Bei umfangreicheren Internetseiten liefert diese Prozedur jedoch recht undurchsichtige Ergebnisse, deren Interpretation wohl den Aufwand nicht lohnt – zumindest dann nicht, wenn es schlicht um die Fortschrittsanzeige beim Laden einer Internetseite im Zeitalter von DSL geht.

4.10.2 Seite fertig geladen?

Dennoch wäre es hilfreich zu erfahren, wann die gewünschte Seite denn nun fertig geladen ist. Ein Grund, diese Information zu erhalten, wäre etwa eine Prozedur, die Informationen aus dieser Seite auslesen soll. Und das ist erst sinnvoll, wenn diese auch vollständig vorliegt.

Die Ereigniseigenschaft *Bei Dokument vollständig* scheint ein Ansatz zu sein. Legen Sie also hierfür eine entsprechende Ereignisprozedur an und füllen Sie diese wie folgt:

```
Private Sub ctlWebbrowser_DocumentComplete(ByVal pDisp As Object, URL As Variant)
    Debug.Print "Complete: " & URL
End Sub
```

Experimente mit einfachen Seiten sind viel versprechend: Das Steuerelement lädt offensichtlich jeweils zunächst die Adresse *about:blank*, um den Browser zu leeren und dann die gewünschte Seite. Nach Fertigstellung erscheint im Direktfenster die entsprechende Meldung (siehe Abbildung 4.30).

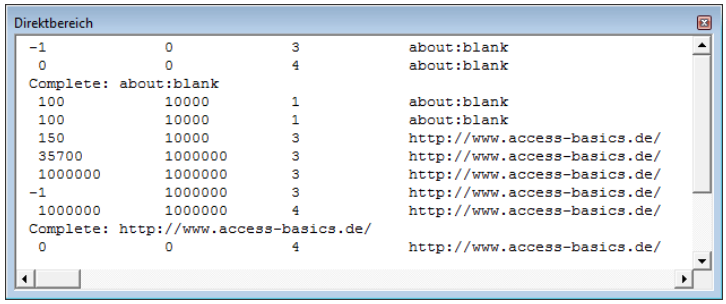


Abbildung 4.30: Laden einer einfachen Seite mit Vollzugsmeldung

Wenn Sie das mal mit einer aufwendigeren Seite wie etwa amazon.de probieren, quillt das Direktfenster schnell über: Da meldet das *Webbrowser*-Steuerelement nämlich auch für jedes Javascript und jedes IFrame das Ende des Ladevorgangs. Wie soll man da erkennen, dass die Seite nun komplett geladen ist?

Es gibt eine Vorgehensweise, mit der Sie zumindest sicherstellen können, dass die Hauptseite unabhängig von darin enthaltenen IFrames geladen ist. Dies funktioniert durch die folgende Erweiterung der obigen Prozedur (Details siehe *http://support.microsoft.com/kb/180366/en-us*):

```
Private Sub ctlWebbrowser_DocumentComplete(ByVal pDisp As Object, URL As Variant)
    Debug.Print "Complete: " & URL
    If pDisp Is ctlWebbrowser.Object And Not URL = "about:blank" Then
        Debug.Print "Komplett fertig!"
    End If
End Sub
```

Statt der *Debug.Print*-Anweisung mit der Meldung *Komplett fertig!* bringen Sie den Code unter, der sich mit der aktuellen Internetseite beschäftigt.

4.10.3 Seitenfehler

Wenn die Seite nicht erfolgreich geladen werden konnte, erscheint üblicherweise eine entsprechende Fehlermeldung im Webbrowser. Solche Fehler fangen Sie mit der Ereignisprozedur ab, die durch das Ereignis *Bei Navigationsfehler* ausgelöst wird.

Wie bei den vorherigen Beispielen soll im Falle eines Fehlers einfach eine Meldung im Direktfenster ausgeben werden. Die Meldung soll die fehlerhafte URL und den Statuscode des Fehlers enthalten:

```
Private Sub ctlWebbrowser_NavigateError(ByVal pDisp As Object, URL As Variant, _
        TargetFrameName As Variant, StatusCode As Variant, Cancel As Boolean)
```

```
    Debug.Print "Fehler:", URL, StatusCode
End Sub
```

Dies sieht beispielsweise wie in Abbildung 4.31 aus.

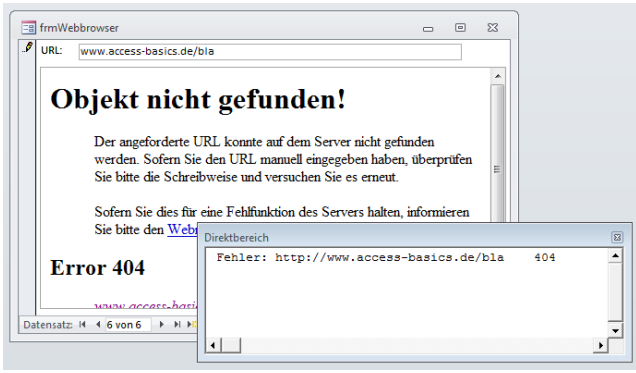


Abbildung 4.31: Fehler beim Laden einer Internetseite

Das Einstellen des Parameters *Cancel* sorgt indes nicht dafür, dass die Fehlermeldung unterbleibt.

4.10.4 Webseiten im ungebundenen Webbrowser-Steuerelement

Wenn das Webbrowser-Steuerelement im Formular einfach nur die in ein Textfeld namens *txtURL* eingegebene Internetadresse aufrufen soll, brauchen Sie nur ein paar Zeilen VBA-Code:

```
Dim objWebbrowser As WebBrowser

Private Sub Form_Load()
    Set objWebbrowser = Me!ctlWebbrowser.Object
End Sub

Private Sub txtURL_AfterUpdate()
    objWebbrowser.Navigate Me!txtURL
End Sub
```

Beim Laden des Formulars wird die Variable *objWebbrowser* mit einem Verweis auf das *Webbrowser*-Steuerelement gefüllt. Nach dem Aktualisieren ruft die entsprechende Ereignisprozedur die *Navigate*-Methode mit der URL als Parameter auf. Dieser umständliche wirkende Vorgehensweise ist nötig, weil nur das hinter dem eingebauten

Webbrowser-Steuerelement steckende ActiveX-Steuerelement die hier notwendige *Navigate*-Methode liefert. Ein Beispiel finden Sie in der Beispieldatenbank unter *frmWebbrowserUngebunden*.

4.11 Weitere Steuerelementeigenschaften

Einige Eigenschaften sind für mehrere Steuerelemente hinzugekommen. Die folgenden Absätze erläutern, worum es sich dabei handelt.

4.11.1 Steuerelemente verankern

Eine der interessantesten Neuerungen von Access 2007 ist das Verankern von Steuerelementen: Damit ist die Ereigniseigenschaft *Bei Größenänderung* quasi obsolet, zumindest was das Anpassen der Steuerelementgröße angeht. Die Abbildung 4.32 und Abbildung 4.33 zeigen, wie sich die verankerten Steuerelemente eines Formulars bei einer Größenänderung des Formulars mit verändern.

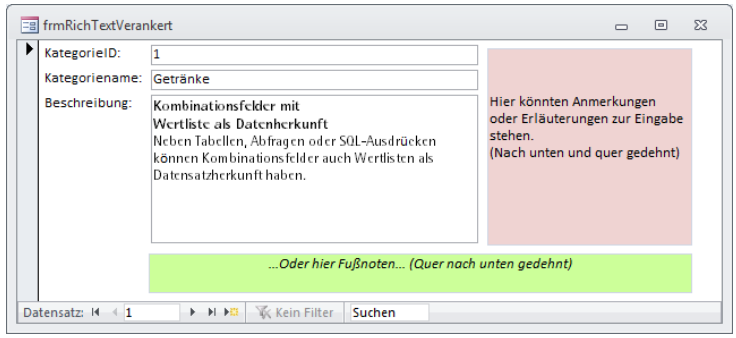


Abbildung 4.32: Ein Formular mit verankerten Steuerelementen im Ausgangszustand ...

Das Verankern der Steuerelemente können Sie über die Benutzeroberfläche oder per VBA vornehmen. In der Benutzeroberfläche finden Sie die passenden Schaltflächen des Ribbons nach Aktivieren des betroffenen Steuerelements in der Entwurfsansicht im Bereich *Anordnen|Position|Anker*. Hier können Sie auswählen, wie ein Objekt verankert werden und wohin es sich ausdehnen soll. Im Eigenschaftsfenster legen Sie die notwendigen Einstellungen mit den Eigenschaften *Horizontaler Anker* und *Vertikaler Anker* (VBA: *HorizontalAnchor* und *VerticalAnchor*, Werte: *acHorizontalAnchorLeft (0)*, *acHorizontalAnchorRight* (*1*) und *acHorizontalAnchorBoth* (*2*) für den horizontalen Anker, *acVerticalAnchorTop (0)*, *acVerticalAnchorBottom* (*1*), *acHorizontalAnchorBoth* (*2*)) fest.

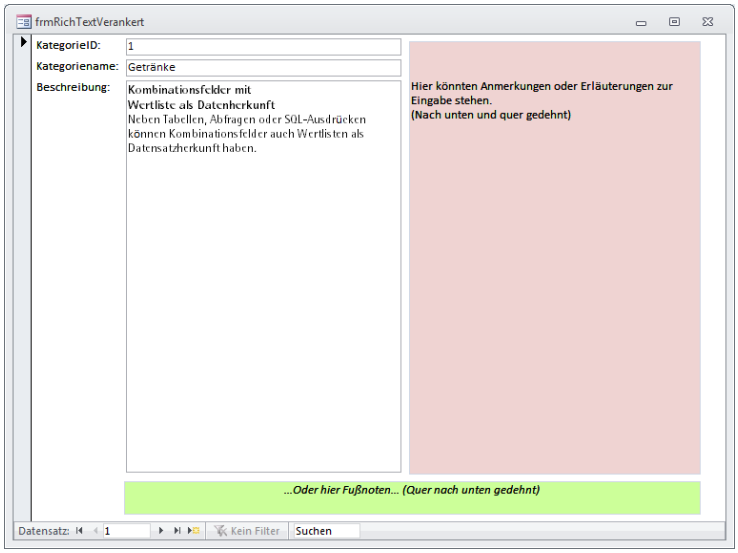


Abbildung 4.33: ... und im vergrößerten Zustand

Dazu eine kurze Erläuterung: Wenn Sie *Horizontaler Anker* für ein Steuerelement auf *Links* und *Vertikaler Anker* auf *Oben* festlegen, tut sich beim Ändern der Formulargröße gar nichts. Stellen Sie *Horizontaler Anker* auf *Rechts* ein, bewegt sich das Steuerelement beim Vergrößern der Breite entsprechend nach rechts. Die Einstellung *Beide* sorgt dafür, dass der Abstand vom linken und vom rechten Rand gleich bleibt und das Steuerelement seine Breite entsprechend dem Formular anpasst.

Gleiches gilt für das vertikale Verankern: *Oben* bedeutet, dass der obere Rand des Steuerelements auf jeden Fall seine vertikale Position beibehält. *Unten* heißt, dass das Steuerelement sich mit dem unteren Formularrand nach unten bewegt, und die Einstellung *Beide* führt zum Vergrößern der Höhe des Steuerelements. Eine Vergrößerung der Breite oder Höhe erfolgt übrigens nicht proportional zur Vergrößerung des Formulars, sondern absolut – das heißt, wenn Sie das Formular um zwei Zentimeter verbreitern, wirkt sich das im gleichen Maße auch auf Steuerelemente aus, für die Sie *Horizontaler Anker* auf *Beide* eingestellt haben. In den Formularen *frmListenfeldMehrfachauswahl* und *frmListViewMitDetaildaten* der Beispieldatenbank finden Sie weitere Beispiele für das Verankern von Steuerelementen.

4.11.2 Layouts

Unter »Hilfreiche Funktionen für den Formularentwurf« ab Seite 171 haben Sie bereits erfahren, wie Sie Steuerelemente zu einem der beiden Layouts *Tabelle* oder *Gestapelt* zusammenfassen. Bezüglich der enthaltenen Steuerelemente gibt es hier einige Besonderheiten.

Layout oder nicht?

Die Eigenschaft *Layout* (nur VBA) gibt an, ob das Steuerelement zu einem der beiden Layouts *Tabelle* oder *Gestapelt* gehört. Da ein Formular mehrere Layouts aufweisen kann, gibt die Eigenschaft *LayoutID* (ebenfalls nur VBA) zusätzlich die ID des Layouts zurück. Beide Eigenschaften sind schreibgeschützt. *Layout* kann die Werte *0* (*acLayoutNone*, kein Layout), *1* (*acLayoutStacked*, gestapeltes Layout) oder *2* (*acLayoutTabular*, Tabellen-Layout) annehmen.

Das Steuerelement Leere Zelle

Seit Access 2010 können Sie hier auch bestehende Zellen horizontal oder vertikal in zwei Zellen aufteilen. Dadurch entstehen Zwischenräume, die aber nur scheinbar leer sind: In Wirklichkeit werden diese Zwischenräume durch das sogenannte *EmptyCell*-Steuerelement erzeugt. Diese Steuerelemente verfügen genauso über eigene Eigenschaften wie alle anderen Steuerelemente. Sie können diese jedoch nicht über das Ribbon hinzufügen.

Wenn Sie beispielsweise das Feld *KundeID* mit dem Ribbon-Eintrag *Anordnen|Zusammenführen/Teilen|Horizontal teilen* aufteilen, sieht das Ergebnis wie in Abbildung 4.34 aus.

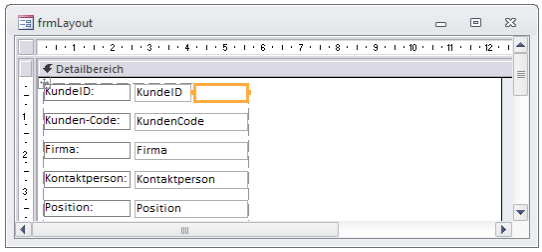


Abbildung 4.34: Leere Zelle nach der Teilung von *KundeID*

Nach einem Klick auf den leeren Bereich zeigt das Eigenschaftsfenster ein Element des Typs *Leere Zelle* mit einem Namen wie *EmptyCell36* an. Dieses enthält die üblichen Format-Eigenschaften. Mehr als die Funktion eines Platzhalters erledigt dieses Steuerelement jedoch nicht. Sie können es auch außerhalb eines Layouts positionieren.

4.11.3 Gitternetzlinien

In den beiden Layouts *Tabelle* und *Gestapelt* können Sie Gitternetzlinien aktivieren (Ribbon-Eintrag *Format|Gitternetzlinien|Gitternetzlinien*). Die Eigenschaften (VBA-Version in Klammern) legen fest, wie das Gitter aussieht:

» *Linienart für Gitternetzlinien oben (GridlineStyleTop)*,

» *Linienart für Gitternetzlinien unten*, (*GridlineStyleBottom),*

» *Linienart für Gitternetzlinien links (GridlineStyleLeft)* und

» *Linienart für Gitternetzlinien rechts* (*GridlineStyleRight*)

» *Gitternetzlinienbreite oben* (*GridlineWidthTop)*,

» *Gitternetzlinienbreite unten (GridlineStyleBottom)*,

» *Gitternetzlinienbreite links (GridlineStyleLeft)* und

» *Gitternetzlinienbreite rechts (GridlineStyleRight*)

Fehlt noch die Farbe: Dafür ist die Eigenschaft *Gitternetzlinienfarbe* (*GridlineColor*) zuständig. Mit dem oben genannten Ribbon-Eintrag erstellen Sie Gitternetzlinien auf die Schnelle, daneben finden Sie noch Einträge zum Einstellen der Breite, Formatvorlage (Linienart) und Farbe per Mausklick.

Diese Eigenschaften sind vorrangig für den Einsatz in Webdatenbanken interessant.

4.11.4 Textabstand

Mit den folgenden Eigenschaften legen Sie die Abstände zu den benachbarten Steuerelementen in den beiden Layouts *Tabelle* und *Gestapelt* fest:

» *Textabstand oben* (*TopPadding)*

» *Textabstand unten (BottomPadding)*

» *Textabstand links (LeftPadding)*

» *Textabstand rechts (RightPadding*)

4.11.5 Effekte für Schaltflächen, Umschaltflächen, Registersteuerelement und Navigationsschaltfläche

Einige Steuerelemente besitzen weitere Eigenschaften zum Einstellen des Erscheinungsbildes. Es handelt sich dabei um die folgenden:

» *Form* (*Shape*)

» *Farbverlauf* (*Gradient*)

» *Schatten* (*Shadow*)

» *Leuchten* (*Glow*)

» *Weiche Kanten* (*SoftEdges*)

» *Abschrägung* (*Bevel*)

Abbildung 4.35 zeigt, wie Sie diese Einstellung über das Ribbon manuell vornehmen können. Den Farbverlauf finden Sie hinter dem Eintrag *Fülleffekt*, die vier Einstellungen *Schatten*, *Leuchten*, *Weiche Kanten* und *Abschrägung* verbergen sich hinter *Formeffekte*.

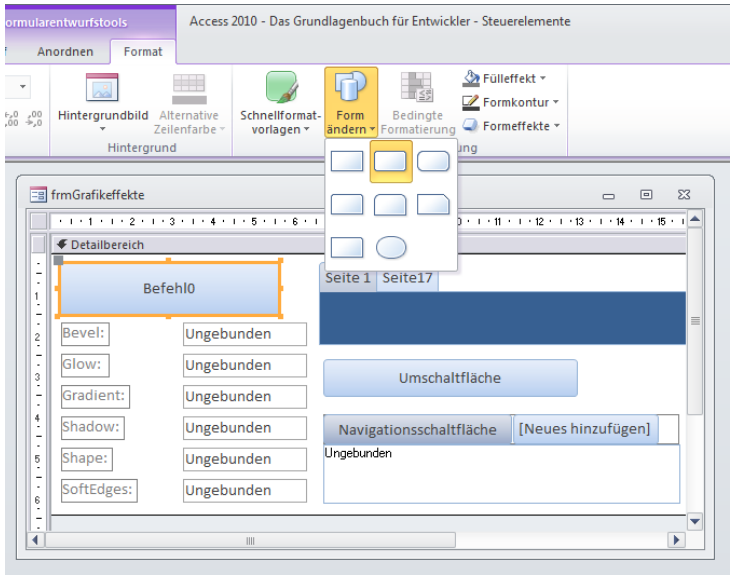


Abbildung 4.35: Einstellen des Aussehens verschiedener Steuerelemente

Das VBA-Modul des in der Abbildung ebenfalls sichtbaren Formulars *frmGrafikeffekte* enthält Beispiele, wie Sie die Einstellungen per VBA vornehmen können. Das Eigenschaftsfenster bietet hierzu keine Möglichkeit.

Beim Register-Steuerelement sind nur die beiden Eigenschaften *Form* und *Farbeffekt* zulässig. Im Formular *frmGrafikeffekte* probieren Sie die verschiedenen Effekte aus, indem Sie das jeweilige Textfeld (etwa *Bevel*) aktivieren und den Wert mit der *Nach oben*- und der *Nach unten*-Taste verändern.

4.11.6 Bedingte Formatierung

Mit Access 2010 gibt es neue Möglichkeiten für die bedingte Formatierung. So können Sie beispielsweise Balken anzeigen, welche das Verhältnis von Zahlenwerten verschiedener Datensätze grafisch aufbereiten (siehe Abbildung 4.36).

Die Optionen zur bedingten Formatierung finden Sie im Ribbon unter *Format|Steuerelementformatierung|Bedingte Formatierung*. Ein Klick auf diesen Eintrag öffnet den Dialog aus Abbildung 4.37. Hier gibt es die üblichen Schaltflächen zum Anlegen, Bearbeiten oder Löschen bedingter Formatierungen sowie die Liste der bisher angelegten Regeln.

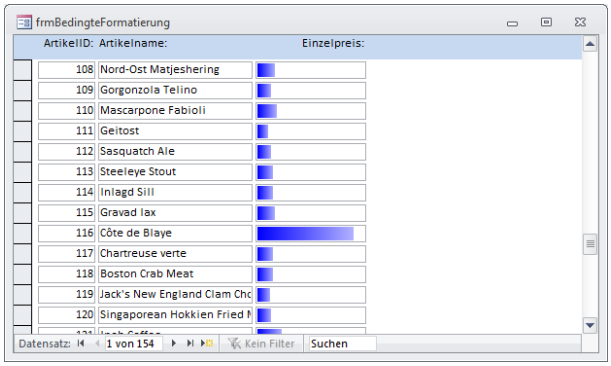


Abbildung 4.36: Balken mit bedingter Formatierung

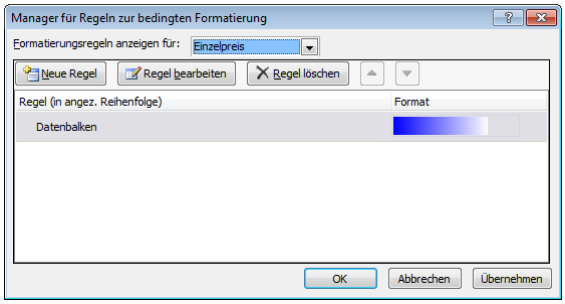


Abbildung 4.37: Der neue Manager für Regeln zur bedingten Formatierung

Die Schaltfläche *Neue Regel* öffnet einen weiteren Dialog namens *Neue Formatierungsregel* (siehe Abbildung 4.38). Dieser sieht auf den ersten Blick kaum anders aus als der aus früheren Versionen bekannte Dialog. Neu ist allerdings die Möglichkeit, zwei verschiedene Regeltypen auszuwählen. Der erste Eintrag *Werte im aktuellen Datensatz prüfen oder einen Ausdruck verwenden* liefert die bereits bekannten Optionen. Neu ist hier jedoch, dass Sie bis zu fünfzig verschiedene Regeln festlegen können – die ärgerliche Beschränkung auf drei Regeln der Vorgängerversionen ist somit obsolet.

Es gibt drei Möglichkeiten für bedingte Formatierungen auf Basis von Werten:

» Feldwert erfüllt eine bestimmte Bedingung

» Ein Ausdruck, der nicht unbedingt das zu formatierende Feld enthält, ist gültig

» Feld hat den Fokus

In allen drei Fällen legen Sie verschiedene Formatierungen fest, die beim Eintreten der jeweiligen Bedingung angezeigt werden sollen (Fett, Kursiv, Unterstrichen, Hintergrundfarbe und Zeichenfarbe).

5

Berichte

Mit Berichten lassen sich Daten in nahezu beliebiger Weise darstellen. Voraussetzung ist, dass Sie genau wissen, wie ein solcher Bericht arbeitet – also wie Sie Gruppierungen und Sortierungen verwenden, wie Sie Unterberichte einsetzen, in welcher Reihenfolge der Bericht die einzelnen Ereignisse abarbeitet, wo Sie innerhalb dieser Ereignisse Aktionen etwa zum Ein- oder Ausblenden von Bereichen oder Steuerelementen unterbringen und so weiter.

Dieses Kapitel hilft Ihnen, die Funktion von Berichten zu verstehen, und zeigt außerdem, wie sich Daten der gängigen Beziehungsarten darstellen lassen.

Außerdem zeigt es, wie Sie die neuen Features seit Access 2007 (Access 2010 liefert kaum Neues in diesem Bereich) am besten nutzen können.

5.1 Berichte erstellen

Microsoft verspricht, dass Sie mit Access 2007/2010 noch schneller zu anspruchsvollen Berichten kommen als mit älteren Versionen. Das gilt in erster Linie für Einsteiger, die normale Ansprüche an das Aussehen und die Funktion eines Berichts haben. Auch erfahrene Access-Entwickler dürften Spaß an dem einen oder anderen Feature haben, das die folgenden Abschnitte vorstellen. Aber wenn es ans Eingemachte geht – wie beispielsweise beim Erstellen aufwändiger Berichte wie etwa der Rechnung, die Sie am Ende dieses Kapitels kennen lernen – helfen die neuen Möglichkeiten zum Erstellen von Berichten auch nicht mehr weiter.

BEISPIELDATENBANKEN

Den Download mit dem Beispielen finden Sie unter *www.acciu.de/aeb2010*. Die Datenbank zu diesem Kapitel heißt *Berichte.accdb*.

Achtung, verwirrende Ansichtsvielfalt!

Berichte unter Access 2007/2010 liefern direkt zwei neue Ansichten: Die erste ist die *Layoutansicht*, die Sie bereits von den Formularen kennen. Die zweite ist die interaktive *Berichtsansicht*. Diese liefert ein ähnliches Bild wie die bereits von älteren Access-Versionen bekannte *Seitenansicht*, wartet aber mit Interaktivität auf: Dafür sorgt eine ganze Reihe neuer Ereigniseigenschaften, die Sie samt der *Berichtsansicht* ganz hinten in diesem Kapitel kennen lernen.

Noch eines ist in diesem Zusammenhang wichtig: Sie können die Ansicht generell über den Ribbon-Eintrag *Start|Ansichten* ändern, nur in der Seitenansicht nicht. Hier finden Sie die passenden Befehle im Kontextmenü.

Und noch ein Hinweis: Sie können die Layoutansicht mit der Berichtseigenschaft *Layoutansicht zulassen* zur Bearbeitung durch den Benutzer freigeben oder sperren. Diese Ansicht erlaubt es dem Benutzer, schnell größere Änderungen an einem Bericht vorzunehmen und ihn damit möglicherweise unbrauchbar zu machen. Sie sollten ihn darauf hinweisen oder diese Ansicht sperren.

5.1.1 Anlegen eines Berichts

Access liefert einige Ribbon-Einträge, über die Sie Berichte erstellen können. Wenn Sie etwa den Eintrag *Erstellen|Berichte|Bericht* verwenden möchten, um einen einfachen Bericht zu erstellen, müssen Sie zuvor eine Tabelle oder Abfrage mit den anzuzeigenden Daten auswählen. Access erstellt dann einen Bericht, den Sie schnell durch Auswählen

eines der zur Verfügung stehenden Designs (Ribbon-Eintrag *Entwurf|Designs*) anpassen können. Wenn Sie einen solchen Bericht erstellen, öffnet Access diesen in der Layoutansicht. Da kommt schon ein wenig Excel-Feeling auf, wenn der Bericht mit gestrichelten Linien andeutet, welche Informationen auf welcher Seite ausgedruckt werden – davon abgesehen ist das aber sehr hilfreich, um die Felder auf das richtige Maß zu stutzen (siehe Abbildung 5.1). Mehr zur Layoutansicht lesen Sie weiter unten.

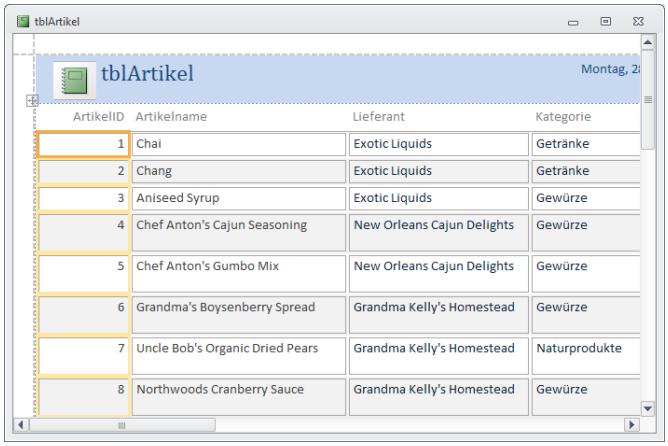


Abbildung 5.1: Ein automatisch erstellter Bericht in der Layoutansicht

Der Assistent zum Anlegen von Berichten hat sich im Vergleich zu den Vorgängerversionen nicht wesentlich geändert; probieren Sie ihn einfach aus und entscheiden Sie, ob Sie Ihre Berichte lieber damit oder von Hand erstellen möchten. Beachten Sie dabei, dass der Assistent immer ein Bild im Berichtskopf einbettet, das eine Menge Speicherplatz braucht. Wenn Sie lieber selbst Hand anlegen, klicken Sie einfach auf den Ribbon-Eintrag *Erstellen|Berichte|Berichtsentwurf*. Sie erhalten dann einen leeren Bericht in der Entwurfsansicht, den Sie von Grund auf neu gestalten können.

Im ersten Schritt wählen Sie eine Tabelle, Abfrage oder einen SQL-Ausdruck als Datensatzquelle für den Bericht aus oder stellen die Daten in einer neuen Abfrage zusammen. Anschließend ziehen Sie die gewünschten Felder aus der Feldliste (Ribbon-Eintrag *Entwurf|Tools|Vorhandene Felder hinzufügen*) in den Berichtsentwurf. Weiter unten erfahren Sie, in welche Bereiche des Berichts welche Daten gehören, und der Abschnitt »Anlegen eines Formulars« ab Seite 162 liefert weitere Informationen über die Feldliste.

Natürlich können Sie wie in Formularen auch andere Steuerelemente als die beim Hinzufügen von Elementen der Feldliste entstehenden Textfelder und sonstige einfache Steuerelemente hinzufügen, indem Sie diese im Ribbon-Bereich *Entwurf|Steuerelemente* markieren und anschließend dem Entwurf hinzufügen.

Diese können Sie anschließend durch Festlegen entsprechender Eigenschaften wie Steuerelementinhalt oder Datensatzherkunft an die dem Bericht zugrunde liegenden Daten oder auch an andere Datenquellen binden.

5.1.2 Vereinfachtes Layouten

Genau wie bei Formularen liefert Access seit Version 2007 einige Features, die eine komfortablere Anordnung der Steuerelemente erlauben. Zum Nachvollziehen der folgenden Techniken erstellen Sie mit dem Ribbon-Eintrag *Erstellen|Berichte|Bericht* einen Bericht auf Basis einer beliebigen Tabelle. Die Tabellenfelder werden dabei automatisch in tabellarischer Form angeordnet (siehe Abbildung 5.1). Außerdem legt Access automatisch das Layout *Tabelle* für die Steuerelemente fest (das können Sie – wenn Sie die Steuerelemente von Hand in den Bericht gezogen haben – durch Markieren der benötigten Steuerelemente und Betätigen der Ribbon-Schaltfläche *Anordnen|Layout bestimmen|Tabelle|Tabelle* erreichen). Dieses Layout verschafft einige Vorteile:

» Sie brauchen nur noch ein Element einer Spalte zu markieren und können ihre Breite durch Ziehen des rechten Rands mit der Maus anpassen – und die rechts davon liegenden Steuerelemente verschieben sich direkt mit. In älteren Access-Versionen mussten Sie alle rechts davon liegenden Steuerelemente manuell verschieben.

» Sie sehen die enthaltenen Daten und können die Breite direkt daran anpassen, ohne wie früher zwischen Entwurfs- und Seitenansicht hin- und herwechseln zu müssen.

» Sie können die Reihenfolge der Spalten durch eine einfache Maus-Aktion einstellen. Um eine Spalte um eine Position nach rechts zu bewegen, ziehen Sie diese einfach so weit nach rechts, bis an der gewünschten Stelle ein vertikaler Balken angezeigt wird, und lassen die Spalte dann fallen (siehe Abbildung 5.2).

Im Layout *Gestapelt* sieht dies ähnlich aus, nur dass die Felder eines Datensatzes hier in »Päckchenform« angezeigt werden – hier können Sie entsprechend die Höhe der Steuerelemente und ihre vertikale Reihenfolge ändern, ohne dass Sie die von der Änderung betroffenen Steuerelemente manuell anpassen müssen.

5.1.3 Einheitliches Design mit Autoformat

Bereits unter Access 2007 wurde mit Autoformat eine Möglichkeit eingeführt, für ein einheitliches Layout in Formularen und Berichten zu sorgen. Dort gestalten Sie einen Bericht nach Wunsch und speichern die vorhandenen Formatierungen in Form eines Autoformats. Diese Funktion ist nun zumindest nicht mehr über die Benutzeroberfläche verfügbar. Wer sie dennoch nutzen möchte, setzt im VBA-Editor den folgenden Befehl ab:

```
RunCommand acCmdAutoFormat
```

Statt der Autoformate stehen seit Access 2010 die sogenannten Designs zur Verfügung. Mehr darüber erfahren Sie unter »Farben und Schriftarten per Design festlegen« ab Seite 176, die Anwendung erfolgt in Berichten auf die gleiche Weise wie in Formularen.

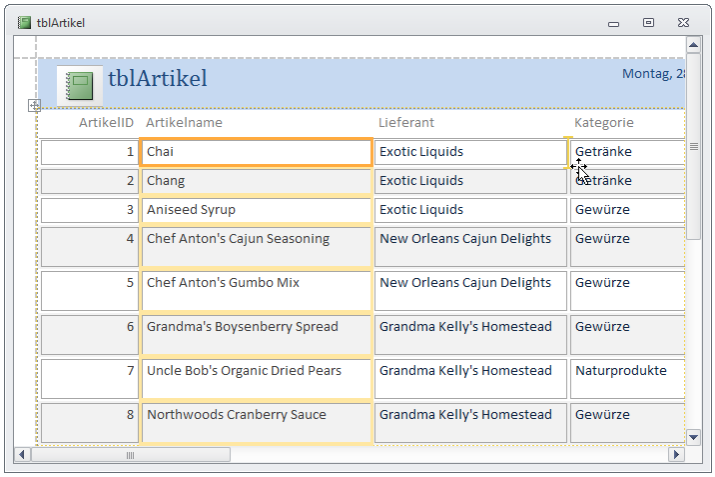


Abbildung 5.2: Ändern der Reihenfolge von Spalten in einem Bericht

5.1.4 Wechselnde Hintergrundfarbe

Im Ribbon-Bereich *Format* finden Sie Schaltflächen in den Bereichen *Schriftart|Hintergrundfarbe* sowie *Hintergrund|Alternative Zeilenfarbe.* Damit können Sie ganz einfach wechselnde Hintergrundfarben für Berichte festlegen (siehe Abbildung 5.3). Markieren Sie zuvor den Detailbereich oder den Gruppenkopf/-fuß, der mit einer alternativen Hintergrundfarbe versehen werden soll, in der Entwurfsansicht des Berichts.

5.1.5 Bedingte Formatierung

Neben den wechselnden Hintergrundfarben können Sie Berichtssteuerelemente mit bedingten Formatierungen ausstatten. Dazu markieren Sie das passende Steuerelement, klicken Sie auf die Ribbon-Schaltfläche *Format|Schriftart|Bedingte Formatierung* und legen in dem nun erscheinenden Dialog die Bedingungen und die passenden Formatierungen fest.

Im Gegensatz zu älteren Access-Versionen können Sie ab Access 2010 bis zu 50 bedingte Formatierungen festlegen und außerdem Balkendiagramme in Abhängigkeit von den enthaltenen Zahlenwerten anzeigen lassen – mehr dazu erfahren Sie unter »Bedingte Formatierung« ab Seite 298.

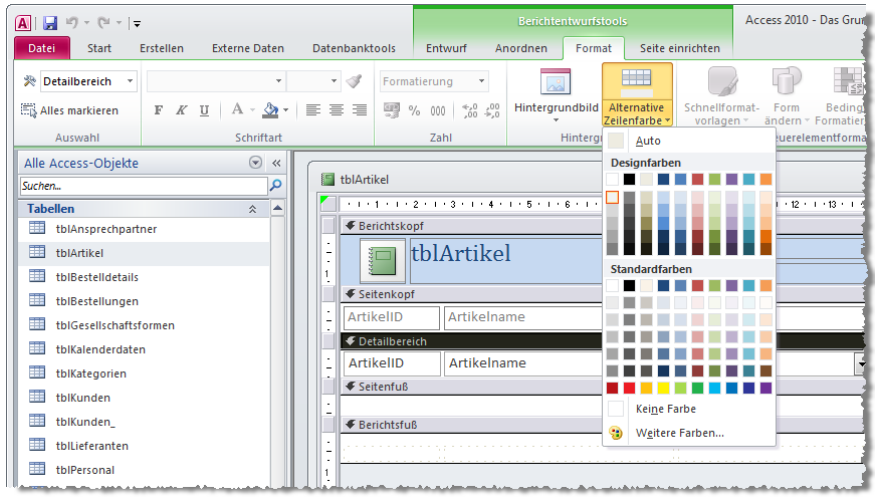


Abbildung 5.3: Festlegen einer alternativen Hintergrundfarbe

5.1.6 Sonstige Layout-Vereinfachungen

Die übrigen Funktionen, wie das Anpassen der Seitenränder, des Abstandes des Texts eines Steuerelements von den Steuerelementrändern, das Ausrichten von Steuerelementen, das Hinzufügen und Entfernen von Steuerelementen in beziehungsweise aus den beiden Layouts, sind intuitiv zu handhaben.

Ein echtes Highlight ist dabei die Möglichkeit, einem Layout Gitternetzlinien hinzuzufügen – wer mit älteren Versionen von Access gearbeitet hat, weiß, welchen Spaß man mit vertikalen Linien in Berichten haben kann (Layoutansicht, Ribbon-Eintrag *Format|Gitternetzlinien|Gitternetzlinien*). Probieren Sie es einfach aus; bei Problemen liefert die Onlinehilfe ausreichend Hilfestellung. Wenn Sie einmal alle Elemente des Layouts markieren möchten, markieren Sie zunächst ein Element und klicken dann auf das links über dem Layout erscheinende Symbol oder die Ribbon-Schaltfläche *Anordnen|Zeilen und Spalten|Layout* auswählen.

5.1.7 Berichtsbereiche

Berichte haben standardmäßig drei Bereiche, die Sie mit Steuerelementen füllen können: der Detailbereich, der Seitenkopf und der Seitenfuß. Hinzu kommen, wenn Sie den Kontextmenüeintrag *Berichtskopf/-fuß* auswählen, zwei weitere Bereiche namens *Berichtskopf* und *Berichtsfuß*. Diese Bereiche haben die folgenden Funktionen:

» *Detailbereich*: Dieser Bereich wird für jeden Datensatz der Datensatzherkunft wiederholt. Hier fügen Sie die eigentlichen Daten ein.

» *Seitenkopf und -fuß*: Diese Bereiche werden im Kopf und im Fuß jeder Seite angezeigt. Sie können darin Angaben wie Seitenzahlen, Datum oder Seitensummen und -überträge anzeigen.

» *Berichtskopf und -fuß*: Diese Bereiche zeigt der Bericht am Anfang und am Ende an. Der Berichtskopf kann Informationen über den Bericht enthalten und gegebenenfalls auch auf eine komplette Seite ausgedehnt und als Deckblatt verwendet werden. Im Berichtsfuß bringen Sie etwa Gesamtsummen unter.

» *Gruppenköpfe und -füße*: Weitere Bereiche können Sie durch Gruppierungen erreichen. Wenn Sie etwa die Artikel der Nordwind-Datenbank nach Kategorien gruppieren und jeweils die Kategorie oberhalb einer Gruppe von Artikeln ausgeben möchten, ist eine Gruppierung genau das Richtige. Zu jeder Gruppierung können Sie einen Gruppenkopf und einen Gruppenfuß einblenden, um beispielsweise Überschriften oder Gruppensummen einzufügen. Mehr zu Gruppierungen erfahren Sie weiter unten.

5.1.8 Berichtsansichten

Seit Access 2007 gibt es vier für die Arbeit mit Berichten geeignete Ansichten:

» Die Seitenansicht zeigt den Bericht so, wie er auch ausgedruckt wird.

» Die Entwurfsansicht dient dem Entwerfen des Berichts. Hier können Sie etwa Steuerelemente hinzufügen und ausrichten, Gruppierungen und Sortierungen oder die Größe von Bereichen festlegen.

» Die Layoutansicht ist eine Mischung aus Seiten- und Entwurfsansicht. Sie liefert ein gutes Bild des beim Ausdruck zu erwartenden Aussehens des Berichts und bietet gleichzeitig die Möglichkeit, den Entwurf von Bericht und Steuerelementen anzupassen.

» Die Berichtsansicht sieht genau so aus wie die Seitenansicht, liefert aber erheblich mehr Möglichkeiten zur Interaktion mit dem Benutzer. Dazu unterstützt sie eine ganze Reihe von Ereigniseigenschaften, die Sie bisher nur von Formularen her kannten.

5.1.9 Gruppieren und sortieren

Eine Gruppierung legen Sie in der Entwurfsansicht im Bereich *Gruppieren, Sortieren und Summe* an, den Sie mit dem Ribbon-Eintrag *Entwurf|Gruppierung und Summen|Gruppieren und Sortieren* anzeigen. In der Layoutansicht blenden Sie diesen Bereich mit dem Ribbon-Eintrag *Format|Gruppierung und Summen|Gruppieren und sortieren* ein. Im Gegensatz zu früheren Access-Versionen ist dieser Bereich fest im unteren Teil des Access-Fensters verankert und steht nicht mehr als eigenes Dialogfenster zur Verfügung. Prinzipiell hat sich aber zumindest bezüglich der Gruppierung und Sortierung nicht viel geändert.

5.2 Berichte anzeigen

Die Anzeige von Berichten erfolgt in VBA meist über die *DoCmd.OpenReport*-Methode. Die Methode hat folgende Parameter (hier nur mit den gebräuchlichsten Konstanten):

» *ReportName*: Name des anzuzeigenden Berichts

» *View*: Ansicht beim Öffnen (*acViewPreview*: Seitenansicht, *acViewNormal*: Drucken, *acViewDesign*: Entwurfsansicht, *acViewLayout*: Layoutansicht, *acViewReport*: Berichtsansicht)

» *FilterName*: Name einer Abfrage auf Basis der Datensatzquelle

» *WhereCondition*: Teil einer Abfrage hinter der *WHERE*-Klausel zum Einschränken der enthaltenen Daten

» *WindowMode*: Fenstereigenschaften (*acDialog*: modales Fenster, *acWindowNormal*: Standard, *acIcon*: minimiert, *acHidden*: nicht sichtbar)

» *OpenArgs*: Öffnungsargument, mit dem zusätzliche Informationen übergeben werden können (Datentyp *Variant*)

Die Parameter können Sie entweder in der angegebenen Reihenfolge durch Kommata getrennt eingeben. Wenn Sie einen Parameter nicht benötigen, lassen Sie diesen einfach weg, ohne aber die Anzahl der Kommata zu verändern. Hinter dem letzten verwendeten Parameter brauchen Sie allerdings keine Kommata mehr anzuhängen.

Alternativ können Sie mit »benannten Argumenten« arbeiten. Dabei geben Sie den Parameternamen gefolgt von Doppelpunkt und Gleichheitszeichen sowie dem Wert an (in einer Zeile):

```
DoCmd.OpenReport "rptKalenderdaten", View:=acViewPreview,
    FilterName:="qryKalenderdaten"
```

Zwischen Parameter und Wert dürfen sich keine Leerzeichen befinden. Die Reihenfolge bei der Verwendung benannter Parameter ist beliebig.

Besonderheit beim Öffnen von Berichten aus modal geöffneten Formularen

Wenn ein Formular, von dem aus Sie etwa per Schaltfläche einen Bericht anzeigen möchten, mit dem Parameter *WindowMode:=acDialog*, also als modaler Dialog geöffnet wurde, müssen Sie auch den Bericht als modales Fenster öffnen.

Anderenfalls wird der Bericht direkt im Hintergrund des Formulars angezeigt und Sie können nicht auf diesen zugreifen.

Berichte in Formularen anzeigen

Seit Access 2010 können Sie Berichte in Unterformular-Steuerelemente einklinken. Das ist erstmal eine gute Nachricht – die kleine Einschränkung ist, dass dies nur in der Berichtsansicht funktioniert (diese mit Access 2007 eingeführte Ansicht lernen Sie im Detail unter »Die Berichtsansicht« ab Seite 408 kennen). Abbildung 5.4 zeigt ein Beispiel für einen Bericht im Formular.

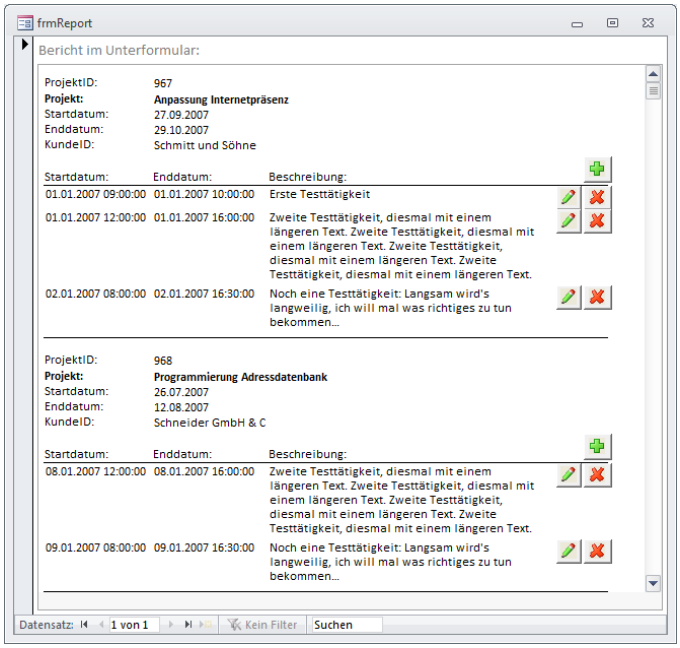


Abbildung 5.4: Ein Bericht in einem Unterformular

5.3 Filtern und sortieren

Berichte enthalten sechs Eigenschaften, mit denen sich Filter- und Sortierkriterien einstellen lassen. Diese können auf unterschiedliche Weise festgelegt werden – beispielsweise auf Basis der zugrunde liegenden Datensatzquelle oder auch per Zuweisung der entsprechenden Eigenschaften mit VBA.

Wenn Sie etwa eine Abfrage als Datensatzquelle festlegen, die eine Sortierung und ein Kriterium enthält, werden diese Angaben einfach in den Bericht übernommen, ohne dass Sie dafür eine Eigenschaft einstellen müssen. Wenn Sie beim Öffnen des Berichts eine Abfrage mit dem Parameter *FilterName* übergeben, die eine Sortierung und ein Kriterium enthält, wird das Kriterium dieser Abfrage in die Eigenschaft *Filter* übernom-

men, aber die Eigenschaft *Beim Laden Filtern* nicht gesetzt. Beim Sortieren funktioniert alles wie erwartet: Die Eigenschaften *Sortiert nach* und *Beim Laden sortieren* erhalten die gewünschten Werte (siehe Abbildung 5.5). Trotzdem wird nach dem angegebenen Artikelnamen gefiltert und sortiert:

```
DoCmd.OpenReport "rptArtikelUebersicht", acPreview, "qryArtikel"
```

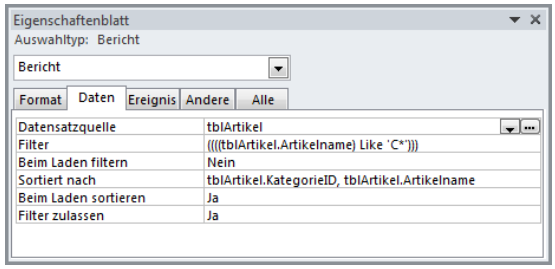


Abbildung 5.5: Einstellungen der Filter- und Sortiereigenschaften nach Zuweisen einer Abfrage mit dem Parameter *FilterName* der *DoCmd.OpenReport*-Methode

Die Eigenschaften in dieser Abbildung lassen sich nebenher auch noch manuell oder per VBA zur Laufzeit einstellen. Per VBA stehen sogar noch zwei weitere Eigenschaften zur Verfügung: *FilterOn* und *SortByOn*. Der Unterschied zwischen *FilterOnLoad* und *FilterOn* ist folgender: Enthält *Filter* einen Kriteriumsausdruck und hat *FilterOnLoad* den Wert *True*, dann wird der Filter bereits beim Laden des Formulars berücksichtigt. Das dürfte der Normalfall sein.

Stellen Sie die Eigenschaft *FilterOn* zur Laufzeit, also bei geöffnetem Bericht, auf *False* ein, wird der angegebene Filter deaktiviert. Stellen Sie den Wert wieder auf *True*, filtert der Bericht die Daten wieder nach dem angegebenen Filterkriterium. Hat *FilterOn* den Wert *True*, können Sie auch das Filterkriterium ändern und die Auswirkungen direkt in der Seitenansicht betrachten.

Im Gegensatz zu älteren Access-Versionen finden Sie die Eigenschaften *FilterOn* und *OrderByOn* nicht mehr im Eigenschaftsfenster (dafür gab es dort die Eigenschaften *FilterOnLoad* und *OrderByOnLoad* noch nicht).

Die Eigenschaften *FilterOn* und *OrderByOn* werden übrigens beide auf den Wert *Ja* eingestellt, wenn man beim Öffnen des Berichts wie oben eine Abfrage mit Sortierung und Filterkriterien für den Parameter *FilterName* übergibt.

Wenn Sie eine der gleich vorgestellten Gruppierungen oder Sortierungen im Bereich *Gruppieren, Sortieren und Summe* festlegen, steht die in der Eigenschaft *Sortiert nach* festgelegte Sortierung ganz hinten an. Wenn Sie in *Sortiert nach* etwa absteigend nach dem Artikelnamen sortieren und im Bereich *Gruppieren, Sortieren und Summe* eine auf-

steigende Sortierung nach dem Artikelnamen festgelegt ist, werden die Datensätze aufsteigend nach dem Artikelnamen sortiert. Die Einstellung der Eigenschaft *Sortiert nach* macht sich erst bemerkbar, wenn im Bereich *Gruppieren, Sortieren und Summe* keine Sortierung für dieses Feld angegeben ist.

5.3.1 Filtern und sortieren in der Seitenansicht

Interessant werden die Eigenschaften *Filter*, *Filter aktiv*, *Sortierung* und *Sortierung aktiv*, wenn die diesbezüglichen Möglichkeiten im Bericht noch nicht durch sonstige Sortierungen und Gruppierungen erschöpft sind.

Ein Beispiel dafür sind einfache Übersichtslisten wie der Bericht *rptArtikelUebersicht* (siehe Beispieldatenbank). Wenn Sie diesen in der Seitenansicht öffnen, können Sie zur Laufzeit per VBA die Sortierung und den Filter anpassen. Um die Datensätze beispielsweise in absteigender Reihenfolge nach dem Artikelnamen zu sortieren, verwenden Sie die folgende Anweisung:

```
Reports!rptArtikelUebersicht.OrderBy = "Artikelname DESC"
```

Falls das nicht wirkt, ist die Sortierung noch nicht aktiv. Schieben Sie in diesem Fall noch folgende Anweisung hinterher:

```
Reports!rptArtikelUebersicht.OrderByOn = True
```

Genauso können Sie auch den Filter einsetzen. Alle Artikel mit dem Anfangsbuchstaben *A* liefert die folgende Anweisung (wiederum bei aktivierter Seitenansicht):

```
Reports!rptArtikelUebersicht.Filter = "Artikelname LIKE 'A*'"
```

Falls der Filter nicht greift, müssen Sie auch diesen anschalten:

```
Reports!rptArtikelUebersicht.FilterOn = True
```

Das Ganze funktioniert seit Access 2007 auch über die Benutzeroberfläche: Sie können per Kontextmenü des gewünschten Feldes einen Filter festlegen.

5.3.2 Filtern, sortieren und gruppieren in der Layoutansicht

Mit der Layoutansicht erhalten Sie noch viel mächtigere Möglichkeiten, die im Bericht angezeigten Daten zur Laufzeit zu filtern und zu sortieren. Abbildung 5.6 zeigt, wie das aussehen kann: Wenn Sie etwa die Daten in tabellarischer Form ausgeben, können Sie in der Layoutansicht einfach mit der rechten Maustaste auf ein Feld klicken und aus dem Kontextmenü eine ganze Reihe zusätzlicher Funktionen auswählen.

Mit den beiden Eigenschaften *Berichtsansicht zulassen* und *Layoutansicht zulassen* können Sie übrigens festlegen, welche Ansichten der Benutzer verwenden darf.

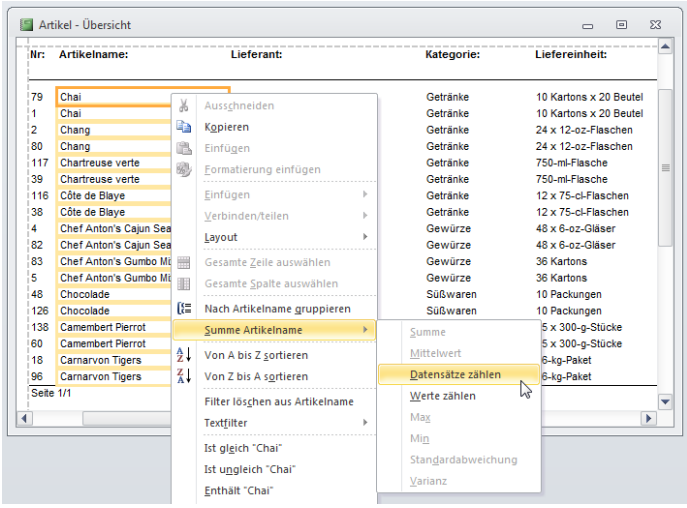


Abbildung 5.6: Das Kontextmenü eines Feldes in der Layoutansicht liefert Sortier- und Filterfunktionen

5.3.3 Filtern in der Layoutansicht

Wenn Sie ein Feld nach bestimmten Kriterien filtern möchten, klicken Sie im Kontextmenü des Feldes entweder einen der Schnellfilter an, die unterschiedliche Vergleichsoperatoren auf den Inhalt des aktuellen Feldes anwenden, oder wählen den für den aktuellen Datentyp angezeigten Eintrag wie Textfilter, Zahlenfilter oder Datumsfilter und einen der für dieses Feld gültigen Vergleichsoperatoren aus. Diese variieren je nach dem Felddatentyp – für Datumsangaben bietet Access etwa direkt eine ganze Latte von Vergleichsmöglichkeiten. In einem weiteren Dialog geben Sie den Vergleichswert ein; Access filtert den Bericht dann nach den angegebenen Kriterien. Diese Funktion ist intuitiv zu bedienen, probieren Sie es einfach aus.

5.3.4 Sortieren in der Layoutansicht

Mit den Kontextmenüeinträgen *Von A bis Z sortieren* oder *Von Z bis A sortieren* legen Sie die Sortierung für das aktuell ausgewählte Feld fest. Dies funktioniert leider nur für das aktuell markierte Feld. Wenn Sie nach mehreren Feldern sortieren wollen, müssen Sie entsprechende Sortierkriterien im Bereich *Gruppieren, Sortieren und Summe* festlegen (Ribbon-Eintrag *Entwurf|Gruppierung und Summen|Gruppieren und sortieren*). Das erledigen Sie mit einem Klick auf die Schaltfläche *Sortierung hinzufügen*. Die Reihenfolge der Sortierung können Sie dann mit den Pfeil-Schaltflächen am rechten Rand des Bereichs festlegen. Die Daten werden zuerst nach dem zuoberst angegebenen Kriterium sortiert.

5.3.5 Gruppieren in der Layoutansicht

Sie können beispielsweise nach einem Feld gruppieren. Im vorliegenden Beispiel macht das etwa für Lieferanten Sinn: Klicken Sie mit der rechten Maustaste auf einen der Einträge in der Spalte *Lieferanten* und wählen Sie den Kontextmenüeintrag *Nach Lieferant gruppieren* aus.

Das Ergebnis sieht wie in Abbildung 5.7 aus: Access zieht die Spalte mit den Lieferanten heraus und fügt sie in einen Gruppenkopf ein, die passenden Artikel werden darunter aufgelistet. Was da passiert ist, lässt sich prima in der Entwurfsansicht nachvollziehen (siehe Abbildung 5.8): Access hat tatsächlich eine neue Gruppierung erstellt, die Sie noch nach Ihren Wünschen anpassen können – einfacher geht es kaum.

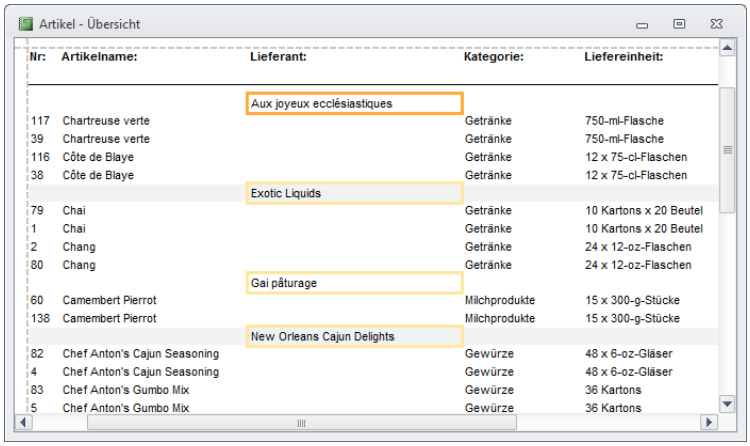


Abbildung 5.7: Die Gruppierung nach Lieferanten ist nicht schön, aber schnell gemacht ...

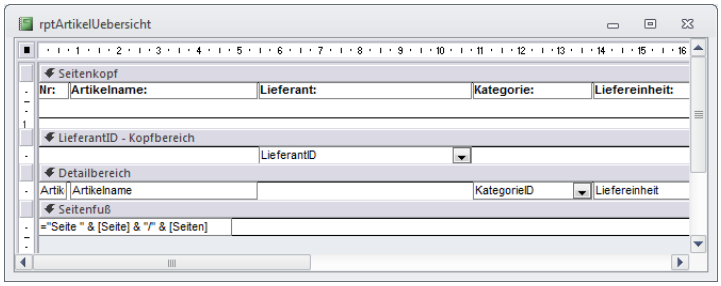


Abbildung 5.8: ... und kann vor allem als Basis für eine dauerhafte Gruppierung verwendet werden, wie die Entwurfsansicht zeigt

In der Layoutansicht wie auch in der Berichtsvorschau können Sie einen in ab Access 2007 neu designten Bereich zum Bearbeiten der Gruppierungs- und Sortierungsebenen

anzeigen lassen (Ribbon-Eintrag *Entwurf|Gruppierung und Summen|Gruppieren und sortieren*). Diese hat leider immer die in Abbildung 5.9 angezeigte Höhe und kann nur in der Breite verändert werden – das ist ein großer Wermutstropfen in Anbetracht der vielen Verbesserung der Berichtsansichten.

Gegenüber dem alten Dialog zum Einstellen dieser Eigenschaften hat sich dennoch einiges verbessert: Sie können Gruppierungen und Sortierungen mit einem Klick löschen oder ihre Priorität anpassen.

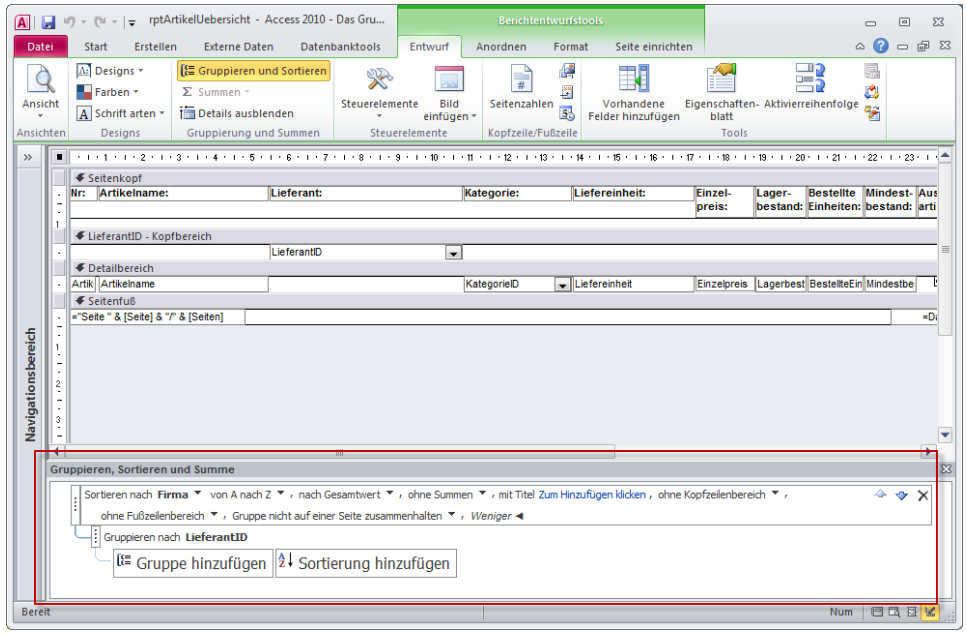


Abbildung 5.9: Die Gruppierungs- und Sortierungseigenschaften lassen sich leichter bearbeiten, allerdings leidet die Übersicht durch die fixe Höhe des passenden Bereichs

Damit Einsteiger nicht mehr mit den verwirrenden Begriffen »aufsteigend« und »absteigend« durcheinanderkommen, lauten die Sortieroptionen nun »vom kleinsten zum größten« und »vom größten zum kleinsten« (siehe Abbildung 5.10). Gruppenkopf und Gruppenfuß heißen nun »Kopfzeilenbereich« und »Fußzeilenbereich«.

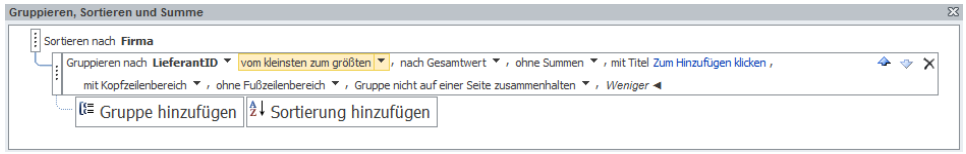


Abbildung 5.10: Eine Gruppierung mit allen Details

5.3.6 Summen in der Layoutansicht

Wenn Sie einem Berichtsfeld in der Layoutansicht eine Summe hinzufügen wollen, wählen Sie aus dem Kontextmenü des passenden Felds den Eintrag *Summe <Feldname>* aus. Dies liefert genau genommen nicht direkt ein Summenfeld, sondern weitere Menüeinträge, mit denen Sie weitere Aggregatberechnungen durchführen können (siehe Abbildung 5.11). Abbildung 5.12 zeigt das neue Berechnungsfeld im Berichtsfuß.

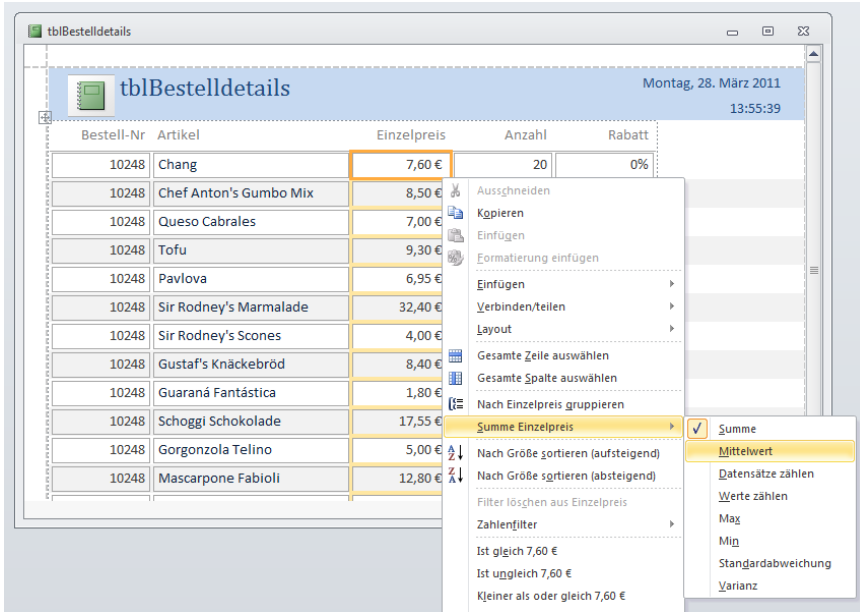


Abbildung 5.11: Berechnungsfunktionen für einzelne Felder eines Berichts

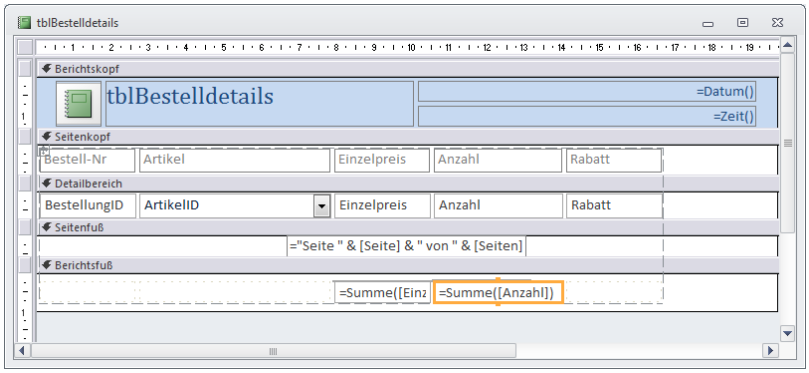


Abbildung 5.12: Berechnungsfelder wie diese lassen sich mit Access 2007/2010 per Kontextmenü anlegen

5.4 Berichtsbereiche und Ereignisse

Von elementarer Bedeutung für die Arbeit mit Berichten – vor allem mit VBA – ist das Verständnis der einzelnen Bereiche eines Berichts und der Ereignisse dieser Berichtsbereiche. Daher finden Sie zunächst noch einmal eine Zusammenfassung der Informationen über die einzelnen Berichtsbereiche, bevor es an die Beschreibung der Ereignisse dieser Bereiche geht.

5.4.1 Berichtsbereiche

Standardmäßig zeigt ein Bericht einen Detailbereich sowie einen Seitenkopf und einen Seitenfuß an. Optional lassen sich noch ein Berichtskopf- und ein Berichtsfuß anzeigen. Letztere werden nur je einmal am Anfang und am Ende des Berichts eingeblendet und können beispielsweise für die Gestaltung einer Titelseite verwendet werden.

Seitenkopf und -fuß erscheinen am oberen und unteren Ende jeder Seite und legen somit fest, wie viel Platz noch für den Detailbereich und andere Bereiche übrig bleibt. Alle Bereiche lassen sich jedoch nach Bedarf ein- und ausblenden. Dazu später mehr.

Eines der wichtigsten Elemente zur Strukturierung von Daten – und hier speziell von verknüpften Daten – liegt in der Möglichkeit, Daten nach bestimmten Kriterien zu gruppieren und zu sortieren.

Sortierungen und Gruppierungen legen Sie in dem bereits weiter oben erwähnten Bereich fest. Mit diesem fügen Sie Sortierungen und Gruppierungen zu einem Bericht hinzu. Im Gegensatz zu älteren Access-Versionen, in denen Sie zunächst Felder in den Dialog gezogen und dann mit den Eigenschaften *Gruppenkopf* und *Gruppenfuß* festgelegt haben, ob es sich nur um eine Sortierung oder eine richtige Gruppierung mit Kopf- oder Fußbereich handelte, können Sie unter Access 2007/2010 per Klick auf die entsprechende Schaltfläche entweder eine Gruppe oder eine Sortierung anlegen.

Der einzige Unterschied ist, dass bei einer Gruppierung gleich ein Kopfzeilenbereich hinzugefügt wird. Entfernen Sie diesen, wird aus der Gruppierung eine Sortierung und umgekehrt (Gleiches gilt für den Fußzeilenbereich).

Um eine Sortierung oder Gruppierung hinzuzufügen, klicken Sie auf eine der Schaltflächen *Gruppe hinzufügen* oder *Sortierung hinzufügen* (siehe Abbildung 5.13) oder ziehen einfach ein Feld aus der Feldliste auf den Gruppierungs-Bereich und stellen dann in der erweiterten Ansicht aus Abbildung 5.14 das zu gruppierende Feld und die übrigen Eigenschaften ein. Die Einstellungen aus dieser Abbildung sorgen übrigens dafür, dass die zugrunde liegenden Artikel zunächst nach Kategorien und dann innerhalb der Kategorien nach Lieferanten gruppiert werden. Die Daten jeder Lieferanten-Gruppe werden dann noch einer aufsteigenden Sortierung unterzogen.

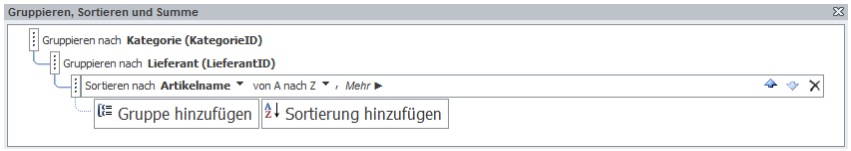


Abbildung 5.13: Dialog zum Festlegen von Sortierungen und Gruppierungen

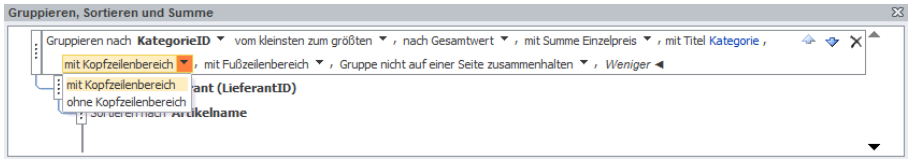


Abbildung 5.14: Festlegen der Bereiche einer Gruppierung

Unter »Wichtige Eigenschaften von Berichten und Berichtsbereichen« ab Seite 381 lernen Sie die übrigen Einstellungen des *Gruppieren, Sortieren und Summe*-Bereichs kennen.

5.4.2 Ereignisse in Berichten

Berichte bieten bereits seit Access 2007 erheblich mehr Ereignisse für Berichte als in den Vorgängerversionen – und fast so viele wie für Formulare. Die Möglichkeiten der Interaktion sind um einiges besser geworden; so können Sie etwa auf einen Datensatz in einem Bericht klicken, um ein Formular mit den Details zum Bearbeiten zu öffnen. Diese Ereignisse beziehen sich aber alle auf eine weitere neue Darstellungsart, nämlich die »Berichtsansicht«. Mehr zu dieser neuen interaktiven Ansicht finden Sie weiter unten.

Bis ein Bericht überhaupt einmal angezeigt ist, geschieht allerdings schon eine Menge und in vielen Fällen können Sie den Ablauf per Ereignisprozedur beeinflussen. Nachfolgend finden Sie zunächst eine Auflistung der Ereignisse eines Berichts und der Bereiche eines Berichts; im Anschluss lernen Sie einige Anwendungsfälle der Ereignisse kennen.

5.4.3 Zusammenfassung der Berichtsereignisse

Der Bericht löst folgende Ereignisse aus, die in den bisherigen Access-Versionen ebenfalls unterstützt wurden und die für die reine Anzeige eines Berichts wichtig sind:

» *Beim Öffnen*: Wird beim Öffnen, aber vor dem Erstellen des Berichts ausgelöst. Dient beispielsweise der Übergabe von Parametern oder zum Abbrechen der Ausgabe des Berichts.

» *Bei Aktivierung*: Bericht wird aktiviert oder gedruckt.

» *Bei Seite*: Wird nach Abschluss der *Bei Formatierung*- und der *Beim Drucken*-Ereignisse der Berichtsbereiche, aber vor dem Anzeigen der Seite ausgelöst.

» *Beim Schließen*: Wird beim Schließen des Berichts ausgelöst – etwa durch Betätigen der *Schließen*-Schaltfläche.

» *Bei Deaktivierung*: Bericht wird deaktiviert, da ein anderes Objekt den Fokus erhält oder der Bericht geschlossen wird.

» *Bei Ohne Daten*: Wird ausgelöst, wenn die Datensatzquelle des Berichts keine Daten enthält.

» *Bei Fehler*: Wird beim Auftreten eines Fehlers ausgelöst.

5.4.4 Zusammenfassung der Bereichsereignisse

Die Bereiche *Berichtskopf*, *Berichtsfuß*, *Seitenkopf*, *Seitenfuß*, *Kopf* und *Fuß* der einzelnen Gruppierungen und der Detailbereiche lösen die folgenden Ereignisse aus.

Dabei werden die Ereignisse *Beim Formatieren* und *Beim Drucken* je Element eines jeden Bereichs mindestens einmal aufgerufen:

» *Beim Formatieren*: Wird ausgelöst, wenn die Daten ermittelt sind, bevor der Bereich formatiert wird. Dieses Ereignis wird gegebenenfalls mehrere Male ausgelöst. Wenn es sich beim aktuell formatierten Bereich um einen Gruppenkopf handelt, haben Sie Zugriff auf die im Gruppenkopf angezeigten Daten und auf die Daten des ersten Datensatzes der Gruppierung im Detailbereich. Beim Gruppenfuß stehen die Daten des Gruppenfußbereichs und des letzten Datensatzes des Bereichs zur Verfügung. Im Detailbereich bietet dieses Ereignis Zugriff auf die Daten des aktuellen Datensatzes.

» *Beim Drucken*: Wird ausgelöst, wenn der Bereich formatiert, aber noch nicht gerendert ist. In diesem Ereignis können Sie je Bereich auf die gleichen Daten zugreifen wie im *Beim Formatieren*-Ereignis. Mit »Drucken« ist hier nicht die Ausgabe auf Papier gemeint, sondern die grafische Ausgabe in das »Dokument«, das dann auf dem Bildschirm oder auf Papier dargestellt werden kann.

» *Bei Rückname*: Wird jedes Mal ausgelöst, wenn Access einen Bereich infolge Positionierungsberechnungen neu formatieren muss (steht nicht für den Seitenkopf zur Verfügung).

Die genaue Abfolge der Ereignisse eines Berichts ist relativ komplex. Mehr Gruppierungen oder spezielle Bedingungen wie etwa das Zusammenhalten von Gruppierungen führen zu einer unüberschaubaren Menge von Ereignissen. Allein der Aufruf eines Berichts

mit einer Gruppierung, der nur einen Datensatz anzeigt, löst die folgenden Ereignisse in der angegebenen Reihenfolge aus:

» Bericht *Beim Öffnen*

» Bericht *Bei Aktivierung*

» Seitenkopfbereich *Beim Formatieren*

» Gruppenfuß0 *Beim Formatieren*

» Detailbereich *Beim Formatieren*

» Gruppenfuß0 *Beim Formatieren*

» Seitenfußbereich *Beim Formatieren*

» Seitenkopfbereich *Beim Formatieren*

» Seitenkopfbereich *Beim Drucken*

» Gruppenfuß0 *Beim Formatieren*

» Gruppenfuß0 *Beim Drucken*

» Detailbereich *Beim Formatieren*

» Detailbereich *Beim Drucken*

» Gruppenfuß0 *Beim Formatieren*

» Gruppenfuß0 *Beim Drucken*

» Seitenfußbereich *Beim Formatieren*

» Seitenfußbereich *Beim Drucken*

» Bericht *Bei Seite*

» Bericht *Beim Schließen*

» Bericht *Bei Deaktivierung*

5.4.5 Zugriff auf die Berichtsbereiche

Der Zugriff auf die Steuerelemente eines Berichts erfolgt genau wie in Formularen. Interessanter sind da die Elemente, die in Formularen nicht vorhanden sind – die einzelnen Bereiche der Berichte. Warum muss man eigentlich wissen, wie man per VBA auf diese Bereiche zugreift? Weil es Situationen gibt, in denen Sie beispielsweise einen Bereich ein- oder ausblenden oder die Eigenschaften eines Bereichs anpassen müssen.

Auf einen Berichtsbereich greifen Sie über die Auflistung *Section* zu. Als Argument geben Sie entweder den Namen, eine Zahl oder – bei den eingebauten Bereichen – eine VBA-Konstante an.

Auf den Detailbereich können Sie beispielsweise mit folgenden Anweisungen zugreifen:

```
Debug.Print Reports!<Berichtsname>.Section(0).Name
Debug.Print Reports!<Berichtsname>.Section(acDetail).Name
Debug.Print Reports!<Berichtsname>.Section("Detailbereich").Name
Debug.print Reports!<Berichtname>.Detailbereich.Name
```

Alle vier Anweisungen geben in der deutschen Version von Access 2007 den Namen »Detailbereich« aus. Dies ist der voreingestellte Name für den Detailbereich. Sie können den Namen der eingebauten Bereiche wie auch der zusätzlichen Gruppierungen leicht im Eigenschaftsfenster des jeweiligen Bereichs anpassen (siehe Abbildung 5.15).

Damit wird deutlich, dass Access für jeden Bereich dynamisch eine Berichts-Eigenschaft mit dem Namen des jeweiligen Bereichs bereitstellt.

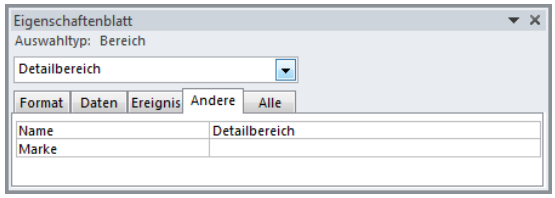


Abbildung 5.15: Anpassen des Namens eines Berichtsbereichs

Die Namen, Zahlenwerte und Konstanten der einzelnen Berichtsbereiche finden Sie in Tabelle 5.5: Die Kopf- und Fußbereiche der bis zu zehn Gruppierungsebenen werden von 5 bis 24 durchnummeriert, wobei die Kopfbereiche jeweils ungerade und die Fußbereiche gerade Zahlen erhalten.

Access legt automatisch Namen für die Bereiche an, die jeweils mit *Gruppenkopf* und *Gruppenfuß* beginnen und genau wie die anderen Steuerelemente eine eigene Nummer erhalten, die nicht zwangsläufig fortlaufend sein muss.

Name	Zahl	Konstante
Detailbereich	0	acDetail
Berichtskopf	1	acHeader
Berichtsfuß	2	acFooter
Seitenkopfbereich	3	acPageHeader
Seitenfußbereich	4	acPageFooter

Tabelle 5.5: Name, Zahlenwert und Konstanten von Berichtsbereichen

5.5 Beispiele für den Einsatz der Berichts- und Bereichsereignisse in der Seitenansicht

Die Berichtsereignisse lassen sich teilweise für recht spezielle Aktionen einsetzen. In den folgenden Abschnitten finden Sie einige Beispiele, damit Sie ein Gefühl für das richtige Einsetzen der Berichtsereignisse erhalten.

5.5.1 Beim Öffnen: Auswertung von Öffnungsargumenten

Der richtige Zeitpunkt zum Auswerten eines Öffnungsarguments ist das Ereignis *Beim Öffnen*. Mit dem Öffnungsargument der *DoCmd.OpenReport*-Anweisung lässt sich beispielsweise ein Parameter zum Filtern der Datensatzquelle übergeben.

Da dies aber relativ langweilig ist, finden Sie nachfolgend ein Beispiel, wie Sie die Gruppierung eines Berichts beim Öffnen verändern können. Voraussetzung ist der Bericht *rptGruppierungenTauschen* aus Abbildung 5.16.

Der Clou an diesem Bericht ist, dass Sie mit wenigen Zeilen Code die Gruppierungsebenen vertauschen können. Dazu sind folgende, in der Abbildung nicht sichtbare Eigenschaften einzustellen:

» Name des Beschriftungsfeldes im Berichtskopf: *lblUeberschrift*

» Name des *LieferantID*-Kopfbereichs: *Gruppenkopf0*

» Name des Beschriftungsfeldes im *LieferantID*-Kopfbereich: *lblUeberschrift Gruppierung0*

» Name des Textfeldes im *LieferantID*-Kopfbereich: *txtUeberschriftGruppierung0*

» Name des *KategorieID*-Kopfbereichs: *Gruppenkopf1*

» Name des Beschriftungsfeldes im *KategorieID*-Kopfbereich: *lblUeberschrift Gruppierung0*

» Name des Textfeldes im *KategorieID*-Kopfbereich: *txtUeberschriftGruppierung1*

Wie bringen Sie nun Dynamik ins Spiel? Betrachten Sie den reinen Ablauf, so rufen Sie den Bericht auf und übergeben mit dem Öffnungsargument Informationen über die im Bericht anzuzeigenden Daten. Das sieht etwa folgendermaßen aus (in einer Zeile):

```
DoCmd.OpenReport "rptGruppierungenTauschen", View:=acViewPreview, OpenArgs:="Artikel
nach Kategorien und Lieferanten;KategorieID;LieferantID;Kategorie;Lieferant"
```

Das Öffnungsargument enthält sogar mehrere Argumente, die durch Semikola voneinander getrennt sind. Das erste enthält den Text, der im Berichtskopf als Überschrift

angezeigt werden soll, das zweite und dritte enthalten die Beschriftungen der Bezeichnungsfelder in den beiden Gruppenköpfen und das vierte und fünfte die Felder der Datensatzquelle, nach denen gruppiert werden soll.

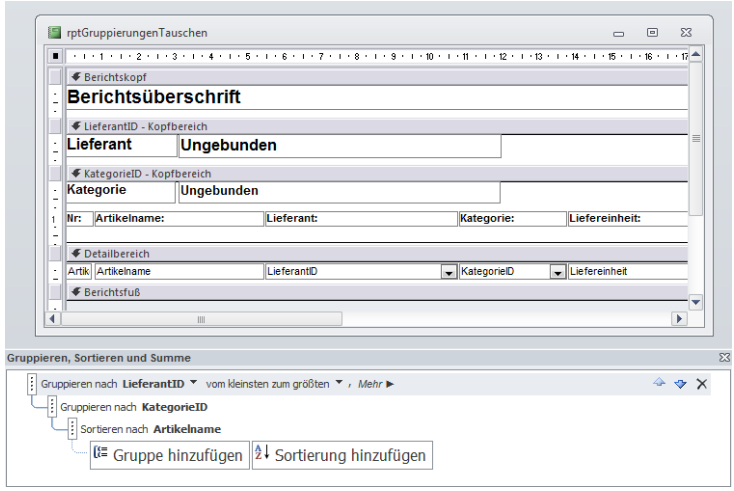


Abbildung 5.16: Bericht mit zwei Gruppierungsebenen

Fehlt noch eine Routine, die diese Informationen auseinandernimmt und den entsprechenden Eigenschaften zuweist. Diese wird – wer hätte es gedacht – durch das *Beim Öffnen*-Ereignis des Berichts ausgelöst:

```
Private Sub Report_Open(Cancel As Integer)
    Dim strOpenArgs As String
    Dim strGruppierungen() As String
    Dim i As Integer
    If IsNull(Me.OpenArgs) Then
        Exit Sub
    End If
    strGruppierungen() = Split(Me.OpenArgs, ";")
    Me!lblUeberschrift.Caption = strGruppierungen(0)
    For i = 1 To 2
        Me("lblUeberschriftGruppierung" & i - 1).Caption = _
            strGruppierungen(i + 2)
        Me.GroupLevel(i - 1).ControlSource = strGruppierungen(i)
    Next i
End Sub
```

Listing 5.1: Diese Prozedur sortiert die Gruppierungen nach den Vorgaben im Öffnungsargument

6

VBA

VBA ist als Basic-Dialekt eine strukturierte und sehr leicht lesbare sowie gut verständliche Programmiersprache.

Das hat Vor- und Nachteile: Es ist sehr einfach, etwas in Basic zu programmieren – selbst Programmiereinsteiger bringen hier schnell erste Resultate zu Stande. Das verleitet natürlich dazu, einfach drauflos zu programmieren – man benötigt nur eine Prozedur und schreibt dort alles hinein, was die aktuelle Aufgabe erfordert.

Auf diese Weise entstehen schnell endlos lange Prozeduren (sogenannter Spaghetti-Code), bei denen man in den letzten Zeilen schon nicht mehr weiß, welche Variablen man zu Beginn der Prozedur deklariert hat, und in denen es vielleicht sogar noch Sprungmarken gibt, zwischen denen munter hin- und hergesprungen wird. Wer seine

Aufgaben mit solchen oder ähnlichen Prozeduren löst, hat aber eines möglicherweise bereits geschafft: Er kennt sich ein wenig mit der Sprache aus.

BEISPIELDATENBANKEN

Den Download mit dem Beispielen finden Sie unter *www.acciu.de/aeb2010*. Die Datenbank zu diesem Abschnitt heißt *VBA.accdb*.

Das ist auch die Voraussetzung für dieses Kapitel: Es bietet keine grundlegende Einführung in die Sprache VBA, sondern setzt grundlegende Kenntnisse voraus. Sie sollten also schon wissen, wie eine Prozedur aufgebaut ist, wie Sie Variablen deklarieren, wie Sie diesen Variablen Werte zuweisen und wie die Konstrukte zum Verzweigen und zur Realisierung von Schleifen aussehen.

Dieses Kapitel soll vielmehr dabei helfen, die Grundkenntnisse ein wenig auszubauen, und vermitteln, wie Sie Code so strukturiert aufbauen, dass dieser leicht verständlich und damit leicht zu pflegen und zu erweitern ist.

6.1 VBA-Neuigkeiten in Access 2010

Access 2010 bringt zwar eine neue VBA-Version mit (von Version 6.5 auf Version 7); es gibt aber keine Änderungen am Sprachumfang der VBA-Bibliothek, sondern lediglich Anpassungen der Oberfläche an die neue Office-Version. Lediglich die 64-bit-Version von Access kommt mit einigen neuen Elementen.

Da die 64-bit-Version von Access 2010 aber nicht mit bestehenden 32-bit-Versionen etwa von ActiveX-Steuerelementen (zum Beispiel TreeView und ListView) und COM-Add-Ins zusammenarbeitet, gehen wir hier nicht auf diese Neuerungen ein.

Access 2010 liefert zwar Neues im VBA-Bereich, die betroffenen Methoden, Eigenschaften und Ereignisse beziehen sich aber auf andere Bibliotheken als die Access-Bibliothek.

So finden Sie etwa in der Bibliothek *Microsoft Access 14.0 Object Library* einige neue Elemente wie etwa die Methoden, Eigenschaften und Ereignisse der neuen Steuerelemente. Und auch in der Bibliothek *Microsoft Office 14.0 Database Engine Object Library* gibt es eine neue Eigenschaft namens *Expression* – damit greifen Sie auf den Berechnungsausdruck eines Tabellenfeldes zu.

Einen Überblick über die Neuerungen finden Sie in »Neues in Access 2010« ab Seite 29, dort finden Sie auch Verweise auf die Kapitel, in denen die Neuerungen behandelt werden.

6.2 Namenskonventionen in VBA

Routinen und Variablen sind die Elemente in VBA, denen der Programmierer nach eigenem Ermessen Namen zuteilen darf. Stopp: Ganz so beliebig geht es hier doch nicht zu. Immerhin gibt es ein paar Konventionen, die bei VBA zu berücksichtigen sind – man darf bestimmte Zeichen nicht verwenden und es dürfen keine Zahlen am Anfang eines Routinen- oder Variablennamens stehen.

Um nicht alle Ausnahmen aufzuzählen: Verwenden Sie nur Zahlen, Buchstaben und den Unterstrich (_) und beginnen Sie den Variablennamen mit einem Buchstaben, dann sind Sie auf der sicheren Seite (das gilt übrigens auch für Access-Objekte wie Tabellen, Abfragen und so weiter).

Auch bei der Wahl der Namen sollten Sie bestimmte Regelmäßigkeiten einhalten. Das hilft Ihnen zum einen, wenn Sie mal aus dem Kopf ein paar weiter oben in der Prozedur deklarierte Variablennamen abrufen möchten, zum anderen ist es nützlich, wenn Sie aus dem Namen einer Variablen oder einer Routine ableiten können, was diese enthält beziehungsweise bewirkt. Bei Variablen wäre es zudem interessant, nicht nur die Art des Inhalts, sondern auch den Variablentyp aus dem Namen ableiten zu können – spätestens hier stoßen Sie dann auf die sogenannte ungarische Notation. Diese stammt von Gregory Reddick und liefert einen Vorschlag für eine Konvention zur Benennung von Variablen. Die komplette Konvention finden Sie im Internet unter [1]. Nach dieser Konvention wird ein Variablenname wie folgt aufgebaut:

```
[prefixes]tag[BaseName[Suffixes]]
```

Dabei legt *Basename* den eigentlichen Inhalt der Variablen fest und *tag* ein Präfix, das den Typ der Variablen festsetzt (beispielsweise *str* für String, *obj* für Objekte, *frm* für Formulare). Der erste Teil *prefixes* enthält spezielle Informationen wie *m* für private Variablen (*member*) oder *g* für öffentliche Variablen (*global*). Der Teil *Suffixes* enthält nähere Informationen zum Inhalt der Variablen wie *Min* oder *Max* für Extremwerte. Die ungarische Notation ist quasi Standard bei der Benennung von Variablen- und Objektnamen. Fast jeder verwendet mehr oder weniger bewusst die hier festgelegten Regeln – manch einer übernimmt vermutlich intuitiv diese Konvention aus Code-Beispielen. Hier und da gibt es sicher Variationen (auch in diesem Buch), aber im Großen und Ganzen hilft diese Notation ganzen Heerscharen von Programmierern, zumindest die Variablen- und Objektnamen ihrer Kollegen zu verstehen.

Keine reservierten Wörter!

Wenn Sie keine Präfixe verwenden, sollten Sie zumindest darauf achten, dass Sie keine reservierten Wörter als Variablennamen verwenden – Begriffe wie *Name*, *Field*, *Links* etc. machen früher (wenn der Compiler sich meldet) oder später Ärger.

6.3 Layout von Code

VBA-Code muss gut lesbar und verständlich sein, damit Sie oder andere ihn leicht warten oder erweitern können. Dazu gehört nicht nur die eigentliche Struktur des Codes, sondern auch seine Darstellung oder das Layout. Man könnte hier nun einige Negativbeispiele anführen, aber dafür ist der Platz zu schade. Daher direkt einige Hinweise, die eine Grundlage für die gute Lesbarkeit des Codes liefern.

6.3.1 Funktionalität vor Schönheit?

Manch ein Programmierer mag nun denken: Eine Routine muss funktionieren und nicht schön aussehen. Tatsache ist aber: Eine Routine muss nicht nur funktionieren, sondern sie soll auch mal gelesen, geändert oder debugged werden, und wenn das dann ein anderer als der Urheber mit seinem geringen ästhetischen Empfinden machen muss, hat dieser sicher nicht gerade helle Freude daran. Tatsache ist, dass schon die Einrückungen bestimmter Teile des Codes diesen verständlicher machen. Allein das Einrücken des eigentlichen Inhalts von Routinen gegenüber der Routinendeklaration und der *End*-Zeile hilft beim schnellen Durchscrollen, das Ende der einen und den Beginn der nächsten Routine zu finden. Genauso verhält es sich mit Kontrollstrukturen wie Verzweigungen, Schleifen oder auch nur dem *With*-Statement.

6.3.2 Code einrücken zur Verdeutlichung der logischen Struktur

Wenn Sie Code einrücken, verwenden Sie dazu am besten die Tabulator-Taste. Im *Optionen*-Dialog der VBA-Entwicklungsumgebung können Sie Ihrem Geschmack entsprechend festlegen, wie viele Leerzeilen die Schrittweite des Tabulators umfassen soll (siehe Abbildung 6.1).

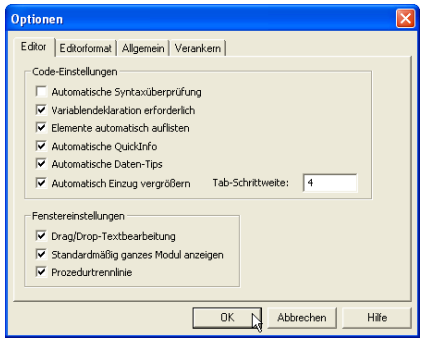


Abbildung 6.1: Optionen-Dialog der VBA-Entwicklungsumgebung

Typische Beispiele für die Einrückung sehen wie folgt aus:

```
Public Sub DatenLesen()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    On Error GoTo DatenLesen_Err
    Set db = CurrentDb
    Set rst = db.OpenRecordset("tblProjekte", dbOpenDynaset)
    Do While Not rst.EOF
        Debug.Print rst!Projekt
        rst.MoveNext
    Loop
DatenLesen_Exit:
    On Error Resume Next
    rst.Close
    Set rst = Nothing
    Set db = Nothing
DatenLesen_Err:
    MsgBox "Es ist ein Fehler aufgetreten!"
    Resume DatenLesen_Exit
End Sub
```

Listing 6.1: Beispiele für Einrückungen

Zunächst einmal sind alle Anweisungen außer der ersten und der letzten Zeile der Routine um einen Tabulatorschritt eingerückt. Innerhalb der *Do While*-Schleife erfolgt eine weitere Einrückung.

Die Sprungpunkte *DatenLesen_Exit* und *DatenLesen_Err* wiederum werden automatisch an den linken Rand gesetzt, damit man diese leicht identifizieren kann. Der geübte VBA-Programmierer sieht hier auf einen Blick: Der eigentliche Kern dieser Routine befindet sich zwischen der ersten Zeile und der ersten Sprungmarke *DatenLesen_Exit*. Andere Beispiele für Einrückungen sind (neben vielen anderen):

» *For...Next*-Schleife:

```
For i = 1 to 10
    'Inhalt der Schleife
Next i
```

» *If...Then...Else*-Verzweigung

```
If a=b then
    'Inhalt der Verzweigung
End If
```

» *Select Case*-Verzweigung

```
Select Case str
    Case "a"
        'Inhalt des Zweiges
    Case "b"
        'Inhalt des Zweiges
End Select
```

» Umbrochene Zeilen

```
MsgBox "Es ist schön, wenn man nicht scrollen muss, " _
    & "um eine Zeile komplett zu lesen.", _
    vbOkOnly + vbExclamation, "Sinnfreie Meldung"
```

Wenn Sie einmal mit dem Code anderer Entwickler oder selbst geschriebenem Code arbeiten müssen, der nicht sauber formatiert ist, sollten Sie dies nachholen – aber nicht von Hand. Es gibt praktische Tools, mit denen sich VBA-Code automatisch in eine ansprechende Form bringen lässt.

Ein Beispiel ist die Freeware *SmartIndenter*, die Sie unter [2] finden. Diese formatiert nicht stur nach Schema, sondern bietet auch einige Einstellungsmöglichkeiten zum Formatieren des Quellcodes. Um mal eben ein paar Zeilen gleichzeitig einzurücken, markieren Sie diese und betätigen dann die *Tab*-Taste (mit *Umschalt + Tab* verschieben Sie die Zeilen wieder nach links).

6.3.3 Leerzeilen für bessere Lesbarkeit

Neben Einrückungen sind Leerzeilen ein sinnvolles Mittel zur Verbesserung der Lesbarkeit des Codes.

Das Beispiel aus Listing 6.1 wäre beispielsweise viel leichter zu lesen, wenn Sie durch Leerzeichen für eine Gruppierung zusammenhängender Codezeilen sorgen.

Dies könnte so aussehen:

```
Public Sub DatenLesen()

    Dim db As DAO.Database
    Dim rst As DAO.Recordset

    On Error GoTo DatenLesen_Err

    Set db = CurrentDb
```

```
    Set rst = db.OpenRecordset("tblProjekte", dbOpenDynaset)

    Do While Not rst.EOF
        Debug.Print rst!Projekt
        rst.MoveNext
    Loop

DatenLesen_Exit:
    On Error Resume Next
    rst.Close
    Set rst = Nothing
    Set db = Nothing

DatenLesen_Err:
    MsgBox "Es ist ein Fehler aufgetreten!"
    Resume DatenLesen_Exit

End Sub
```

Listing 6.2: Optisches Strukturieren des Codes mit Leerzeilen

Mit den hier eingefügten Leerzeilen wächst die Lesbarkeit deutlich. Die Deklarationszeilen werden mit dem Prozedurkopf zusammengefasst und einige weitere Blöcke werden ebenfalls sinnvoll gruppiert. Wichtig ist bei der Verwendung von Leerzeilen, dass Sie diese nicht einsetzen, um Zeilen voneinander zu trennen, sondern um zusammenhängende Anweisungen zu gruppieren.

6.3.4 Zeilenumbrüche

Extrem lange Codezeilen sind im Quellcode eher selten, aber wenn sie dennoch auftauchen, sollten Sie diese umbrechen. So ersparen Sie dem Leser des Codes unnötiges Scrollen. Paradebeispiel für lange Zeilen sind die Deklarationen von API-Funktionen.

Ob man diese nun im Detail lesen muss, ist eine andere Frage, das Umbrechen schadet jedenfalls nicht. Wie weiter oben bereits erwähnt, sollten Sie Fortsetzungen umbrochener Zeilen einrücken, um diese als solche kenntlich zu machen.

In einem Fall ist sogar eine folgende Leerzeile angezeigt: Wenn der Prozedurkopf einen Zeilenumbruch erfordert, sollten Sie vor der folgenden Zeile eine Leerzeile einfügen. Die Fortsetzungszeile und die eingerückte Zeile lassen sich optisch sonst nur schwer auseinanderhalten. Es gibt zwei Varianten zum Umbrechen von Zeilen: innerhalb und außerhalb von Zeichenketten. Außerhalb einer Zeichenkette können Sie eine Zeile fast überall umbrechen, außer mitten in Schlüsselwörtern.

Dazu fügen Sie an der Stelle des gewünschten Umbruchs ein Leerzeichen, einen Unterstrich und einen Zeilenumbruch ein – und vergessen Sie das Einrücken nicht. Die folgenden zwei Zeilen enthalten ein Beispiel für eine Zeile mit allen möglichen Umbrüchen:

```
Set rst = db.OpenRecordset("tblProjekte", dbOpenDynaset)
```

Die extrem umbrochene Variante sieht so aus – lesbar ist der Code so natürlich nicht mehr, aber er zeigt die Möglichkeiten auf. Allerdings sind auch hier Grenzen gesetzt: Sie können eine Anweisung nicht auf mehr als 25 Zeilen aufteilen:

```
Set _
    rst _
    = _
    db _
    . _
    OpenRecordset _
    ( _
    "tblProjekte" _
    , _
    dbOpenDynaset _
    )
```

Die zweite Variante von Zeilenumbrüchen betrifft Zeichenketten innerhalb von Anführungszeichen. Diese lassen sich an mehr Stellen umbrechen, nämlich nach jedem Buchstaben. Allerdings sind die Regeln geringfügig umfangreicher.

Nehmen Sie die folgende Zeile als Ausgangspunkt:

```
MsgBox "Es ist ein Fehler aufgetreten!"
```

Der Zeilenumbruch erfolgt zum besseren Verständnis in zwei Schritten. Erst wird die Zeichenkette in zwei Teilzeichenketten aufgeteilt und verknüpft:

```
MsgBox "Es ist ein " & "Fehler aufgetreten!"
```

Dann fügen Sie wie oben einfach den Umbruch ein:

```
MsgBox "Es ist ein " _
    & "Fehler aufgetreten!"
```

Ob Sie den Umbruch vor oder nach dem *Und*-Zeichen durchführen, bleibt Ihnen überlassen. Wenn Sie einmal einen sehr langen Text in einer *String*-Variablen unterbringen möchten, können Sie diesen auch in mehreren Schritten zusammensetzen:

```
Dim str As String
str = "Erster Teil"
str = str & "Zweiter Teil"
```

6.3.5 Anweisungen zusammenfassen

Es gibt in VBA verschiedene Möglichkeiten, Anweisungen in einer Zeile zusammenzufassen. So können Sie beispielsweise die *If...Then*-Anweisung ohne *Else*-Teil in eine Zeile schreiben. Ausgangspunkt sind die folgenden Zeilen:

```
If a = 1 Then
    Debug.Print "A ist gleich 1"
End If
```

Daraus wird diese Variante:

```
If a = 1 Then Debug.Print "A ist gleich 1"
```

Und mehrere einzelne Anweisungen lassen sich durch einen Doppelpunkt getrennt in einer einzigen Zeile eingeben:

```
rst.Close: Set rst = Nothing
```

Diese Varianten sparen zwar Zeilen ein, übersichtlicher und lesbarer machen sie den Code aber nicht unbedingt. Außerdem gibt es Probleme, wenn Sie beim Dokumentieren von Fehlern die Zeilennummer mit einbeziehen – Sie können dann nicht erkennen, welche Anweisung den Fehler ausgelöst hat (weitere Informationen zum Ermitteln der Zeilennummer fehlerhafter Zeilen finden Sie in »Wichtige Fehlerinformationen« ab Seite 824).

6.4 Kommentare

Mit dem Hochkomma (') leiten Sie einen Kommentar ein. Kommentare können als komplette Zeilen oder als Anhängsel an bestehende Codezeilen eingesetzt werden. Seltener zum Einsatz kommt das Schlüsselwort *Rem*, das allerdings grundsätzlich am Anfang einer Zeile stehen muss.

Wenn Sie Kommentare in einer Routine verwenden, beziehen sich diese meist auf eine einzelne Zeile oder eine Gruppe von Zeilen. Setzen Sie den Kommentar direkt über die betroffenen Zeilen und rücken Sie den Kommentar genauso weit wie die kommentierte Zeile ein. Auf diese Weise machen Sie die Zugehörigkeit gut kenntlich:

```
Public Sub DatenLesen()
    …
    'Alle Datensätze durchlaufen und Projekte ausgeben
    Do While Not rst.EOF
        Debug.Print rst!Projekt
        rst.MoveNext
```

```
        Loop
        …
    End Sub
```

Listing 6.3: Kommentare rücken Sie am besten genauso weit wie die kommentierte Zeile ein

Kommentare wie im vorherigen Beispiel benötigen Sie normalerweise nicht. Die kommentierte *Do While*-Schleife verwendet jeder Access-Entwickler vermutlich täglich. Setzen Sie Kommentare nur dort ein, wo diese wirklich benötigt werden – etwa wenn Sie nicht triviale Methoden oder einen Trick verwenden, um zu einem bestimmten Ergebnis zu gelangen. Ein Beispiel für Kommentare im Anschluss an Zeilen ist der Deklarationsbereich. In vielen Fällen würde das Unterbringen aller benötigten Informationen zu lange Variablennamen erfordern – etwa wenn der Variableninhalt eine bestimmte Einheit hat:

```
Dim sngLaenge As Single 'Länge in Meter [m]
```

Weitere sinnvolle Einsatzmöglichkeiten sind folgende:

» Hinweise auf noch zu bearbeitende Code-Bereiche:

```
'ToDo: …
```

» Auskommentieren von Zeilen, etwa um eine alte Version nicht endgültig zu löschen, während man eine neue Variante ausprobiert

» Kommentierter Bereich im Kopf einer Routine, um deren Eingangs- und Ausgangsparameter, die enthaltene Funktionalität und weitere Informationen etwa zur Änderungshistorie anzugeben

Das Kommentieren und Auskommentieren von Code brauchen Sie übrigens nicht zeilenweise von Hand vorzunehmen. Die Menüleiste *Bearbeiten* (Menüeintrag *Ansicht|Symbolleisten|Bearbeiten*) liefert zwei Schaltflächen, mit denen Sie mehrere Zeilen gleichzeitig ein- und auskommentieren können (siehe Abbildung 6.2).

Abbildung 6.2: Menüleiste mit zwei Befehlen zum Kommentieren und Entkommentieren von VBA-Code

6.5 Konstanten

Konstanten bieten eine Möglichkeit, Werte, die zur Laufzeit nicht verändert werden, zentral zu speichern und durch einen aussagekräftigen Ausdruck zu ersetzen.

VBA und die in Access üblicherweise verwendeten Bibliotheken enthalten Hunderte, wenn nicht Tausende Konstanten. Schauen Sie sich allein die *MsgBox*-Anweisung an:

```
MsgBox "Meldungstext", vbOKCancel Or vbCritical Or vbDefaultButton1
```

Hinter den dort verwendeten Konstanten verbergen sich völlig harmlose Zahlenwerte. Ein gutes Einsatzgebiet für Konstanten in Ihren eigenen Anwendungen sind beispielsweise Optionsgruppen, deren Wert Sie nach der Auswahl in einer *Select Case*-Verzweigung auswerten (siehe Abbildung 6.3).

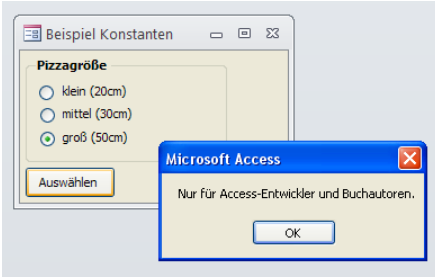


Abbildung 6.3: Optionsgruppe zur Anzeige von Detailinformationen

Die folgende Prozedur legt für die drei Optionen aussagekräftige Konstanten fest, die in der Routine *cmdAuswaehlen_Click* eingesetzt werden können.

```
Const pizKlein = 1
Const pizMittel = 2
Const pizGross = 3

Private Sub cmdAuswaehlen_Click()
    Select Case Me!ogrPizzagroesse
        Case pizKlein
            MsgBox "Für den kleinen Hunger zwischendurch."
        Case pizMittel
            MsgBox "Normale Größe."
        Case pizGross
            MsgBox "Nur für Access-Entwickler und Buchautoren."
    End Select
End Sub
```

Listing 6.4: Einsatz benutzerdefinierter Konstanten

Ein anderes sinnvolles Einsatzgebiet von Konstanten ergibt sich, wenn Sie bestimmte Kennzahlen im Code einsetzen – das gilt erst recht, wenn diese Kennzahlen an mehr als einer Stelle vorkommen.

Ein gutes Beispiel für D-Mark-Liebhaber wäre der Umrechnungsfaktor zwischen DM und Euro:

```
Public Const EUROFAKTOR = 1.95583
```

Andere Beispiele, die den Code lesbarer machen, sind folgende:

```
Public Const ANZAHL_MONATE = 12
Public Const ANZAHL_STUNDEN_PRO_TAG = 24
```

Auch oft verwendete Zeichenketten sollten Sie in Konstanten packen. Wenn Sie beispielsweise eine Anwendung entwickeln und sich über ihren Namen noch nicht im Klaren sind, diesen aber an mehreren Stellen ausgeben wollen, legen Sie einfach eine Konstante mit dieser Information an und verwenden diese an den entsprechenden Stellen statt des Variablennamens:

```
Public Const ANWENDUNGSNAME = "Beispieldatenbank VBA"
```

Sie können diese Konstante dann im Begrüßungsformular, Meldungsfenster und an anderen Orten anstatt des hart codierten Anwendungsnamens zuweisen und brauchen sie bei Bedarf nur an einer einzigen Stelle zu ändern. Konstanten können Sie auch nur innerhalb einer einzigen Prozedur verfügbar machen, indem Sie diese innerhalb der Prozedur deklarieren.

Konvention für Konstanten

Die ungarische Notation sieht für Konstanten die gleichen Regeln vor wie für Variablen. An den obigen Beispielen haben Sie schon erkennen können, dass diese Konvention hier keine Beachtung findet. Das ist wiederum Geschmackssache, aber aufgrund der sehr unterschiedlichen Anwendungszwecke verdienen Konstanten eine wesentlich auffälligere Notation als ein zwischengeschobenes »c«, wie in der ungarischen Notation vorgeschlagen.

Aufzählungstypen

Aufzählungstypen sind ein probates Mittel, Konstanten für eine Variable zur Verfügung zu stellen, deren Wertebereich bekannt und begrenzt ist. Dies ist vor allem für solche Werte interessant, die in oft genutzten Routinen als Parameter zum Einsatz kommen – die möglichen Werte werden dann durch *IntelliSense* angezeigt (siehe Abbildung 6.4).

Damit eine Routine eine solche Liste zur Verfügung stellt, verwenden Sie eine Enumeration und einen Funktionskopf wie im folgenden Listing:

```
Public Enum ePizzagroesse
    ePizzagroesse_Klein = 1
    ePizzagroesse_Mittel = 2
```

```
    ePizzagroesse_Gross = 3
End Enum

Public Function Teigmenge(intPizzagroesse As ePizzagroesse) As Integer
    Select Case intPizzagroesse
        Case ePizzagroesse_Klein
            Teigmenge = 500
        Case ePizzagroesse_Mittel
            Teigmenge = 750
        Case ePizzagroesse_Gross
            Teigmenge = 1250
    End Select
End Function
```

Listing 6.5: Bereitstellen einer Enumeration als Parameter einer Routine

Die Mitglieder einer Enumeration werden, wenn Sie keine expliziten Werte angeben, mit dem Wert *1* beginnend durchnummeriert, als Datentyp wird standardmäßig *Long* angenommen.

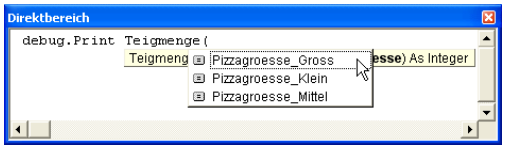


Abbildung 6.4: Einsatz einer Enumeration

6.6 Variablen

In diesem Abschnitt erfahren Sie, wie Sie Variablen benennen, wie Sie diese optimal einsetzen und welche speziellen Variablentypen es gibt. Das Wichtigste vorneweg: Sie sollten unbedingt darauf achten, dass Sie ausschließlich Variablen benutzen, die Sie auch deklariert haben. VBA ermöglicht auch den anderen Weg – nämlich einfach eine Variable einzusetzen, die man nicht deklariert hat.

Dann erhält diese automatisch den Datentyp *Variant* – und auch der Compiler meckert beim Kompilieren nicht, wenn die Variable nicht deklariert ist. Damit dies nicht passiert, fügen Sie zu Beginn eines Moduls die folgende Zeile ein:

```
Option Explicit
```

Wenn Sie nun eine nicht deklarierte Variable verwenden, schlägt Access beim Kompilieren des Moduls Alarm.

Damit man nicht unter Umständen mal vergisst, diese sehr wichtige Zeile an den Anfang des Moduls zu setzen, stellt VBA eine Option bereit, mit der diese Zeile standardmäßig beim Anlegen eines neuen Moduls hinzugefügt wird: Öffnen Sie dazu den *Optionen*-Dialog (*Extras|Optionen*) und aktivieren Sie dort die Option *Variablendeklaration erforderlich* im Bereich *Code-Einstellungen* der Registerseite *Editor.*

6.6.1 Variablennamen

Variablennamen sollten zunächst einmal den in »Namenskonventionen in VBA« ab Seite 417 genannten Konventionen entsprechen. Noch nicht besprochen wurde dort der Teil zwischen Präfix und Suffix, also der eigentlich wichtigste Teil. Dieser Teil kann aus einem Wort oder mehreren Wörtern bestehen. Schreiben Sie jedes neue Wort groß, damit man es als neues Wort erkennen kann, etwa *intAnzahlMitarbeiter*. Gestalten Sie den Variablennamen so lang wie nötig und so kurz wie möglich, allerdings ohne mit wilden und nicht nachvollziehbaren Abkürzungen zu arbeiten. So ist *sngMwStSatz* statt *sngMehrwertsteuersatz* sicher sinnvoll, aber *sngMS* ist ein wenig kurz und selbst im richtigen Zusammenhang schwer zu deuten. Gleichzeitig sollten Sie eine Variable so benennen, dass man unmittelbar erkennen kann, was diese Variable für einen Wert enthält. Beispiele:

» *strSQL*

» *datAktuellesDatum*

» *curBetrag*

» *rstMitarbeiter*

Diese Variablen weisen noch einen weiteren Vorteil auf: Sie lassen sich allesamt leicht einprägen. Das ist besonders wichtig, denn Sie wollen sicher nicht bei jeder Anwendung einer Variablen zum Deklarationsbereich scrollen, um die genaue Schreibweise dieser Variablen zu ermitteln.

Zahlen in Variablennamen

Wenn Sie in einer Routine Variablennamen wie *strMitarbeiter1*, *strMitarbeiter2* oder ähnliche finden, sollten Sie das Design des Codes überprüfen. Entweder ließe sich hier besser ein Array verwenden oder es handelt sich tatsächlich um zwei verschiedene Variablen, die aber mit wesentlich aussagekräftigeren Namen ausgestattet werden sollten.

6.6.2 Spezielle Variablennamen

Es haben sich einige Variablennamen eingebürgert, die der weiter oben erwähnten Konvention widersprechen. Dennoch werden sie immer wieder benutzt – eben weil sie gängig sind.

Lauf- oder Zählervariablen

Das beste Beispiel ist sicher die Variable *i* als Zählervariable. Wer nicht mit der Konvention brechen möchte, mag vielleicht eine Variante wie *intZaehler* verwenden – das ist letzten Endes Geschmackssache.

Ein alternativer Name für eine Laufvariable ist definitiv angezeigt, wenn dieser Wert beispielsweise anschließend weiter verwendet wird. Auch wenn es sich um Laufvariablen in verschachtelten Schleifen handelt, sollte man über eine andere Benennung als *i* und *j* nachdenken.

Temporäre Variablen

Temporäre Variablen verdienen meist einen aussagekräftigeren Namen als *temp* oder *tmp*. Man könnte zumindest die Bezeichnung dessen, was sie beinhalten, vorne anfügen, dann hieße eine temporäre Variable beispielsweise *rstMitarbeiterTemp*.

Statusvariablen

Wenn Sie eine Variable verwenden, um einen Status zu speichern, sollten Sie auch dieser einen brauchbaren Namen geben. Viele Entwickler nennen solche Variablen lieblos *Flag*.

Spätestens, wenn Sie mal zwei »Flags« in einer Routine oder in einem Modul benötigen, müssen Sie sich zwei unterschiedliche Namen ausdenken – und dann tun Sie sich selbst den Gefallen und nennen diese nicht *Flag1* und *Flag2*.

6.6.3 Arrays

Arrays sind Datenfelder zum Speichern mehrerer Daten gleichen Datentyps. Diese Datenfelder können auch mehrdimensional ausgelegt werden. Sie deklarieren ein Datenfeld entweder direkt mit der gewünschten Anzahl möglicher Werte oder lassen diesen Parameter offen:

```
Dim strVornamen() As String
```

oder

```
Dim strVornamen(10) As String
```

In beiden Fällen können Sie die Anzahl später noch mit der *ReDim*-Anweisung ändern. Diese gibt es in zwei Ausführungen: Ohne das Schlüsselwort *Preserve* werden alle enthaltenen Daten gelöscht, mit diesem Schlüsselwort behält das Array die vorhandenen Daten bei, was normalerweise gewünscht sein dürfte:

```
ReDim Preserve strVornamen(15) As String
```

7

Access-SQL

SQL (*Structured Query Language*, strukturierte Abfragesprache) ist eine weitgehend standardisierte Abfragesprache für relationale Datenbanken.

SQL dient der Auswahl und der Manipulation von Daten aus den Tabellen einer relationalen Datenbank und dem Entwerfen des Datenmodells.

Der Sprachumfang von Access-SQL ist in zwei Bereiche unterteilt: Die *Data Manipulation Language* (*DML*) liefert die Befehle zum Auswählen und Manipulieren von Daten, die *Data Definition Language* (*DDL*) die Anweisungen zum Erstellen, Bearbeiten und Löschen der Elemente des Datenmodells selbst. Die Befehle der DML finden Sie unter »Daten auswählen« ab Seite 455 und »Daten manipulieren« ab Seite 478 und die der DDL unter »Datenmodell erstellen und manipulieren« ab Seite 481.

7.1 SQL-Versionen

Die für den Zugriff auf Access-Datenbanken verwendete Sprache wird in diesem Buch Access-SQL genannt. Access-SQL bietet eine Mischung aus den Standards SQL-89, SQL-92 und einigen Access-spezifischen Erweiterungen. So lassen sich unter Access beispielsweise VBA-Ausdrücke in SQL-Anweisungen integrieren. Damit ist etwa die Verwendung von Standardfunktionen oder Bezügen auf Formulare und Steuerelemente möglich. Die Bestandteile aus SQL-92 werden nur an bestimmten Stellen unterstützt. In den folgenden Abschnitten finden Sie hauptsächlich die Möglichkeiten von Access-SQL, die sowohl in der SQL- und Abfrage-Entwurfsansicht als auch unter DAO und ADO ohne Einschränkung verwendet werden können.

Soweit die Erweiterungen die hier beschriebenen Funktionen von SQL betreffen, finden Sie ihre Erläuterung und einen entsprechenden Hinweis auf die Version.

Für den Zugriff auf andere Datenbanksysteme etwa über PassThrough-Abfragen müssen Sie den SQL-Dialekt des jeweiligen Systems verwenden. So erwartet der Microsoft SQL Server beispielsweise nicht das Sternchen (*), sondern das Prozentzeichen (%) als Platzhalter.

Um die erst unter SQL-92 enthaltenen Funktionen einsetzen zu können, müssen Sie eine Einstellung in den Optionen der Datenbank vornehmen (siehe Abbildung 7.1). Diese Änderung hat weit reichende Folgen: Unter anderem ändert sich die Syntax in einigen Punkten – zum Beispiel werden die unter Access verwendeten Platzhalter Sternchen (*) und Fragezeichen (?) nicht mehr unterstützt, sondern das Prozentzeichen (%) und der Unterstrich (_) verwendet.

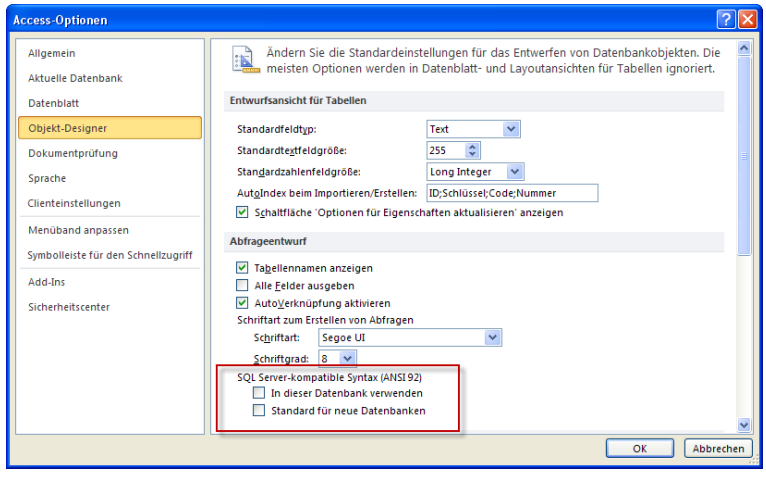


Abbildung 7.1: Aktivieren der SQL-92-Erweiterungen

Sie können die Erweiterungen auch ohne Ändern dieser Einstellung verwenden, allerdings nicht an allen Stellen. Das Einsatzgebiet beschränkt sich dann auf die Anwendung in VBA-Code und dort auf die ADO-Objektbibliothek. Sie finden an entsprechender Stelle eine Beispielprozedur für den Aufruf von SQL-Anweisungen mit SQL-92-spezifischen Sprachelementen.

BEISPIELDATENBANKEN

Den Download mit dem Beispielen finden Sie unter *www.acciu.de/aeb2010*. Die Datenbank zu diesem Abschnitt heißt AccessSQL.accdb.

7.2 SQL und Access

SQL wird in Access an den verschiedensten Stellen eingesetzt – teilweise völlig unbemerkt. Wer mit einem Grundlagenbuch in Access einsteigt, wird unter Umständen erst in den hinteren Kapiteln – wenn überhaupt – mit SQL in Berührung kommen. Und wenn man es darauf anlegt, kann man sich eine ganze Weile um die erste selbst geschriebene SQL-Anweisung herumdrücken. Schließlich bietet Access mit der Abfrage-Entwurfsansicht ein ausgezeichnetes Hilfsmittel für die Erstellung von Abfragen (siehe Abbildung 7.2).

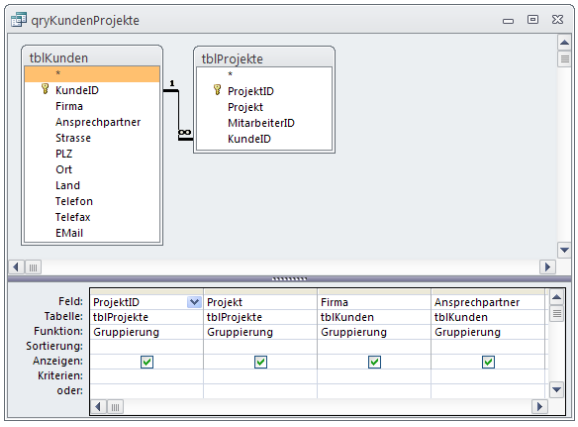


Abbildung 7.2: Die Entwurfsansicht zur Erstellung von Abfragen

Dieses Werkzeug ermöglicht es, die in einer Abfrage anzuzeigenden Tabellen und Felder auszuwählen, Kriterien direkt oder als Parameter anzugeben, Sortierungen und Gruppierungen festzulegen und Funktionen zur Summierung, Mittelwertbildung, Ermittlung von Extremwerten oder Anzahlen anzuwenden. Wer bis dato noch keinen Kontakt mit

SQL hatte und sich darum vielleicht auch keine größeren Gedanken gemacht hat, ist möglicherweise überrascht, dass alles, was man in der Abfrage-Entwurfsansicht anstellt, in die Erstellung einer SQL-Anweisung mündet. Beweis gefällig? Dann wählen Sie doch einfach aus dem Kontextmenü des Abfrageentwurfs den Eintrag *SQL-Ansicht* aus – das nun erscheinende Fenster offenbart das wahre Aussehen der Abfrage (siehe Abbildung 7.3).

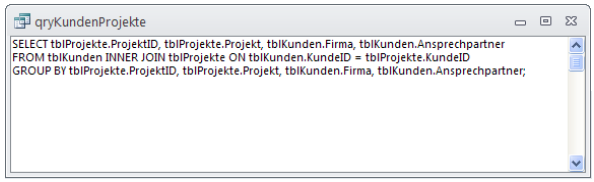


Abbildung 7.3: Abfrage in der SQL-Ansicht

Die Schriftart dieses Fensters und somit auch des Entwurfsrasters der Entwurfsansicht von Abfragen können Sie in den Access-Optionen unter *Objektdesigner|Abfrageentwurf|Schriftart zum Erstellen von Abfragen* anpassen.

7.2.1 Wozu trotz Abfrage-Entwurfsansicht SQL lernen?

Warum sollten Sie sich eigentlich mit SQL beschäftigen, obwohl die Entwurfsansicht für Abfragen so eine Erleichterung beim Erstellen von Abfragen ist? Dafür gibt es mehrere Gründe:

» Nicht jede Abfrage lässt sich in der Entwurfsansicht darstellen, darunter UNION-Abfragen, Datendefinitions-Abfragen und PassThrough-Abfragen.

» Die Erstellung einer Abfrage per SQL-Code geht manchmal einfach schneller. Die Eingabe der folgenden Zeile im Direktfenster bekommt manch einer fixer hin, als die entsprechende Tabelle zu öffnen, alle Datensätze zu markieren, den Ribboneintrag *Entwurf|Abfragetyp|Löschen* auszuwählen und die Tabelle wieder zu schließen:

```
CurrentDB.Execute "DELETE FROM tblKunden"
```

» Gelegentlich ist es aus Performancegründen sinnvoll, eine SQL-Abfrage in den VBA-Code einzuarbeiten und von dort auszuführen – beispielsweise, wenn sich die in den betroffenen Tabellen enthaltenen Daten oft ändern und der Vorteil einer kompilierten und optimierten gespeicherten Abfrage nicht mehr vorhanden ist (siehe »Gespeicherte Abfragen versus Ad-hoc-Abfragen« ab Seite 847).

» Manchmal ist es auch unabwendbar, eine SQL-Anweisung im VBA-Code zusammenzusetzen – beispielsweise, weil die Abfrage das Ergebnis einer Suche liefern soll, die je nach Kriterien Felder aus wechselnden Tabellen enthält.

7.2.2 Wo lässt sich SQL überall einsetzen?

Von Hand erstellte SQL-Ausdrücke lassen sich praktisch an allen Stellen einsetzen, an denen auch Tabellen oder Abfragen als Datenquelle angegeben werden können. Aber auch »normale« Abfragen lassen sich statt über die Entwurfsansicht über die SQL-Ansicht eingeben und können anschließend wie gewohnt über die Entwurfsansicht angepasst werden. Diese Vorgehensweise bietet sich an, wenn Sie im VBA-Code eine SQL-Abfrage zusammensetzen, die ihren Dienst verweigert. Sie können sich dann die entsprechende Zeichenkette im Direktfenster ausgeben lassen, den SQL-Ausdruck in die SQL-Ansicht einer Abfrage eingeben und dann nach Fehlern suchen. SQL-Ausdrücke lassen sich an folgenden Stellen verwenden:

- SQL-Ansicht von Abfragen
- Datensatzquelle von Formularen
- Datensatzquelle von Berichten
- Datensatzherkunft von Kombinationsfeldern
- Datensatzherkunft von Listenfeldern
- Datensatzherkunft von Nachschlagefeldern in Tabellen
- Makro *AusführenSQL*
- *RunSQL*-Anweisung des *DoCmd*-Objekts in VBA
- *Execute*-Anweisung des *Database*-Objekts (ADO) in VBA
- *Execute*-Anweisung des *Connection*-Objekts (ADO) in VBA
- *Source*-Eigenschaft des *Recordset*-Objekts (ADO) in VBA
- *CommandText* des *Command*-Objekts (ADO) in VBA

7.3 Daten auswählen

SQL-Abfragen dienen der Auswahl von Daten aus einer oder mehreren Tabellen. Dabei können Sie sowohl die Anzahl auszugebender Felder als auch der auszugebenden Datensätze einschränken. Die einfachste Form einer Auswahlabfrage liefert den kompletten Inhalt einer Tabelle wie im folgenden Beispiel:

```
SELECT * FROM tblKunden
```

Das Ergebnis dieser Abfrage entspricht genau dem Bild, das Sie auch beim Öffnen einer Tabelle erhalten. Es enthält alle Felder und alle Datensätze des Originals.

Um die Datenblatt-Ansicht dieser und der folgenden SQL-Beispiele anzusehen, gehen Sie folgendermaßen vor:

» Klicken Sie im Ribbon auf den Eintrag *Erstellen|Andere|Abfrageentwurf*.

» Schließen Sie den Dialog *Tabelle anzeigen*, ohne eine Tabelle auszuwählen.

» Wählen Sie im nun aktiven Ribbon-Tab namens *Entwurf* die Schaltfläche *SQL* aus (siehe Abbildung 7.4).

» Geben Sie im nächsten Fenster den SQL-Ausdruck ein und wählen Sie anschließend statt der SQL-Ansicht die Datenblatt-Ansicht aus oder klicken Sie auf die *Ausführen*-Schaltfläche im Ribbon.

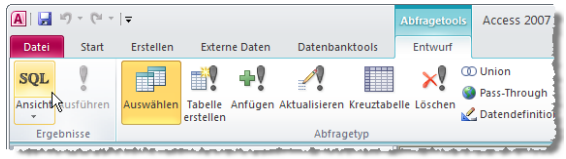


Abbildung 7.4: Auswählen der SQL-Ansicht einer Abfrage

Wie Sie bereits gesehen haben, besteht die einfachste Form einer SQL-Abfrage aus mindestens zwei Teilen: Die *SELECT*-Klausel leitet den ersten Teil ein und enthält die Auflistung der auszuwählenden Felder. Die FROM-Klausel legt fest, aus welcher beziehungsweise welchen Tabellen die Daten stammen. Die weiteren Teile einer Abfrage dienen dem Einschränken (*WHERE*-Klausel) und Sortieren des Ergebnisses (*ORDER BY*-Klausel). Diese beiden Klauseln sind optional.

7.3.1 Festlegen der anzuzeigenden Felder

Der erste Teil einer Abfrage, der durch die *SELECT*-Klausel eingeleitet wird, legt fest, welche Felder die Ergebnismenge enthält. Die Felder werden im Anschluss an das *SELECT*-Schlüsselwort aufgelistet und durch Kommata voneinander getrennt. Unter Umständen schiebt man noch einige Elemente zwischen *SELECT* und Feldliste – dazu später mehr. Die folgende Abfrage gibt die drei Felder *KundeID* und *Firma* der Tabelle *tblKunden* aus:

```
SELECT KundeID, Firma FROM tblKunden;
```

Eine etwas ausführlichere Variante würde lauten:

```
SELECT tblKunden.KundeID, tblKunden.Firma FROM tblKunden;
```

Die Angabe des Tabellennamens vor dem jeweiligen Feld ist sinnvoll, wenn die Abfrage ihre Daten aus mehreren Tabellen bezieht, und Pflicht, wenn sich in den zugrunde lie-

genden Tabellen mehrere Felder gleichen Namens befinden. Wenn Sie einen SQL-Ausdruck ohne Angabe der Tabelle zusammen mit den Feldnamen in der SQL-Ansicht einer Abfrage eingeben, anschließend zur Entwurfsansicht wechseln und von dort aus die Abfrage speichern, enthält die SQL-Ansicht beim nächsten Anzeigen jeweils den zu den Feldern gehörenden Tabellennamen. Wenn Sie andersherum die Entwurfsansicht zum Erstellen der Abfrage verwenden, enthält die SQL-Ansicht immer die vollständigen Feldnamen – also mit Tabellennamen.

Alle Felder einer Tabelle ausgeben

Um die Inhalte aller Felder einer Tabelle zu ermitteln, geben Sie entweder alle Felder explizit an oder verwenden das Sternchen (*) an Stelle der Auflistung der Felder:

```
SELECT * FROM tblKunden;
```

oder ausführlicher:

```
SELECT tblKunden.* FROM tblKunden;
```

Sonderzeichen in Tabellen- und Feldnamen

Vom Gebrauch von Sonderzeichen in Tabellen- und Feldnamen ist grundsätzlich abzuraten. Manchmal lässt sich das aber nicht verhindern, weil man etwa eine bestehende Datenbank weiterentwickelt, bei der das Kind schon in den Brunnen gefallen ist und eine nachträgliche Änderung zu aufwändig wäre. In diesem Fall fassen Sie den Feldnamen in eckige Klammern ein:

```
SELECT [Kunde-ID], Firma FROM tblKunden;
```

Auch hier die ausführliche Variante mit Tabellennamen:

```
SELECT tblKunden.[Kunde-ID], tblKunden.Firma FROM tblKunden;
```

Feldnamen ersetzen

Wenn Sie für einen bestimmten Zweck einen anderen Feldnamen in der Ausgabe der Abfrage wünschen, verwenden Sie das *AS*-Schlüsselwort, um einem Feld einen alternativen Namen zuzuweisen:

```
SELECT [Kunde-ID] AS KundeID, Firma AS Kunde FROM tblKunden;
```

Wie Sie sehen, lassen sich die unbequemen Feldnamen mit Sonderzeichen auf diese Weise zumindest für den weiteren Gebrauch auf Basis des SQL-Ausdrucks umbenennen. Auch für Felder, die Sie aus anderen Feldern berechnen oder zusammensetzen, verwenden Sie das *AS*-Schlüsselwort:

```
SELECT MitarbeiterID, Nachname & ", " & Vorname AS Mitarbeiter FROM tblMitarbeiter;
```

Das Ergebnis dieser Abfrage sieht wie in Abbildung 7.5 aus.

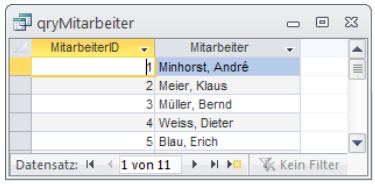


Abbildung 7.5: Abfrage mit zusammengesetztem Feld

7.3.2 Festlegen der enthaltenen Tabellen

Neben den Feldern, die ein SQL-Ausdruck enthält, müssen Sie natürlich auch noch die Tabellen angeben, aus denen die Felder stammen. Das gilt übrigens nicht nur für die angezeigten Felder, sondern auch für die Felder, die als Kriterien- oder Sortierfeld dienen. Die Tabellen listet man unmittelbar hinter der *FROM*-Klausel auf.

Bei einer einzigen Tabelle ist der Ausdruck noch recht übersichtlich:

```
SELECT MitarbeiterID, Vorname, Nachname FROM tblMitarbeiter;
```

Wenn die Daten aber aus mehreren Tabellen stammen, reicht es unter Umständen nicht aus, einfach nur die beteiligten Tabellen aufzulisten. Der einzige Fall, in dem Sie so verfahren, liefert alle Kombinationen der Felder aus den angegebenen Tabellen:

```
SELECT tblMitarbeiter.MitarbeiterID, tblMitarbeiter.Vorname, tblMitarbeiter.Nachname,
tblProjekte.ProjektID, tblProjekte.Projekt FROM tblMitarbeiter, tblProjekte;
```

Dieser SQL-Ausdruck würde alle Kombinationen der Datensätze der Tabelle *tblMitarbeiter* und der Tabelle *tblProjekte* ausgeben. Normalerweise wird man nur bestimmte Kombinationen aus Projekten und Mitarbeitern anzeigen wollen – etwa einen Mitarbeiter, der in der Projekt-Tabelle als Projektleiter eingetragen ist.

Um einen derartigen Zusammenhang festzulegen, verwendet man entweder eine *JOIN*-Klausel, die Informationen über eine Verknüpfung zwischen den beteiligten Tabellen angibt, oder eine *WHERE*-Klausel, die festlegt, welche Felder der beiden Tabellen übereinstimmen müssen, damit eine Kombination ausgegeben wird.

Vereinfachen der Schreibweise

Um die Schreibweise und Übersichtlichkeit von SQL-Ausdrücken mit mehreren Tabellen zu vereinfachen, können Sie den Tabellennamen auch alternative Bezeichnungen zuweisen. Diese werden dann im Rest des SQL-Ausdrucks verwendet. Im folgenden Beispiel soll die Tabelle *tblMitarbeiter* die Bezeichnung *t1* erhalten und die Tabelle *tblProjekte* die Bezeichnung *t2*:

```
SELECT t1.MitarbeiterID, t1.Vorname, t1.Nachname, t2.ProjektID, t2.Projekt FROM tblMit-
arbeiter AS t1, tblProjekte AS t2;
```

In der Entwurfsansicht stellen Sie einen alternativen Namen übrigens über die Eigenschaft *Alias* ein (siehe Abbildung 7.6).

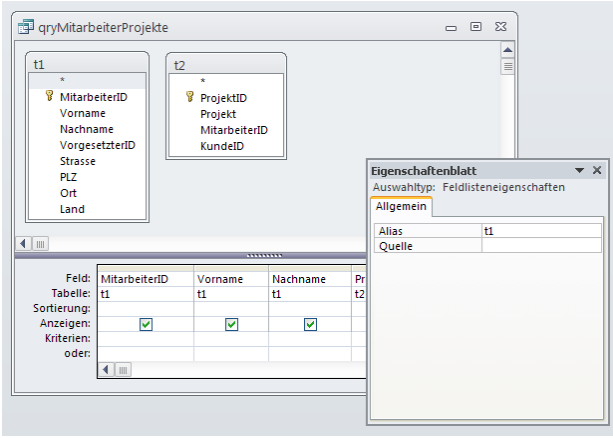


Abbildung 7.6: Eingeben eines Alias-Namens für eine Tabelle

7.3.3 Festlegen von Bedingungen

Bisher haben Sie nur die Felder ausgewählt, die Sie im Abfrageergebnis der SQL-Abfrage anzeigen möchten. Nun legen Sie fest, welche Datensätze angezeigt werden sollen. Dazu verwenden Sie die *WHERE*-Klausel von SQL.

Die *WHERE*-Klausel enthält einen oder mehrere Ausdrücke, die durch *AND* oder *OR* voneinander getrennt sind. Damit können Sie etwa alle Datensätze der Tabelle *tblMitarbeiter* herausfinden, bei denen das Feld *MitarbeiterID* den Wert *1* enthält:

```
SELECT * FROM tblMitarbeiter WHERE MitarbeiterID = 1;
```

Mit Hilfe der *OR*-Verknüpfung können Sie Datensätze zurückgeben, die mindestens eine von mehreren Bedingungen erfüllen:

```
SELECT * FROM tblMitarbeiter WHERE MitarbeiterID = 1 OR MitarbeiterID = 2;
```

Mit der *AND*-Verknüpfung schränken Sie die zurückgegebenen Datensätze so ein, dass diese alle aufgeführten Bedingungen erfüllen:

```
SELECT * FROM tblMitarbeiter WHERE Vorname = 'Klaus' AND Nachname = 'Wild';
```

Wenn der SQL-Ausdruck mehr als einen Verknüpfungsoperator enthält, können Sie die Reihenfolge der Abarbeitung durch das Setzen von Klammern beeinflussen.

7.3.4 Vergleichsausdrücke

Bei der Zusammenstellung der Vergleichsausdrücke sind der Fantasie praktisch keine Grenzen gesetzt. In den folgenden Abschnitten finden Sie die Grundlagen und die wichtigsten Möglichkeiten für Vergleichsausdrücke.

Operatoren

Für Vergleiche stellt SQL die folgenden Operatoren zur Verfügung:

» gleich (=)

» größer als (>)

» größer als oder gleich (>=)

» kleiner als (<)

» kleiner als oder gleich (<=)

» ungleich (<>)

» Vergleichsumkehr (*NOT*)

» zwischen zwei Werten (*BETWEEN ... AND*)

» Vergleich mit Platzhaltern wie Sternchen (*, beliebig viele Zeichen) oder Fragezeichen (?, ein Zeichen): *LIKE*

» Vergleich mit Null-Werten: *IS NULL*

» Vergleich mit den Werten einer Menge: *IN*

Vergleiche mit Zahlen

Wenn Sie einen der obigen Vergleichsoperatoren für den Vergleich mit Zahlen verwenden, können Sie alle Varianten außer Vergleichen mit Platzhaltern einsetzen.

Beispiele:

```
SELECT * FROM tblArtikel WHERE Preis > 10;
SELECT * FROM tblArtikel WHERE Preis BETWEEN 5 AND 15;
```

Geben Sie Zahlen immer ohne Formatierungen wie Prozentzeichen (%), Währungsangaben (€) oder dergleichen ein.

Verwenden Sie nur die nackten Zahlen als Vergleichsausdrücke und setzen Sie den Punkt als Dezimaltrennzeichen:

```
SELECT * FROM tblArtikel WHERE Mehrwertsteuer IN (0.7, 0.16);
```

Vergleiche mit Zeichenketten

Wichtig für die Verwendung von Zeichenketten ist vor allem, dass Sie den Vergleichswert in einfache (') oder doppelte (") Anführungszeichen einschließen.

Probleme werden Sie jedoch bekommen, wenn innerhalb der Zeichenkette selbst Anführungszeichen enthalten sind, weil der SQL-Interpreter diese dann als Trennzeichen der Zeichenkette interpretiert.

Um auf der sichereren Seite zu sein, sind doppelte Anführungszeichen meist günstiger. Dann gibt es auch keine Fehlermeldung, wenn ein Name wie »D'Hondt« ausgewertet wird. Weiterhin sind alle Vergleichsoperationen möglich.

Am sinnvollsten ist jedoch der *LIKE*-Vergleich, da er die Verwendung von Platzhaltern wie Sternchen (*), das stellvertretend für beliebig viele Zeichen steht, und Fragezeichen (?), das stellvertretend für je ein Zeichen steht, unterstützt.

Die folgenden Beispiele verdeutlichen dies:

» Alle Mitarbeiter, deren Vorname mit A beginnt:

```
SELECT * FROM tblMitarbeiter
WHERE Vorname LIKE "A*";
```

» Alle Mitarbeiter, deren Nachname die Zeichenfolge EI enthält:

```
SELECT * FROM tblMitarbeiter
WHERE Nachname LIKE "*EI*";
```

» Alle Mitarbeiter, deren Vorname mit einem Buchstaben von A–J beginnt:

```
SELECT * FROM tblMitarbeiter
WHERE Vorname >= "A" AND Vorname < "K";
```

Vergleiche mit Datumsangaben

Datumsangaben sind ein fehlerträchtiges Gebiet – vor allem in Abfragen. Auf Nummer sicher gehen Sie mit den folgenden beiden Datumsformaten:

» Amerikanisches Datumsformat: *#mm/dd/yyyy#*

» ISO-Datumsformat: *#yyyy/mm/dd#*

Verwenden Sie eines dieser Formate als Grundlage für den Vergleichswert. Gehen Sie kein Risiko ein. Lassen Sie sich auch nicht davon irritieren, dass das Datum in der Tabelle ganz anders eingegeben und angezeigt wird.

Von dieser Feinheit abgesehen können Sie für Vergleiche mit dem Datum alle Vergleichsvarianten mit Ausnahme des Vergleichs mit Platzhaltern heranziehen.

Beispiele:

```
SELECT * FROM tblBestellungen WHERE Bestelldatum = #2005/05/15#;

SELECT * FROM tblBestellungen WHERE Bestelldatum In (#5/5/2005#, #5/7/2005#);

SELECT * FROM tblBestellungen WHERE Lieferdatum BETWEEN #2005/5/10# AND #2005/5/15#;
```

Vergleiche mit dem Null-Wert

Der *Null*-Wert besitzt eine besondere Funktion: Liefert ein Vergleich eines Feldes mit dem Wert *Null* den Wert *True*, ist das Feld leer. Das folgende Beispiel liefert alle Mitarbeiter, die keine E-Mail-Adresse haben:

```
SELECT * FROM tblMitarbeiter WHERE Email IS NULL;
```

Umgekehrt ergibt das nächste Beispiel alle Mitarbeiter, die eine E-Mail-Adresse besitzen:

```
SELECT * FROM tblMitarbeiter WHERE NOT EMail IS NULL;
```

Vergleiche mit Funktionen

Access-SQL bietet die Möglichkeit, auch Funktionen als Vergleichswert zu verwenden. Eine oft als Vergleichswert verwendete Funktion ist *Date()*. Die folgende Abfrage ermittelt alle Bestellungen, deren Lieferdatum mit dem aktuellen Datum übereinstimmt:

```
SELECT * FROM tblBestellungen WHERE Lieferdatum = DATE();
```

Auch berechnete Ausdrücke sind erlaubt. Alle Bestellungen der letzten 30 Tage ermitteln Sie folgendermaßen:

```
SELECT * FROM tblBestellungen WHERE Bestelldatum = DATE() - 30;
```

Standard-SQL bietet einige Funktionen wie etwa die Aggregatfunktionen *Sum*, *Count*, *Max* oder *Min* (siehe »Aggregatfunktionen« ab Seite 463). Unter Access-SQL lassen sich wie im Beispiel auch eingebaute VBA-Funktionen, benutzerdefinierte Funktionen und Verweise auf Objekte verwenden. Letzteres ist vor allem im Zusammenhang mit der Einbindung der Steuerelementinhalte von Formularen interessant.

7.3.5 Sortieren von Daten

Die *ORDER BY*-Klausel dient der Angabe der Sortierreihenfolge für beliebig viele Felder. Nach der *ORDER BY*-Klausel, die sich übrigens immer am Ende der Abfrage befindet, geben Sie den Namen eines oder mehrerer Felder an, nach denen das Abfrageergebnis sortiert werden soll. Mehrere Feldnamen trennt man durch Kommata. Um anzugeben,

ob aufsteigende oder absteigende Sortierung verwendet werden soll, ergänzen Sie den Feldnamen um einen der beiden Ausdrücke *ASC* (aufsteigend) oder *DESC* (absteigend). Ohne Angabe von *ASC* oder *DESC* wird automatisch *ASC* angenommen.

Wenn Sie die Datensätze der Mitarbeitertabelle aufsteigend nach den Nachnamen sortieren möchten, verwenden Sie folgende Abfrage:

```
SELECT * FROM tblMitarbeiter ORDER BY Nachname;
```

Die nächste Variante legt die Sortierreihenfolge explizit fest:

```
SELECT * FROM tblMitarbeiter ORDER BY Nachname ASC;
```

Da Nachnamen gegebenenfalls mehrfach vorkommen, macht eine zusätzliche Sortierung nach dem Vornamen Sinn:

```
SELECT * FROM tblMitarbeiter ORDER BY Nachname, Vorname;
```

Die explizite Variante hieße dann:

```
SELECT * FROM tblMitarbeiter ORDER BY Nachname ASC, Vorname ASC;
```

7.3.6 Aggregatfunktionen

Sie können SQL-Aggregatfunktionen verwenden, um Datensätze zu zählen, Mittelwerte zu bilden oder Summen zu ermitteln. Diese Aggregatfunktionen beziehen sich auf das komplette Abfrageergebnis oder auf einzelne Gruppierungen. Mehr über den Einsatz von Aggregatfunktionen auf das komplette Abfrageergebnis erfahren Sie im Anschluss an die Auflistung der Funktionen; der Einsatz von Aggregatfunktionen mit gruppierten Daten wird in »Gruppieren von Daten« ab Seite 464 vorgestellt.

SQL bietet die folgenden Aggregatfunktionen:

» *Avg(<Ausdruck>)*: Berechnet den arithmetischen Mittelwert des in *<Ausdruck>* enthaltenen Wertes.

» *Count*: Ermittelt die Anzahl von Datensätzen. Es gibt zwei Varianten: *Count(*)* ermittelt die Anzahl aller enthaltenen Datensätze, und *Count(<Ausdruck>)* ermittelt die Anzahl aller Datensätze, in denen der *<Ausdruck>* nicht den Wert *NULL* besitzt.

» *First(<Ausdruck>)*: Ermittelt den Wert von *<Ausdruck>* des ersten Datensatzes des Abfrageergebnisses.

» *Last(<Ausdruck>)*: Ermittelt den Wert von *<Ausdruck>* des letzten Datensatzes des Abfrageergebnisses.

» *Max(<Ausdruck>)*: Ermittelt den größten Wert von *<Ausdruck>* des Abfrageergebnisses.

» *Min(<Ausdruck>)*: Ermittelt den kleinsten Wert von *<Ausdruck>* des Abfrageergebnisses.

» *StDev(<Ausdruck>)*, *StDevP(<Ausdruck>)*: Ermittelt die Standardabweichung von *<Ausdruck>* bezogen auf die Grundgesamtheit (*StDevP*) oder eine Stichprobe (*StDev*).

» *Sum(<Ausdruck>)*: Summiert die Werte von *<Ausdruck>*, *Null*-Werte werden ignoriert.

» *Var(<Ausdruck>)*, *VarP(<Ausdruck>)*: Berechnet die Varianz von *<Ausdruck>* bezogen auf die Grundgesamtheit (*VarP*) oder eine Stichprobe (*Var*).

Aggregatfunktionen ohne Gruppierung

Die Aggregatfunktionen lassen sich ohne Gruppierung auf die komplette Ergebnismenge der Abfrage anwenden. Wenn Sie beispielsweise den Durchschnittspreis von Produkten ermitteln möchten, verwenden Sie etwa folgende SQL-Abfrage:

```
SELECT Avg(Preis) AS Durchschnittspreis
FROM tblProdukte;
```

Das Ergebnis der Abfrage sieht wie in Abbildung 7.7 aus.

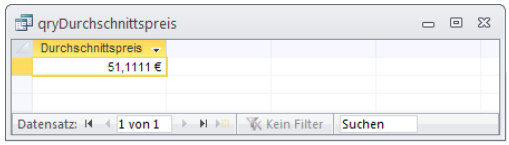


Abbildung 7.7: Ergebnis der Abfrage eines Durchschnittswertes

Wichtig ist, dass Sie für den durch die Aggregatfunktion entstandenen Ausdruck einen Alias-Namen angeben – in diesem Fall als *AS Durchschnittspreis* –, sonst vergibt Access automatisch einen Namen der Form *Durchschnitt von Preis*, *Anzahl von ProduktID* ...

7.3.7 Gruppieren von Daten

Daten lassen sich mit SQL innerhalb einer einzigen Abfrage gruppieren. Dadurch erhalten Sie die Möglichkeit, Aggregatfunktionen auf gruppierte Datensätze mit bestimmten Eigenschaften auszuführen.

Bei der Erstellung von Abfragen mit Gruppierungen sind zwei Regeln zu beachten:

» Gruppierungen können entweder Tabellenfelder, berechnete Felder oder Konstanten enthalten.

» Jedes Feld, das der *SELECT*-Bereich der Abfrage enthält, muss entweder mit einer Aggregatfunktion versehen oder ein Element des *GROUP BY*-Abschnitts sein.

Das fällt auch beim Erstellen von Gruppierungen mit der Abfrage-Entwurfsansicht auf: Dort wird für jedes hinzugefügte Feld standardmäßig die Funktion *Gruppierung* festgelegt. Es ist nicht möglich, in der Spalte *Funktion* weder eine Gruppierung noch eine Aggregatfunktion auszuwählen.

Einfache Gruppierungen

Das folgende Beispiel zeigt eine Abfrage, die Produkte nach Kategorien gruppiert und die Anzahl der Produkte je Kategorie ausgibt:

```
SELECT KategorieID, Count(ProduktID) AS AnzahlProdukte
FROM tblProdukte
GROUP BY KategorieID;
```

Das Ergebnis der Abfrage sieht wie in Abbildung 7.8 aus.

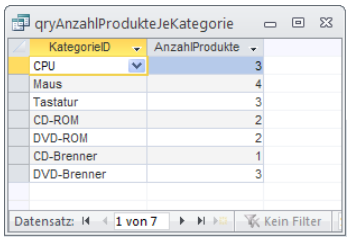


Abbildung 7.8: Abfrageergebnis einer Gruppierung von Datensätzen mit der Anzahl Datensätze jeder Gruppe

Sie können auch nach mehreren Feldern gruppieren. Ist die Abfrage aus dem vorherigen Beispiel nicht aussagekräftig genug, können Sie auch noch den Hersteller in die Gruppierung einbeziehen:

```
SELECT KategorieID, HerstellerID, Count(ProduktID) AS AnzahlProdukte
FROM tblProdukte
GROUP BY KategorieID, HerstellerID;
```

Diese Variante gruppiert nach allen vorhandenen Kombinationen aus Kategorie und Hersteller, wie das Ergebnis in Abbildung 7.9 zeigt.

Gruppierungen einschränken

Es gibt zwei verschiedene Möglichkeiten, das Ergebnis einer gruppierten Abfrage einzuschränken:

» vor dem Gruppieren der enthaltenen Datensätze und

» nach dem Gruppieren der enthaltenen Datensätze.

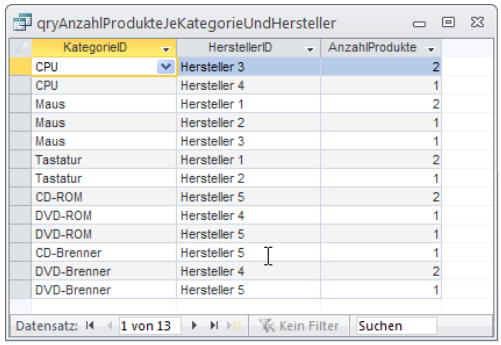


Abbildung 7.9: Gruppierung nach einer Kombination aus mehreren Feldern

Die erste Möglichkeit kennen Sie bereits: Mit der *WHERE*-Klausel leiten Sie einen Bereich ein, der Kriterien für die anzuzeigenden Datensätze enthält. Den *WHERE*-Abschnitt fügen Sie dabei hinter dem *FROM*-Bereich, aber vor dem *GROUP BY*-, *HAVING*- oder *ORDER BY*-Bereich ein.

Das folgende Beispiel ermittelt die Anzahl der Produkte teurer als 50 EUR je Kategorie. Dabei werden durch die *WHERE*-Bedingung zunächst alle Produkte ermittelt, die mehr als 50 EUR kosten, und anschließend wird die Gruppierung durchgeführt:

```
SELECT KategorieID, HerstellerID, Count(ProduktID) AS AnzahlProdukte

FROM tblProdukte

WHERE Preis > 50

GROUP BY KategorieID, HerstellerID;
```

Die zweite Möglichkeit zum Einschränken einer gruppierten Abfrage führt zunächst die Gruppierung durch und wertet dann das Ergebnis einer Aggregatfunktion als Kriterium aus. Dazu wird – analog zur *WHERE*-Klausel – die *HAVING*-Klausel zum Bereitstellen des Kriteriums beziehungsweise der Kriterien verwendet.

Im folgenden Beispiel sollen die Produkte nach Kategorie und Hersteller gruppiert und die Anzahl der Produkte je Gruppierung ermittelt werden, bevor die Abfrage diejenigen Kombinationen herausfiltert, die mit weniger als zwei Produkten vertreten sind.

Abbildung 7.10 zeigt das Ergebnis des folgenden SQL-Ausdrucks an:

```
SELECT KategorieID, HerstellerID,
Count(ProduktID) AS AnzahlProdukte
FROM tblProdukte
GROUP BY KategorieID, HerstellerID
HAVING Count(ProduktID) > 1;
```

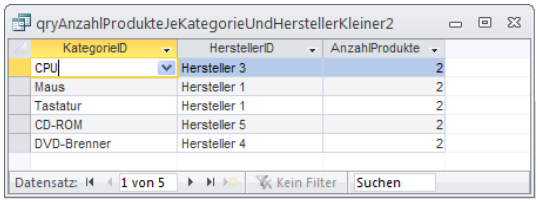


Abbildung 7.10: Abfrageergebnis eines SQL-Ausdrucks mit HAVING-Klausel

7.3.8 WHERE, GROUP BY, HAVING und ORDER BY im Überblick

Die vielfältigen Möglichkeiten zum Einschränken, Gruppieren und Sortieren der Ergebnismenge eines SQL-Ausdrucks bergen ein Problem: Wie soll man sich die Reihenfolge der unterschiedlichen Bereiche merken? Folgendes hilft vielleicht:

» *WHERE* schränkt die Daten ein und kommt daher zuerst.

» *GROUP BY* gruppiert die übrig gebliebenen Datensätze.

» *HAVING* schränkt ebenfalls Daten ein, aber nur auf Basis von Gruppierungen.

» *ORDER BY* kommt zum Schluss, weil alles andere vergeudete Rechenleistung wäre.

Diese Reihenfolge scheint intuitiv richtig zu sein – das reicht als Eselsbrücke für den Aufbau von SQL-Anweisungen aus.

Tatsächlich werden Abfragen mitunter auch anders abgearbeitet – mehr dazu erfahren Sie unter »Abfragen und die ACE-Engine« ab Seite 840.

7.3.9 Verknüpfen von Tabellen in Abfragen

Wenn Sie mehrere Tabellen im *FROM*-Bereich als Datenherkunft eines SQL-Ausdrucks angeben, werden diese von der ACE-Engine als völlig losgelöst von jeglichen zuvor angelegten Beziehungen zwischen den Tabellen betrachtet.

Manuelles Hinzufügen einer Verknüpfung

Sie können durchaus eine 1:n-Beziehung mit referentieller Integrität zwischen Tabellen wie *tblProjekte* und *tblMitarbeiter* im Beziehungsfenster angelegt haben – sobald Sie diese beiden Tabellen in der folgenden Form in einem SQL-Ausdruck angeben, ist alles vergessen:

```
SELECT tblProjekte.ProjektID, tblMitarbeiter.MitarbeiterID FROM tblProjekte,
tblMitarbeiter;
```

Die Abfrage-Entwurfsansicht hat da ein etwas besseres Gedächtnis. Sie erkennt direkt bestehende Beziehungen und zeigt diese auch an, wie in Abbildung 7.11. Allerdings können Sie auch diese Möglichkeit ausschalten:

Dazu deaktivieren Sie auf der Registerseite *Objekt-Designer* des *Access-Optionen*-Dialogs die Eigenschaft *AutoVerknüpfung aktivieren*.

Ein Wechsel von hier aus in die SQL-Ansicht zeigt, welchen zusätzlichen Code die Festlegung einer Beziehung zwischen zwei Tabellen mit sich bringt:

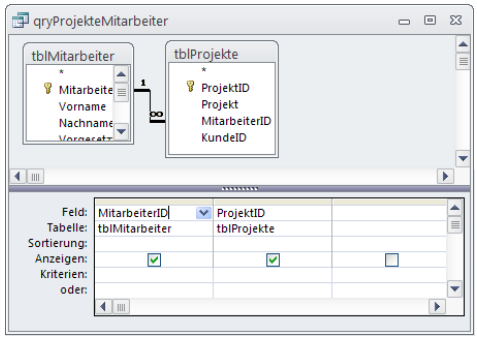


Abbildung 7.11: Die Abfrage-Entwurfsansicht übernimmt bestehende Beziehungen nach dem Hinzufügen von verknüpften Tabellen

```
SELECT tblMitarbeiter.MitarbeiterID, tblProjekte.ProjektID
FROM tblMitarbeiter INNER JOIN tblProjekte
ON tblMitarbeiter.MitarbeiterID = tblProjekte.MitarbeiterID;
```

Es gibt noch eine weitere Möglichkeit, zwei Tabellen innerhalb eines SQL-Ausdrucks zu verknüpfen. Dabei erfolgt die Verknüpfung im *WHERE*-Teil der Abfrage:

```
SELECT tblMitarbeiter.MitarbeiterID, tblProjekte.ProjektID
FROM tblMitarbeiter, tblProjekte
WHERE tblMitarbeiter.MitarbeiterID = tblProjekte.MitarbeiterID;
```

Diese Variante schreibt sich zwar etwas kürzer, kann aber keine *OUTER JOIN*-Verknüpfungen abbilden – mehr zu dieser Verknüpfungsart später. Noch gravierender ist der Nachteil, dass das Abfrageergebnis nicht aktualisiert werden kann – das zeigt ein Blick auf die Datenblatt-Ansicht einer solchen Abfrage (siehe Abbildung 7.12).

Dort findet sich keine Möglichkeit, einen neuen Datensatz anzulegen, und auch eine Änderung der enthaltenen Daten ist nicht möglich.

Im Entwurfsfenster einer solchen Abfrage erscheint übrigens keine Verknüpfungslinie, die *WHERE*-Bedingung wird wie üblich im Entwurfsraster angezeigt (siehe Abbildung 7.13).

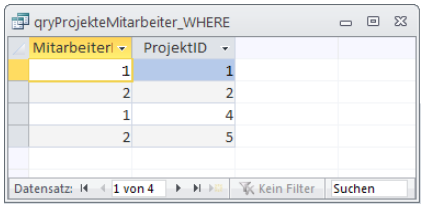


Abbildung 7.12: WHERE-Verknüpfungen sind nicht aktualisierbar

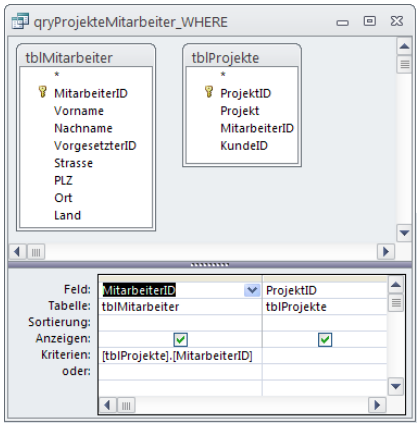


Abbildung 7.13: Beziehung per Kriterium

Aufbau von Verknüpfungen (INNER JOIN)

Einfache Verknüpfungen zwischen zwei Tabellen definiert man in SQL folgendermaßen:

```
<Tabelle1> INNER JOIN <Tabelle2>
ON <Tabelle1>.<Verknüpfungsfeld1> = <Tabelle2>.<Verknüpfungsfeld2>;
```

Kommen weitere Tabellen hinzu, setzt man den Ausdruck für die erste Verknüpfung in Klammern und behandelt ihn für die Erstellung der zweiten Verknüpfung wie eine einzige Tabelle.

Die neue Verknüpfung baut man einfach um den bestehenden Ausdruck herum wie im folgenden Beispiel (neue Teile fett gedruckt):

```
<Tabelle3> INNER JOIN (<Tabelle1>
INNER JOIN <Tabelle2>
ON <Tabelle1>.<Verknüpfungsfeld1> = <Tabelle2>.<Verknüpfungsfeld2>)
ON <Tabelle3>.<Verknüpfungsfeld3>;
```

Beispiel: Die klassische m:n-Beziehung zwischen Bestellungen, Bestelldetails und Artikeln enthält zwei *INNER JOIN*-Verknüpfungen (siehe Abbildung 7.14).

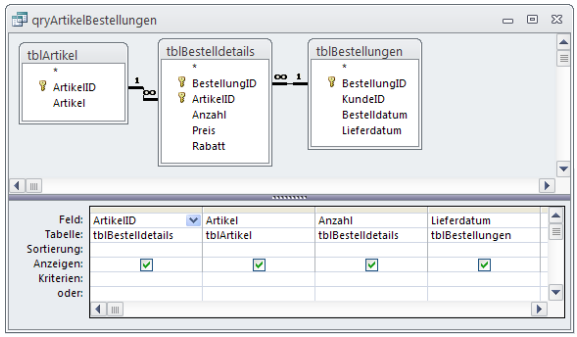


Abbildung 7.14: Abfrage mit zwei Verknüpfungen

Der entsprechende SQL-Ausdruck sieht folgendermaßen aus:

```
SELECT tblBestelldetails.ArtikelID, tblArtikel.Artikel, tblBestelldetails.Anzahl, tbl-
Bestellungen.Lieferdatum FROM tblBestellungen
INNER JOIN (
    tblArtikel INNER JOIN tblBestelldetails
    ON tblArtikel.ArtikelID = tblBestelldetails.ArtikelID
) ON tblBestellungen.BestellungID = tblBestelldetails
```

Weitere Verknüpfungsarten

Neben den *INNER JOINS*, die alle Datensätze ausgeben, bei denen die Inhalte des Verknüpfungsfeldes gleich sind, gibt es noch weitere Verknüpfungsarten.

Ein *OUTER JOIN* liefert alle Datensätze der ersten Tabelle zurück und nur die Datensätze der zweiten Tabelle, die mit einem Datensatz der ersten Tabelle verknüpft sind. *OUTER JOIN*-Abfragen treten als *LEFT OUTER JOIN* und *RIGHT OUTER JOIN* auf. *LEFT* beziehungsweise *RIGHT* legt dabei fest, ob alle Datensätze von der links vom Schlüsselwort *JOIN* stehenden Tabelle ausgegeben werden oder von der rechts davon stehenden Tabelle.

Beispiel: Der folgende SQL-Ausdruck soll alle Mitarbeiter und ihre Projekte ausgeben, aber auch die Mitarbeiter berücksichtigen, denen kein Projekt zugeordnet ist.

```
SELECT Nachname & ", " & Vorname AS Mitarbeiter, tblProjekte.Projekt
FROM tblMitarbeiter LEFT JOIN tblProjekte
ON tblMitarbeiter.MitarbeiterID=tblProjekte.MitarbeiterID;
```

Abbildung 7.15 zeigt das Ergebnis dieses SQL-Ausdrucks.

Für die Verwendung von *OUTER JOIN*-Verknüpfungen gibt es zwei wichtige Regeln: Die rechte Tabelle eines *LEFT OUTER JOIN* kann nicht mit anderen Tabellen per *INNER JOIN* verknüpft werden und kann nicht die linke Tabelle eines anderen *RIGHT OUTER JOIN* oder

8

ADO

Im vorherigen Kapitel haben Sie bereits erfahren, dass DAO die bevorzugte Datenzugriffs-Bibliothek ist, wenn es um die Entwicklung reiner Access-Anwendungen geht. ADO (ActiveX Data Objects) war von Microsoft als Nachfolger von DAO geplant.

Es eröffnet wesentlich mehr Möglichkeiten, die sich aber vor allem dann bemerkbar machen, wenn Sie ein alternatives Backend wie beispielsweise Microsoft SQL Server oder MySQL verwenden.

Der Fokus dieses Buchs richtet sich auf die Entwicklung reiner Access-Datenbankanwendungen. Daher sollen die Möglichkeiten von Access in Zusammenarbeit mit dem SQL Server hier nicht betrachtet werden.

Entwickler, die jetzt noch reine Access-Anwendungen programmieren, aber diese im Hinblick auf einen späteren Wechsel auf den SQL Server direkt auf dieses Backend vorbereiten möchten, sollen natürlich nicht außen vor bleiben – zumal ADO auch für reine Access-Anwendungen einige Features bereithält, die DAO nicht bietet.

BEISPIELDATENBANK

Die Beispieldatei zu diesem Kapitel finden Sie unter dem Namen ADO.*accdb* auf *www.acciu.de/aeb2010*.

Wegen des gegenüber DAO wesentlich größeren Funktionsumfangs könnte über ADO ein eigenes Buch geschrieben werden. Aus Platzgründen wird das Thema hier jedoch ein wenig eingeschränkt – und zwar so, dass Sie die Techniken, die Sie im vorherigen Kapitel über DAO kennen gelernt haben, auch mit ADO einsetzen können. Natürlich soll auch die eine oder andere Spezialität von ADO nicht unerwähnt bleiben. Viele bereits in »DAO« ab Seite 527 enthaltene Informationen gelten auch für den Umgang mit ADO. Dies bezieht sich vor allem auf formale Techniken wie den Umgang mit Auflistungen, die Verwendung von Punkt oder Ausrufezeichen für den Bezug auf Elemente und Eigenschaften oder das Deklarieren und Instanzieren von Objekten. Wenn Sie Informationen zu diesen Themen benötigen, schlagen Sie am besten im oben genannten Kapitel nach.

ADO-Neuigkeiten

Mit Access 2010 kommt nach Access 2007 wiederum keine neue ADO-Version. Der mit Access 2007 eingeführte JET-OLEDB-Provider namens *Microsoft Office 12.0 Access Database Engine OLE DB Provider* (*Microsoft.ACE.OLEDB.12.0*) findet weiterhin Verwendung. Im Gegensatz zu DAO mit den Objekten *Recordset2* und *Field2* gibt es unter ADO keine Möglichkeit, die in Attachment- oder mehrwertigen Feldern gespeicherten Daten über ein zusätzliches Recordset auszulesen. Daher kann man nur über die »Unterfelder« auf die in den verknüpften, verborgenen Tabellen enthaltenen Daten zugreifen. Wie dies funktioniert, erfahren Sie unter »Daten eines Recordsets mit mehrwertigen Feldern ausgeben« auf Seite 506 und »Daten eines Recordsets mit Attachment-Feldern ausgeben« auf Seite 507.

8.1 Zugriff auf eine Datenquelle herstellen

Obgleich Sie im Folgenden feststellen werden, dass viele Vorgehensweisen unter DAO und ADO gleich ablaufen, unterscheidet sich das Objektmodell von ADO in einigen Punk-

ten vom DAO-Objektmodell. Das macht sich beim Zugriff auf die gewünschte Datenbank sofort bemerkbar, wie der folgende Abschnitt zeigen wird.

Connection und ConnectionString

Das beginnt damit, dass nicht das *Database*-Objekt, sondern das *Connection*-Objekt Ursprung aller lesenden, schreibenden und sonstigen Zugriffe auf die Tabellen der Datenbank ist. Das *Connection*-Objekt enthält immer einen *ConnectionString*, der Informationen über die Verbindung zur gewünschten Datenbank enthält. Diesen *ConnectionString* stellt man entweder selbst zusammen oder man lässt sich dabei unterstützen. Die einfachste Methode dazu ergibt sich beim Zugriff auf die aktuelle Datenbank. In diesem Fall brauchen Sie einfach nur auf das *Connection*-Objekt des bestehenden *CurrentProject*-Objekts zuzugreifen.

```
Public Sub Verbindung()
    Dim cnn As ADODB.Connection
    Set cnn = CurrentProject.Connection
    With cnn
        Debug.Print cnn.ConnectionString
    End With
End Sub
```

Listing 8.1: Verweisen auf eine Verbindung zur aktuellen Datenbank und Ausgabe der ConnectionString-Eigenschaft

Die Verbindungszeichenfolge für die aktuelle Datenbank sieht etwa so aus – gut, dass Sie diesen Ausdruck nicht selbst zusammenstellen müssen:

```
Provider=Microsoft.ACE.OLEDB.12.0Microsoft.ACE.OLEDB.12.0;User ID=Admin;Data Source=E:\
ADO.accdbmdb;Mode=Share Deny None;Extended Properties="";Jet OLEDB:System database=C:\
Dokumente und Einstellungen\Administrator\Anwendungsdaten\Microsoft\Access\System.
mdw;Jet OLEDB:Registry Path= Software\Microsoft\Office\12.0\Access\Access Connectivity
EngineSoftware\Microsoft\Office\11.0\Access\Jet\4.0;Jet OLEDB:Database Password="";Jet
OLEDB:Engine Type=65;Jet OLEDB:Database Locking Mode=1;Jet OLEDB:Global Partial Bulk
Ops=2;Jet OLEDB:Global Bulk Transactions=1;Jet OLEDB:New Database Password="";Jet
OLEDB:Create System Database=False;Jet OLEDB:Encrypt Database=False;Jet OLEDB:Don't
Copy Locale on Compact=False;Jet OLEDB:Compact Without Replica Repair=False;Jet
OLEDB:SFP=False;Jet OLEDB:Support Complex Data=True
```

Was die Parameter im Einzelnen bedeuten, soll hier gar nicht aufgeschlüsselt werden. Für die Verwendung der Datenbank, zu der auch das VBA-Projekt gehört, reicht die Kenntnis, dass *CurrentProject.Connection* die richtige Verbindung liefert.

Für den Fall, dass Sie einmal eine Verbindung zu einer externen Datenquelle herstellen möchten, brauchen Sie den *ConnectionString* auch nicht unbedingt manuell zusammen-

zustellen. Zur Ermittlung der passenden Parameter können Sie auch einen Dialog verwenden, der bei der Festlegung der Parameter behilflich ist.

Die folgende Routine erzeugt eine Instanz des *DataLinks*-Objekts und zeigt mit der Methode *PromptNew* den Dialog zum Anlegen einer neuen Verbindungszeichenfolge an (siehe Abbildung 8.1).

```
Public Function ConnectionStringErmitteln()
    ConnectionStringErmitteln = CreateObject("DataLinks").PromptNew
End Function
```

Listing 8.2: Aufruf des Dialogs zum Ermitteln einer Verbindungszeichenfolge

Mit dem gleichen Dialog können Sie nicht nur die Eigenschaften neuer Verbindungen auswählen und zurückgeben lassen, sondern auch die Eigenschaften von *CurrentProject.Connection* etwas übersichtlicher ausgeben (siehe Abbildung 8.2). Dazu müssen Sie die aufrufende Routine geringfügig abändern:

```
Public Function ConnectionStringBearbeiten()
    ConnectionStringBearbeiten = _
        CreateObject("DataLinks").PromptEdit(CurrentProject.Connection)
End Function
```

Listing 8.3: Aufrufen des Dialogs Datenverknüpfungseigenschaften für eine bestehende Verbindungszeichenfolge

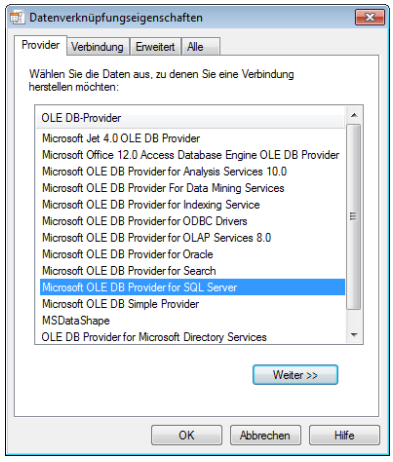


Abbildung 8.1: Anlegen einer neuen Verbindung

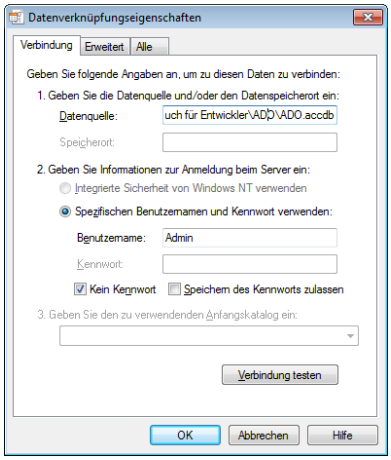


Abbildung 8.2: Bearbeiten von CurrentProject.Connection

8.2 Manipulation des Datenmodells

Die Manipulation von Tabellen, Feldern und anderen Datenbankobjekten erfolgt nicht über ADO selbst, sondern über die Objekte, Methoden und Eigenschaften der Bibliothek ADOX. Die entsprechende Bibliothek im *Verweise*-Dialog heißt *Microsoft ADO Ext. 2.8 for DDL and Security* (oder älter).

8.2.1 Anlegen einer Tabelle

Zum Anlegen einer Tabelle mit reinen ADO-Mitteln benötigen Sie das *Catalog*-Objekt der ADOX-Bibliothek (es ginge ja auch mit *CurrentProject.Execute "CREATE TABLE..."*). Dieses wird mit der Eigenschaft *ActiveConnection* auf die aktuelle Datenbank eingestellt. Zum Löschen einer eventuell schon vorhandenen gleichnamigen Tabelle stellt das *Catalog*-Objekt die *Delete*-Methode der *Tables*-Auflistung zur Verfügung.

Anschließend wird ein neues *Table*-Objekt erstellt und mit dem Namen *tblUnternehmen* versehen. Bevor die Tabelle an die *Tables*-Auflistung angehängt und damit verfügbar gemacht wird, fügen Sie zwei Felder an: *UnternehmenID* und *Unternehmen*. Dabei werden zwei unterschiedliche Vorgehensweisen verwendet: Die erste instanziert zunächst ein neues *Column*-Objekt, füllt dessen Eigenschaften mit den entsprechenden Daten und fügt es dann an die *Columns*-Auflistung des *Table*-Objekts an. Die zweite übergibt die benötigten Informationen direkt beim Anfügen eines neuen Feldes an das *Table*-Objekt.

Nach dem Anlegen werden die *Tables*-Auflistung und der Navigationsbereich aktualisiert.

```
Public Sub TabelleAnlegen_Unternehmen()
    Dim cat As ADOX.Catalog
    Dim tbl As ADOX.Table
    Dim col As ADOX.Column
    Set cat = New ADOX.Catalog
    cat.ActiveConnection = CurrentProject.Connection
    'Bestehende Tabelle löschen
    cat.Tables.Delete "tblUnternehmen"
    'Verweis auf neues Table-Objekt
    Set tbl = New ADOX.Table
    'Name der Tabelle zuweisen
    tbl.Name = "tblUnternehmen"
    'Feld neu erstellen und per Objektvariable referenzieren
    Set col = New ADOX.Column
    col.Name = "UnternehmenID"
    col.Type = adInteger
    tbl.Columns.Append col
    'Noch ein Feld erstellen, kurze Fassung
    tbl.Columns.Append "Unternehmen", adVarWChar, 30
    'Tabelle anhängen und Katalog aktualisieren
    With cat.Tables
        .Append tbl
        .Refresh
    End With
    'Navigationsbereich aktualisieren
    Application.RefreshDatabaseWindow
    Set tbl = Nothing
    Set cat = Nothing
End Sub
```

Listing 8.4: Anlegen einer Tabelle mit ADOX

Konstanten für Datentypen unter ADO und ADOX

Tabelle 1.1 zeigt die häufigsten ADOX-Konstanten für die unterschiedlichen Datentypen.

Konstante	Datentyp
adBigInt	Big Integer
adBinary	Binary
adBoolean	Boolean
adUnsignedTinyInt	Byte
adChar	Char

Konstante	Datentyp
adCurrency	Currency
adDate	Date/Time
adNumeric	Decimal
adDouble	Double
adGUID	GUID
adSmallInt	Integer
adInteger	Long
adLongVarBinary	Long Binary (OLE Object), Attachment.FileData
adLongVarWChar	Memo
adNumeric	Numeric
adSingle	Single
adWChar, adVarWChar	Text (Unicode)
adDBTime	Time
adIDispatch	Attachment, ComplexTypes
adDBTimeStamp	Time Stamp

Tabelle 1.1: Konstanten für den Datentyp

8.2.2 Autowert anlegen

Wenn Sie das Feld *UnternehmenID* als Autowert festlegen möchten, müssen Sie in obiger Routine noch einige Zeilen hinter dem Anlegen des Feldes hinzufügen:

```
'Anlegen eines Autowertes
 With tbl.Columns("UnternehmenID")
     .ParentCatalog = cat
     .Properties("AutoIncrement") = True
 End With
```

Die Eigenschaft *AutoIncrement* gilt nur für Access-Datenbanken.

8.2.3 Löschen einer Tabelle

Das Löschen einer Tabelle erfolgt über die *Delete*-Methode der *Tables*-Auflistung. Die folgende Routine löscht die soeben erstellte Tabelle und aktualisiert die *Tables*-Auflistung sowie den Navigationsbereich.

```
Public Sub TabelleLoeschen_Unternehmen()
    Dim cat As ADOX.Catalog
    Set cat = New ADOX.Catalog
```

```
    cat.ActiveConnection = CurrentProject.Connection
    'Bestehende Tabelle löschen
    On Error Resume Next
    cat.Tables.Delete "tblUnternehmen"
    On Error GoTo 0
    'Katalog aktualisieren
    cat.Tables.Refresh
    'Navigationsbereich aktualisieren
    Application.RefreshDatabaseWindow
    Set cat = Nothing
End Sub
```

Listing 8.5: Löschen einer Tabelle mit ADOX

8.2.4 Erstellen eines Indexes

Mit der folgenden Routine fügen Sie der Tabelle *tblUnternehmen* einen Primärindex auf dem Feld *UnternehmenID* hinzu.

```
Public Sub IndexErstellen()
    Dim cat As ADOX.Catalog
    Dim idx As ADOX.Index
    Dim tbl As ADOX.Table
    Dim col As ADOX.Column
    Dim idxs As ADOX.Indexes
    'Catalog instanzieren und auf aktuelle Datenbank einstellen
    Set cat = New ADOX.Catalog
    cat.ActiveConnection = CurrentProject.Connection
    'Tabelle festlegen
    Set tbl = cat.Tables("tblUnternehmen")
    'Verweis auf Indexes-Auflistung erstellen
    Set idxs = tbl.Indexes
    'Neues Index-Objekt erstellen
    Set idx = New ADOX.Index
    With idx
        'Index-Objekt mit Eigenschaften ausstatten
        .Name = "PrimaryKey"
        .PrimaryKey = True
        .Unique = True
        'Column-Objekt mit zu indizierendem Feld erzeugen
        'und zur Auflistung der indizierten Columns hinzufügen
        Set col = New ADOX.Column
```

```
        col.Name = "UnternehmenID"
        .Columns.Append col
    End With
    'Index an die Auslistung Indexes anfügen
    idxs.Append idx
    Set col = Nothing
    Set idx = Nothing
    Set idxs = Nothing
    Set tbl = Nothing
    Set cat = Nothing
End Sub
```

Listing 8.6: Anlegen eines Index mit ADOX

8.2.5 Löschen eines Indexes

Zum Entfernen eines Indexes verwenden Sie die folgende Routine. Sie setzt die *Delete*-Methode der *Indexes*-Auflistung zum Entfernen des Indexes ein.

```
Public Sub IndexLoeschen()
    Dim cat As ADOX.Catalog
    Dim tbl As ADOX.Table
    Dim idxs As ADOX.Indexes
    Set cat = New ADOX.Catalog
    cat.ActiveConnection = CurrentProject.Connection
    Set tbl = cat.Tables("tblUnternehmen")
    Set idxs = tbl.Indexes
    'Index aus der Auflistung löschen
    idxs.Delete "PrimaryKey"
    Set tbl = Nothing
    Set idxs = Nothing
    Set cat = Nothing
End Sub
```

Listing 8.7: Löschen eines Indexes aus der Auflistung der Indizes einer Tabelle

8.2.6 Erstellen einer Beziehung

Um eine Beziehung zwischen zwei Tabellen herzustellen, verwenden Sie die folgende Routine.

Voraussetzung ist, dass das Fremdschlüsselfeld der Detailtabelle den gleichen Datentyp wie das Primärschlüsselfeld der Mastertabelle hat. Außerdem muss der Primär-

schlüssel der Mastertabelle eindeutig sein. Sind die Voraussetzungen erfüllt (und die angegebenen Tabellen beziehungsweise Felder vorhanden), legt die Routine die Beziehung aus Abbildung 8.3 an.

Wenn Sie zusätzlich Löschweitergabe oder Aktualisierungsweitergaben definieren möchten, müssen Sie die Eigenschaften *DeleteRule* und *UpdateRule* des *key*-Objekts mit den entsprechenden Werten bestücken. Mit der Konstanten *adRICascade* sorgen Sie für die Weitergabe der jeweiligen Aktion:

```
Public Sub BeziehungErstellen()
    Dim cat As ADOX.Catalog
    Dim tbl As ADOX.Table
    Dim key As ADOX.key
    Set cat = New ADOX.Catalog
    cat.ActiveConnection = CurrentProject.Connection
    'Tabelle mit dem Fremdschlüsselfeld festlegen
    Set tbl = cat.Tables("tblMitarbeiter")
    'Neuen Key instanzieren und Eigenschaften zuweisen
    Set key = New ADOX.key
    key.Name = "ForeignKey"
    key.Type = adKeyForeign
    'Tabelle mit Primärschlüssel festlegen
    key.RelatedTable = "tblUnternehmen"
    'Verknüpfungsfeld der Detailtabelle angeben
    key.Columns.Append "UnternehmenID"
    'Optional: Lösch- oder Aktualisierungsweitergabe
    key.DeleteRule = adRICascade
    key.UpdateRule = adRICascade
    'Verknüpfungsfeld der Mastertabelle angeben
    key.Columns("UnternehmenID").RelatedColumn = "UnternehmenID"
    'Key an die Keys-Auflistung anhängen
    tbl.keys.Append key
    Set key = Nothing
    Set tbl = Nothing
    Set cat = Nothing
End Sub
```

Listing 8.8: Anlegen einer Beziehung zwischen zwei Tabellen

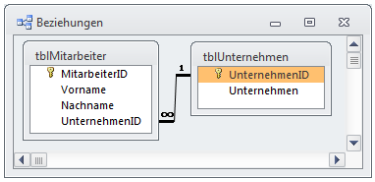


Abbildung 8.3: Mit ADOX erstellte Beziehung

8.2.7 Löschen einer Beziehung

Die Beziehung beziehungsweise den Fremdschlüssel löschen Sie mit der folgenden Routine:

```
Public Sub BeziehungLoeschen()
    Dim cat As ADOX.Catalog
    Set cat = New ADOX.Catalog
    cat.ActiveConnection = CurrentProject.Connection
    'Beziehung in Form des Fremdschlüssels löschen
    cat.Tables("tblMitarbeiter").keys.Delete "ForeignKey"
    Set cat = Nothing
End Sub
```

Listing 8.9: Löschen einer Beziehung

8.3 Zugriff auf Tabellen, Abfragen und die darin enthaltenen Daten

In den folgenden Abschnitten erfahren Sie, wie Sie auf die Daten der Datenbank zugreifen können.

8.3.1 Ausgeben aller Tabellen

Im vorigen Kapitel zum Thema DAO haben Sie erfahren, wie Sie mit DAO alle Tabellen ausgeben oder prüfen, ob eine Tabelle vorhanden ist. Unter ADO erledigen Sie dies wie in folgendem Beispiel:

```
Sub DBInfoADO()
    Dim rs As New ADODB.Recordset
    Dim fld As ADODB.Field
    Set rs = CurrentProject.Connection.OpenSchema(adSchemaTables)
    Do While Not rs.EOF
```

```
        For Each fld In rs.Fields
            Debug.Print , fld.Name, "=", fld.Value
        Next fld
        rs.MoveNext
        Debug.Print
    Loop
    rs.Close
End Sub
```

Das liefert beispielsweise solche Informationen:

```
TABLE_CATALOG =             Null
TABLE_SCHEMA  =             Null
TABLE_NAME    =             tblDateien
TABLE_TYPE    =             TABLE
TABLE_GUID    =             Null
DESCRIPTION   =             Null
TABLE_PROPID  =             Null
DATE_CREATED  =             21.01.2007 14:52:08
DATE_MODIFIED =             21.01.2007 14:52:11
```

Wenn Sie nur die benutzerdefinierten Tabellen ausgeben möchten, erledigen Sie das etwa so:

```
Sub DBInfoADO()
    Dim rst As New ADODB.Recordset
    Dim fld As ADODB.Field
    Set rst = CurrentProject.Connection.OpenSchema(adSchemaTables)
    Debug.Print "Tabellen:"
    Do While Not rst.EOF
        For Each fld In rst.Fields
            If fld.Name = "TABLE_NAME" And rst("TABLE_TYPE") = "TABLE" Then
                Debug.Print fld.Value
            End If
        Next fld
        rst.MoveNext
    Loop
    rst.Close
End Sub
```

Es gibt noch einige weitere Konstanten, die Sie der *OpenSchema*-Methode übergeben können, zum Beispiel die Folgenden – experimentieren Sie einfach mal damit:

» *adSchemaColumns*

- *adSchemaConstraintColumnUsage*
- *adSchemaForeignKeys*
- *adSchemaIndexes*
- *adSchemaKeyColumnUsage*
- *adSchemaPrimaryKeys*
- *adSchemaProcedures* und *adSchemaViews* (Abfragen)
- *adSchemaProviderTypes*
- *adSchemaReferentialConstraints*
- *adSchemaStatistics* (Beziehungen)
- *adSchemaTableConstraints*
- *adSchemaTrustees* (DB-Benutzer)

8.3.2 Prüfen, ob eine Tabelle vorhanden ist

Auch hier brauchen Sie nicht mit ADO zu arbeiten, sondern können die *AllTables*-Auflistung verwenden:

```
Public Function IstTabelleVorhanden_ADO(strTabellenname As String) _
        As Boolean
    Dim objTable As AccessObject
    On Error Resume Next
    Set objTable = CurrentData.AllTables (strTabellenname)
    IstTabelleVorhanden_ADO = Not objTable Is Nothing
End Function
```

Listing 8.10: Prüfen des Vorhandenseins einer Tabelle

8.3.3 Datensatzgruppe auf Basis einer Tabelle öffnen

Das Öffnen einer Datensatzgruppe erfolgt anders als unter DAO. Die Methode zum Öffnen ist keine Methode des übergeordneten Objekts (bei DAO das Database-Objekt), sondern eine Methode des ADO-*Recordset*-Objekts selbst. Die *Open*-Methode erwartet den Namen der zu öffnenden Tabelle oder Abfrage beziehungsweise einen SQL-Ausdruck, den *Connection*-String sowie zwei Parameter zur Angabe des Cursor-Typs und des Sperrmechanismus. Die Beschreibung der verschiedenen Möglichkeiten finden Sie im Anschluss an die Routine zum Öffnen einer Datensatzgruppe. Hier tritt offen zu Tage, weshalb es sich lohnt, bei der Variablendeklaration explizit die Bibliothek mit

anzugeben, aus der die Objekte stammen: Wenn Sie das versäumt haben und die DAO-Bibliothek ist in der Liste der Verweise oberhalb der ADO-Bibliothek angeordnet, sucht die folgende Routine vergeblich die *Open*-Methode des Recordset-Objekts.

```
Public Sub DatensatzgruppeOeffnen()
    Dim cnn As ADODB.Connection
    Dim rst As ADODB.Recordset
    Set cnn = CurrentProject.Connection
    Set rst = New ADODB.Recordset
    rst.Open "tblUnternehmen", cnn, adOpenDynamic, adLockOptimistic
    With rst
        'etwas mit der Datenatzgruppe machen
    End With
    rst.Close
    Set rst = Nothing
    Set cnn = Nothing
End Sub
```

Listing 8.11: Öffnen einer Datensatzgruppe mit ADO

Das eigentliche Öffnen der Datensatzgruppe kann auch so erfolgen, wobei zunächst die Eigenschaften festgelegt werden und erst dann die Datensatzgruppe geöffnet wird:

```
rst.ActiveConnection = cnn
rst.LockType = adLockOptimistic
rst.CursorType = adOpenKeyset
rst.Open "tblUnternehmen"
```

8.3.4 Cursor-Typen

Beim Öffnen einer Datensatzgruppe unter ADO können Sie folgende Cursor-Typen für die Eigenschaft *CursorType* verwenden:

» *adOpenDynamic*: Liefert eine Gruppe von Datensätzen und zeigt Datensatzänderungen anderer Benutzer an (entspricht *dbOpenDynaset* unter DAO).

» *adOpenForwardOnly*: Liefert einen Snapshot der gewünschten Datensätze zum Zeitpunkt des Öffnens des *Recordset*-Objekts, kann nur vorwärts durchlaufen werden (entspricht *dbOpenForwardOnly* unter DAO).

» *adOpenKeyset*: Liefert Verweise auf die Datensätze der zugrunde liegenden Tabellen (entspricht keiner DAO-Konstante genau, ist aber fast äquivalent zu *dbOpenDynaset* und etwas schneller als *adOpenDynamic*).

» *adOpenStatic*: Liefert einen Snapshot der gewünschten Datensätze zum Zeitpunkt des Öffnens des *Recordset*-Objekts (entspricht *dbOpenSnapshot* unter DAO).

Achtung: Bei Recordsets, die mit Connections geöffnet werden, die nicht *CurrentProject* entsprechen, müssen Sie zusätzlich die *CursorLocation* auf den Wert *adUseClient* festlegen:

```
cnn.CursorLocation = adUseClient
```

8.3.5 Sperrung von Daten

Mit dem Parameter *LockType* legen Sie fest, wie die Daten beim Schreiben gesperrt werden sollen:

» *adLockReadonly*: Öffnet ein schreibgeschütztes Recordset.

» *adLockPessimistic*: Sperrt die komplette Speicherseite, in der sich der von einer Änderung betroffene Datensatz befindet, sobald die Bearbeitung beginnt.

» *adLockOptimistic*: Sperrt die komplette Speicherseite, in der sich der von einer Änderung betroffene Datensatz befindet, erst, wenn der Datensatz aktualisiert wird.

» *adLockBatchOptimistic*: Wie *adLockOptimistic*, aber für die *UpdateBatch*-Methode.

8.3.6 Datensätze eines Recordsets durchlaufen

Zum Durchlaufen der Datensätze eines Recordsets verwenden Sie beispielsweise die *Do While*-Schleife, in der Sie nach dem Durchführen der gewünschten Aktion jeweils mit der *MoveNext*-Methode einen Datensatz weiter springen. Als Abbruchbedingung dient die *EOF*-Eigenschaft der Datensatzgruppe, die den Wert *True* erhält, wenn der Datensatzzeiger über den letzten Datensatz hinaus verschoben wurde.

```
Public Sub DatensaetzeDurchlaufen()
    Dim cnn As ADODB.Connection
    Dim rst As ADODB.Recordset
    Set cnn = CurrentProject.Connection
    Set rst = New ADODB.Recordset
    rst.Open "tblUnternehmen", cnn, adOpenKeyset, adLockOptimistic
    Do While Not rst.EOF
        With rst
            'etwas mit dem aktuellen Datensatz tun
        End With
        rst.MoveNext
    Loop
    rst.Close
```

```
    Set rst = Nothing
    Set cnn = Nothing
End Sub
```

Listing 8.12: Datensatzgruppe durchlaufen

Das Pendant zur *EOF*-Eigenschaft ist die *BOF*-Eigenschaft. Sie erhält den Wert *True*, wenn der Datensatzzeiger sich vor dem ersten Datensatz der Datensatzgruppe befindet. Neben der *MoveNext*-Methode gibt es noch die Methoden *MoveFirst*, *MoveLast* und *MovePrevious* zum Bewegen innerhalb der Datensatzgruppe.

8.3.7 Daten eines Recordsets mit mehrwertigen Feldern ausgeben

ADO bietet keine besondere Möglichkeit, um auf die Inhalte mehrwertiger Felder zuzugreifen – es gibt also kein *Recordset2*- oder *Field2*-Objekt wie unter DAO. Daher können Sie nur auf die in solchen Feldern enthaltenen Daten zugreifen, wenn Sie diese in der Datenherkunft über die »Unterfelder« referenzieren, wie das folgende Beispiel zeigt:

```
Public Sub DatensaetzeDurchlaufen_MehrwertigesFeld()
    Dim cnn As ADODB.Connection
    Dim rst As ADODB.Recordset
    Set cnn = CurrentProject.Connection
    Set rst = New ADODB.Recordset
    rst.Open "SELECT MehrwertigesFeld.Value " _
        & "FROM tblMehrwertigesFeld", _
        cnn, adOpenKeyset, adLockOptimistic
    Do While Not rst.EOF
        With rst
            Debug.Print rst.Fields("MehrwertigesFeld.Value")
        End With
        rst.MoveNext
    Loop
    rst.Close
    Set rst = Nothing
    Set cnn = Nothing
End Sub
```

Listing 8.13: Zugriff auf mehrwertige Felder mit einem ADO-Recordset

8.3.8 Daten eines Recordsets mit Attachment-Feldern ausgeben

Der Umgang mit den in *Attachment*-Feldern gespeicherten Daten sieht prinzipiell wie bei den mehrwertigen Feldern aus:

```
Public Sub DatensaetzeDurchlaufen_Attachment()
    Dim cnn As ADODB.Connection
    Dim rst As ADODB.Recordset
    Set cnn = CurrentProject.Connection
    Set rst = New ADODB.Recordset
    rst.Open "SELECT DateiID, Datei.FileType, " _
        & "Datei.Filename FROM tblDateien", _
        cnn, adOpenKeyset, adLockOptimistic
    Do While Not rst.EOF
        With rst
            Debug.Print rst!DateiID, rst.Fields("Datei.FileType"), _
                rst.Fields("Datei.Filename")
        End With
        rst.MoveNext
    Loop
    rst.Close
    Set rst = Nothing
    Set cnn = Nothing
End Sub
```

Listing 8.14: Zugriff auf Attachment-Felder

8.3.9 Anzahl der Datensätze in einer Datensatzgruppe ermitteln

Um die Anzahl der Datensätze zu ermitteln, dürfen Sie nicht die Konstante *adOpenForwardOnly* für die Eigenschaft *CursorType* einsetzen. Außerdem müssen Sie für die Eigenschaft *CursorLocation* den Parameter *adUseClient* verwenden, da serverseitige Datensatz-Cursor unter ACE das Zählen von Datensätzen nicht unterstützen.

```
Public Sub Datensatzanzahl()
    Dim cnn As ADODB.Connection
    Dim rst As ADODB.Recordset
    Set cnn = CurrentProject.Connection
    Set rst = New ADODB.Recordset
    rst.CursorLocation = adUseClient
```

```
    rst.CursorType = adOpenForwardOnly
    rst.LockType = adLockOptimistic
    rst.ActiveConnection = cnn
    rst.Open "tblUnternehmen"
    Debug.Print rst.RecordCount
End Sub
```

Listing 8.15: Ermitteln der Datensatzanzahl eines Recordsets

8.3.10 Prüfen, ob eine Datensatzgruppe leer ist

Eine einfache Möglichkeit, um herauszufinden, ob eine Datensatzgruppe leer ist, besteht im Prüfen der *EOF*- und der *BOF*-Eigenschaften der Datensatzgruppe:

```
Public Sub LeereDatensatzgruppe()
    Dim cnn As ADODB.Connection
    Dim rst As ADODB.Recordset
    Set cnn = CurrentProject.Connection
    Set rst = New ADODB.Recordset
    rst.Open "tblMitarbeiter", cnn, adOpenForwardOnly, adLockOptimistic
    If rst.BOF And rst.EOF Then
        MsgBox "Die Datensatzgruppe ist leer."
    End If
    rst.Close
    Set rst = Nothing
    Set cnn = Nothing
End Sub
```

Listing 8.16: Prüfen, ob eine Datensatzgruppe leer ist

8.3.11 Ausgabe des Inhalts eines Recordsets

Manchmal möchte man den Inhalt eines *Recordsets* auf die Schnelle betrachten oder etwa in einer Textdatei speichern. Dabei leistet die *GetString*-Methode gute Dienste.

Diese Methode ist sehr flexibel und kann, mit zusätzlichen Parametern ausgestattet, die Spalten auch als formatierten Text zurückgeben.

Informationen zu den (optionalen) Parametern finden Sie unter *http://msdn.microsoft.com/en-us/library/ms676975%28v=vs.85%29.aspx*.

```
Public Sub DatensatzgruppeAusgeben()
    …
    rst.Open "tblUnternehmen", cnn, adOpenForwardOnly, adLockOptimistic
```

```
    Debug.Print rst.GetString
    …
End Sub
```

Listing 8.17: Ausgeben der Daten einer Datensatzgruppe

8.3.12 Speichern der Daten in einem Array

Wenn Sie die Daten einer Datensatzgruppe in einem Array weiter verarbeiten möchten, können Sie mit der *GetRows*-Methode ein zweidimensionales Array mit den in der Datensatzgruppe enthaltenen Daten füllen.

Die *GetRows*-Methode hat drei Parameter: *Rows* gibt an, wie viele Datensätze eingelesen werden sollen, *Start* enthält ein Bookmark auf den ersten einzulesenden Datensatz und *Fields* die Position oder den Namen des einzulesenden Feldes beziehungsweise ein Array mit den Namen der einzulesenden Felder. Ohne die Angabe von Parametern liest *GetRows* alle Datensätze mit allen Feldern ein.

```
Public Sub DatensatzgruppeInArray()
    Dim cnn As ADODB.Connection
    Dim rst As ADODB.Recordset
    Dim varRecordset() As Variant
    Dim i As Integer
    Dim j As Integer
    Set cnn = CurrentProject.Connection
    Set rst = New ADODB.Recordset
    rst.Open "tblUnternehmen", cnn, adOpenForwardOnly, adLockOptimistic
    varRecordset = rst.GetRows()
    For i = 0 To UBound(varRecordset, 2)
        For j = 0 To UBound(varRecordset, 1)
            Debug.Print varRecordset(j, i)
        Next j
    Next i
    rst.Close
    Set rst = Nothing
    Set cnn = Nothing
End Sub
```

Listing 8.18: Einlesen der Daten einer Datensatzgruppe in ein Array

9

DAO

DAO (*Data Access Objects*) ist neben *ADO* (*Active Data Objects*) eine der beiden Bibliotheken für den Zugriff auf die Daten in einer Access-Datenbank. In Access 97 war DAO noch Alleinunterhalter auf diesem Gebiet, ab Access 2000 kam dann ADO hinzu. ADO war vor langer Zeit eigentlich als legitimer Nachfolger von DAO vorgesehen, denn es ermöglicht sowohl die Verwendung von herkömmlichen Access-Datenbanken als auch von Access-Projekten (*.adp*), die ebenfalls mit Access 2000 eingeführt wurden. Diese Access-Projekte wurden seit Access 2007 nicht mehr wesentlich weiterentwickelt, sondern nur noch dem Einsatz mit dem SQL Server 2005/2008 angepasst – beispielsweise für die Verwendung der neuen Datentypen in SQL Server 2008. Und es kommt noch besser: Microsoft empfiehlt in Zusammenhang mit dem SQL Server den Einsatz von ODBC.

Im Gegensatz zu ADO hat Microsoft nur noch DAO für die neue Access-Version weiterentwickelt und so ist im Rahmen der neuen Office-Version auch eine neue DAO-Version entstanden, die sich nunmehr *Microsoft Office 14.0 Access database engine Objects* nennt.

In Access-Projekten, die den Microsoft SQL Server als Datenbank-Backend verwenden, ist ADO die einzige Wahl. Um von einer Access-Datenbank, also etwa einer *.accdb*-Datei, an die Daten in einer SQL Server-Datenbank heranzukommen, müssen Sie die benötigten Tabellen zunächst per ODBC verknüpfen und können dann per DAO oder ADO darauf zugreifen. Das ist in jedem Fall nur die zweitbeste Wahl, wird aber von Microsoft seit Access 2007 so propagiert.

DAO hat also seit Access 2007 an Bedeutung zurückgewonnen, was auch damit zusammenhängt, dass DAO seit dieser Version nunmehr allein Bestandteil von Office und nicht mehr von Windows ist. Daneben gibt es eine Menge weiterer Gründe, die alle genauso die Frage beantworten könnten, warum DAO sich letzten Endes – zumindest für den Einsatz mit der neuen Engine ACE – durchgesetzt hat:

» DAO gibt es bereits einige Zeit und es läuft mittlerweile stabil und fehlerfrei.

» DAO ist für den Einsatz mit der ACE-Engine optimiert und in Zusammenarbeit damit in vielen Fällen schneller als ADO.

» Einige Features von Access, etwa die Verwendung der *RecordsetClones* in Formularen oder das Komprimieren und Reparieren per VBA, erfordern den Einsatz von DAO und auch die *Recordset*-Eigenschaft von Formularen und liefert standardmäßig ein DAO-Recordset zurück.

Aus diesen Gründen lässt sich schon erkennen, wann die Anwendung von DAO Sinn macht und wann man eher ADO den Vorzug geben sollte.

Wenn Sie eine Access-Anwendung mit einem überschaubaren Lebenszyklus planen, deren Anwendungszweck keine Wünsche nach einer Erweiterung in Richtung eines »größeren« Systems, also etwa Microsoft SQL Server, beinhaltet, sind Sie mit DAO gut bedient. Teilweise ist ADO sicher performanter (etwa beim Schreiben von Daten), aber es unterstützt auch nicht die neuen Features von Access.

BEISPIELDATENBANK

Die Beispieldatei zu diesem Kapitel finden Sie unter dem Namen *DAO2010.accdb* auf *www.acciu.de/aeb2010*.

Wenn Sie eine bestehende Access-Anwendung weiterentwickeln, die mit DAO arbeitet, hängt es ebenfalls von den Zukunftsplänen mit dieser Anwendung ab – bei einer geplanten Vergrößerung in Richtung SQL Server macht ein Umstieg auf ADO definitiv

Sinn – je früher, desto besser. Je mehr hier noch in DAO hinzuprogrammiert wird, desto mehr ist anschließend nach ADO umzuwandeln. Sie können natürlich auch mit beiden Bibliotheken parallel arbeiten – und so bei Bedarf neue Bestandteile mit ADO entwickeln und Bestehendes Stück für Stück umwandeln.

Für den Fall, dass Sie aufgrund der oben genannten Gründe DAO für die richtige Datenzugriffstechnik unter VBA halten, finden Sie in den nächsten Abschnitten alles Wichtige zu diesem Thema. Anderenfalls überspringen Sie dieses Kapitel einfach und wenden sich direkt dem Kapitel »ADO« zu, das Sie im Download zu diesem Buch finden.

9.1 DAO und ADO im Einsatz

In der Access-Welt gibt es Datenbanken, die nur DAO, nur ADO oder auch beide Objektbibliotheken verwenden. Wenn Sie niemals Probleme bekommen möchten, weil Sie die (etwas anderen) Methoden etwa eines *Recordset*-Objekts von DAO unter ADO einsetzen, sollten Sie direkt klarstellen, mit welchem Objektmodell Sie aktuell arbeiten (in Teilen ist dies mit der neuen DAO-Bibliothek kein Problem mehr, da diese ein neues *Recordset2*- und ein *Field2*-Objekt mitliefert – dazu später mehr).

Im *Verweise*-Dialog (zu öffnen mit dem Menüeintrag *Extras/Verweise* der VBA-Entwicklungsumgebung) finden Sie gegebenenfalls beide Bibliotheken (siehe Abbildung 9.1). Hier wird deutlich, dass Microsoft tatsächlich auf DAO als Datenzugriffstechnologie für Access setzt: Standardmäßig ist lediglich noch ein Verweis auf die neue DAO-Bibliothek vorhanden, ADO wird gar nicht mehr automatisch berücksichtigt.

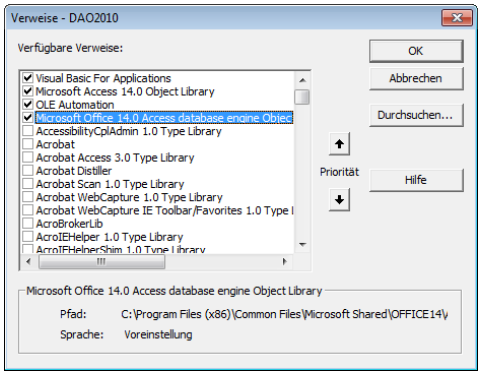


Abbildung 9.1: Aktivierter DAO-Verweis

Wenn Sie beide Objektbibliotheken gleichzeitig eingebunden haben, kann es vorkommen, dass Sie eines der gleichnamigen Elemente einer der Objektbibliotheken verwenden und auf eine nicht vorhandene oder anders einzusetzende Methode oder Eigenschaft

zugreifen. Falls Sie nämlich beide Objektmodelle einsetzen und ein in beiden vorkommendes Objekt benutzen, greift Access auf das im Verweise-Fenster weiter oben angeordnete Objekt zu. Dem lässt sich vorbeugen: Deklarieren Sie die Objekte einfach direkt mit Bezug auf das richtige Objektmodell, indem Sie den VBA-Namen der entsprechenden Bibliothek voranstellen:

```
'Deklaration eines Recordset der DAO-Bibliothek
 Dim rst As DAO.Recordset
'Deklaration eines Recordset der ADO-Bibliothek
 Dim rst As ADODB.Recordset
```

9.2 Das DAO-Objektmodell

Der erste Teil dieses Kapitels verschafft Ihnen einen kleinen Überblick über die Objekte von DAO und zeigt Ihnen, wie Sie auf diese zugreifen. Sie finden hier ausdrücklich keine Referenz mit allen Objekten, Eigenschaften und Methoden, sondern erhalten eine Vorstellung von den wichtigsten Elementen dieser Objektbibliothek. Das Verständnis gerade der obersten Objekte in der DAO-Hierarchie ist in vielen Fällen sehr wichtig. Man kommt zwar lange Zeit ohne großartiges Hintergrundwissen damit aus, einfach ein *Database*-Objekt und eines oder mehrere Recordsets zu verwenden, aber irgendwann tauchen die ersten Fragen auf. Im Anschluss an diesen Teil erfahren Sie anhand häufig vorkommender Anwendungen mehr über den Praxiseinsatz von DAO.

9.2.1 Zugriff auf die Elemente des Objektmodells

Abbildung 9.2 zeigt das DAO-Objektmodell ohne Eigenschaften und Methoden. In der Hierarchie folgt immer eine Auflistung auf ein Objekt und umgekehrt, wobei Auflistungen durch kursive Schreibweise hervorgehoben sind. Da die neuen Klassen *Recordset2* und *Field2* von den älteren Objekten *Recordset* und *Field* abgeleitete Klassen sind, finden Sie diese stellvertretend für beide Klassen in der Abbildung. *Recordset* und *Field* sind jedoch nach wie vor vorhanden. Alle Zugriffe auf die Objekte erfolgen theoretisch über das oberste Element der Hierarchie namens *DBEngine*. Es gibt allerdings Möglichkeiten für den Quereinstieg: Der oft benötigte Zugriff auf die aktuelle Datenbank erfordert beispielsweise nur die Instanzierung eines *Database*-Objekts durch Zuweisung eines Verweises per *CurrentDB*-Methode. Jedes Objekt verfügt zusätzlich über eine *Properties*-Auflistung, in der eingebaute wie auch benutzerdefinierte Eigenschaften des Objekts gespeichert sind.

Die grau gezeichneten Objekte werden im Access 2007-Datenbankformat nicht mehr unterstützt (auch Access 2010 verwendet dieses .accdb-Format), nur noch in Datenbanken, die im *.mdb*-Format von Access 2000 oder 2002/2003 geöffnet werden. Regulär kann der

Zugriff auf beliebige Klassen des Objektmodells auf verschiedenen Wegen erfolgen. Der Zugriff auf ein *Database*-Objekt lässt sich etwa auf folgende Arten realisieren:

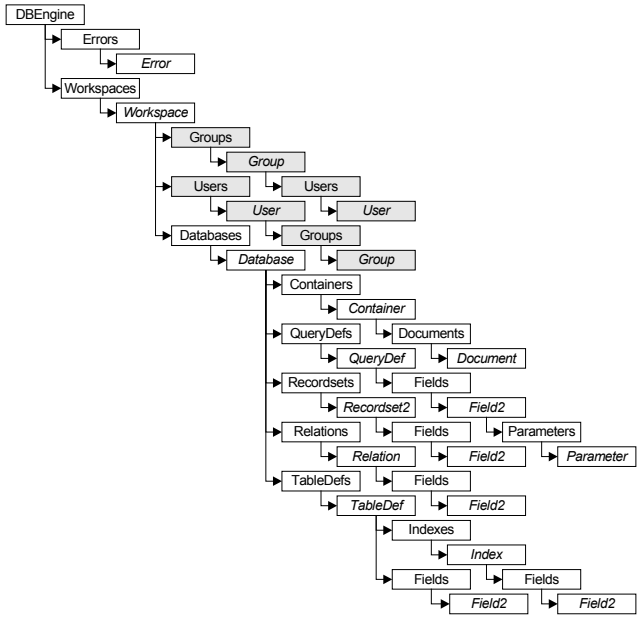


Abbildung 9.2: Das DAO-Objektmodell im Überblick

» Über den Namen des Objekts in der Auflistung als Zeichenkette oder Variable (zu verwenden, wenn der Name variabel ist):

```
DBEngine.Workspaces("<Workspacename>").Databases("<Datenbankname>")
DBEngine.Workspaces(strWorkspacename).Databases(strDatenbankname)
```

» Über den Objektnamen:

```
DBEngine.Workspaces!<Workspacename>.Databases!<Datenbankname>
```

» Über die Ordinalzahl:

```
DBEngine.Workspaces(0).Databases(0)
```

» Über die Ordinalzahl (abgekürzte Variante):

```
DBEngine(0)(0)
```

Die Abkürzung *DBEngine(0)(0)* profitiert gleich von zwei Abkürzungen: Erstens ist *Workspaces* das Default-Element des *DBEngine*-Objekts und *Databases* das Default-Element des *Workspace*-Elements, und zweitens liefern die jeweils ersten, mit der Zahl *0* versehenen Auflistungselemente den Verweis auf das hier benötigte aktuelle *Database*-Objekt.

9.2.2 Deklarieren und instanzieren

Theoretisch kann man auf Objekte des DAO-Objektmodells zugreifen, ohne überhaupt eine Variable zu verwenden. Die folgende Anweisung gibt etwa die Anzahl Datensätze einer Tabelle aus, wenn sie im Direktfenster abgesetzt wird (in einer Zeile):

```
Debug.Print DBEngine.Workspaces(0).Databases(0).TableDefs("tblMitarbeiter").RecordCount
```

In VBA-Routinen ist dies auch möglich, aber es ist kein guter Programmierstil und außerdem leidet unter Umständen die Performance darunter – nämlich dann, wenn Sie mehr als einmal auf ein Objekt zugreifen möchten. Dann macht es Sinn, eine Objektvariable anzulegen und mit dieser auf das gewünschte Objekt zu verweisen. Die Routine aus Listing 9.1 deklariert zunächst die gewünschten Objekte und weist diese anschließend den Objektvariablen zu – dabei arbeitet sie sich in der Objekthierarchie von oben nach unten durch. Während die Objekte *Workspace*, *Database* und *Recordset* explizit zugewiesen werden, erfolgt der Zugriff auf die *Field*-Elemente des *TableDef*-Objekts per *For Each*-Schleife über dessen *Fields*-Auflistung.

```
Public Sub Tabelleneigenschaften()
    Dim wrk As DAO.Workspace
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    Dim fld As DAO.Field
    Set wrk = DBEngine.Workspaces(0)
    Set db = wrk.Databases(0)
    Set tdf = db.TableDefs("tblMitarbeiter")
    For Each fld In tdf.Fields
        Debug.Print fld.Name
    Next fld
    Set tdf = Nothing
    Set db = Nothing
    Set wrk = Nothing
End Sub
```

Listing 9.1: Beispiel für das Deklarieren und Instanzieren von Objekten des DAO-Objektmodells

Kürzer – und in vielen Fällen sinnvoller, wie sich weiter unten zeigen wird – ist folgende Variante. Dabei wird die Referenz direkt über die Methode *CurrentDB* erzeugt. Diese Variante ist die meistverbreitete:

```
Public Sub TabelleneigenschaftenKurz()
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    Dim fld As DAO.Field
```

```
    Set db = CurrentDb
    Set tdf = db.TableDefs("tblMitarbeiter")
    For Each fld In tdf.Fields
        Debug.Print fld.Name
    Next fld
    Set tdf = Nothing
    Set db = Nothing
End Sub
```

Listing 9.2: Erstellen einer Objektreferenz auf ein Database-Objekt per CurrentDB

9.2.3 Auf Auflistungen zugreifen

In Listing 9.2 haben Sie bereits ein Beispiel für das Durchlaufen einer Auflistung des DAO-Objektmodells kennen gelernt. Es gibt zwei Möglichkeiten, die zahlreichen Auflistungen dieses Objektmodells zu durchlaufen: die *For...Next*-Schleife und die *For Each*-Schleife.

Die *For...Next*-Schleife erscheint als eines der in fast allen Programmiersprachen vertretenen Konstrukte möglicherweise intuitiver:

```
Public Sub TabellenfelderMitForNext()
    …
    Dim i As Integer
    Set db = CurrentDb
    Set tdf = db.TableDefs("tblMitarbeiter")
    For i = 0 To tdf.Fields.Count - 1
        Set fld = tdf.Fields(i)
        Debug.Print fld.Name
    Next i
    …
End Sub
```

Listing 9.3: Auflistung per For...Next-Schleife durchlaufen

Alternativ dazu finden Sie hier die *For Each*-Variante. Diese Variante spart die Laufvariable *i*, die Ermittlung der Anzahl der Elemente der Auflistung und die explizite Zuweisung des *Field*-Objekts ein. Dafür ist die folgende Prozedur aber unmerklich langsamer als die vorherige:

```
Public Sub TabellenfelderKurz()
    …
    Set db = CurrentDb
    Set tdf = db.TableDefs("tblMitarbeiter")
```

```
    For Each fld In tdf.Fields
        Debug.Print fld.Name
    Next fld
    …
End Sub
```

Listing 9.4: Zugriff auf eine Auflistung per *For Each*-Schleife

9.2.4 Punkte und Ausrufezeichen

DAO und ADO haben eines mit den übrigen Objekten von Access gemeinsam: die gemischte Verwendung von Punkten (.) und Ausrufezeichen (!) als Trennzeichen zwischen zwei miteinander in Beziehung stehenden Objekten, zum Beispiel:

```
Debug.Print rstMitarbeiter!MitarbeiterID.Value
```

Bezüge zu Steuerelementen in Formularen sehen beispielsweise so aus:

```
Debug.Print Forms!frmMitarbeiter!MitarbeiterID
```

Dort würde aber auch Folgendes funktionieren:

```
Debug.Print Forms.frmMitarbeiter.MitarbeiterID
```

Wo setzt man nun welches Zeichen ein und warum funktionieren manchmal beide? Diese Fragen sind relativ leicht zu beantworten. Den Punkt verwendet man immer, wenn das nachgestellte Element ein Access-internes Objekt ist, das Ausrufezeichen kommt bei nachgestellten benutzerdefinierten Elementen zum Zuge. Dabei gilt als benutzerdefiniert alles, was Sie selbst zur Datenbank hinzugefügt haben – Tabellen, Tabellenfelder, Indizes, Formulare und Berichte und die enthaltenen Steuerelemente. Alles andere wird von Access gestellt: eingebaute Objekte, Eigenschaften, Methoden und Ereignisse.

Warum in manchen Fällen Punkt und Ausrufezeichen funktionieren? Das geschieht nur scheinbar. Erfahrungen zeigen, dass sporadisch Probleme auftauchen, wenn man die Punkt-Notation an Stelle der Ausrufezeichen-Notation verwendet, obwohl Letztere angezeigt ist. Um allen Problemen aus dem Wege zu gehen, schreiben Sie einfach vor alle selbst erstellten Objekte ein Ausrufezeichen.

9.3 DBEngine

Das *DBEngine*-Objekt ist der »Kopf« des DAO-Objektmodells. Beim Start von Access wird automatisch eine Instanz dieses Objekts erzeugt. Es enthält zwei Auflistungen: Die *Workspaces*-Auflistung und die *Errors*-Auflistung.

DAO-Fehler verfolgen

Die *Errors*-Auflistung ist ein DAO-eigenes Fehlerobjekt, das Fehler separat vom *Err*-Objekt von VBA speichert. Die *Errors*-Auflistung von DAO kann allerdings mehrere Fehler speichern. Der Grund hierfür ist, dass manche DAO-Operationen mehr als einen Fehler auslösen können. Um dennoch alle Fehler auszulesen, speichert DAO diese in der *Errors*-Auflistung. Der oder die Fehler einer Operation werden so lange in der *Errors*-Auflistung gespeichert, bis die nächste fehlerhafte Anweisung ausgeführt wird.

Mehrere Instanzen

In manchen Fällen benötigen Sie eine zusätzliche Instanz des *DBEngine*-Objekts – etwa, wenn Sie auf eine geschützte Datei zugreifen möchten. In diesem Fall erzeugen Sie die neue Instanz mit den folgenden zwei Zeilen:

```
Dim objDBEngine As DAO.DBEngine
Set objDBEngine = New DAO.DBEngine
```

9.4 Workspace — Arbeitsbereich oder Sitzung?

Die zweite Auflistung unterhalb des *DBEngine*-Objekts enthält die Workspaces einer Datenbank. Workspace ist eigentlich die Übersetzung für Arbeitsbereich, aber Benutzersitzung trifft in diesem Fall eher den Punkt. Ein Workspace wird mit dem Anmelden eines Benutzers an eine Datenbank erzeugt und mit dessen Abmeldung wieder zerstört.

Sie können in einer einzigen Access-Instanz mehrere *Workspace*-Objekte erzeugen – das macht vor allem Sinn, wenn Sie automatisch die Wechselwirkungen von Sperrmechanismen oder Transaktionen im Mehrbenutzerbetrieb testen möchten.

Um ein neues *Workspace*-Objekt zu erzeugen, reicht die *CreateWorkspace*-Methode aus, an die *Workspaces*-Auflistung wird es allerdings nicht automatisch angehängt. Das Erzeugen eines *Workspace*-Objekts und das Anhängen an die entsprechende Auflistung zeigt folgendes Beispiel:

```
Public Sub WorkspacesSample()
    Dim wrk As DAO.Workspace
    Dim wrkExt As DAO.Workspace
    'Neues Workspace-Objekt erzeugen...
    '(Arbeitsgruppe ist in diesem Fall die Standardsarbeitsgruppe)
    Set wrkExt = DBEngine.CreateWorkspace("#New Workspace#", "admin", "")
    '...und an die Workspaces-Auflistung anhängen
    Workspaces.Append wrkExt
    'Namen der Workspaces ausgeben
```

```
    Debug.Print "Workspaces:"
    For Each wrk In DBEngine.Workspaces
        Debug.Print wrk.Name
    Next wrk
End Sub
```

Listing 9.5: Anlegen eines neuen Workspace-Objekts und Anhängen an die Workspaces-Auflistung

9.4.1 Auflistungen des Workspace-Objekts

Das *Workspace*-Objekt enthält drei Auflistungen, die weiter unten ausführlicher beschrieben werden. Für den Einsatz mit Datenbanken im Access 2007/2010-Format ist nur noch die *Databases*-Auflistung interessant, die anderen beiden und das damit verbundene Sicherheitssystem werden nicht mehr unterstützt (derweil diese Einträge durchaus noch verwendet werden können – eben für Datenbanken, die mit älteren Versionen von Access erstellt wurden):

» *Databases*: Enthält alle *Database*-Objekte, die im Kontext der Sitzung geöffnet wurden.

» *Users*: Enthält eine Auflistung aller Benutzer der aktuellen Arbeitsgruppe.

» *Groups*: Enthält eine Auflistung aller Benutzergruppen der aktuellen Arbeitsgruppe.

9.4.2 Aufgaben des Workspace-Objekts

Das *Workspace*-Objekt stellt eine ganze Reihe Funktionen zur Verfügung. Eine davon ist besonders wichtig: das Verwalten von Transaktionen. Weitere Informationen hierzu finden Sie unter »Transaktionen« ab Seite 587 .

Darüber hinaus enthält es Funktionen zum Erzeugen neuer Datenbanken (*CreateDatabase*) oder zum Öffnen weiterer Datenbanken im gleichen Workspace (*OpenDatabase*). Informationen zu weiteren Elementen finden Sie in der Onlinehilfe.

9.4.3 Datenbanken erzeugen und öffnen

Im Kontext eines *Workspace*-Objekts lassen sich neue Datenbanken erzeugen und bestehende Datenbanken öffnen. Eine neue Datenbank erzeugen Sie mit der *CreateDatabase*-Methode, eine bestehende Datenbank öffnen Sie mit *OpenDatabase*.

Eine aus der aktuellen Datenbank heraus erzeugte neue Datenbank könnte beispielsweise dem Auslagern temporärer Tabellen dienen. Auf diese Weise würden temporäre Daten die Datenbank nicht unnötig aufblähen. Besonders interessant ist diese Vorgehensweise, wenn die Datenbankgröße ein kritisches Maß erreicht hat und die tempo-

rären Daten diese sprengen könnten. Um eine neue Datenbank im Format von Access 2007/2010 zu erstellen, verwenden Sie die folgende Anweisung:

```
DbEngine.CreateDatabase (<Datenbankname>, dbLangGeneral, dbVersion140
```

9.5 Aktuelle Datenbank referenzieren

Um per DAO auf die aktuelle Datenbank zuzugreifen, gibt es zwei Möglichkeiten: Entweder man arbeitet sich vom obersten Element der Objekthierarchie bis zum *Database*-Objekt vor und verweist darauf oder man verwendet direkt die *CurrentDB*-Methode, um einen Verweis zu erhalten:

```
DBEngine.Workspaces(0).Databases(0)
```

oder abgekürzt

```
DBEngine(0)(0)
```

verweisen dabei auf die aktuelle Datenbank, während

```
CurrentDB
```

einen Verweis auf eine neue Objektinstanz erstellt und diesen zurückliefert.

Welche Vor- und Nachteile gibt es nun und welche Variante setzt man wann ein? Die *DBEngine*-Version ist die einzige der beiden Möglichkeiten, mit der Sie von außerhalb, also etwa von Excel oder Word aus, auf eine Access-Datenbank zugreifen können. Intern sind beide Varianten möglich. *CurrentDB* hat dabei den Vorteil, dass es immer auf die aktuelle Datenstruktur zugreift, während dies bei *DBEngine(0)(0)* nicht immer der Fall ist. *CurrentDB* aktualisiert nämlich sämtliche im *Database*-Objekt enthaltenen Auflistungen, während bei der Verwendung von *DBEngine* ein zusätzlicher Aufruf der *Refresh*-Methode erforderlich ist. Den von *CurrentDB* zurückgegebenen Verweis muss man speichern, um länger als für die Dauer der aktuellen Anweisung auf das *Database*-Objekt und untergeordnete Objekte zugreifen zu können; greift man über *CurrentDB* auf untergeordnete Objekte zu, sind diese nur für die Dauer der Anweisung existent. Der Nachteil von *CurrentDB* ist, dass es minimal langsamer als *DBEngine(0)(0)* ist – was sich allerdings im normalen Betrieb kaum bemerkbar macht.

Ein Beispiel für dieses Verhalten finden Sie in Abbildung 9.3. Ausnahmen für dieses Verhalten gibt es auch: Wenn Sie mit folgendem Quellcode eine Datensatzgruppe erzeugen, verschwindet das Objekt nicht von alleine wieder in die ewigen Jagdgründe:

```
Public Sub CurrentDBPersistenzMitRecordset()
    Dim rst As Recordset
    Set rst = CurrentDb.OpenRecordset("tblMitarbeiter", dbOpenDynaset)
```

```
    Debug.Print rst.RecordCount
    Set rst = Nothing
End Sub
```

Listing 9.6: Erfolgreiches Erzeugen eines Objektverweises mit *CurrentDB* und untergeordnetem Recordset-Objekt

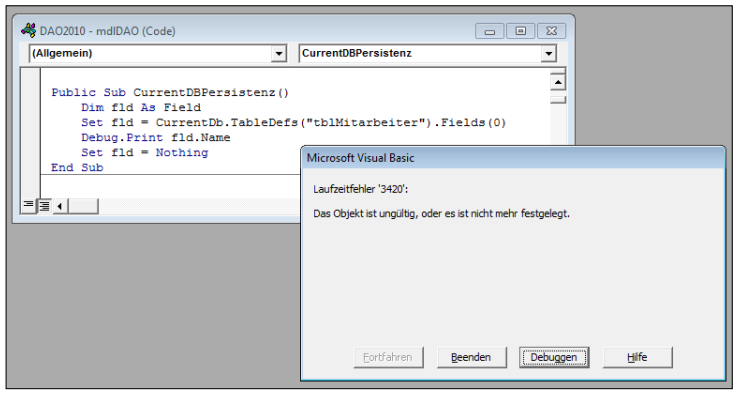


Abbildung 9.3: Fehler beim Versuch, einen Verweis auf ein untergeordnetes Objekt mit *CurrentDB* herzustellen

9.5.1 Users und Groups

Diese beiden Auflistungen werden in Datenbanken, die mit Access 2010 erstellt wurden, nicht mehr unterstützt. Sie können mit Access 2010 zwar durchaus Anwendungen öffnen, die mit Access 2000 oder Access 2002/2003 erstellt wurden und damit das Sicherheitssystem noch unterstützen. Da Access das Sicherheitssystem aber nun bereits seit zwei Versionen nicht mehr unterstützt, gehen wir in diesem Buch nicht mehr darauf ein.

9.6 Das Database-Objekt

Das *Database*-Objekt enthält einige Auflistungen, Eigenschaften und Methoden, von denen vier Kategorien in den folgenden Abschnitten besonderes Augenmerk erhalten:

» Manipulation des Datenmodells

» Zugriff auf Auflistungen wie *QueryDefs*, *Recordsets*, *Relations* und *TableDefs*

» Anwenden der *OpenRecordset*-Methode

» Ausführen von Aktionsabfragen

9.6.1 Manipulation des Datenmodells

Tabellen, Felder und Beziehungen lassen sich nicht nur mit den *Data Definion Language* (DDL)-Anweisungen von SQL erzeugen, bearbeiten und löschen, sondern auch per DAO. Dazu stehen die folgenden Methoden zur Verfügung:

» *CreateDatabase* (*Workspace*-Objekt): Erzeugen einer Datenbankdatei

» *CreateTableDef* (*Database*-Objekt): Erstellen einer Tabelle

» *CreateProperty* (*Database*-Objekt): Anlegen einer Datenbank-Eigenschaft

» *CreateQueryDef* (*Database*-Objekt): Erstellen einer Abfrage

» *CreateRelation* (*Database*-Objekt): Erstellen einer Tabellenbeziehung

» *CreateField* (*TableDef*-Objekt, *Relations*-Objekt, *Index*-Objekt): Erstellen eines Feldes

» *CreateIndex* (*TableDef*-Objekt): Erstellen eines Indexes

» *CreateProperty* (*TableDef*-Objekt, *Index*-Objekt): Erstellen einer Tabellen- oder Index-Eigenschaft

9.6.2 Erstellen einer Tabelle

Die folgende Routine zeigt, wie man eine neue Tabelle namens *tblUnternehmen* und die beiden Felder *UnternehmenID* und *Unternehmen* anlegt und in der Datenbank verfügbar macht. Zu beachten ist hier, dass Sie eine Menge Tabellen und Felder mit *CreateTable* und *CreateField* erzeugen können – aber wenn Sie diese nicht an die entsprechenden Auflistungen *TableDefs* beziehungsweise *Fields* anhängen, sind alle erzeugten Objekte mit Ablauf der Routine wieder verschwunden. Abbildung 9.4 zeigt die Abhängigkeiten der an der Erstellung einer Tabelle beteiligten Elemente der DAO-Objektbibliothek. Wenn Sie sich merken, dass Sie einfach zuerst die einzelnen Elemente erzeugen und diese dann an die Auflistung des übergeordneten Elements anhängen müssen, haben Sie das Objektmodell bezüglich der Erstellung von Objekten schon fast im Griff (für die Erstellung einer Tabelle können Sie statt des *Field2*-Objekts (kursiv abgebildet) auch das ältere *Field*-Objekt verwenden).

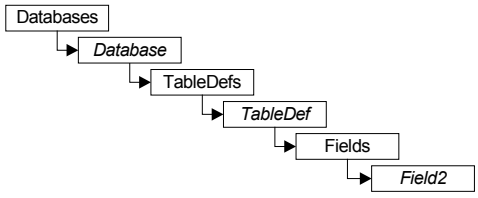


Abbildung 9.4: Hierarchie der beim Erstellen einer Tabelle beteiligten Elemente der DAO-Objektbibliothek

Tabelle 9.6 enthält alle Konstanten, die Sie für die Konstante *Type* der *CreateField*-Anweisung verwenden können.

```
Public Sub TabelleAnlegen()
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    Dim fld As DAO.Field
    'Verweis auf aktuelle Datenbank
    Set db = CurrentDb
    'Verweis auf neues TableDef-Objekt
    Set tdf = db.CreateTableDef
    'Name der Tabelle zuweisen
    tdf.Name = "tblUnternehmen"
    'Feld neu erstellen und per Objektvariable referenzieren
    Set fld = tdf.CreateField("UnternehmenID", dbDouble)
    'Feld an die Feldliste der Tabelle anhängen
    tdf.Fields.Append fld
    'Gleiches Spiel mit einem zweiten Feld
    Set fld = tdf.CreateField("Unternehmen", dbText, 255)
    tdf.Fields.Append fld
    'Die bisher nicht vorhandene Tabelle zur TableDefs-Auflistung hinzufügen
    db.TableDefs.Append tdf
    'Auflistung aktualisieren
    db.TableDefs.Refresh
    'Navigationsbereich aktualisieren
    Application.RefreshDatabaseWindow
    Set db = Nothing
End Sub
```

Listing 9.7: Erstellen einer Tabelle mit DAO

9.6.3 Autowert anlegen

Wenn Sie das Primärschlüsselfeld der Tabelle als Autowertfeld auslegen möchten, brauchen Sie lediglich ein einziges Attribut zu setzen.

Dazu fügen Sie zwischen der *CreateField*-Anweisung und der *Append*-Anweisung zum Hinzufügen des Feldes die folgende Zeile ein:

```
'Attributes-Eigenschaft auf Autowert einstellen
fld.Attributes = dbAutoIncrField
```

Konstante	Zahlenwert	Datentyp
dbBoolean	1	Boolean
dbByte	2	Byte
dbInteger	3	Integer
dbLong	4	Long
dbCurrency	5	Currency
dbSingle	6	Single
dbDouble	7	Double
dbDate	8	Date/Time
dbBinary	9	Binary
dbText	10	Text
dbLongBinary	11	Long Binary (OLE Object)
dbMemo	12	Memo
dbGUID	15	GUID
dbBigInt	16	Big Integer
dbVarBinary	17	VarBinary (OLE-Objekt)
dbChar	18	Char
dbNumeric	19	Numeric
dbDecimal	20	Decimal
dbFloat	21	Float
dbTime	22	Time
dbTimeStamp	23	Time Stamp
dbAttachment	101	Attachment
dbComplexByte	102	Byte (mehrwertiges Feld)
dbComplexInteger	103	Integer (mehrwertiges Feld)
dbComplexLong	104	Long (mehrwertiges Feld)
dbComplexSingle	105	Single (mehrwertiges Feld)
dbComplexDouble	106	Double (mehrwertiges Feld)
dbComplexGUID	107	GUID (mehrwertiges Feld)
dbComplexDecimal	108	Decimal (mehrwertiges Feld)
dbComplexText	109	Text (mehrwertiges Feld)

Tabelle 9.6: Konstanten für den Datentyp in der *CreateField*-Anweisung

9.6.4 Attachment-Feld anlegen

Die folgende Routine erzeugt eine Tabelle mit einem Feld des Typs *Attachment*. Die passende Konstante finden Sie in Tabelle 9.6.

```
Public Sub TabelleMitAttachmentfeldAnlegen()
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    Dim fld As DAO.Field2
    Set db = CurrentDb
    Set tdf = db.CreateTableDef
    tdf.Name = "tblDateien"
    Set fld = tdf.CreateField("DateiID", dbLong)
    fld.Attributes = dbAutoIncrField
    tdf.Fields.Append fld
    Set fld = tdf.CreateField("Datei", dbAttachment)
    tdf.Fields.Append fld
    db.TableDefs.Append tdf
    db.TableDefs.Refresh
    Application.RefreshDatabaseWindow
    Set db = Nothing
End Sub
```

Listing 9.8: Anlegen von Feldern mit den neuen Datentypen von Access 2007

Abbildung 9.5 zeigt die Tabelle mit dem neuen Attachment-Feld.

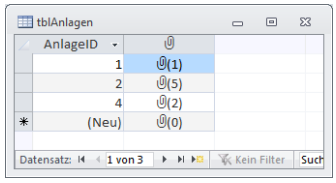


Abbildung 9.5: Eine per DAO erstellte Tabelle mit Attachment-Feld

9.6.5 Mehrwertige Felder anlegen

Das Anlegen von mehrwertigen Feldern ist nicht ganz so einfach wie bei Attachment-Feldern – zumindest nicht, wenn man diese Felder anschließend auch Gewinn bringend einsetzen möchte. Aber der Reihe nach: Schauen Sie sich zunächst einmal das folgende Listing an, das genau wie in den vorhergehenden Beispielen eine Tabelle mit einem ID-Feld und einem weiteren Feld eines bestimmten Datentyps anlegt. In diesem Fall handelt es sich dabei um ein mehrwertiges Feld zum Speichern von Text.

```
Public Sub TabelleMitKomplexemFeldAnlegen()
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    Dim fld As DAO.Field2
```

```
    Set db = CurrentDb
    Set tdf = db.CreateTableDef
    tdf.Name = "tblKomplexesFeld"
    Set fld = tdf.CreateField("KomplexesFeldID", dbLong)
    fld.Attributes = dbAutoIncrField
    tdf.Fields.Append fld
    Set fld = tdf.CreateField("KomplexesFeld", dbComplexText)
    tdf.Fields.Append fld
    db.TableDefs.Append tdf
    db.TableDefs.Refresh
    Application.RefreshDatabaseWindow
    Set db = Nothing
End Sub
```

Listing 9.9: Anlegen einer Tabelle mit einem komplexen Feld

Das Anlegen der Tabelle und des *ComplexText*-Feldes funktioniert, aber der Versuch, diesem Feld in der Datenblattansicht der Tabelle Einträge hinzuzufügen, schlägt fehl.

Kein Wunder, wie ein Blick in die Entwurfsansicht und die *Nachschlagen*-Eigenschaften dieses Feldes zeigt: Die für das Hinzufügen verantwortliche Eigenschaft *Wertlistenbearbeitung zulassen* hat den Wert *Nein*. Dies ändern Sie schnell wie in Abbildung 9.6.

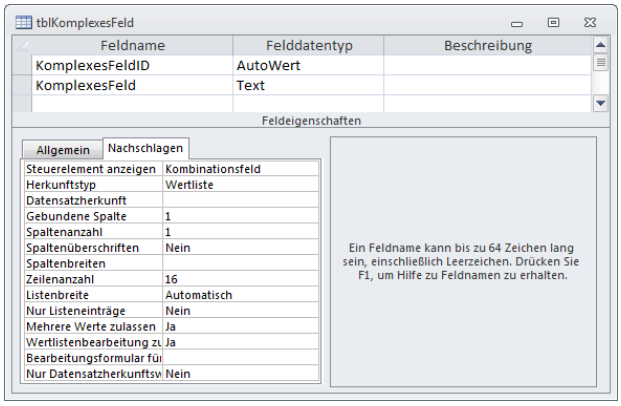


Abbildung 9.6: Eigenschaften eines per DAO und VBA erstellten mehrwertigen Feldes

Wie aber soll man diese Eigenschaften beim Anlegen eines solchen Feldes per DAO einstellen, wenn diese offensichtlich noch nicht vorhanden sind?

Die Lösung ist der Einsatz von *Properties*, die Sie zunächst festlegen und dann auf die gewünschten Werte einstellen. Das folgende Beispiel legt die gleiche Tabelle wie oben an, nur dass die Eigenschaft *Wertlistenbearbeitung zulassen* auf den Wert *Ja* eingestellt wird:

```
Public Sub TabelleMitAenderbaremKomplexemFeldAnlegen()
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    Dim fld As DAO.Field2
    Dim prp As DAO.Property
    Set db = CurrentDb
    Set tdf = db.CreateTableDef
    CurrentDb.TableDefs.Delete "tblKomplexesFeld"
    tdf.Name = "tblKomplexesFeld"
    Set fld = tdf.CreateField("KomplexesFeldID", dbLong)
    fld.Attributes = dbAutoIncrField
    tdf.Fields.Append fld
    Set fld = tdf.CreateField("KomplexesFeld", dbComplexText)
    tdf.Fields.Append fld
    db.TableDefs.Append tdf
    Set prp = fld.CreateProperty("AllowValueListEdits", dbBoolean)
    prp.Value = True
    fld.Properties.Append prp
    Set prp = fld.CreateProperty("RowSourceType", dbText)
    prp.Value = "Wertliste"
    fld.Properties.Append prp
    db.TableDefs.Refresh
    RefreshDatabaseWindow
    Set db = Nothing
End Sub
```

Listing 9.10: Anlegen einer Tabelle mit einem mehrwertigen Feld, das auf einer Wertliste basiert und benutzerdefinierte Einträge zulässt

Die Tabelle arbeitet nun in der Datenblattansicht wie erwartet: Beim Klick auf das Kombinationsfeld zum Auswählen des Inhalts für das mehrwertige Feld öffnet sich nicht nur die Liste, sondern es erscheint auch das Symbol zum Öffnen des Dialogs zum Ändern der Listeneinträge (siehe Abbildung 9.7).

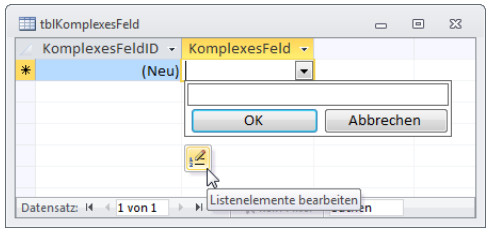


Abbildung 9.7: Eine per DAO erstellte Tabelle mit erweiterbarem, mehrwertigem Feld

Weitere Properties von Nachschlage- und mehrwertigen Feldern

Tabelle 9.7 zeigt die übrigen Eigenschaften von Nachschlage- und mehrwertigen Feldern mit der englischen Bezeichnung und dem Datentyp.

Eigenschaft	Englische Bezeichnung	Datentyp
Herkunftstyp	RowSourceType	dbText/String
Datensatzherkunft	RowSource	dbText/String
Gebundene Spalte	BoundColumn	dbLong/Long
Spaltenanzahl	ColumnCount	dbInteger/Integer
Spaltenüberschriften	ColumnHeads	dbBoolean/Boolean
Spaltenbreiten	ColumnWidth	dbText/String
Zeilenanzahl	ListRows	dbInteger/Integer
Listenbreite	ListWidth	dbText/String
Nur Listeneinträge	LimitToList	dbBoolean/Boolean
Mehrere Werte zulassen	AllowMultipleValues	dbBoolean/Boolean
Wertlistenbearbeitung zulassen	AllowValueListEdits	dbBoolean/Boolean
Bearbeitungsformular für Listenelemente	ListItemsEditForm	dbText/String
Nur Datensatzherkunftswerte anzeigen	ShowOnlyRowSourceValues	dbBoolean/Boolean

Tabelle 9.7: Eigenschaften für Nachschlage- und mehrwertige Felder

9.6.6 Berechnete Felder anlegen

Berechnete Felder legen Sie natürlich nicht nur über den Tabellenentwurf, sondern auch per DAO an. Dafür gibt es eine neue Eigenschaft namens *Expression*. Den Datentyp legen Sie entsprechend dem Datentyp des Ergebnisses fest, also beispielsweise *dbText* (*10*) oder *dbLong* (*4*). Im folgenden Beispiel soll ein Feld namens *BerechnetesFeld* den Inhalt der beiden Zahlenfelder *Zahl1* und *Zahl2* addieren.

```
Public Sub TabelleAnlegenBerechnetesFeld()
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    Dim fld As DAO.Field2
    Dim prp As DAO.Property
    Set db = CurrentDb
    Set tdf = db.CreateTableDef
    On Error Resume Next
    CurrentDb.TableDefs.Delete "tblBerechnetesFeld"
    On Error GoTo 0
    tdf.Name = "tblBerechnetesFeld"
    Set fld = tdf.CreateField("BerechnetesFeldID", dbLong)
```

```
    fld.Attributes = dbAutoIncrField
    tdf.Fields.Append fld
    Set fld = tdf.CreateField("Zahl1", dbLong)
    tdf.Fields.Append fld
    Set fld = tdf.CreateField("Zahl2", dbLong)
    tdf.Fields.Append fld
    Set fld = tdf.CreateField("BerechnetesFeld", dbLong)
    fld.Expression = "Zahl1 + Zahl2"
    tdf.Fields.Append fld
    db.TableDefs.Append tdf
    db.TableDefs.Refresh
    Application.RefreshDatabaseWindow
    Set db = Nothing
End Sub
```

Neben Expression gibt es noch eine weitere Eigenschaft, die das Verhalten berechneter Felder beeinflusst. Dabei handelt es um die benutzerdefinierte Eigenschaft *ResultType*, der Sie eine der Datentyp-Konstanten zuweisen können.

Beim Anlegen eines berechneten Feldes mit der obigen Prozedur legt Access diese Eigenschaft automatisch entsprechend dem Datentyp an. Im Beispiel wurde hier *dbLong* angegeben, also wird die entsprechende Eigenschaft im Tabellenentwurf als *Long Integer* angezeigt.

Wenn das berechnete Feld jedoch beispielsweise Texte verketten soll, führt dies zu einem Fehler beziehungsweise zur Anzeige des Ausdrucks *#Typ!*.

Lassen Sie den Datentyp beim Erstellen des Feldes einfach wie in der folgenden Zeile weg, ermittelt Access selbst den Datentyp:

```
Set fld = tdf.CreateField("BerechnetesFeld")
```

Dabei kommen offensichtlich verschiedene Kriterien zum Zuge. Beispiele:

» Beteiligte Felder sind Textfelder, Operator ist &: Text

» Beteiligte Felder sind Textfelder, Operator ist +: Text

» Beteiligte Felder sind Zahlenfer, Operator ist &: Text

» Beteiligte Felder sind Zahlenfelder, Operator ist +: Zahl

9.6.7 Spaltenberechnungen festlegen

Seit Access 2007 können Sie in der Datenblattansicht eine zusätzliche Zeile mit Berechnungen über die gesamte Spalte anzeigen (siehe Abbildung 9.8).

9

Makros

Bis zu dieser Version von Access haben Entwickler zugunsten von VBA auf den Einsatz von Makros verzichtet. Mit Access 2010 könnte sich dies ändern: Erstens hat Microsoft die Entwicklungsumgebung für das Erstellen von Makros erheblich aufgepeppt.

Zweitens wurde der Befehlsumfang der Makros erweitert: Sie können damit nun auch direkt auf die in Tabellen gespeicherten Daten zugreifen – genau so, wie Sie es auch mit DAO oder ADODB erledigt haben.

Drittens hat Microsoft endlich ein Pendant zu den Triggern aus dem SQL Server oder anderen Datenbanksystemen geschaffen: die sogenannten Datenmakros. Und viertens sind Makros das Mittel der Wahl, wenn es um die Programmierung von Webdatenbanken geht – den neuen Datenbanktyp, den Sie mittels SharePoint ins Internet portieren können

und den wir in Kapitel 10, »Webdatenbanken«, genauer unter die Lupe nehmen. Daher wird an manchen Stellen von Client-Datenbanken (also herkömmlichen Datenbanken) und Webdatenbanken die Rede sein.

Begriffsklärung

Dieses Kapitel verwendet die folgenden Begriffe:

» *Makro*: Ein herkömmliches Makro, wie Sie es von früheren Access-Versionen her kennen – zwar mit neuem Makro-Editor und mehr Aktionen, aber immer noch ein Makro.

» *Datenmakro*: Ein Makro, das einer Tabelle hinzugefügt und durch ein bestimmtes Ereignis dieser Tabelle (etwa *Vor Löschung* oder *Nach Aktualisierung*) ausgelöst wird. Diese Makros werden ausschließlich durch diese Ereignisse gestartet.

» *Benanntes Makro*: Ein Makro, das zwar im Kontext einer Tabelle erstellt werden muss, aber sowohl von Makros als auch von Datenmakros aus aufgerufen werden kann. Seine Besonderheit ist die Möglichkeit, Parameter zu empfangen und auszuwerten.

Die letzten beiden Makroarten wurden mit Access 2010 neu eingeführt.

In diesem Kapitel lernen Sie den neu gestalteten Makro-Editor kennen und finden eine Übersicht der Makro-Befehle und -Strukturen. Außerdem werfen wir einen umfassenden Blick auf die neuen Datenmakros. Wir gehen nach wie vor davon aus, dass Entwickler von Client-Datenbanken VBA statt Makros verwenden. Daher beschreiben wir nicht alle Techniken, die Sie mit Makros in Client-Datenbanken durchführen können. Stattdessen finden Sie viele Beispiele für den Praxiseinsatz von Makros im Kapitel »Webdatenbanken« ab Seite 637.

BEISPIELDATENBANKEN

Den Download mit dem Beispielen finden Sie unter *www.acciu.de/aeb2010*. Die Datenbank zu diesem Kapitel heißt *Makros.accdb*.

Makros – ein Muss für Webdatenbanken

Wenn Sie eine Access-Datenbank als Webdatenbank veröffentlichen und diese in irgendeiner Weise programmieren möchten, sind Makros das Mittel der Wahl. Sie können Makros als eine abgespeckte VBA-Variante betrachten, die zwar wesentlich weniger Möglichkeiten bietet, dafür aber die Kompatibilität mit einer Webdatenbank sicherstellt.

Die folgenden Abschnitte zeigen zunächst, wie Sie Makros von den verschiedenen Stellen aus aufrufen. Anschließend lernen Sie die Möglichkeiten von Makros kennen

und erfahren, wie Sie gängige Vorgehensweisen, die Sie sonst mit VBA erledigen, mit Makros programmieren.

Makros in Client-Datenbanken

Sie können natürlich auch Client-Datenbanken mit Makros versehen: Sie können diese sowohl als unabhängige Makros speichern als auch für die Ereignisse von Formularen, Berichten und Steuerelementen hinterlegen.

Außerdem können Sie auch für benutzerdefinierte Ribbon-Steuerelemente Makros anstelle von Callback-Funktionen angeben.

9.1 Makros aufrufen

Als Nächstes erfahren Sie, wie Sie Makros, benannte Makros und Datenmakros aufrufen beziehungsweise wodurch diese ausgelöst werden.

Herkömmliche Makros aufrufen

Legen Sie für die folgenden Beschreibungen ein einfaches Beispielmakro an, das lediglich eine Meldung ausgibt. Betätigen Sie den Ribbon-Eintrag *Erstellen|Makros und Code|Makro*. Es erscheint der leere Makro-Editor.

Wählen Sie im Kombinationsfeld mit dem Inhalt *Neue Aktion auswählen* den Eintrag *Meldungsfeld* aus, tragen Sie für die beiden Parameter *Meldung* und *Titel* die Werte *Dies ist eine Testmeldung* und *Test* ein und speichern Sie es unter *macTest* (siehe Abbildung 9.1).

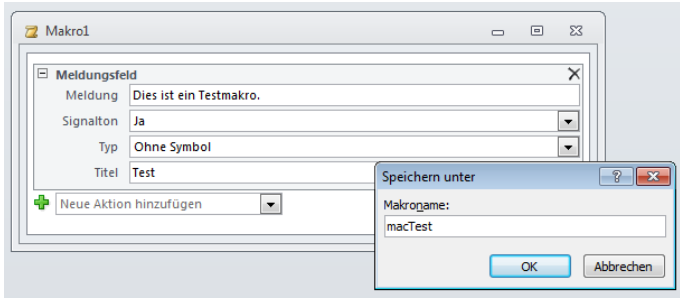


Abbildung 9.1: Anlegen eines einfachen Testmakros

Sie haben nun die folgenden Möglichkeiten, das Makro aufzurufen (die offensichtlichen, also per Doppelklick auf den Eintrag im Navigationsfenster oder über den Ribbon-Eintrag *Entwurf|Tools|Ausführen* lassen wir außen vor):

» Per VBA (etwa aus dem Direktfenster oder aus einem Modul heraus) mit der *DoCmd*-Methode *RunMacro*. Mit dem Parameter *RepeatCount* können Sie festlegen, wie oft das Makro aufgerufen werden soll, mit *RepeatExpression* legen Sie einen Ausdruck fest, der zum Abbruch führt, wenn er wahr ist:

```
DoCmd.RunMacro "macTest"
```

» Von einem anderen herkömmlichen Makro aus mit der Aktion *AusführenMakro* (siehe Abbildung 9.2). Diese Aktion entspricht der obigen VBA-Methode *DoCmd.RunMacro*.

» Von einem Ribbon aus. Wenn Sie einer Callback-Funktion eines Steuerelements (bevorzugt *OnAction*) den Namen eines Makros zuweisen, wird dieses beim Betätigen des Steuerelements ausgeführt (Beispiel siehe Formular *frmTestRibbonUndMakro*).

» Von Kontextmenüs aus. Geben Sie beim Erstellen des Kontextmenüs (unter Access 2010 vermutlich per VBA) den Namen des Makros für die Eigenschaft *OnAction* an.

» Durch Ereignisse von Formularen, Berichten und die darin enthaltenen Steuerelemente. Geben Sie einfach den Namen des Makros für die entsprechende Ereigniseigenschaft an (siehe Abbildung 9.3).

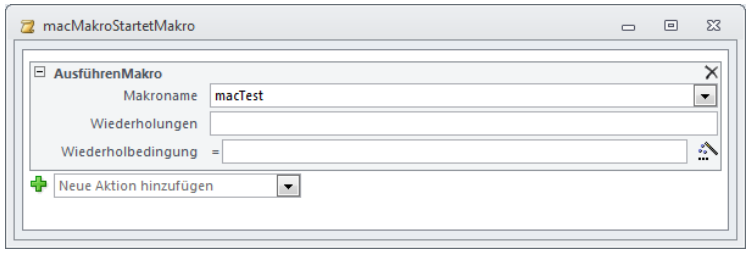


Abbildung 9.2: Aufrufen eines Makros von einem anderen Makro aus

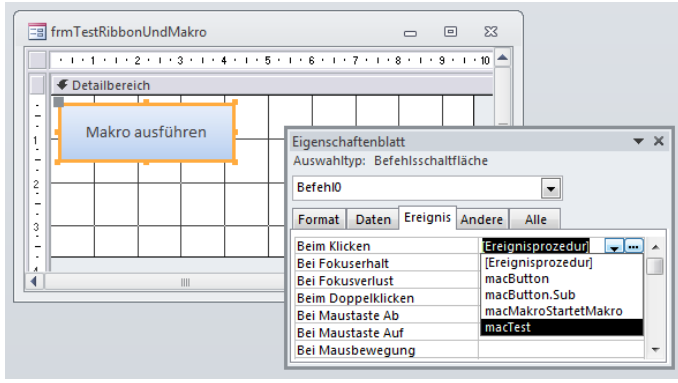


Abbildung 9.3: Auswahl von Makros, die durch eine Ereignisprozedur ausgelöst werden sollen

Datenmakros aufrufen

Da Datenmakros etwas andere Aktionen verwenden als herkömmliche Makros (unter anderem kennen sie kein Meldungsfenster) und es etwas aufwendiger ist, auf die Schnelle ein Datenmakro zu erstellen, das sich nach dem Aufrufen bemerkbar macht, hier nur kurz die Möglichkeiten zum Aufrufen eines Datenmakros:

Datenmakros, die Sie für ein Ereignis einer Tabelle erstellt haben, werden ausschließlich durch das entsprechende Ereignis ausgelöst. Benannte Datenmakros erstellen Sie zwar im Kontext einer Tabelle, Sie können diese jedoch auch von anderen Stellen aus aufrufen:

» Per VBA über die Methode *RunDataMacro* des *DoCmd*-Objekts

» Mit der Makroaktion *AusführenDatenmakro von* herkömmlichen Makros aus

» Mit der gleichen Makroaktion auch von anderen Datenmakros aus

9.2 Die Makro-Entwicklungsumgebung

Alle Makros werden in der gleichen Entwicklungsumgebung angelegt und bearbeitet, und zwar im sogenannten Makro-Editor. Sie rufen diesen aber von verschiedenen Stellen aus auf:

» Ein neues Makro erstellen Sie über den Ribbon-Eintrag *Erstellen|Makros und Code|Makro*.

» Ein Datenmakro erstellen Sie in der Datenblattansicht von Tabellen über den Ribbon-Eintrag *Tabelle|Vorabereignisse|<Makroname>* beziehungsweise *Tabelle|Nachfolgeereignisse|<Makroname>* oder in der Entwurfsansicht über den Ribbon-Eintrag *Entwurf|Feld-, Datensatz- und Tabellenereignisse|Datenmakros erstellen|<Makroname>*.

» Ein benanntes Makro erstellen Sie in der Datenblattansicht von Tabellen mit dem Ribbon-Eintrag *Tabelle|Benannte Makros|Benanntes Makro|Benanntes Makro erstellen* und in der Entwurfsansicht mit dem Ribbon-Eintrag *Entwurf|Feld-, Datensatz- und Tabellenereignisse|Datenmakros erstellen|Benanntes Makro erstellen*.

» Ein Makro, das durch ein Ereignis etwa in einem Formular einer Webdatenbank ausgelöst werden soll, erstellen Sie über das Auswählen der Ereignisprozedur und einen Klick auf die dann erscheinende Schaltfläche mit den drei Punkten.

Sie sehen bereits, dass Sie Datenmakros von der Entwurfs- und der Datenblattansicht von Tabellen aus erstellen können. In der Praxis hat es sich bewährt, dies gleich von der Datenblattansicht aus zu erledigen, weil man dann die neuen/geänderten Datenmakros auf der Stelle testen kann.

9.2.1 Das Makro-Ribbon

Die Makro-Entwicklungsumgebung besteht aus einem Ribbon-Tab mit einigen Befehlen zum Debuggen und zum Steuern der Ansicht der Makros und des Aktionskatalogs, aus dem Makro-Fenster selbst sowie einem Aktionskatalog, über den man bequem alle verfügbaren Aktionen in das Makro-Fenster ziehen kann.

Das Ribbon-Tab *Entwurf* erlaubt das Starten des aktuellen Makros, das schrittweise Ausführen sowie das Konvertieren des Makros in ein VBA-Modul (andersherum funktioniert dies allerdings nicht).

Die zweite Gruppe dieses Tabs liefert einige Möglichkeiten zum Ein- und Ausblenden der verschiedenen Ebenen des aktuellen Makros und seiner Aktionen.

Schließlich blenden Sie mit den beiden Schaltflächen der letzten Gruppe den Aktionskatalog ein und aus und legen fest, ob der Aktionskatalog und die Kombinationsfelder zur Auswahl von Makros im Makro-Editor alle oder nur die »sicheren« Makroaktionen anzeigen sollen (siehe Abbildung 9.4).

Bei Anzeige der unsicheren Makroaktionen werden diese durch ein gelbes Warnsymbol gekennzeichnet.

Jede weitere Erläuterung zum Thema Sicherheit kann man sich in diesem Zusammenhang eigentlich sparen: Sie als Entwickler werden Ihre eigenen Anwendungen immer bei geringster Sicherheitsstufe programmieren, und auch die Benutzer werden wohl kaum ständig Meldungen wegklicken wollen, die bei zu empfindlichen Sicherheitseinstellungen auftauchen.

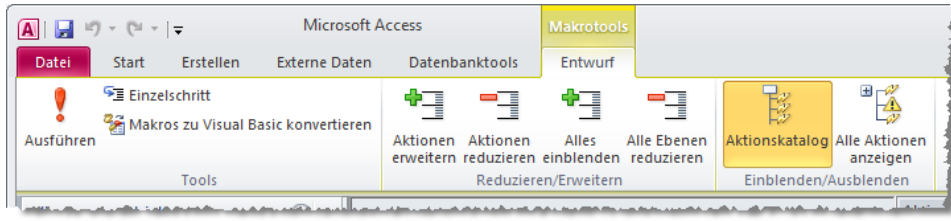


Abbildung 9.4: Das Ribbon zur Steuerung des Makro-Editors

9.2.2 Der Makro-Editor

Das zweite interessante Element der Programmierumgebung für Makros ist der Makro-Editor. Er ist im Prinzip das Pendant zum Codefenster des VBA-Editors.

Hier legen Sie die einzelnen Makro-Aktionen untereinander und teilweise verschachtelt in der Reihenfolge an, in der Access sie abarbeiten soll.

Wenn Sie mit dem Kombinationsfeld eine Makro-Aktion hinzugefügt haben, erscheint woanders ein neues Kombinationsfeld zum Anlegen einer neuen Aktion. Manchmal gibt es mehrere gleichzeitig, weil es verschiedene Stellen zum Anlegen neuer Makro-Aktionen gibt (siehe Abbildung 9.5).

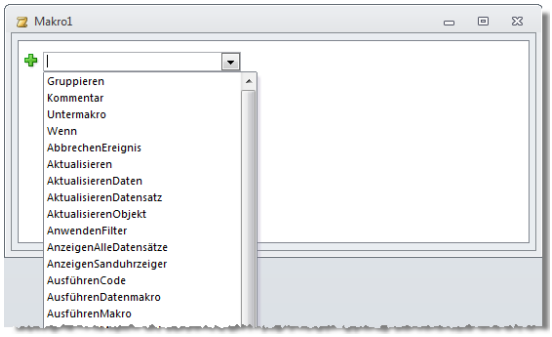


Abbildung 9.5: Der Makro-Editor mit der Auswahl aller verfügbaren Aktionen

Wem das Auswählen der neuen Aktionen mit dem Kombinationsfeld nicht gefällt, der kann die Aktionen aus dem Aktionskatalog direkt an die gewünschte Stelle im Makro-Editor ziehen. Beim Überfahren der vorhandenen Elemente im Makro-Editor unterlegt dieser potenzielle Abwurfstellen farbig.

Der Aktionskatalog kategorisiert die verschiedenen Makroaktionen grob und führt zusätzlich alle Strukturen für den Programmablauf sowie eine Liste aller bereits vorhandenen Makros auf (siehe Abbildung 9.6).

Einen kleinen Vorteil hat das Kombinationsfeld – zumindest beim Erstellen von Datenmakros und benannten Makros: Dort sind je nach Kontext nicht immer alle Aktionen erlaubt. Das Kombinationsfeld zeigt immer nur die zulässigen Aktionen an.

Makros eingeben

Grundsätzlich sollten Sie sich ein paar Minuten mit dem Makro-Editor beschäftigen – Sie werden sich dort schnell einleben. Dennoch hier ein paar Tipps zur schnelleren Eingabe und Bearbeitung:

» Wenn Sie Makroaktionen über das Kombintionsfeld anlegen, können Sie schnell die ersten Buchstaben des gewünschten Makros eintippen und dann mit der *Tab*-Taste direkt zur Eingabe der Parameter übergehen.

» Zum Ändern der Reihenfolge von Makroaktionen verwenden Sie die *Nach oben*- und *Nach unten*-Pfeile im Makro-Editor. Wenn Sie jedoch gern mit der Tastatur arbeiten, können Sie die aktuelle Makroaktion beziehungsweise den aktuell markierten Block auch mit der Tastenkombination *Strg + Nach oben* und *Strg + Nach unten* verschieben.

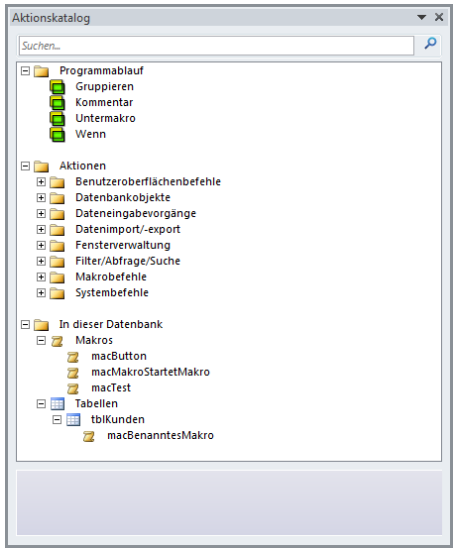


Abbildung 9.6: Der Aktionskatalog mit sortierten Aktionen und den erreichbaren Makros der aktuellen Datenbank

9.2.3 Kopieren, ausschneiden und einfügen

Wenn Sie im VBA-Editor Code an eine andere Stelle im gleichen Modul oder in ein anderes Modul verschieben möchten, markieren Sie den Code, schneiden ihn aus, positionieren die Einfügemarke am Zielort und kopieren den Code dorthin. Dies gelingt auch im Makro-Editor: Mit den Tastenkombinationen *Strg + C* beziehungsweise *Strg + X* kopieren Sie den aktuell grau hinterlegten Bereich des Makros oder schneiden diesen aus. Wenn Sie dann einfach im aktuellen oder einem anderen Makro die Tastenkombination *Strg + V* betätigen, wird der Inhalt der Zwischenablage an die Stelle des aktuell aktiven Kombinationsfeldes *Neue Aktion hinzufügen* kopiert (wenn das Makro Strukturen wie etwa Untermakros enthält, können durchaus mehrere solcher Kombinationsfelder sichtbar sein). Sie können auch ein vorhandenes Element im Makro markieren und dann den Inhalt der Zwischenablage einfügen. Dieser landet dann unmittelbar hinter dem markierten Element.

9.2.4 Makros debuggen

Mit Access 2010 gibt es erstmalig die Möglichkeit, Makros zu debuggen. Dazu öffnen Sie das betroffene Makro in der Entwurfsansicht, aktivieren im Ribbon den Eintrag *Macrotools|Tools|Einzelschritt* und klicken dann auf den Befehl *Ausführen* in der gleichen Ribbon-Gruppe.

Anschließend zeigt Access für jeden Schritt den Dialog aus Abbildung 9.7 mit allen notwendigen Informationen an.

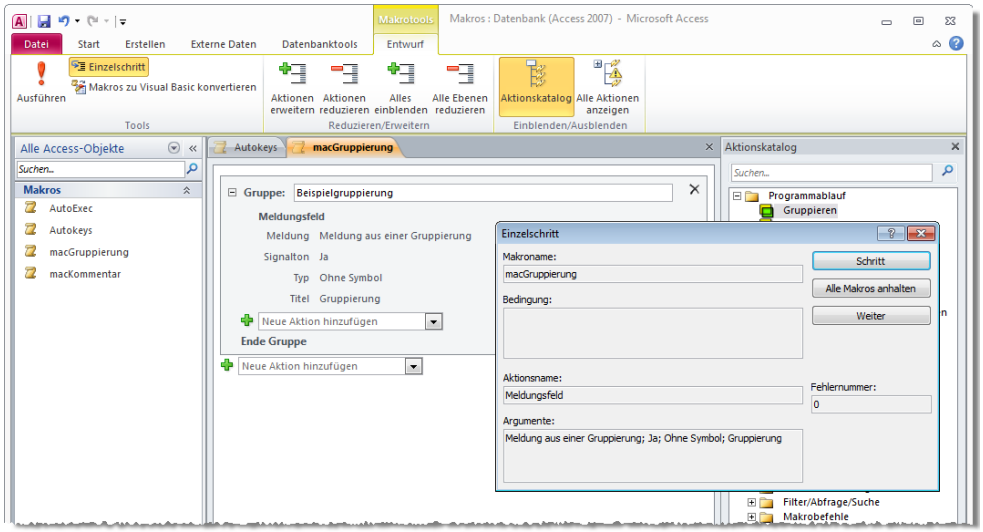


Abbildung 9.7: Debuggen im Makro-Editor

9.3 Makros in Client-Datenbanken

Es gibt zwei Gründe, Makros in Client-Formularen anzulegen: Entweder, Sie wollen damit eine Aktion oder eine Prozedur beim Öffnen der Datenbank ausführen oder Sie legen damit anwendungsweit gültige Tastenkombinationen fest.

Im ersten Fall erstellen Sie ein Makro, das die gewünschten Makrobefehle ausführt und speichern es unter dem Namen *Autoexec*. Dieses Makro wird beim Start von Access automatisch ausgeführt (siehe Abbildung 9.8).

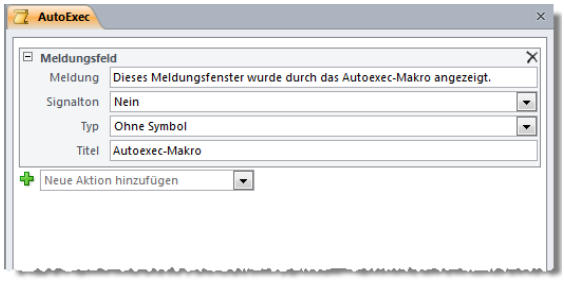


Abbildung 9.8: Beispiel für ein Autoexec-Makro

Im zweiten Fall legen Sie ein Makro mit einer speziellen Notationsweise an und speichern es unter dem Namen *Autokeys*. Diese Notationsweise besteht darin, dass Sie für jede Tastenkombination ein eigenes Untermakro anlegen und diesem statt einem Namen eine Formel für die zu verwendende Tastenkombination zuweisen. Innerhalb des Untermakros fügen Sie dann die eigentlichen Befehle ein (siehe Abbildung 9.9).

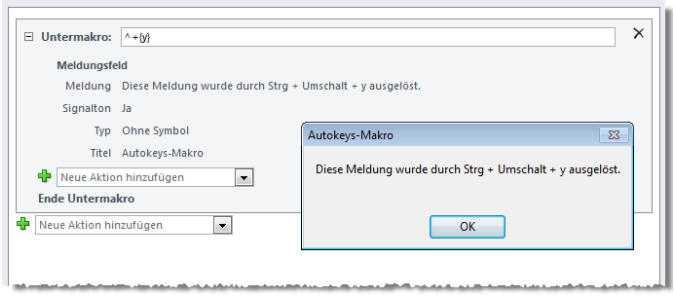


Abbildung 9.9: Ein Tastaturmakro

Im Beispiel lautet der Name der Makrogruppe *^+{y}*, was zunächst kryptisch aussieht, aber schnell erklärt ist: ^ steht für die *Strg*-Taste, + für die *Umschalt*-Taste und der Ausdruck in geschweiften Klammern ist der zu betätigende Buchstabe. Es gibt noch ein paar Spezialausdrücke: Funktionstasten (*{F1}*...), Einfügen (*{Insert}*), Löschen (*{Delete}*) et cetera. Eine genaue Übersicht liefert die Onlinehilfe.

9.4 Programmablauf- und Strukturbefehle

Unter die Programmablaufbefehle fallen solche, die entweder den Ablauf des Makros beeinflussen oder die Makrobefehle zusammenfassen. Aufgeführt wird hier auch der Kommentar-Befehl. Er gehört zwar eigentlich nicht in diese Gruppe, aber offenbar wollte Microsoft für diesen Befehl keine eigene Gruppe eröffnen.

9.4.1 Kommentar

Der *Kommentar*-Befehl ist eigentlich gar kein Befehl, sondern erlaubt schlicht das Einfügen eines Kommentars. Tragen Sie einen Text ein, wird dieser später wie in Abbildung 9.10 angezeigt.

9.4.2 Gruppieren

Mit der *Gruppieren*-Struktur können Sie einen oder mehrere Makrobefehle gruppieren. Das hat zum einen den Zweck, zusammenhängende Befehle zu benennen, und erlaubt

es andererseits, die gruppierten Befehle nur noch als eine einzige Zeile im Makro-Editor anzuzeigen. Dies gelingt übrigens auch nachträglich, indem Sie alle zu gruppierenden Makroaktionen markieren und dann aus dem Kontextmenü den Eintrag *Gruppenblock erstellen* auswählen. Sie können auch verschachtelte Gruppierungen anlegen.

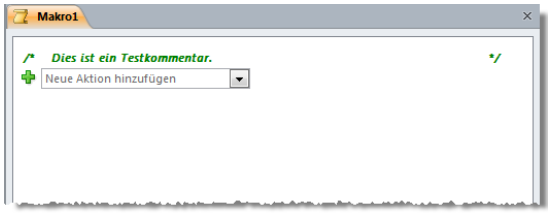


Abbildung 9.10: Ein Kommentar im Makro

9.4.3 Untermakro

Mit Untermakros erhalten Sie ein ähnliches Resultat wie mit einzelnen Prozeduren in einem VBA-Modul. Ein Untermakro kann genau wie jedes Makro einen oder mehrere Befehle enthalten, die beim Aufrufen des Untermakros ausgelöst werden. Die Analogie zwischen Modul und Makro bezüglich Prozeduren und Untermakros hat jedoch Grenzen: Ein Makro kann nämlich, wie in den bisherigen Beispielen, auch Anweisungen außerhalb von Untermakros enthalten. Diese führt Access aus, wenn Sie einfach nur das Makro aufrufen. Das Makro aus Abbildung 9.11 enthält eine normale *Meldungsfeld*-Anweisung und eine in einem Untermakro befindliche. Wenn Sie dieses Makro nun einfach ausführen, wird nur das erste Meldungsfeld ausgeführt, das zweite nicht.

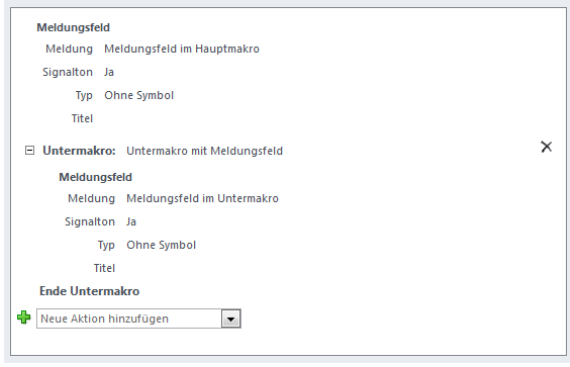


Abbildung 9.11: Ein reguläres Meldungsfeld und eines in einem Untermakro (*macUntermakros*)

Damit dies geschieht, müssen Sie dieses gezielt mit der Anweisung *AusführenMakro* aufrufen. Dazu legen Sie diese Anweisung an und legen dann das Untermakro fest. Dies

geschieht in der Syntax *<Makro>.<Untermakro>*, wobei Access freundlicherweise alle Makronamen samt Untermakros in einem Kombinationsfeld anbietet.

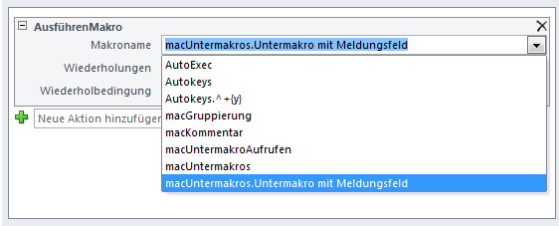


Abbildung 9.12: Aufruf eines Untermakros

Untermakros können Sie zum Beispiel nutzen, indem Sie für jedes Formular einer Webdatenbank ein eigenes Makro anlegen und Sie darin alle Untermakros speichern, die durch die Ereignisse der Webdatenbank aufgerufen werden. Dies ist eine Alternative zum Speichern eines jeden Ereignisses eines Formulars oder der enthaltenen Steuerelemente in einem eigenen, mit dem Formular gespeicherten Makro.

Sie können dann über das Kombinationsfeld der entsprechenden Ereigniseigenschaft auf alle in der Datenbank enthaltenen Makros und Untermakros zugreifen.

Bestehende Makroaktionen können Sie auch nachträglich zu einem Untermakro zusammenfassen. Dazu markieren Sie die betroffenen Aktionen und wählen dann aus dem Kontextmenü den Befehl *Untermakroblock erstellen* aus.

9.4.4 Wenn

Der *Wenn*-Befehl entspricht dem *If* unter VBA. Im einfachsten Fall gibt es nur einen *Wenn*-Zweig. Angenommen, ein Formular namens *frmZahl* mit einem Textfeld *txtZahl* wäre geöffnet. Dann würde das Makro aus Abbildung 9.13 das Meldungsfenster mit dem Text *Geben Sie eine Zahl ein.* anzeigen. Den in der *Wenn*-Bedingung enthaltenen Ausdruck können Sie mit dem Ausdrucks-Generator erstellen.

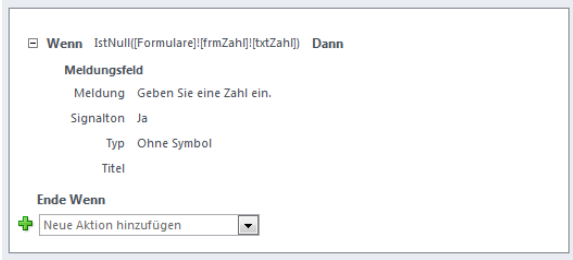


Abbildung 9.13: Eine einfache *If...Then*-Bedingung

Sie können auch eine *If...Then...Else*-Bedingung nachbilden. Dies erledigen Sie, indem Sie im Entwurf des Makros auf den Link *Sonst hinzufügen* klicken. Im nun erscheinenden neuen Bereich geben Sie die Anweisungen an, die Access ausführen soll, wenn die erste Bedingung nicht eintrifft.

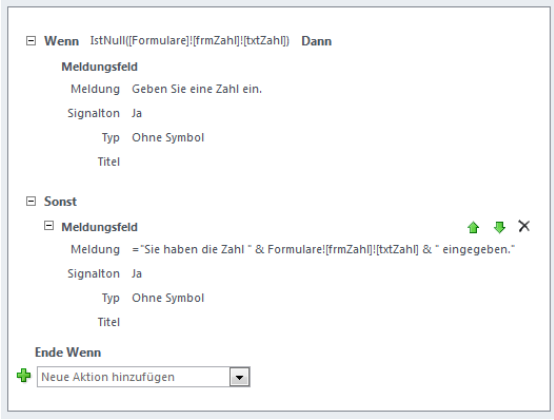


Abbildung 9.14: Eine *If...Then...Else*-Bedingung in einem Makro

Schließlich fügen Sie mit dem Link *Sonst Wenn hinzufügen* einen oder mehrere weitere Bedingungen hinzu.

Und genau wie bei Gruppierungen und Untermakros liefert das Kontextmenü einen Eintrag zum Einfassen einer oder mehrerer markierter Makroaktionen in eine Wenn-Bedingung.

9.4.5 StoppMakro und StoppAlleMakros

Die beiden Makrobefehle *StoppMakro* und *StoppAlleMakros* brechen das aktuelle Makro beziehungsweise alle aktuell laufenden Makros ab.

9.5 Weitere Makrobefehle

Die übrigen Makro-Anweisungen sind die eigentlichen Befehle. Sie werden in den folgenden Abschnitten vorgestellt.

9.5.1 Meldungsfeld

Die *Meldungsfeld*-Anweisung entspricht einem abgespeckten *MsgBox*-Befehl. Sie erwartet die Parameter *Meldung*, *Signalton*, *Typ* und *Titel*.

Wenn Sie Literale mit Variablen mischen wollen (etwa temporäre oder lokale Variablen, wie sie gleich im Anschluss vorgestellt werden), tragen Sie ein führendes Gleichheitszeichen als Meldungstext ein und umschließen die Literale mit Anführungszeichen.

Außerdem verbinden Sie die einzelnen Elemente mit dem Kaufmanns-Und (&), zum Beispiel:

```
="Der Wert der Beispielvariablen lautet: " & LokaleVar!Beispielvariable
```

Zum Hinzufügen von Zeilenumbrüchen im Meldungstext verwenden Sie das @-Zeichen. Sie können maximal zwei Zeilenumbrüche einbauen. Sollten Sie drei @-Zeichen angeben, wird der Text hinter dem dritten Auftreten dieses Zeichens abgeschnitten.

Beachten Sie, dass der *Meldungsfeld*-Befehl in Webdatenbanken nur den Parameter *Meldung* zulässt. Außerdem darf der Meldungstext inklusive Variablen nur 256 Zeichen lang sein – dies gilt für Web- und Client-Datenbanken.

9.5.2 “

In Client- und Webdatenbanken können Sie Werte in temporären Variablen zwischenspeichern, die anwendungsweit zur Verfügung stehen. Dafür verwenden Sie die folgenden Befehle:

» *FestlegenTempVar*: Legt eine temporäre Variable fest und erwartet den Namen der Variablen und den zu speichernden Ausdruck als Parameter.

» *EntfernenAlleTempVar*: Leert alle temporären Variablen.

» *EntfernenTempVar*: Leert eine temporäre Variable, deren Name mit dem einzigen Parameter angegeben wird.

Bleibt noch die Frage, wie Sie auf eine temporäre Variable zugreifen. Dies geschieht durch einen Zugriff auf die *TempVar*-Auflistung mit folgender Syntax: *TempVar!<Variablenname>*. Dabei ist *<Variablenname>* in eckige Klammern einzufassen, wenn es Leer- und Sonderzeichen enthält.

Über eine temporäre Variable können Sie übrigens Ergebnisse von VBA-Funktionen abrufen. Legen Sie in einem Standardmodul diese Funktion an:

```
Public Function TempVarFuellen() As String
    TempVarFuellen = CurrentProject.Path
End Function
```

Das Ergebnis dieser Funktion können Sie dann in eine temporäre Variable einlesen, indem Sie als Ausdruck den Namen der Funktion angeben (siehe Abbildung 9.15).

Die anschließende Meldungsfeld-Aktion gibt dann den Funktionswert aus.

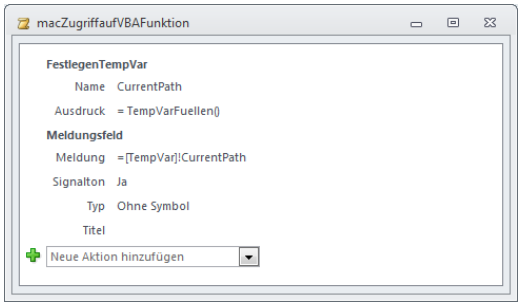


Abbildung 9.15: Ausführen einer VBA-Funktion per Makro

Auf temporäre Variablen können Sie übrigens auch von VBA aus zugreifen – mehr dazu erfahren Sie unter »TempVars« ab Seite 448.

9.5.3 Lokale Variablen

Neben den global verfügbaren Variablen können Sie auch Variablen verwenden, die nur im jeweiligen Makro bekannt sind – die sogenannten lokalen Variablen. Hier gibt es nur einen Befehl:

» *FestlegenLokaleVar*: Legt eine lokale Variable fest und erwartet den Namen der Variablen und den zu speichernden Ausdruck als Parameter.

Das Makro aus Abbildung 9.16 zeigt ein Beispiel für das Zuweisen und Ausgeben einer lokalen Variablen. Der Zugriff auf lokale Variablen erfolgt analog zu den temporären Variablen mit einem Ausdruck wie *[LocaleVar]![<Variablenname>]*.

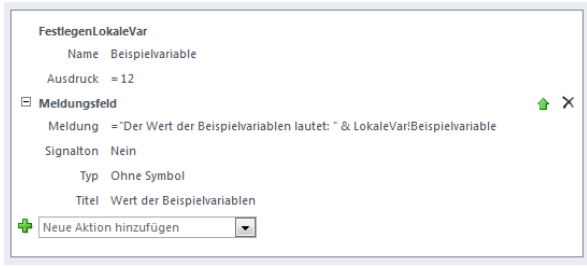


Abbildung 9.16: Speichern und ausgeben einer lokalen Variablen

9.5.4 Makros zum Arbeiten mit Daten

Die folgenden Makroaktionen dienen der Bearbeitung von Daten, zum Beispiel zum Löschen, Aktualisieren oder Speichern.

Datensatz löschen

Das Makro *DatensatzLöschen* löscht den aktuellen Datensatz. Es erwartet keine Parameter.

Wenn sich die Schaltfläche zum Auslösen dieser Makroaktion nicht im gleichen Formular befindet, das auch die zu löschenden Daten anzeigt, müssen Sie zuvor noch den Fokus auf das entsprechende Formular verschieben. Ein Beispiel finden Sie unter »Datensatz löschen« ab Seite 663.

Aktuellen Datensatz speichern

Die Makroaktion *DatensatzSpeichern* schreibt den aktuellen Datensatz in die zugrunde liegende Tabelle.

Angezeigte Daten aktualisieren

Die Makroaktion *AktualisierenDaten* entspricht der *Requery*-Anweisung von Formularen und Steuerelementen. Ohne Angabe von Parametern aktualisiert diese Aktion die Datenherkunft des Formulars, von dem aus die Aktion aufgerufen wird.

Sollte etwa ein Unterformular durch einen Klick auf eine Schaltfläche im Hauptformular aktualisiert werden, geben Sie für den Parameter *Steuerelementname* den Namen des Unterformularsteuerelements an, in dem sich das Unterformular befindet.

AktualisierenDatensatz

Diese Aktion aktualisiert nur die aktuell in der Datensatzgruppe enthaltenen Daten.

Filter für Formulare, Berichte und Steuerelemente

Mit der Makroaktion *FestlegenFilter* definieren Sie für den Parameter *Bedingung* eine *Where*-Bedingung, mit der Sie das im zweiten Parameter *Steuerelementname* angegebene Steuerelement filtern.

Sortierung für Formulare, Berichte und Steuerelemente

Die Makroaktion *FestlegenSortiertNach* dient dem Festlegen einer Sortierreihenfolge für das angegebene Steuerelement.

9.5.5 Fehlerbehandlung

Die Fehlerbehandlung wird in Makros durch die *BeiFehler*-Anweisung abgehandelt. Diese Anweisung gibt an, wohin im Falle eines Fehlers verzweigt werden soll. Dabei gibt es die folgenden Möglichkeiten:

» *Nächster*: Führt einfach die folgende Anweisung aus (entspricht *On Error Resume Next*).

» *Makroname*: Gibt den Namen eines Makros an, das im Falle eines Fehlers aufgerufen werden soll.

» *Fehlgeschlagen*: Beendet das Makro und zeigt den Access-eigenen Fehlerdialog an.

Im Beispiel aus Abbildung 9.17 soll beim Auftreten eines Fehlers das Untermakro *Fehlerbehandlung* aufgerufen werden. Dieses zeigt eine benutzerdefinierte Fehlermeldung an. Im Meldungstext können Sie die Eigenschaften des Objekts *[MacroError]* verwenden, um genaue Fehlerinformationen auszugeben. Die einzelnen Eigenschaften dieses Objekts lauten:

» *ActionName*: Name des Makrobefehls, der den Fehler auslöste

» *Arguments*: Argumente des Makrobefehls

» *Condition*: Bedingung beim Aufruf des Makrobefehls

» *Description*: Fehlermeldung (entspricht *Err.Description*)

» *MacroName*: Name des Makros, das den Fehler auslöste

» *Number*: Fehlernummer (entspricht *Err.Number*)

Nach dem Auftreten eines Fehlers löschen Sie die Fehler-Informationen mit dem Makro-Befehl *LöschenMakroFehler*. Wenn Sie in der Fehlermeldung alle Informationen unterbringen möchten, wird der Platz etwas eng:

```
="Nr: " & [MacroError].[Number] & "Beschreibung: " & [MacroError].[Description] & "
 Makroname: " & [MacroError].[MacroName] & " Befehl: " & [MacroError].[ActionName] &
"Argumente: " & [MacroError].[Arguments] & " Bedingung: " & [MacroError].[Condition]
```

Vor allem aber lassen sich, wenn Sie Variablen in die Meldung einfließen lassen und somit einen gemischten Text aus Literalen und Variablen verwenden wollen, keine Zeilenumbrüche mit dem &-Zeichen integrieren.

9.5.6 Formulare und Berichte öffnen und schließen

Das Öffnen von Formularen und Berichten erfolgt mit den beiden Makrobefehlen *ÖffnenFormular* und *ÖffnenBericht*.

Formular öffnen

Die *ÖffnenFormular*-Anweisung beschränkt sich in Webdatenbanken auf vier Parameter:

» *Formularname*: Name des Formulars

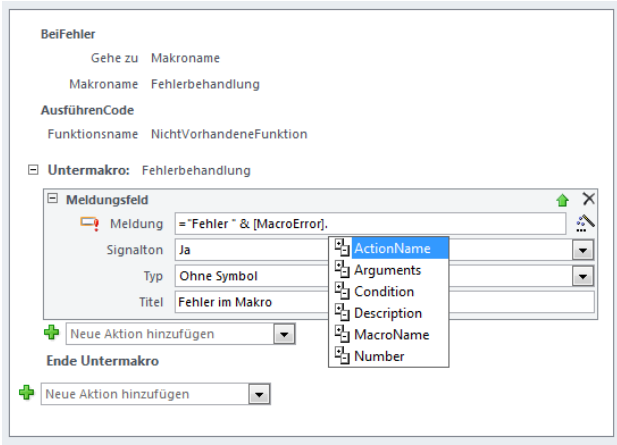


Abbildung 9.17: Erstellen einer Fehlerbehandlung innerhalb eines Makros

» *Bedingung*: Filterkriterium wie etwa *KundeID = 1*

» *Datenmodus*: Einer der Werte *Hinzufügen*, *Bearbeiten* oder *Nur lesen*

» *Fenstermodus*: Unsinniger Parameter für Webdatenbanken, da hier nur der Wert *Dialog* möglich ist

Bericht öffnen

Der Befehl *ÖffnenBericht* erwartet die folgenden drei Parameter:

» *Berichtsname*: Name des zu öffnenden Berichts

» *Bedingung*: Filterkriterium wie etwa *KundeID = 1*

» *Fenstermodus*: Einer der Werte *Normal* oder *Dialog*

Fenster schließen

Die Makroaktion *FensterSchließen* schließt das aktuelle Fenster. In Client-Datenbanken erwartet diese Aktion die von der *DoCmd.Close*-Methode her bekannten Parameter, in Webdatenbanken hingegen gar keine. Dafür ist in Webdatenbanken Folgendes zu beachten:

» Wenn Sie etwa einer *OK*-Schaltfläche in einem als Dialog geöffneten Formular die Aktion FensterSchließen zuweisen, wird dieses Formular geschlossen.

» In normal geöffneten Formularen hingegen wird beim Ausführen von *FensterSchließen* nach Rückfrage das aktuelle Register im Browser beziehungsweise das ganze Browserfenster geschlossen.

9.5.7 In Formularen arbeiten

Die folgenden Aktionen dienen dem Steuern von Formularen und insbesondere dem Eingeben von Daten in die Steuerelemente von Formularen. Außerdem ist es wichtig, die Syntax für das Abfragen der in den Steuerelementen enthaltenen Werte kennenzulernen.

Durch die Datensätze navigieren

Die Aktion *GeheZuDatensatz* erwartet für den Parameter *Datensatz* einen der folgenden Werte:

» *Erster*

» *Letzter*

» *Vorheriger*

» *Nächster*

» *Neuer*

Sie können diese Aktion beispielsweise zum Bestücken einer Schaltfläche verwenden, die einen neuen Datensatz anzeigen soll. Wichtig ist, dass Sie zuvor den Fokus auf das Formular mit den Datensätzen verschieben. Dazu verwenden Sie die gleich vorgestellte Aktion *GeheZuSteuerelement*.

Fokus auf Steuerelement setzen

Die Aktion *GeheZuSteuerelement* verschiebt den Fokus auf das im Parameter *Steuerelementname* angegebene Steuerelement. Damit dies funktioniert, muss das Formular, in dem sich das Steuerelement befindet, bereits den Fokus besitzen. Sollten Sie diese Aktion also etwa vom Hauptformular aus aufrufen und das betroffene Steuerelement befindet sich ebenfalls im Hauptformular, reicht der Aufruf von *GeheZuSteuerelement* unter Angabe des Steuerelementnamens.

Eigenschaften einstellen

Steuerelemente haben eine Reihe von Eigenschaften, darunter auch den Wert. Vorrangig werden Sie die Makroaktion *FestlegenEigenschaft* wahrscheinlich dazu verwenden, Feldern Werte zuzuweisen. Aber auch die übrigen Eigenschaften wie *Aktiviert*, *Sichtbar* et cetera können Sie damit einstellen. Die Methode erwartet die folgenden Parameter:

» *Steuerelementname*: Name des Steuerelements

» *Eigenschaft*: Name der Eigenschaft (zum Beispiel *Wert*)

» *Wert*: Einzustellender Wert

Zu einem anderen Formular wechseln

Mit der *WechselnZu*-Aktion können Sie auf verschiedene Arten ein neues Formular anzeigen. Dazu verwenden Sie die folgenden Parameter:

» *Objekttyp*: Soll ein Formular oder ein Bericht angezeigt werden?

» *Objektname*: Wie heißt das anzuzeigende Objekt?

» *Pfad zu Unterformular-Steuerelement*: Wo soll das Objekt angezeigt werden? Mehr dazu unter »Navigieren in Webanwendungen« ab Seite 666.

» *Bedingung*: Kriterium für das Filtern der Datenherkunft des Formulars/Berichts

» *Seite*: Gibt die Seite innerhalb eines Endlosformular an.

» *Datenmodus*: *Hinzufügen*, *Bearbeiten* oder *Nur lesen*

Datenmakro ausführen

Die Makroaktion *AusführenDatenmakro* wird unter »Benanntes Makro aufrufen« ab Seite 621 ausführlich beschrieben.

AusführenMakro

Wie bereits weiter oben unter »Untermakro« ab Seite 601 erwähnt, können Sie mit dieser Anweisung Makros oder gegebenenfalls in einem Makro enthaltene Untermakros aufrufen. Dabei geben Sie den Namen des Makros und, falls nötig, des durch einen Punkt getrennten Untermakros als ersten Parameter an. Der zweite Parameter erwartet eine konkrete Anzahl der Wiederholung dieses Aufrufs, der dritte gibt eine Abbruchbedingung an.

Damit lassen sich zwei verschiedene Schleifen abbilden – eine *For...Next*-Schleife und die *Do...While*-Schleife.

Die *For...Next*-Schleife erreichen Sie durch die Angabe eines entsprechenden Wertes für die Anzahl der Wiederholungen. Im Beispiel aus Abbildung 9.18 wird das Untermakro *Zaehlerschleife* nach dem Voreinstellen der Zählervariablen *Zaehlervariable* auf den Wert *0* mit fünf Wiederholungen aufgerufen. Den Beweis, dass dies funktioniert hat, liefert die abschließende *Meldungsfeld*-Anweisung: Diese gibt den Wert der im Untermakro hochgezählten Zählervariablen aus.

Die *Do...While*-Schleife soll hingegen so lange laufen, bis eine Abbruchbedingung erreicht ist. Ein Beispiel zeigt das Makro aus Abbildung 9.19. Dieses Makro ruft das Untermakro *Zaehlerschleife* so lange auf, bis die temporäre Variable *Zaehlervariable* den Wert *10* erreicht hat. Danach gibt ein Meldungsfenster den aktuellen Wert der temporären Variablen aus.

10

Webdatenbanken

Webdatenbanken sind Datenbanken, die über den Internetbrowser bedient werden können. Sie sind der neue Versuch von Microsoft, Access-Entwicklern und -Benutzern eine Möglichkeit zu bieten, Datenbanken samt Benutzeroberfläche vom Desktop in das Internet zu befördern.

Hintergrund dieser Technik ist SharePoint 2010, das die Tabellen der Datenbank in SharePoint-Listen überführt und Webseiten zur Bearbeitung der enthaltenen Daten anbietet, die es auf Basis der von Ihnen erstellten Formulare erzeugt. Dazu verwendet SharePoint 2010 die neuen *Access Services*, die es ermöglichen, eine Access-Datenbank komplett ohne Installation einer Access-Vollversion oder Runtime einzusetzen. Genau genommen arbeiten Sie dabei auch gar nicht mehr mit Access, sondern mit gängigen

Webtechniken, die ihre Daten aus den genannten SharePoint-Listen beziehen. Access dient nur noch als Entwicklungstool für die später im Internetbrowser angezeigte Anwendung.

Damit gehen natürlich eine Reihe von Einschränkungen einher. Die erste und wichtigste ist: Sie können kein VBA verwenden, sondern nur Makros. Die zweite ist: Nach aktuellem Stand benötigen Sie für jeden einzelnen Benutzer, der auf die Webdatenbank zugreifen möchte, eine eigene Lizenz.

Und der Zugriff auf eine Datenbank setzt zwingend die Anmeldung an den entsprechenden SharePoint-Server voraus – Sie können also nicht mal eben einen Shop mit Access erstellen und diesen dann über das Internet der Allgemeinheit zur Verfügung stellen.

Beispielserver

Wenn Sie selbst mit SharePoint 2010 und Webdatenbanken experimentieren möchten, haben Sie grundsätzlich zwei Möglichkeiten:

» Sie setzen selbst einen entsprechenden Server auf (entweder lokal oder auch gleich auf einem entsprechenden Webserver). Ohne MSDN-Abo oder vergleichbarem Deal müssen Sie für Windows 2008 64-bit Server, SharePoint 2010 Enterprise und entsprechenden Benutzerlizenzen ca. 6.000,- EUR einplanen.

» Sie nutzen das Angebot eines Internetdienstleisters, der das Hosting von Access-Datenbanken im Web als maßgeschneidertes Paket offeriert. Hier gibt es nur einen Anbieter, der seinen Dienst allerdings schon mit Erscheinen der Beta-Version von Access 2010 aufgenommen hat und mittlerweile über entsprechende Erfahrungen verfügt. Die monatlichen Preise liegen je nach Anforderungen zwischen 19,- und 99,- $ und es gibt die Möglichkeit, den Dienst für 30 Tage kostenlos zu testen. Sie gelangen über die Adresse *www.acciu.de/accesshosting* zu diesem Anbieter. Sollten im Laufe der Zeit weitere Anbieter hinzukommen, werden wir sie unter der genannten Adresse aufführen.

Wie Sie später sehen werden, wird für die Beispiele in diesem Kapitel ein Dienstleister in Anspruch genommen.

Begriffsklärung

Zur Unterscheidung zwischen herkömmlichen Datenbanken und den in diesem Kapitel vorgestellten Webdatenbanken sprechen wir nachfolgend von *Client-Datenbanken* und *Webdatenbanken*.

In Webdatenbanken gibt es wiederum zwei Arten von Objekten: Web-Objekte und Client-Objekte. Web-Objekte unterliegen einigen Einschränkungen, können dafür aber zum SharePoint-Server verschoben und dort für die Anwendung im Webbrowser verwendet

werden. Client-Objekte sind herkömmliche Access-Objekte, die allerdings nicht auf den SharePoint-Server verschoben und dementsprechend nur für administrative Aufgaben verwendet werden können. Der Funktionsumfang der Web-Objekte ist im Gegensatz zu den Client-Objekten meist stark eingeschränkt – dazu später mehr.

BEISPIELDATENBANKEN

Den Download mit dem Beispielen finden Sie unter *www.acciu.de/aeb2010*. Die Datenbank zu diesem Kapitel heißt *Webdatenbank.accdb*.

10.1 Anlegen einer Webdatenbank

Eine Webdatenbank können Sie auf drei Arten anlegen:

» als komplett neue, leere Datenbank,

» auf Basis einer Vorlage oder

» auf Basis einer bestehenden Client-Datenbank.

10.1.1 Leere Webdatenbank erstellen

Zum Anlegen einer leeren, neuen Webdatenbank wählen Sie in einer frisch gestarteten Access-Instanz zunächst die Vorlage *Leere Webdatenbank* aus und klicken dann nach dem Festlegen des Speicherorts auf die Schaltfläche *Erstellen* (siehe Abbildung 10.1).

10.1.2 Webdatenbank auf Basis einer Vorlage erstellen

Um beim Erstellen der Webdatenbank eine Vorlage zu verwenden, klicken Sie zunächst auf *Beispielvorlagen*. Es erscheint eine neue Seite, die einige Vorlagen anzeigt. Leider sind die Vorlagen für Desktop- und Webdatenbanken bunt gemischt, sodass Sie diese nur am Zusatz *Webdatenbank* erkennen.

Klicken Sie hier beispielsweise auf den Eintrag *Kontakte-Webdatenbank*, erstellt Access flugs eine Anwendung auf Basis der gewählten Vorlage. Diese erscheint gleich mit dem jeweiligen Startformular, etwa wie in Abbildung 10.2.

Ob Sie diese Datenbankvorlagen als Basis für eine eigene Anwendung verwenden können, hängt von Ihren Anforderungen ab. Als Anschauungsbeispiele für einige Techniken dienen sie aber auf jeden Fall.

Wir machen jedoch zunächst einen Schritt zurück und wenden uns im übernächsten Abschnitt einer komplett leeren Webdatenbank zu.

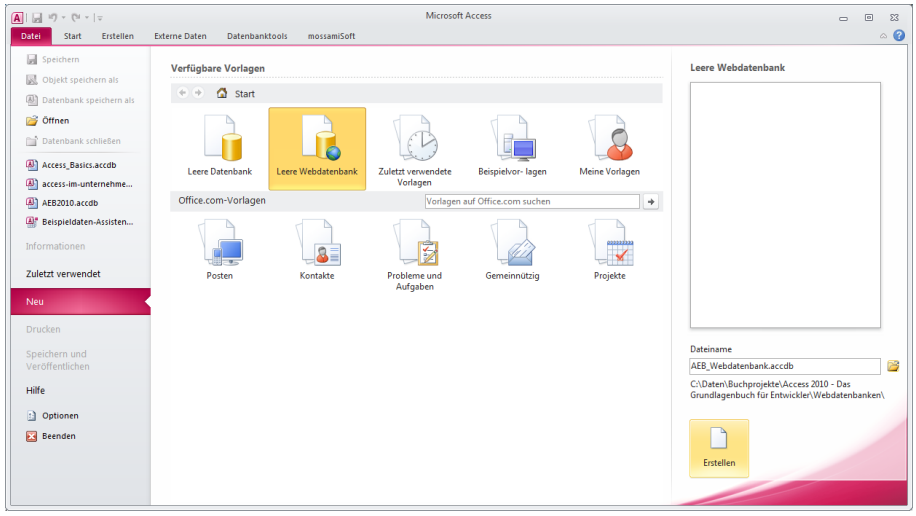


Abbildung 10.1: Anlegen einer leeren Webdatenbank

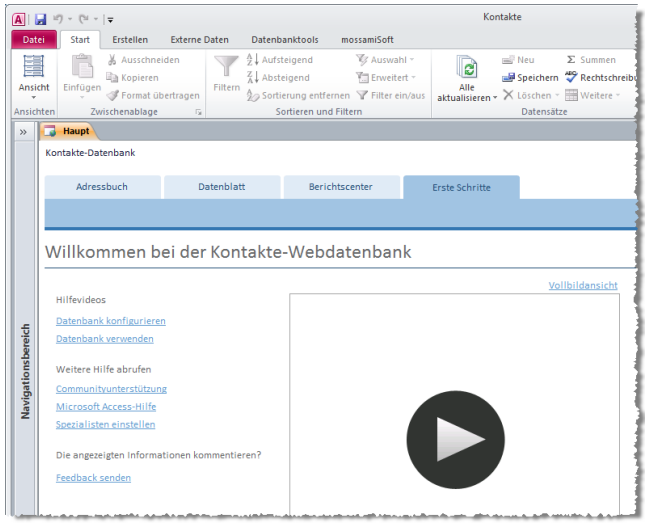


Abbildung 10.2: Webdatenbank auf Basis einer Vorlage

10.1.3 Erstellen einer Webdatenbank aus einer Client-Datenbank

Hier öffnen Sie einfach die betroffene Client-Datenbank und wählen dann im Backstage-Bereich den Eintrag *Speichern und Veröffentlichen|In Access-Services veröffentlichen* aus.

Dies zieht zunächst eine Kompatibilitätsprüfung der in der Datenbank enthaltenen Objekte nach sich.

10.1.4 Unterschiede der Benutzeroberfläche

Das Erste, was beim Anlegen einer Webdatenbank auffällt, ist die fehlende Entwurfsansicht beim Erstellen von Tabellen (siehe Abbildung 10.3). Somit ist gewährleistet, dass das Erstellen von Tabellen anfängerkompatibel ist und sich niemand in der komplizierten Entwurfsansicht verirrt. Die Benutzung ist intuitiv und soll hier nicht weiter erläutert werden. Neu ist hier, dass jede Tabelle auf jeden Fall ein Primärschlüsselfeld enthalten muss. Sie können dieses zwar umbenennen, aber nicht löschen.

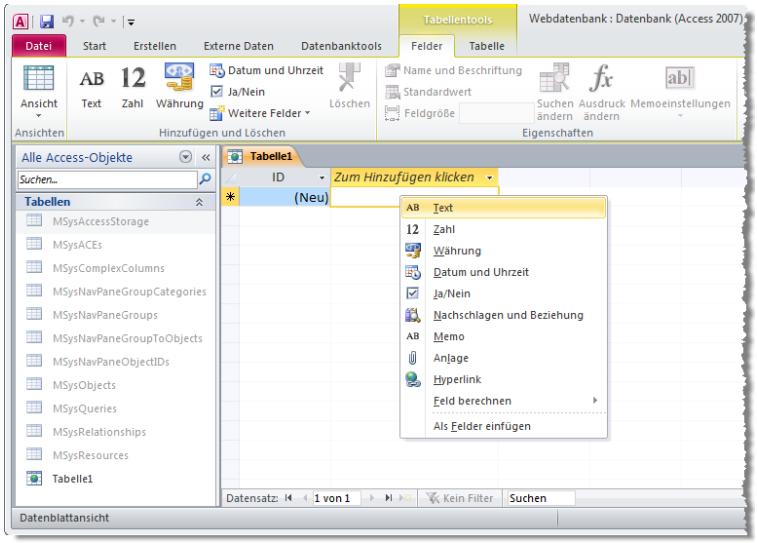


Abbildung 10.3: Anlegen einer neuen Tabelle einer Webdatenbank

Nach dem Anlegen einer ersten kleinen Tabelle mit einem einzigen Feld soll gleich der erste Test erfolgen: Die Datenbank soll ins Internet gestellt werden. Das ist allerdings gar nicht so einfach, denn dazu sind einige Voraussetzungen zu erfüllen. Sie brauchen nämlich einen Provider, der Ihnen einen Webserver mit den nötigen Komponenten bereitstellt. Wir haben zum Zeitpunkt der Erstellung dieses Kapitels nur einen Anbieter gefunden, auf den wir über *www.acciu.de/accesshosting* verlinken (sollten weitere hinzukommen, weisen wir unter dieser Adresse darauf hin). Bei diesem Anbieter (*www.accesshosting.com*) gab es zum Zeitpunkt der Drucklegung dieses Buchs 30-tägige Testzugänge, in denen Sie Ihre Datenbank dort hochladen und über das Internet darauf zugreifen konnten. 30 Tage sind eine kurze Zeit, Sie sollten diesen Zugang also möglichst so legen, dass Sie ihn auch voll ausnutzen können.

Wenn Sie sich für einen dauerhaften Zugang entscheiden, fallen monatliche Kosten an, die von der Menge des verwendeten Speicherplatzes sowie von der Anzahl der Benutzer abhängen.

Womit wir gleich beim nächsten Thema angelangt sind: Standardmäßig können Benutzer nur mit entsprechenden Zugangsdaten auf eine Webdatenbank zugreifen. Eine öffentliche Darbietung des Inhalts von Tabellen ist somit nicht vorgesehen. Wir werden später schauen, ob sich dies gegebenenfalls umgehen lässt.

10.2 Schnellstart: Beispieldatenbank veröffentlichen

Die kleine Beispieldatenbank soll zunächst nur eine Tabelle, ein Formular und einen Bericht enthalten. Die Tabelle *tblKunden* enthält nur das Feld *Firma*.

10.2.1 Formular anlegen

Beim Erstellen eines Formulars in einer Webdatenbank finden Sie eine etwas andere Auswahl von Werkzeugen im Ribbon (siehe Abbildung 10.4). Die mit einem Web-Symbol versehenen Elemente dienen dazu, Objekte für die Anzeige im Internet-Browser zu erstellen. Unter *Clientformulare* finden Sie die für herkömmliche Datenbanken verfügbaren Formularvorlagen.

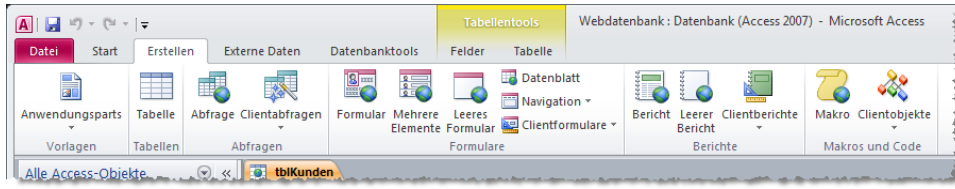


Abbildung 10.4: Das Ribbon zum Anlegen von Formularen für Webdatenbanken

Markieren Sie einfach die Tabelle, für die Sie ein Formular anlegen möchten, und klicken Sie anschließend auf *Erstellen|Formulare|Formular*, um ein Formular in der Detailansicht zu erstellen (siehe Abbildung 10.5). Sie können nur direkt an Tabellen oder Abfragen gebundene Formulare erstellen, ungebundene Formulare gibt es in Webdatenbanken nicht (mit Ausnahme von Navigationsformularen). Webformulare werden immer auf Basis des aktuell markierten Objekts erstellt. Wenn Sie gerade ein Formular ausgewählt haben, erstellt Access ein neues Objekt auf Basis der Datenherkunft des markierten Formulars.

Webformulare haben keine Entwurfsansicht und können nur in der Layoutansicht geändert werden. Diese Ansicht wurde mit Access 2007 für Formulare und Berichte eingeführt, um die enthaltenen Daten gleich beim Entwurf anzuzeigen.

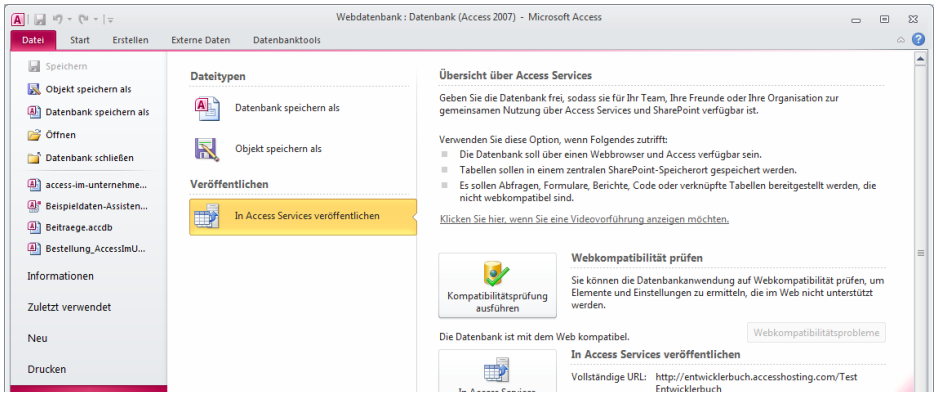


Abbildung 10.5: Ein erstes Formular auf Basis der Tabelle *tblKunden*

Wenn Sie Webformulare statt in der Registerkartenansicht als einzelne Fenster einblenden möchten, können Sie deren Größe nicht wie bei Clientformularen ändern – was sicher gewöhnungsbedürftig ist.

10.2.2 Bericht anlegen

Das Anlegen von Berichten geschieht noch etwas restriktiver als bei Formularen: Sie können hier nur die Datenherkunft festlegen, der Rest ist weitgehend vorbestimmt – zumindest, wenn der Bericht nur auf einer einzigen Tabelle basiert, was in unserem kleinen Beispielbericht (*rptKunden*) der Fall ist.

Natürlich können Sie den Bericht nach der erstmaligen Erstellung nach Belieben anpassen (siehe Abbildung 10.6).

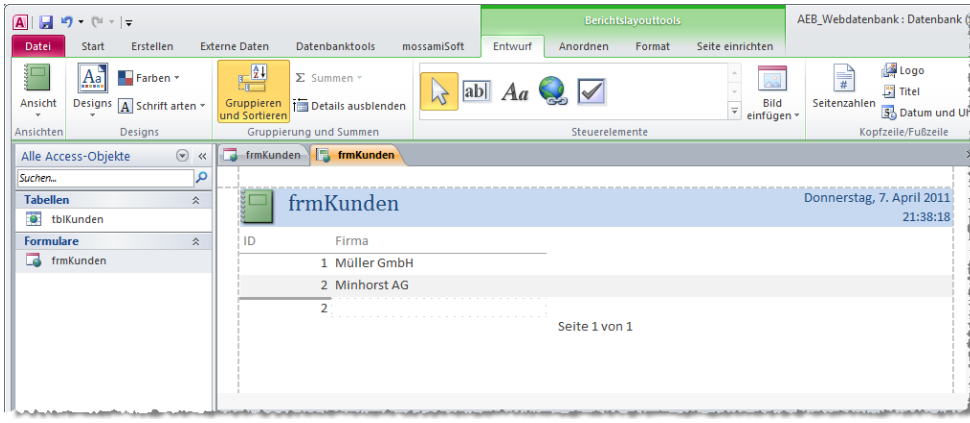


Abbildung 10.6: Web-Bericht in der Layoutansicht

10.2.3 Beispieldatenbank im Internet veröffentlichen

Nach dem Erstellen dieser drei Objekte veröffentlichen wir die Datenbank im Internet. Dazu zeigen Sie mit einem Klick auf den Ribbon-Tab *Datei* den Backstage-Bereich an und betätigen dann die Schaltfläche *In Access Services veröffentlichen* (siehe Abbildung 10.7).

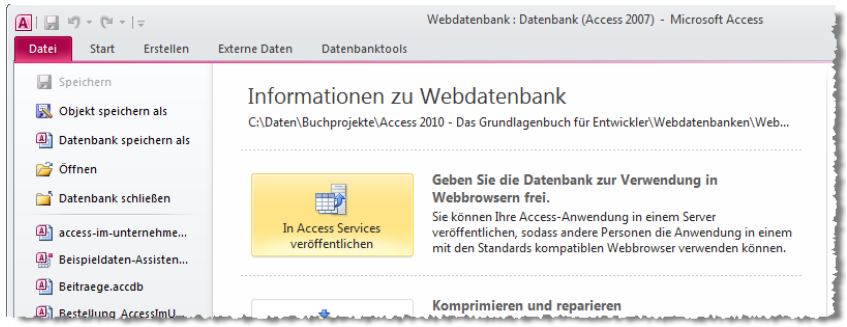


Abbildung 10.7: Eine Webdatenbank veröffentlichen

Es erscheint ein weiterer Bereich mit einer kurzen Beschreibung des Vorgangs. Mit der Schaltfläche *Kompatibilitätsprüfung ausführen* vergewissern Sie sich, dass die erstellten Objekte für die Verwendung in einer Webdatenbank geeignet sind. Anschließend tragen Sie die vom Provider übermittelten Daten in die beiden Felder *Server-URL* und *Websitename* ein (siehe Abbildung 10.8).

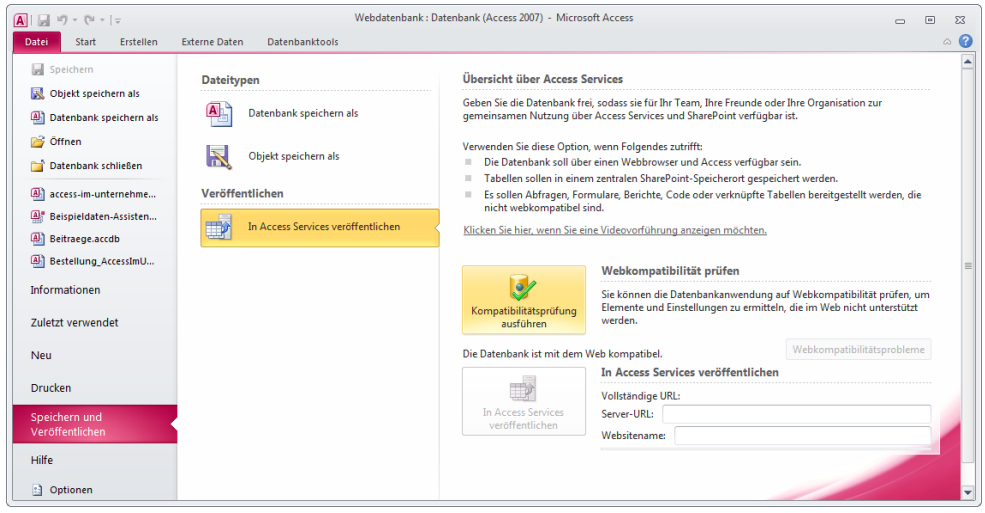


Abbildung 10.8: Veröffentlichen der Webdatenbank

Danach erscheint ein Dialog zur Eingabe der Benutzerkennung, gefolgt von einem Fortschrittsbalken, der über die aktuellen Schritte informiert. Schließlich erhalten Sie eine Meldung über die erfolgreiche Veröffentlichung der Webdatenbank.

Die Domain, unter der die Anwendung zu finden ist, besteht aus der soeben eingegebenen Server-URL und dem durch einen Schrägstrich getrennten Websitenamen.

Wenn Sie diese Adresse im Internet-Browser aufrufen, erscheint der bereits bekannte Anmeldedialog.

Nach Eingabe der Zugangsdaten stellen wir fest, dass noch etwas fehlt – wir haben kein Startformular ausgewählt (siehe Abbildung 10.9). Dafür finden wir jedoch eine Liste aller bereits angelegten Objekte vor.

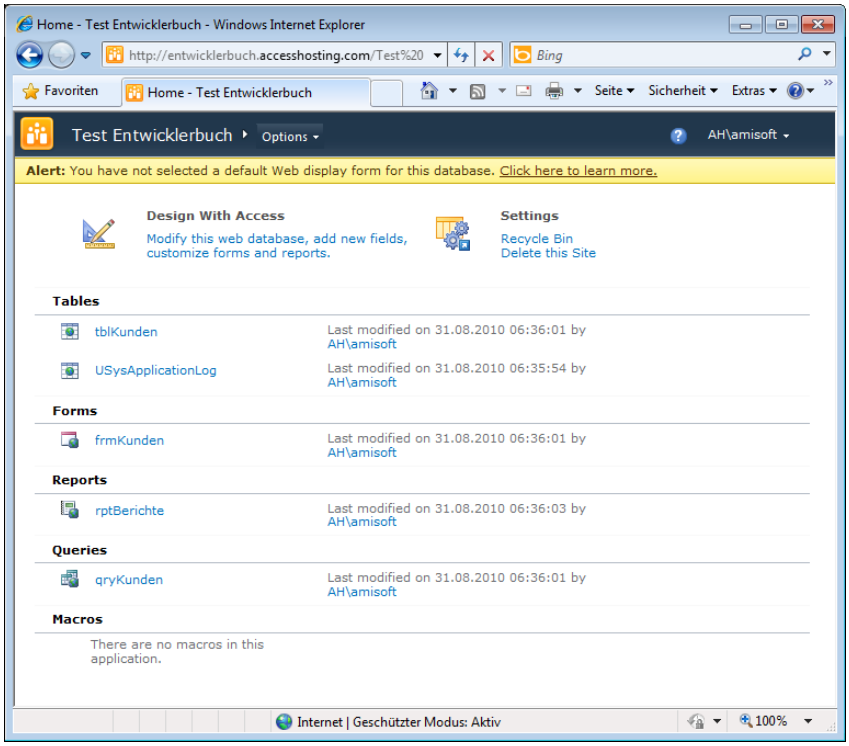


Abbildung 10.9: Der erste Blick auf unsere Webdatenbank

Auf der Internetseite können Sie jedoch alle Objekte anklicken und öffnen. So bleibt uns ein erster Blick auf das Webformular *frmKunden* nicht verwehrt (siehe Abbildung 10.10).

Mit diesem Formular können Sie durch die vorhandenen Datensätze blättern und diese bearbeiten, neue Datensätze anlegen und Datensätze löschen.

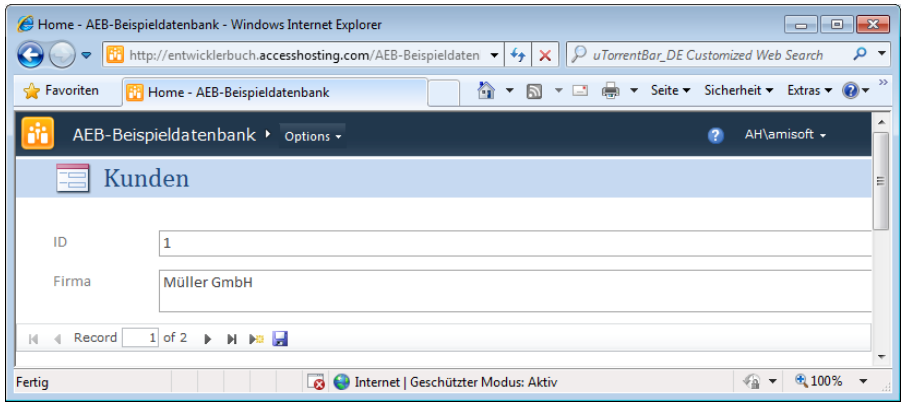


Abbildung 10.10: Das erste Webformular

10.2.4 Einmal im Web, immer im Web?

Wenn Sie die Datenbank einmal veröffentlicht haben, fragt Access bei jedem Öffnungsvorgang der Client-Datenbank nach den Zugangsdaten, weil es Änderungen an den Objekten gern direkt zum Server übertragen und daher mit diesem verbunden sein möchte.

Verständlicherweise kann Access aber nicht davon ausgehen, dass immer eine Online-Verbindung vorliegt, sodass das wiederholte Betätigen der *Abbrechen*-Schaltfläche dazu führt, dass Sie die Datenbank auch ohne Verbindung zum SharePoint-Server öffnen und die Objekte bearbeiten können – mit einer Ausnahme: Tabellen lassen sich ohne Verbindung zum SharePoint-Server nicht ändern.

Haben Sie sich am SharePoint-Server angemeldet und fügen etwa ein Feld zu einer Tabelle hinzu, werden diese Änderungen direkt zum SharePoint-Server übertragen. Auch Änderungen an den Daten werden direkt weitergeleitet.

Bei Bearbeitungen von Formularen und Berichten ist Access nicht so schnell bei der Sache: Erst, wenn Sie im Backstage-Bereich die Schaltfläche *Alles synchronisieren* angeklickt haben, überträgt Access auch Änderungen am Entwurf der übrigen Objekte zum Server.

Erweiterung der Tabellen

Durch das Speichern der Tabellen auf dem SharePoint-Server wurden auch an den nach wie vor in der Client-Datenbank vorhandenen Tabellen Änderungen vorgenommen.

Im Detail handelt es sich um einige neue Felder, die allerdings nur in der Feldliste und nicht in der Tabellenansicht erscheinen (siehe Abbildung 10.11).

Wichtig ist das Feld *_OldID*: Wenn eine Tabelle zum SharePoint-Server übertragen wurde, legt dieser ein neues Autowertfeld an und nummeriert die Datensätze hier von 1 an fortlaufend durch. Die alten Werte werden im Feld *_OldID* gespeichert. Keine Sorge: Eventuell vorhandene Fremdschlüsselfelder, die auf ein solches Autowertfeld verweisen, werden gleichzeitig geändert.

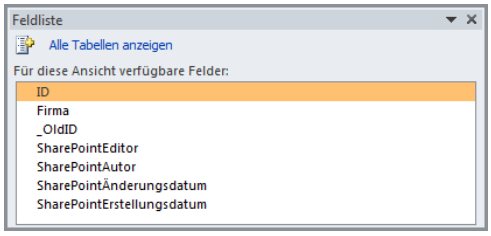


Abbildung 10.11: Tabellen, die nach SharePoint übertragen wurden, erhalten einige neue Felder.

10.2.5 Die SharePoint-Benutzeroberfläche

Das Veröffentlichen der Beispieldatenbank im Web führt dazu, dass wir uns die Benutzeroberfläche einer typischen Webdatenbank ansehen können. Das *Options*-Menü bietet etwa folgende Optionen an (siehe Abbildung 10.12):

» Öffnen der Webdatenbank mit Access auf dem lokalen Rechner. Damit laden Sie eine *.accdw*-Datenbank herunter und öffnen sie wie eine herkömmliche Datenbank. Nach dem Durchführen von Änderungen veröffentlichen Sie diese erneut.

» Einstellen der Berechtigungen (siehe Abbildung 10.13)

» Übersicht der Webdatenbank (liefert das gleiche Bild, als wenn Sie kein Startformular angegeben haben)

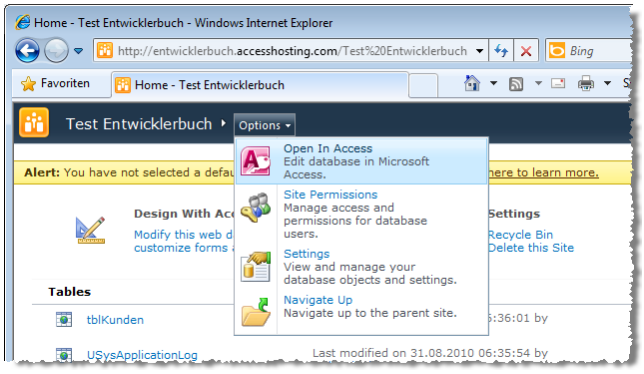


Abbildung 10.12: Optionen der Webdatenbank

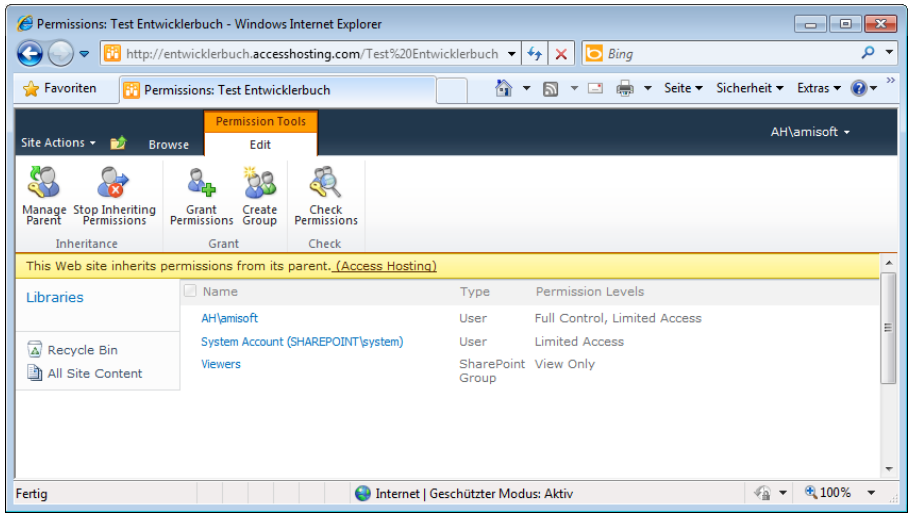


Abbildung 10.13: Einstellen der Berechtigungen für die Webdatenbank

Für die Beispiele dieses Kapitels reicht es aus, dass uns die Benutzeroberfläche von SharePoint über das Fehlen des Startformulars informiert hat. Nachdem wir dieses gleich nachgereicht haben, arbeiten wir direkt mit der Anwendung und nicht mehr mit der eigentlichen SharePoint-Oberfläche.

10.2.6 Startformular erstellen und festlegen

Das Startformular für die Webdatenbank legen Sie fast genau wie für eine Client-Datenbank über den entsprechenden Eintrag der Access-Optionen im Bereich *Aktuelle Datenbank* fest. Für Webdatenbanken gibt es hier jedoch einen weiteren Eintrag namens *Webanzeigeformular* – Sie können also sowohl für die Client-Anwendung als auch für die Webdatenbank ein eigenes Startformular angeben.

Zuvor wollen wir jedoch ein geeignetes Formular erstellen. Dieses ist nicht an eine Datenherkunft gebunden und soll lediglich Steuerelemente zur Navigation enthalten. Hier bietet sich das neue Navigationssteuerelement an. Innerhalb des Steuerelements können Sie dann später Unterformulare und -berichte einfügen. Im Ribbon finden Sie einige Templates für Navigationsformulare (siehe Abbildung 10.14).

Für unsere Zwecke reicht zunächst ein einfaches Navigationsformular mit horizontalen Registerkarten. Speichern Sie dieses unter dem Namen *frmStart* und legen Sie es wie oben beschrieben als Startformular der Anwendung fest (siehe Abbildung 10.15).

Damit sich das Synchronisieren mit dem SharePoint-Server lohnt, fügen Sie dem Navigationsformular gleich noch ein erstes Formular mit Daten hinzu, und zwar *frm-*

Kunden. Dazu öffnen Sie einfach das Formular *frmStart* in der Layoutansicht und ziehen dann den Eintrag *frmKunden* aus dem Navigationsbereich auf die Navigationsleiste von *frmStart*. Das Ergebnis sieht unter Access wie in Abbildung 10.16 aus.

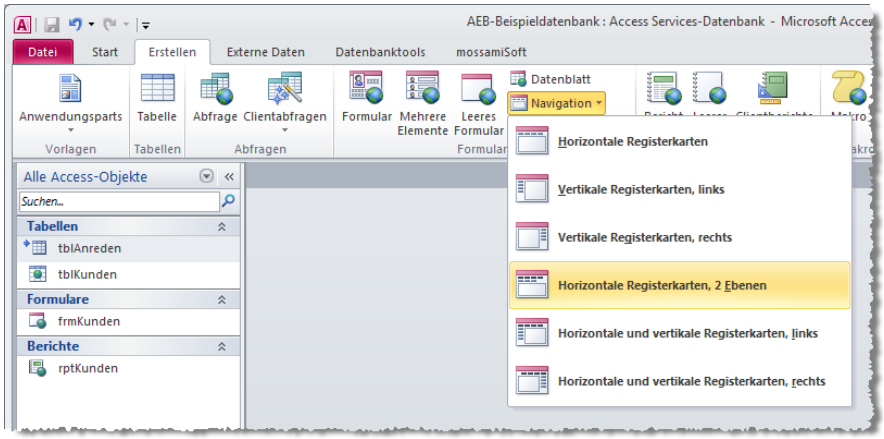


Abbildung 10.14: Auswahl eines Layouts für das Navigationsformular

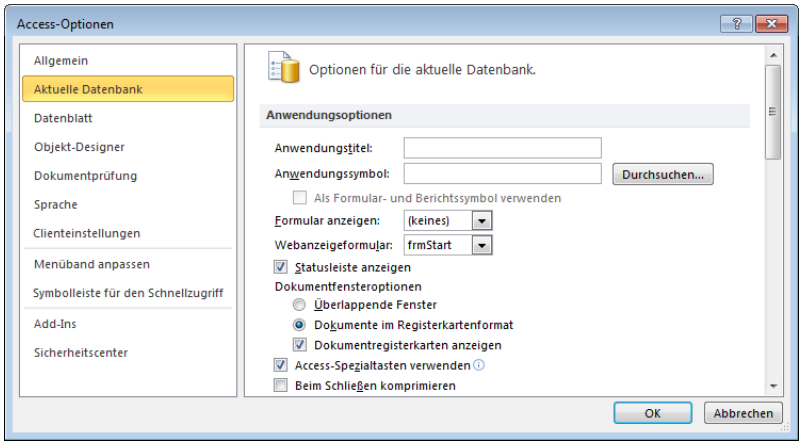


Abbildung 10.15: Festlegen des Startformulars der Webdatenbank

Beim nächsten Aufruf der Webdatenbank im Browser zeigt diese gleich das neue Startformular an (siehe Abbildung 10.17).

Wenn man wie in diesem Beispiel gleich zu einem Hosting-Dienstleister gegriffen hat und nicht erst selbst einen SharePoint-Server einrichten muss, fällt das Fazit positiv aus: Wie haben in wenigen Minuten ein erstes Access-Formular eins zu eins über das Internet verfügbar gemacht.

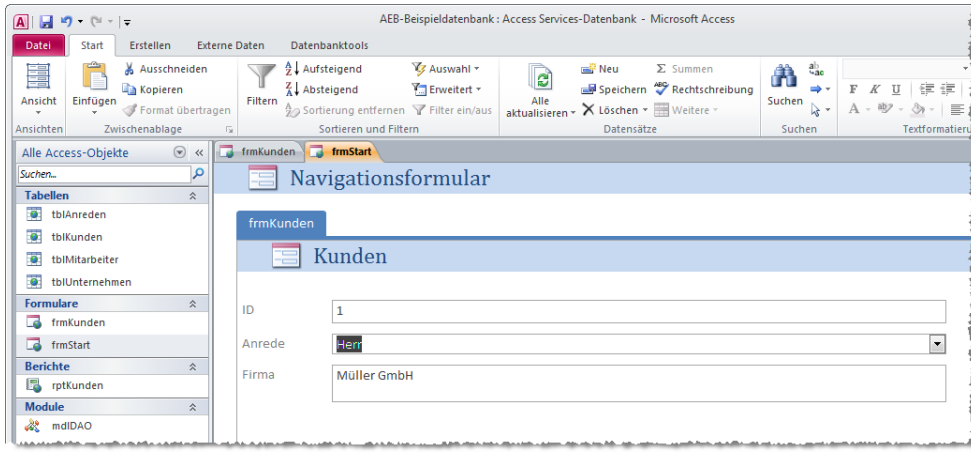


Abbildung 10.16: Navigationsformular mit Kunden-Unterformular

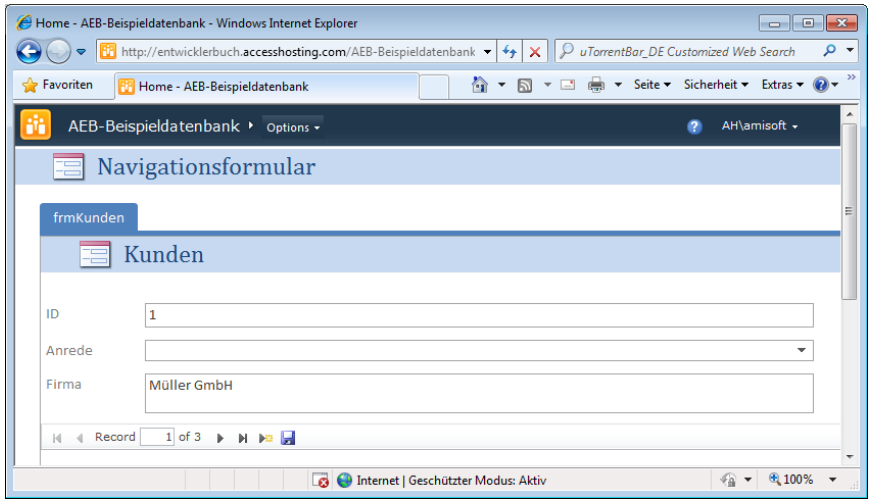


Abbildung 10.17: Das erste Formular im Internet Explorer

Das war der Schnellstart – in den folgenden Abschnitten erfahren Sie nun mehr über die Möglichkeiten von Webdatenbanken.

10.3 Tabellen in Webdatenbanken

Auch wenn Microsoft das Erstellen von Tabellen in Webdatenbanken gegenüber dem üblichen Vorgehen weiter vereinfacht hat, gibt es doch einige nützliche Informationen.

10.3.1 Tabellen erstellen und bearbeiten

Der Tabellenentwurf und die Datenblattansicht sind in Webdatenbanken eins. Das Einstellen von Eigenschaften der Tabelle und der Felder erfolgt über zwei Ribbon-Tabs. Das erste heißt *Tabellen* und enthält im Wesentlichen Schaltflächen zum Anlegen und Bearbeiten von Datenmakros (siehe »Einführung in Datenmakros« ab Seite 612). Einige weitere Eigenschaften stellen Sie über einen Dialog ein, der nach einem Klick auf die Schaltfläche *Tabelleneigenschaften* erscheint. Abbildung 10.18 zeigt das Ribbon-Tab *Tabelle*.

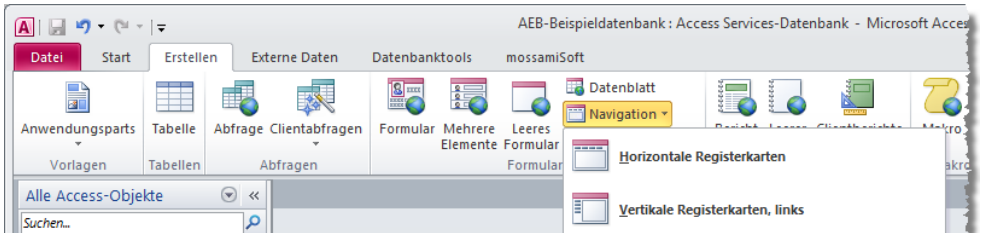


Abbildung 10.18: Das Ribbon-Tab zum Einstellen der Tabelleneigenschaften

Da die Entwurfsansicht in Webdatenbanken wegfällt, finden Sie alle zum Anlegen und Anpassen notwendigen Eigenschaften im Ribbon unter *Felder* (siehe Abbildung 10.19). Die Funktionen sind weitgehend selbsterklärend. Warum Microsoft allerdings ein *Splitbutton*-Steuerelement zum Auswählen einer einzigen Ansicht vorgesehen hat, bleibt offen.

Abbildung 10.19: Das Ribbon-Tab zum Anpassen der Felder und ihrer Eigenschaften

10.3.2 Beziehungen herstellen

Der Befehl *Datenbanktools|Beziehungen|Beziehungen* im Ribbon ist deaktiviert und es ist tatsächlich nicht möglich, Beziehungen über den entsprechenden Dialog anzulegen. Die Alternative ist der Nachschlage-Assistent, den Sie wie in Abbildung 10.20 aufrufen. Sie können dies auch über den Ribbon-Eintrag *Felder|Hinzufügen und Löschen|Weitere Felder|Nachschlagen und Beziehung* erledigen.

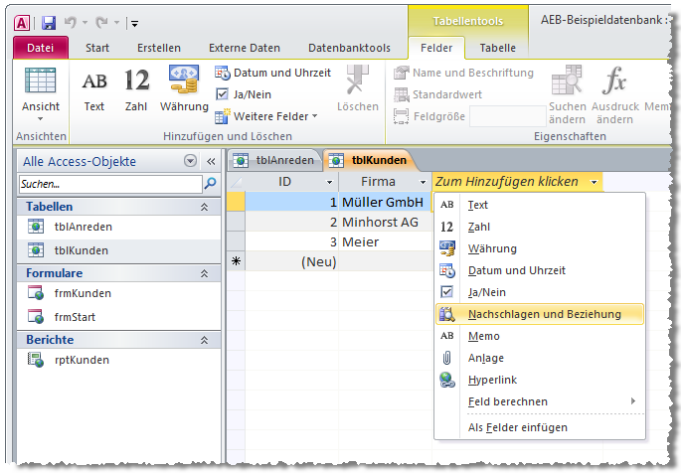


Abbildung 10.20: Aufrufen des Nachschlage-Assistenten

Die einzelnen Schritte des Assistenten entsprechen dem einer Client-Datenbank – mit Ausnahme des letzten Schrittes. Hier können Sie, ersatzweise für das fehlende Beziehungen-Fenster, referentielle Integrität und Löschweitergabe definieren (siehe Abbildung 10.21).

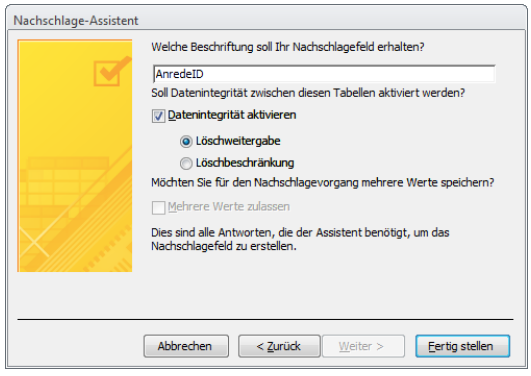


Abbildung 10.21: Festlegen refentieller Integrität

10.3.3 Besonderheiten bei Primärschlüsselfeldern in Webdatenbanken

Primärschlüsselfelder in Webdatenbanken haben immer den Datentyp *Zahl*. Beziehungen werden ausschließlich dann hergestellt, wenn eines der beteiligten Felder ein Primärschlüsselfeld einer Tabelle ist. Die Werte von Primärschlüsselfeldern lassen

11

Bilder und binäre Dateien

Probleme mit dem Speichern von Dateien und insbesondere von Bildern in Dateien gehören in den Newsgroups und Foren zum täglichen Leben. Access bietet mit dem Datentyp *OLE-Objekt* die Möglichkeit, auch Dateien in einer Tabelle zu speichern. Leider gab es bisher in Access keine eingebaute einfache Funktion, um Dateien ohne Weiteres in einem solchen Feld abzulegen – zumindest keine, die nicht früher oder später die Größe der Datenbankdatei explodieren ließ. Mit Access 2007 kam die Wende: Microsoft hat die Access-Gemeinde erhört und nicht nur einen neuen Felddatentyp namens *Anlage*, sondern auch noch ein neues Steuerelement zum Anzeigen von Bildern in Formularen und Berichten spendiert. Und dabei können Sie im Anlage-Feld sogar Dateien speichern, und sogar mehrere pro Datensatz. Access 2010 schließlich bringt weitere Verbesserungen wie etwa eine Systemtabelle zum Speichern von Bildern, die innerhalb der Anwendung verwendet werden können. Und mit unserem Standardmodul

zum Verwenden von Bilddateien in allen Teilen einer Access-Anwendung machen Sie aus der langweiligen Benutzeroberfläche mit Office 97-Charm endlich eine Software, deren Bedienung auch für die Augen des Benutzers eine Freude ist. In diesem Kapitel lernen Sie aber nicht nur, wie das Anlage-Feld und das passende Steuerelement funktionieren, sondern auch alternative Techniken kennen. Außerdem finden Sie einige VBA-Module, die wichtige Techniken für den Umgang mit Bildern und binären Daten liefern. Die Module stammen aus der Feder von Sascha Trowitzsch, der auch bei der Erstellung der Beispiele maßgeblich beteiligt war.

Die Beispiele in diesem Kapitel drehen sich ausschließlich um Bilder, weil dies wohl der primäre Einsatzbereich für das Speichern von Dateien und binären Daten in Tabellen ist. Tatsächlich können Sie die Techniken, die sich nicht explizit auf Bilder beziehen, auch für das Speichern und Wiederherstellen von normalen Dateien verwenden. Ein Einsatzzweck abseits der bunten Welt der Bilder ist etwa das Speichern von Dateien, die Sie für die Ausführung einer Datenbankanwendung benötigen, und das Wiederherstellen solcher Dateien zur Laufzeit. So können Sie zum Beispiel für den Ablauf notwendige DLL-Dateien mitliefern, ohne dass der Benutzer sich mit zusätzlichen Dateien herumschlagen muss.

BEISPIELDATENBANK

Die Beispieldatei zu diesem Kapitel finden Sie unter dem Namen *BilderUndBinaere-Dateien.accdb* auf *www.acciu.de/aeb2010*.

11.1 Bilder für Formulare und Berichte

Access 2010 bietet ein neues Feature: Sie können im Entwurf von Formularen und Berichten Bilder in der Datenbank speichern, die dann in Formularen und Berichten verwendet werden können. Das Hinzufügen erfolgt beispielsweise über den entsprechenden Ribbon-Eintrag in der Entwurfsansicht (siehe Abbildung 11.1).

Das Feature ist natürlich vor allem für die Kompatibilität mit Webdatenbanken gedacht. Die zentral gespeicherten und von Formularen, Berichten und Steuerelementen aus referenzierbaren Bilder lassen sich schließlich wesentlich leichter auf einen Webserver transferieren als eine Mischung aus eingebetteten, verknüpften und in Tabellen gespeicherten Bildern.

Sollten Sie planen, eine Access-Datenbank mit Access 2010 zu entwickeln und diese auch unter Access 2007 einzusetzen, müssen Sie die Bilder auf eine der anderen in diesem Kapitel besprochenen Arten speichern. Unter Access 2007 werden über die Bildergalerie eingebundene Bilder nicht angezeigt.

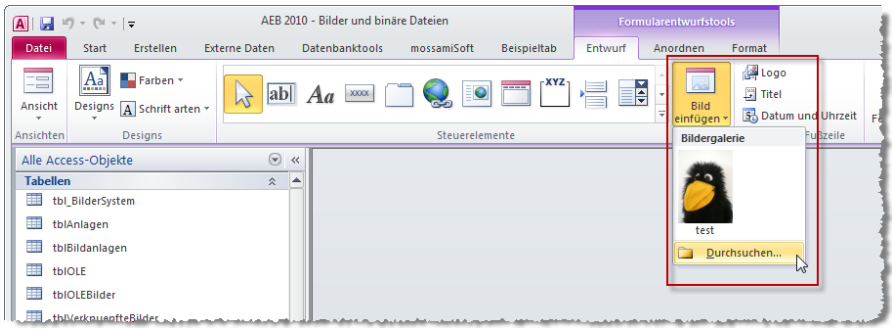


Abbildung 11.1: Hinzufügen von Bildern zu einer Datenbank

Wenn Sie ein Bild zur Bildergalerie hinzugefügt haben, können Sie dieses auf verschiedene Arten verwenden:

» Wählen Sie das Bild aus der Galerie aus und klicken Sie an eine freie Stelle im Formular. Access legt ein neues Bildsteuerelement an und stellt dessen Eigenschaft *Bild* auf den Namen der Bilddatei ohne Dateiendung ein.

» Oder ändern Sie das in einem Bildsteuerelement angezeigte Bild, indem Sie seine Eigenschaft *Bild* anklicken und über das Kombinationsfeld eines der in der Bildergalerie gespeicherten Bilder auswählen. Auf die gleiche Weise verfahren Sie mit weiteren Steuerelementen, die über die *Bild*-Eigenschaft verfügen.

Bilder zur Bildergalerie hinzufügen

Wenn Sie bereits Bilder für Schaltflächen, Bildsteuerelemente et cetera festgelegt haben, können Sie auch diese in die Bildgalerie der Anwendung überführen. Dazu wählen Sie das betroffene Steuerelement aus und stellen für die Eigenschaft *Bildtyp* den Wert *Freigegeben* ein (siehe Abbildung 11.2). Access fügt das Bild zur Bildergalerie hinzu und gibt diesem den Namen des Steuerelements, aus dem das Bild stammt. Hier ist also noch ein wenig Nacharbeit erforderlich.

Leider hat Microsoft dies nicht optimal gelöst: Sie können die Bezeichnungen von Bildern in der Bildergalerie zwar über den entsprechenden Kontextmenüeintrag ändern, aber der neue Name wird nicht automatisch in die betroffenen Steuerelemente übernommen.

Bilder aktualisieren

Sie können jedoch ein in der Bildergalerie gespeichertes Bild durch ein anderes Bild ersetzen. Dazu verwenden Sie den Kontextmenüeintrag *Aktualisieren* des jeweiligen Elements der Bildergalerie. Access zeigt hier auch gleich das aktuelle Bild im Formular oder Bericht an.

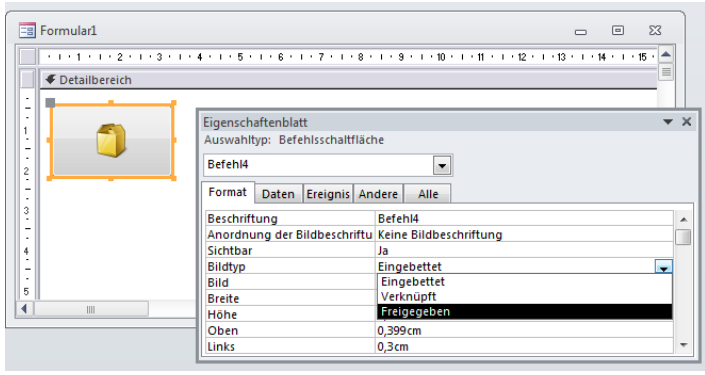


Abbildung 11.2: Hinzufügen eines eingebetteten Bildes zur Bildergalerie

Bilder aus anderen Datenbanken

Wenn Sie ein Formular oder einen Bericht aus einer anderen Datenbank importieren, werden die darin verwendeten Bilder ebenfalls zur aktuellen Datenbank hinzugefügt.

Im Hintergrund

Die zur Bildergalerie hinzugefügten Bilddateien landen in einer zwar ausgeblendeten, aber ungeschützten Systemtabelle namens *MSysResources*. Diese enthält ein Anlagen-Feld sowie einige weitere Felder zum Speichern des Namens und der Endung der Bilddatei (siehe Abbildung 11.3).

Weiter unten unter »Mehrere Bilder gleichzeitig zur Bildergalerie hinzufügen« ab Seite 689 stellen wir eine Funktion vor, mit der Sie mehr als eine Datei gleichzeitig zu dieser Tabelle hinzufügen können. Sie können damit beispielsweise ein komplettes Icon-Verzeichnis einlesen und brauchen dies nicht für jedes Icon einzeln zu erledigen.

Bilder per VBA zur Bildergalerie hinzufügen

Das Access-Objektmodell bietet eine neue Methode des Objekts *CurrentProject* beziehungsweise *CodeProject*, mit dem Sie per VBA schnell eine Bilddatei zur Bildergalerie hinzufügen können. Die Methode heißt *AddSharedImage* und verwendet zwei Parameter:

» *SharedImageName*: Name, der in der Bildergalerie angezeigt werden soll

» *Filename*: Name der Quelldatei

Der folgende Aufruf fügt beispielsweise eine im gleichen Verzeichnis gespeicherte Datei namens *cats.jpg* unter den Namen *Katzen* zur Bildergalerie hinzu:

```
CurrentProject.AddSharedImage "Katzen", CurrentProject.Path & "\cats.jpg"
```

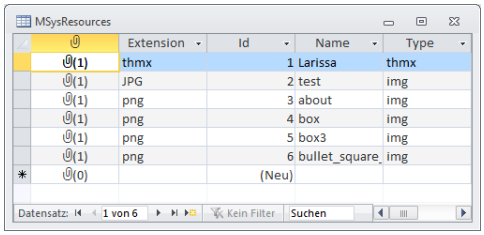


Abbildung 11.3: Die Tabelle *MSysResources* speichert die Bilder der Bildergalerie

Auf die gespeicherten Resourcen greifen Sie dann über die Auflistung *Resources* der Objekte *CurrentProject* beziehungsweise *CodeProject* zu. Die folgenden Anweisungen geben beispielsweise alle Resourcen mit Name und Typ aus:

```
Dim objSharedResource As SharedResource
For Each objSharedResource In CurrentProject.SharedResources
    With objSharedResource
        Debug.Print .Name, .Type
    End With
Next objSharedResource
```

Die Eigenschaft *Type* kann die folgenden beiden Werte annehmen:

» *acResourceTheme* (*0*): Schema

» *acResourceImage* (*1*): Bilddatei

Sie können die Elemente der Bildergalerie auch löschen, und zwar mit der *Delete*-Methode des jeweiligen *SharedResource*-Objekts.

11.2 Bilder und Dateien als Anlage speichern

Die einfachste und naheliegendste Möglichkeit zum Speichern von Bildern bietet der neue Datentyp *Anlage*. Abbildung 11.4 zeigt eine Tabelle mit einem Anlage-Feld in der Entwurfsansicht. In Aktion sieht dieser neue Felddatentyp wie in Abbildung 11.5 aus. Statt eines Feldnamens zeigt die Tabelle ein Büroklammer-Symbol an, das Sie aber durch die Angabe eines Wertes für die Eigenschaft *Bezeichnung* überschreiben können. Im Feld selbst sehen Sie das gleiche Symbol, gefolgt von einer in Klammern eingefassten Zahl. Diese gibt die Anzahl der enthaltenen Anhänge an. Klicken Sie doppelt auf das Feld, erscheint der Dialog *Anlagen*, mit dem Sie Folgendes erledigen können:

» Anlagen hinzufügen

» Anlagen entfernen

» Eine Anlage öffnen

» Eine Anlage aus dem Anlage-Feld auf der Festplatte speichern

» Alle Anlagen auf der Festplatte speichern

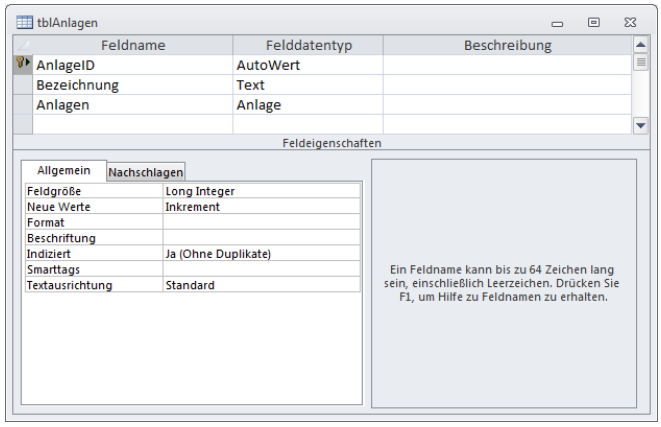


Abbildung 11.4: Eine Tabelle mit einem Anlage-Feld in der Entwurfsansicht

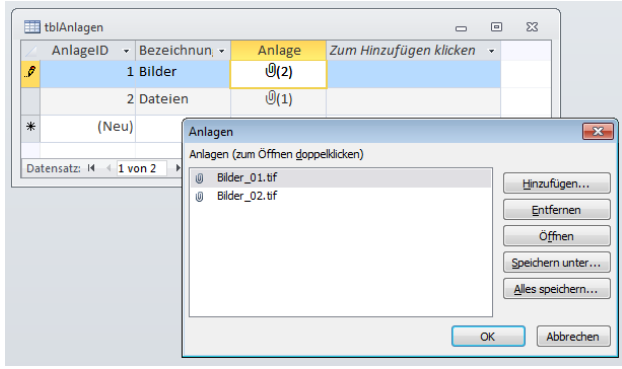


Abbildung 11.5: Das Anlage-Feld in Aktion

Diese Aktionen sind sämtlich selbsterklärend, weshalb das Buch an dieser Stelle nicht weiter darauf eingeht. Als Hinweis nur dies: Als Anlage können Sie beliebige Dateien aufnehmen, bis auf jene, die aus Sicherheitsgründen verboten sind und von Access blockiert werden. Dabei ist ausschließlich die Dateiendung von Belang. Welche Endungen das sind, erfahren Sie in der Onlinehilfe von Access unter *Anlagen|Anfügen von Dateien|Referenzinformationen zu Anlage|Blockierte Dateiformate*. Wie Sie später erfahren werden, lässt sich diese Limitierung mit VBA-Tricks umgehen (siehe »Importieren von Dateien in Anlage-Felder« ab Seite 687).

Eigenschaften des Anlage-Steuerelements

Das neue Anlage-Steuerelement finden Sie in der Entwurfsansicht von Formularen und Berichten im Ribbon *Entwurf|Steuerelemente|Anlage (Büroklammer)*. Neben den üblichen Eigenschaften, die auch andere Steuerelemente besitzen, sind die folgenden erwähnenswert:

» *Standardbild* (VBA: *DefaultPicture*): Laden Sie über diese Eigenschaft eine Bilddatei, die das Steuerelement standardmäßig anzeigen soll. Die Eigenschaft wirkt sich immer dann aus, wenn das Anlage-Feld des angezeigten Datensatzes leer ist – also auch beim Anlegen eines neuen Datensatzes.

» *Hintergrundart* (*BackStyle*): Kann die Werte *Normal* (*1*) oder *Transparent* (*0*) annehmen. Wenn Transparenz eingestellt ist, scheint der Formularhintergund durch transparente Bereiche von Bildern (32bit) hindurch.

» Ereignis *On Attachment Current* (*AttachmentCurrent*): Das Ereignis wird ausgelöst, wenn per Kontextmenü oder mit dem beim Klicken des Bildes erscheinenden Navigationsbar auf eine andere Anlage des Datensatzes geschaltet wird. Das ist nützlich, weil damit etwa der Dateiname des aktuell angezeigten Bildes wie im folgenden Listing in einem Bezeichnungsfeld oder einem anderen Steuerelement angezeigt werden kann:

```
Private Sub Bildanlage_AttachmentCurrent()
    Me!lblFilename.Caption = Me!Bildanlage.FileName
End Sub
```

Die folgenden Eigenschaften stehen nur unter VBA zur Verfügung:

» *FileName*: Dateiname der aktuell angezeigten Anlage (schreibgeschützt)

» *FileData*: Binärdaten der aktuell angezeigten Anlage (Byte-Array, ebenfalls schreibgeschützt)

» *AttachmentCount*: Anzahl der im Datensatz gespeicherten Anlagen

» *CurrentAttachment*: Index der aktuell angezeigten Anlage

» *PictureDisp* (versteckte schreibgeschützte Eigenschaft): Gibt bei Bildanhängen das *Picture*-Objekt des aktuell angezeigten Bildes zurück. Dieses können Sie dann zur weiteren Verarbeitung in VBA verwenden.

» *Back*, *Forward*: Mit diesen Methoden schalten Sie per VBA zwischen den einzelnen Anhängen eines Datensatzes um. Unter Zuhilfenahme der Eigenschaften *AttachmentCount* und *CurrentAttachment* können Sie diese Methoden dazu verwenden, um über selbst gebaute Navigationsschaltflächen zwischen den Bildern eines Datensatzes zu navigieren.

» *SizeToFit*: Die Methode weist das Anlage-Feld an, sich den Ausmaßen des angezeigten Bildes anzupassen. Stellen Sie die Eigenschaft *AutoHeight* des Detailbereichs außerdem auf *True*, dann passt Access auch die Formulargröße dem Bild an.

11.3 Bilder aus Anlage-Feldern in Formularen anzeigen

In Formularen zeigt sich das neue Anlage-Steuerelement genauso benutzerfreundlich wie in der Datenblattansicht von Tabellen. Sie brauchen für ein jungfräuliches Formular nur eine Tabelle mit einem Anlage-Feld als Datensatzquelle auszuwählen und das Feld *Bildanlage* dieser Tabelle in den Formularentwurf zu ziehen (siehe Abbildung 11.6). Mit diesem Entwurf erhalten Sie beispielsweise die Formularansicht aus Abbildung 11.7.

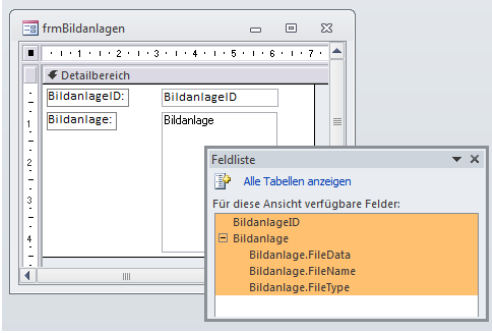


Abbildung 11.6: Hinzufügen des Primärschlüssels und des Anlage-Feldes zu einem Formular

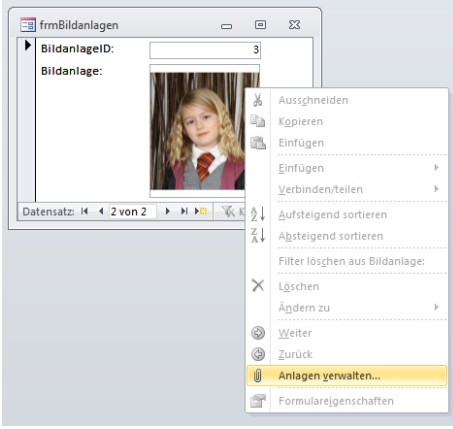


Abbildung 11.7: Über das Kontextmenü des Anlage-Feldes können Sie den Anlagen-Dialog öffnen oder, wenn mehrere Anlagen vorhanden sind, darin blättern

Es geht aber auch anders als im ersten Beispiel: Vielleicht möchten Sie einmal direkt durch alle im Anlage-Feld verschiedener Datensätze enthaltenen Bilder blättern. Dann ziehen Sie nicht das komplette Anlage-Feld, sondern die in der dahinter verborgenen Tabelle enthaltenen Felder aus der Feldliste in den Formularentwurf (siehe Abbildung 11.8). Das Ergebnis sieht ähnlich aus (siehe Abbildung 11.9), unterscheidet sich aber in zwei wesentlichen Punkten:

» Sie können direkt im Formular durch alle Bilder blättern, egal, ob mehrere Bilder im Anlage-Feld eines einzigen Datensatzes verborgen sind.

» Sie können so nicht auf den *Anlage*-Dialog zugreifen.

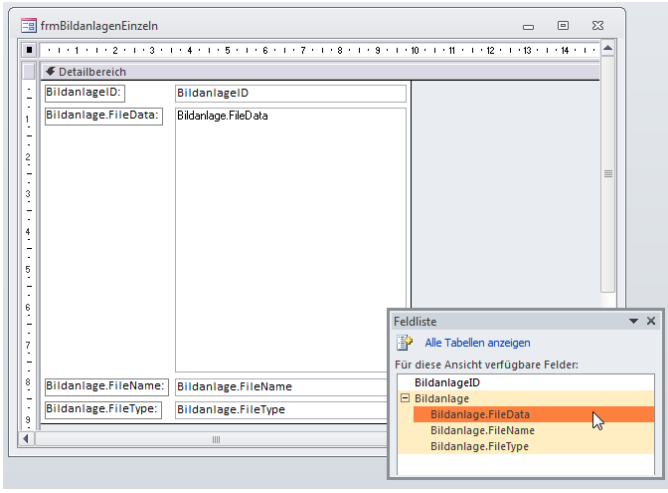


Abbildung 11.8: Entwurf eines Formulars mit Anlage-Feld

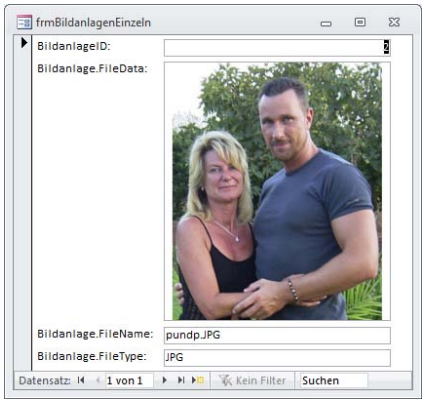


Abbildung 11.9: Ein Bild im Anlage-Steuerelement in einem Formular

Sie müssen beachten, dass Access einen Datensatz mit mehreren Bildern hier wie zwei verknüpfte Tabellen behandelt: Eine Tabelle mit den Basisdaten (also den Daten aus den Feldern, die nicht die Anlage enthalten) und eine Tabelle mit den im Anlage-Feld enthaltenen Dateien.

Diese sind über eine 1:n-Beziehung miteinander verknüpft. Wenn die oben verwendete Tabelle *tblBildanlagen* also im ersten Datensatz zwei Bildanlagen enthält, zeigt das Formular zwei Datensätze mit den gleichen Basisdaten, aber unterschiedlichen Anlagen beziehungsweise Anlage-Informationen an.

11.4 Bilder aus Anlage-Feldern in Berichten anzeigen

In Berichten funktioniert dies genauso einfach: Sie stellen die passende Tabelle als Datensatzquelle des Berichts ein und fügen das Anlage-Feld selbst zum Bericht hinzu. Das Ergebnis überzeugt nicht – zumindest nicht in der Seitenansicht: Der Bericht zeigt nur das jeweils erste Bild zu jedem Datensatz an (siehe Abbildung 11.10).

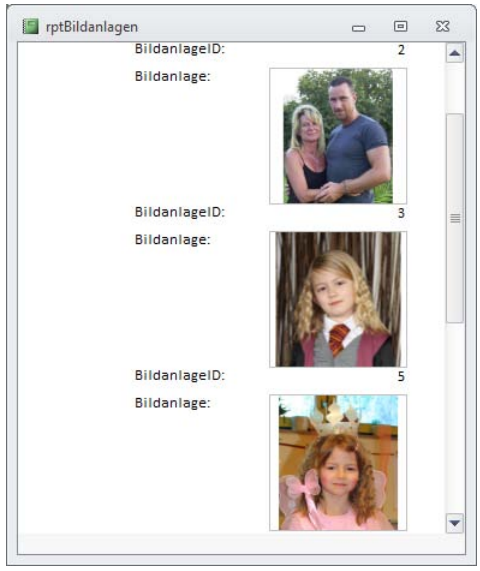


Abbildung 11.10: Berichte zeigen nur je ein Bild eines Anlage-Feldes pro Datensatz an, auch wenn dieses mehrere Bilder enthält

Wenn Sie alle Bilder der enthaltenen Datensätze in der Seitenansicht des Berichts ausgeben möchten, fügen Sie wiederum das dem eigentlichen Anlage-Feld untergeordne-

te *<Name des Anlage-Felds>.FileData*-Feld zum Berichtsentwurf hinzu. Auch hier zeigt Access die Grunddaten von Datensätzen, die mehr als ein Bild enthalten, jeweils doppelt an (siehe Abbildung 11.11). Dem können Sie entgegenwirken, indem Sie die Eigenschaft *Duplikate ausblenden* für solche Steuerelemente, die je Datensatz nur einfach angezeigt werden sollen, auf *Ja* einstellen und – falls diese Steuerelemente sich allein in einer Zeile befinden – mit dem Wert *Ja* für die Eigenschaft *Verkleinerbar* dafür sorgen, dass keine unschönen Lücken entstehen.

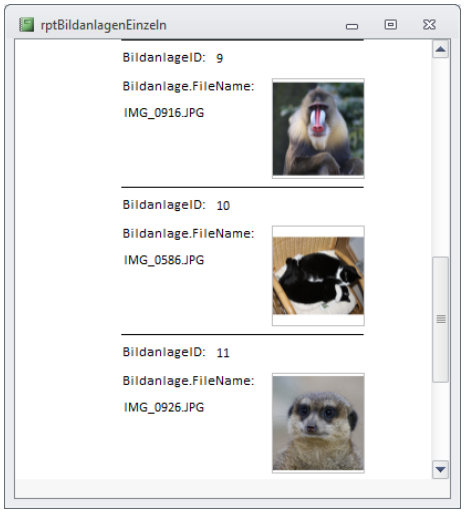


Abbildung 11.11: Alle Bilder eines Anlage-Felds in der Berichtsansicht eines Berichts

Komfortabler geht dies mit einer Gruppierung nach dem Primärschlüssel der Tabelle mit dem Anlage-Feld. Das kann im Entwurf etwa so wie in Abbildung 11.12 aussehen. Dass die Berichtsansicht so wie in Abbildung 11.13 aussieht, ist ein weiteres Indiz dafür, dass Access die Anlagen intern in einer verborgenen Tabelle speichert.

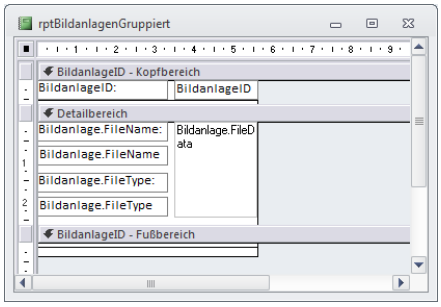


Abbildung 11.12: Entwurf eines nach dem Primärschlüssel der Datensatzquelle gruppierten Berichts mit Bildanlagen ...

11.5 Bilder und Dateien aus Anlage-Feldern auf der Festplatte speichern

Das Speichern von Anlagen aus dem Anlage-Feld ist äußerst trivial: Sie öffnen einfach den Anlagen-Dialog per Doppelklick auf das jeweilige Feld im Formular und führen dort die erforderlichen Schritte aus. Das funktioniert aber leider nur, wenn Sie das Anlage-Feld wie in der ersten in »Bilder aus Anlage-Feldern in Formularen anzeigen« ab Seite 682 beschriebenen Methode anlegen.

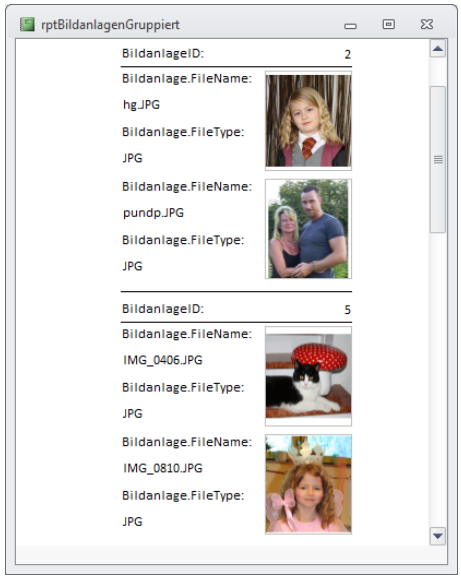


Abbildung 11.13: ... der in der Seitenvorschau so aussieht

11.6 Dateien per VBA in Anlage-Felder importieren und exportieren

Natürlich können Sie Bilder und Dateien auch per VBA in einem Anlage-Feld speichern und von dort aus wieder ins Dateisystem exportieren.

BEISPIELDATENBANK

Die Prozeduren dieses Abschnitts finden Sie im Modul *mdlBLOBs0710* der Beispieldatenbank *DAO2010.accdb* auf *www.acciu.de/aeb2010*.

11.6.1 Importieren von Dateien in Anlage-Felder

Die nachfolgend vorgestellte Funktion können Sie universell für das Importieren beliebiger Dateien in Anlage-Felder verwenden. Sie erwartet die folgenden Parameter:

» *strFileName*: Verzeichnis und Name der zu importierenden Datei

» *strTable*: Name der Tabelle, in der sich das Anlage-Feld befindet

» *strFieldAttach*: Name des Anlage-Feldes

» *boolEdit*: Gibt an, ob die Anlage in einem bestehenden Datensatz gespeichert werden soll

» *strIDField*: Name des Primärschlüsselfeldes der Tabelle

» *varID*: Wert des Primärschlüsselfeldes für den Datensatz, zu dem eine Anlage hinzugefügt werden soll

» *strAttachment*: Name des Attachments, das gegebenenfalls überschrieben werden soll

Wenn Sie eine Datei in einem neuen Datensatz speichern möchten, verwenden Sie etwa diesen Aufruf:

```
StoreBLOB0710 CurrentProject.Path & "\Bilder_01.tif", "tblAnlagen", "Anlagen"
```

Zum Anfügen einer Datei in einem vorhandenen Datensatz setzen Sie diese Anweisung ab – wobei *2* der Primärschlüsselwert des Zieldatensatzes ist:

```
StoreBLOB0710 CurrentProject.Path & "\Bilder_01.tif", "tblAnlagen", "Anlagen", True,
"AnlageID", 2
```

Schließlich können Sie auch ein Attachment mit einem bestimmten Dateinamen in der Tabelle überschreiben. Auch dies erfordert die Angabe des Datensatzes und zusätzlich den Namen des bereits vorhandenen Attachments:

```
StoreBLOB0710 CurrentProject.Path & "\Bilder_02.tif", "tblAnlagen", "Anlagen", True,
"AnlageID", 2, "Bilder_01.tif"
```

Die Routine hat übrigens noch ein weiteres Feature, das eine völlig unverständliche Eigenschaft von Anlage-Feldern ausschaltet: Und zwar dürfen nur Dateien mit bestimmten Dateiendungen importiert werden. Und das ist wörtlich zu nehmen: Der Inhalt der Datei spielt keine Rolle, Sie brauchen die Datei nur vor dem Speichern im Anlage-Feld umzubenennen und dies anschließend wieder rückgängig zu machen. Und genau das macht diese Funktion auch, indem sie den beim Speichern einer Datei mit einer nicht erlaubten Dateiendung auftretenden Fehler entsprechend behandelt.

Der Zugriff auf die Anlagen erfolgt mit DAO. Der grundlegende Ablauf ist, dass Sie zunächst ein *Recordset*-Objekt auf Basis der Tabelle mit dem Anlage-Feld erstellen und ein weiteres auf Basis der *Value*-Eigenschaft des Anlage-Feldes. Letzteres muss unbedingt ein *Recordset2*-Objekt sein, ebenso wie das *Field*-Objekt mit dem Anlage-Feld die neue Version *Field2* braucht.

Die Prozedur sieht wie folgt aus:

```
Function StoreBLOB0710(strFilename As String, strTable As String, _
                           strFieldAttach As String, Optional boolEdit As Boolean, _
                       Optional strIDField As String, Optional varID As Variant, _
                                   Optional strAttachment As String) As Boolean
    Dim fld2 As DAO.Field2
    Dim rstDAO As DAO.Recordset2
    Dim rstACCDB As DAO.Recordset2
    On Error GoTo ErrHandler
    Set rstDAO = CurrentDb.OpenRecordset("SELECT * FROM [" & strTable & "]", _
                                                                    dbOpenDynaset)
    If boolEdit Then
        If IsNull(varID) Then Err.Raise vbObjectError + 1, , _
                                                    "Keine Datensatz-ID angegeben!"
        rstDAO.FindFirst "CStr([" & strIDField & "])='" & CStr(varID) & "'"
        If rstDAO.NoMatch Then Err.Raise vbObjectError + 2, , _
                                    "Datensatz mit ID " & varID & " nicht gefunden!"
        rstDAO.Edit
    Else
        rstDAO.AddNew
    End If
    Set rstACCDB = rstDAO(strFieldAttach).Value
    If boolEdit Then
        If rstACCDB.EOF Then 'Fall 1: Es gibt noch keine Anlagen; > neue Anlage
            rstACCDB.AddNew
        Else
            Do While Not rstACCDB.EOF
                rstACCDB.MoveNext
            Loop
            rstACCDB.FindFirst "[FileName]='" & strAttachment & "'"
            'Fall2: Es gibt keine Anlage mit dem Namen in sAttachment: > neue Anlage
            'Fall3: Anlage gefunden; dann editieren
            If rstACCDB.NoMatch Then rstACCDB.AddNew Else rstACCDB.Edit
        End If
    Else
```

```
        rstACCDB.AddNew
    End If
    Set fld2 = rstACCDB.Fields!FileData
    On Error Resume Next
    fld2.LoadFromFile (strFilename)
    If Err.Number = -2146697202 Then    'Unerlaubte Dateiendung! Spezialbehandlung...
        On Error GoTo ErrHandler
        Name strFilename As strFilename & ".dat"  'Datei mit Endung ".dat" anfügen
        fld2.LoadFromFile (strFilename & ".dat")  'Datei laden
        Name strFilename & ".dat" As strFilename  'Umbenennung rückgängig machen
        'Anlagename setzen
        rstACCDB.Fields!FileName = Mid(strFilename, InStrRev(strFilename, "\") + 1
        rstACCDB.Update
    Else
        On Error GoTo ErrHandler
        rstACCDB.Update
    End If
    rstDAO.Update
    StoreBLOB0710 = True    'Rückgabe True = Alles ok.
Finally:
    On Error Resume Next
    rstACCDB.Close
    rstDAO.Close
    Set rstACCDB = Nothing
    Set rstDAO = Nothing
    Set fld2 = Nothing
    Exit Function
ErrHandler:
    MsgBox Err.Description, vbCritical
    Resume Finally
End Function
```

Listing 11.1: Flexibles Speichern von Dateien im Anlage-Feld

Mehrere Bilder gleichzeitig zur Bildergalerie hinzufügen

Die Bildergalerie von Access 2010, die wir weiter oben vorgestellt haben, hat einen entscheidenden Nachteil: Es lassen sich nicht mal eben mehrere Bilder gleichzeitig hinzufügen. Das ist insbesondere ärgerlich, weil der Dialog zur Auswahl eines Bildes immer wieder den Ordner der aktuellen Datenbank anzeigt. Mit der obigen Funktion und einer kleinen Erweiterung fügen Sie schnell beliebig viele Bilder eines Verzeichnisses zur Bildergalerie hinzu. Dazu brauchen Sie zunächst eine Funktion, die einen Dateidialog

zur Auswahl der hinzuzufügenden Dateien öffnet und für jede ausgewählte Datei eine weitere Funktion aufruft:

```
Public Sub BilderEinfuegen()
    Dim objFiledialog As FileDialog
    Dim obj As Object
    Dim i As Integer
    Set objFiledialog = Application.FileDialog(msoFileDialogFilePicker)
    With objFiledialog
        .AllowMultiSelect = True
        .Filters.Add "Images", "*.png; *.gif; *.jpg; *.jpeg", 1
        .Show
        For i = 1 To .SelectedItems.count
            BildEinfuegen .SelectedItems.Item(i)
        Next i
    End With
End Sub
```

Die zweite Funktion erwartet schlicht den Namen der einzulesenden Bilddatei. Sie verwendet dann die oben vorgestellte Prozedur *StoreBLOB0710*, um das Bild in einem neuen Datensatz der Tabelle *MSysResources* zu speichern.

Zu unserem Glück fehlen noch ein paar weitere Felder, nämlich *Extension*, *Name* und *Type*. Um die entsprechenden Werte zum soeben angelegten Datensatz hinzuzufügen, ermittelt die Funktion zunächst die *ID* des zuletzt angelegten Datensatzes.

Die folgenden Anweisungen lesen dann den Namen und die Erweiterung aus der kompletten Pfadangabe aus.

Diese Informationen landen dann schließlich per *Execute*-Anweisung in der Tabelle *MSysResource*:

```
Public Function BildEinfuegen(strBild As String)
    Dim db As DAO.Database
    Dim lngID As Long
    Dim strName As String
    Dim strFilename As String
    Dim strExtension As String
    Set db = CurrentDb
    StoreBLOB0710 strBild, "MSysResources", "Data", False
    lngID = db.OpenRecordset("SELECT @@IDENTITY").Fields(0)
    strFilename = Mid(strBild, InStrRev(strBild, "\") + 1)
    strName = Split(strFilename, ".")(0)
    strExtension = Split(strFilename, ".")(1)
```

12

Ribbon

Wenn Sie von Access 2007 auf Access 2010 gewechselt sind, haben Sie den größten Kulturschock bereits hinter sich. Seit Access 2007 heißt die neue Menüleiste der Microsoft Office-Anwendungen nämlich Ribbon. Die deutsche Bezeichnung für Ribbon (wörtlich übersetzt: »Band«) lautete in Office 2007 Multifunktionsleiste, in Office 2010 hat der Übersetzer sich für den Ausdruck »Menüband« entschieden. In diesem Buch wird daher ausschließlich von Ribbon die Rede sein – das ist erstens kürzer und zweitens wesentlich cooler. Das Ribbon ist Teil der runderneuerten Benutzeroberfläche und übernimmt die Rolle von Menü- und Symbolleisten gleichzeitig. Während es unter Office 2007 noch einen Office-Button gab, mit dem Sie das Office-Menü anzeigen konnten, bietet uns Office 2010 etwas, von dem die Fans von Musikgruppen träumen: den Backstage-Bereich. Leider laufen dort keine Promis und Groupies herum, sondern es handelt sich

dabei um eine Erweiterung des Ribbons. Für Sie als Entwickler ist natürlich besonders interessant, wie Sie das Ribbon für Ihre Zwecke anpassen können, um etwa die eingebauten Befehle auszublenden und eigene Tabs, Groups und Steuerelemente hinzuzufügen. Und wenn Sie professionelle Anwendungen entwickeln, freuen Sie sich möglicherweise darauf, den Backstage-Bereich etwa für die Unterbringung von Anwendungsoptionen zu verwenden.

Deshalb hält sich dieses Kapitel nicht mit einer Beschreibung der Bedienung des Ribbons auf (wenn Sie es bis zu diesem Kapitel geschafft haben, sollten Sie das schon drauf haben), sondern konzentriert sich auf die Anpassung und Programmierung des Ribbons.

Vorab zu Ihrer Beruhigung: Das Anpassen funktioniert ohne Weiteres, aber erstens völlig anders und zweitens nicht so einfach wie bei den *CommandBars* älterer Access-Versionen. Der Hauptgrund dafür, dass es nicht so leicht geht, ist die Tatsache, dass Microsoft noch keine grafische Benutzeroberfläche zum Anpassen des Ribbons mitliefert. Kommen Sie von Access 2007, brauchen Sie ebenfalls keinen allzu großen Respekt zu haben: Auch der Backstage-Bereich wird schlicht und einfach per XML definiert und später per VBA automatisiert.

Während die Entwicklergemeinde seit der Version von Access 2007 auf Unterstützung beim Bau von Ribbons durch ein Tool von Microsoft hofft, geschieht dies auch mit Access 2010 noch nicht. Außerdem lernen Sie das vom Autor dieses Buchs entwickelte Tool zum Programmieren des Ribbons kennen. Wir würden es gern als kostenlose Dreingabe zum Buch liefern, aber die Entwicklungszeit beträgt bis jetzt bereits mehrere Monate. Daher finden Sie im Buch-Download eine Testversion, mit der Sie Ribbons mit einer begrenzten Anzahl von Elementen einfach programmieren können – so viel, dass es zumindest für alle in diesem Kapitel beschriebenen Beispiele reicht. Wenn Sie auf der Webseite zum Buch unter *http://www.access-entwicklerbuch.de/2010/ribbon* den Registrierungsschlüssel eingeben, erhalten Sie außerdem einen Rabatt von 50% auf den Listenpreis.

BEISPIELDATENBANKEN

Den Download mit dem Beispielen finden Sie unter *www.acciu.de/aeb2010*. Die Datenbank zu diesem Kapitel heißt *Ribbons.accdb*.

12.1 Anpassen des Ribbons/CustomUI

Wie schwer Ihnen das Anpassen des Ribbons fällt, hängt in erster Linie davon ab, ob Sie bereits XML-Grundkenntnisse besitzen oder sich diese aneignen wollen. Sie müssen allerdings kein XML-Guru sein, um das Ribbon anzupassen. Genau genommen brauchen

Sie sich noch nicht einmal überhaupt mit XML zu beschäftigen, wenn Sie die eingeschränkte Version des *Ribbon-Admin 2010* aus dem Buch-Download verwenden.

Begriffsklärung

Die Gesamtheit der Elemente, die Sie per XML-Dokument anpassen können, nennen wir im Folgenden *CustomUI*. Dazu gehören diese Bereiche:

» Backstage-Bereich: der Bereich, den Sie durch einen Klick auf den Registerreiter *Datei* des Ribbons aktivieren

» Schnellstart-Leiste: Schaltflächen ganz oben neben dem Access-Symbol

» Ribbon: Die eigentliche Leiste mit den Tabs und Menübefehlen

Eine eigene *CustomUI*-Anpassung legen Sie in einem XML-Dokument fest, das Sie auf verschiedene Arten anwenden können. Dieses XML-Dokument speichern Sie am einfachsten in einer speziellen Tabelle und vergeben dabei auch gleich einen Namen für diese Anpassung. Mit bestimmten Eigenschaften legen Sie dann fest, welche Ihrer Anpassungen direkt beim Start der Anwendung durchgeführt werden sollen. Außerdem können Sie dafür sorgen, dass beim Öffnen von Formularen und Berichten andere oder weitere Anpassungen durchgeführt werden sollen – zum Beispiel, um einen weiteren Registerreiter hinzuzufügen, der spezielle Steuerelemente zum Aufrufen der Funktionen von Formularen oder Berichten enthält.

Aufbau des Ribbons

Das Ribbon besteht aus einem oder mehreren Registerreitern, den sogenannten Tabs. Jedes *tab*-Element enthält eine oder mehrere Gruppen (*group*) mit Steuerelementen (*button*, *comboBox*, ...). Einige Steuerelemente können wiederum andere Steuerelemente enthalten. Die Schnellstartleiste besteht aus einfachen Schaltflächen zum Aufrufen eingebauter oder benutzerdefinierter Funktionen. Der Backstage-Bereich ist der Bereich, den Sie durch einen Klick auf den Registerreiter *Datei* des Ribbons öffnen. Ihm ist ein eigenes Kapitel gewidmet (siehe »Backstage« ab Seite 777).

12.2 Schnellstart

Die grundsätzlichen Schritte für die Anpassung des Ribbons, das beim Start der Anwendung erscheinen soll, sehen so aus:

» Erstellen einer Tabelle zum Speichern einer oder mehrerer Ribbon-Anpassungen beziehungsweise XML-Dokumente

» Definieren der Anpassung durch ein entsprechendes XML-Dokument

» Eintragen dieser Anpassung in einen neuen Datensatz der dafür vorgesehenen Tabelle

» Schließen und öffnen der aktuellen Access-Datei

» Auswählen des beim Start anzuzeigenden Ribbons in den Access-Optionen

» Erneutes Schließen und Öffnen der Access-Datei – die Ribbon-Anpassung ist da!

Sie können auch Ribbon-Anpassungen definieren, die beim Öffnen von Formularen oder Berichten erscheinen sollen. In diesem Fall gehen Sie fast genauso wie oben vor – mit dem Unterschied, dass Sie den Namen der Ribbon-Anpassung in der Eigenschaft *Name des Menübands* des Formulars oder Berichts auswählen.

12.2.1 Tabelle USysRibbons erstellen

Schauen wir uns an, wie das im Detail abläuft. Erstellen Sie zunächst eine Tabelle namens *USysRibbons* mit den drei Feldern *ID* (Primärschlüsselfeld), *RibbonName* (Textfeld), *RibbonXML* (Memofeld). Die Tabelle sieht im Entwurf wie in Abbildung 12.1 aus. Nach dem Speichern erscheint die Tabelle normalerweise nicht im Navigationsbereich. Das liegt daran, dass Access Tabellen mit Namen wie *MSys...* oder *USys...* als Systemtabellen einstuft und ausblendet. Sie können diese Tabellen einblenden, indem Sie mit der rechten Maustaste auf den Titel des Navigationsbereichs klicken und aus dem Kontextmenü den Eintrag *Navigationsoptionen* auswählen. Im nun erscheinenden Dialog aktivieren Sie die Option *Systemobjekte anzeigen*.

12.2.2 customUI-Definition erstellen

Erstellen Sie dann ein XML-Dokument mit der Beschreibung der Ribbon-Anpassung. Das folgende Beispiel fügt schlicht ein *tab*-Element mit einem *group*-Element und einer Schaltfläche (*button*) hinzu. Einige *id*- und *label*-Attribute sorgen dafür, dass die Elemente eindeutig benannt und mit einer Beschriftung versehen werden. Die umschließenden *customUI*-, *ribbon*- und *tabs*-Elemente sind für Anpassungen des Ribbons obligatorisch:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon>
    <tabs>
      <tab id="tab1" label="Beispieltab">
        <group id="grp1" label="Beispielgruppe">
          <button id="btn1" label="Beispielbutton"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

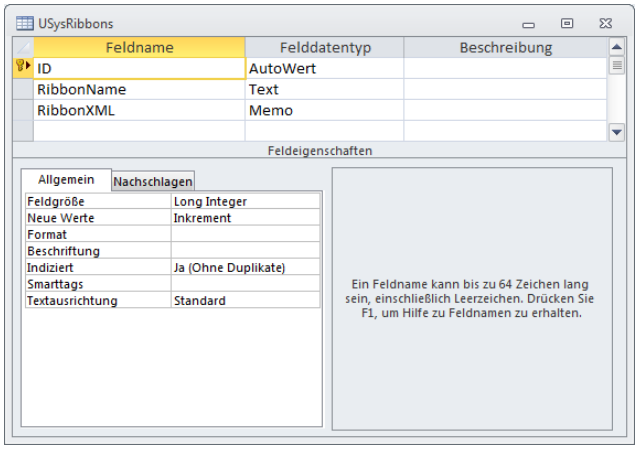


Abbildung 12.1: Diese Tabelle speichert später Ihre Ribbon-Anpassungen.

Legen Sie einen neuen Datensatz in der Tabelle *USysRibbons* an. Tragen Sie als Namen den Wert *Beispielribbon* ein und fügen Sie dieses XML-Dokument zum Feld *RibbonXML* hinzu. Die Tabelle sieht nun wie in Abbildung 12.2 aus.

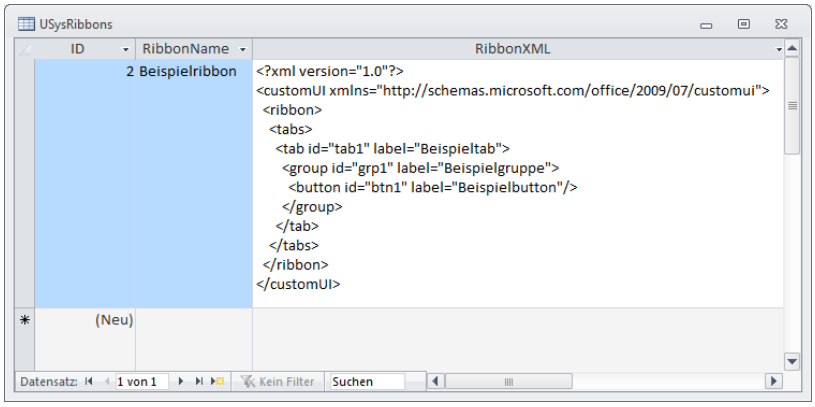


Abbildung 12.2: Die Tabelle *USysRibbons* mit einem Beispiel-CustomUI

EINFACHE RIBBON-ERSTELLUNG

Mit dem Tool *Ribbon-Admin 2010* erstellen Sie ganz einfach Ribbon- und Backstage-Anpassungen. Mehr dazu erfahren Sie am Ende des Kapitels. Im Download finden Sie eine Version dieses Kapitels, dass um die Beschreibung der Funktionen des Ribbon-Admin 2010 angereichert ist.

12.2.3 customUI-Anpassungen anwenden

Nun schließen Sie die aktuelle Access-Anwendung und öffnen diese erneut. Aktivieren Sie mit *Datei|Optionen* den Optionen-Dialog von Access und wählen Sie im Bereich *Aktuelle Datenbank* unter *Menüleisten- und Symbolleistenoptionen* den Wert *Neues Ribbon* für *Name des Menübands* aus (siehe Abbildung 12.3).

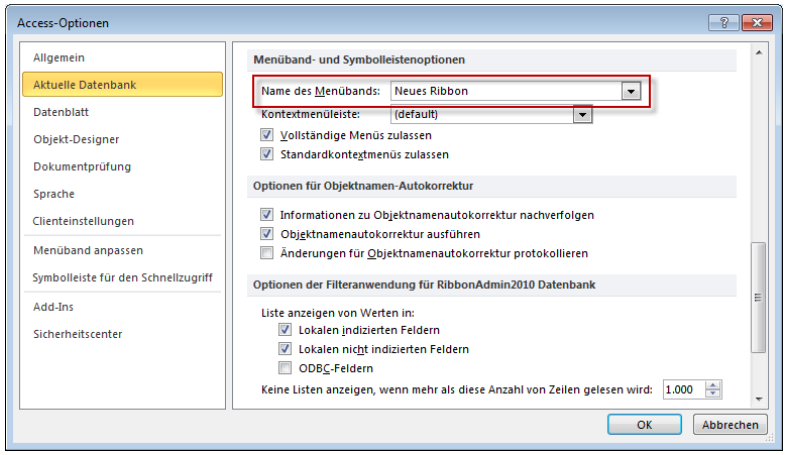


Abbildung 12.3: Auswählen eines *customUI* für die aktuelle Anwendung

Schließen Sie die Anwendung nochmals und öffnen Sie diese wieder. Nun ist das *tab*-Element mit der Beschriftung *Beispieltab* sichtbar. Ein Klick darauf zeigt auch die gewünschte Gruppe mit der Schaltfläche (siehe Abbildung 12.4).

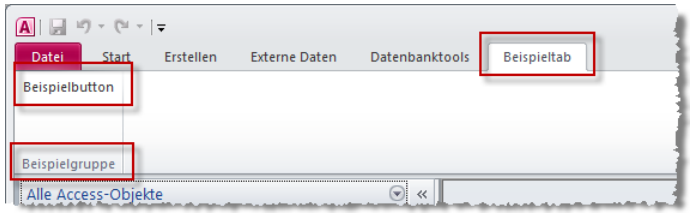


Abbildung 12.4: Access-Anwendung mit Beispielribbon

12.3 Manuelles Anpassen des customUI

Bevor Sie mit dem Erstellen von XML-Dokumenten beginnen, schauen wir uns noch schnell an, wie Sie die Benutzeroberfläche von Hand anpassen können. Auch hierzu liefert Access mittlerweile einige Möglichkeiten mehr als in der Version 2007.

Damit können Sie zumindest die eingebauten Elemente des *customUI* beeinflussen. Ausgangspunkt hierfür ist der Bereich *Menüband anpassen* des Dialogs *Access-Optionen*. Voraussetzung für die Anpassung ist außerdem eine geöffnete Access-Datenbank.

Damit Sie gleich ein praxisnahes Beispiel erhalten, schauen wir uns das *tab*-Element *Entwurf* der Entwurfsansicht von Formularen an. Hier hat Microsoft eine Änderung gegenüber Access 2007 vorgenommen, die viele Entwickler nicht wahrhaben wollen. Schauen Sie selbst: Abbildung 12.5 zeigt, dass in der Formularansicht nicht alle Steuerelement-Schaltflächen gleichzeitig sichtbar sind. Wenn Sie beispielsweise ein Kontrollkästchen, ein Listenfeld oder ein Unterformularsteuerelement zum Formular hinzufügen möchten, müssen Sie die Galerie erst wie in Abbildung 12.6 ausklappen.

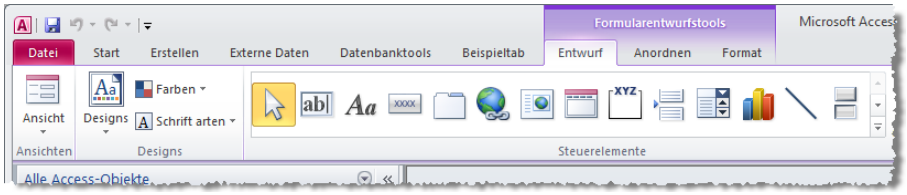


Abbildung 12.5: Steuerelemente im Ribbon vor ...

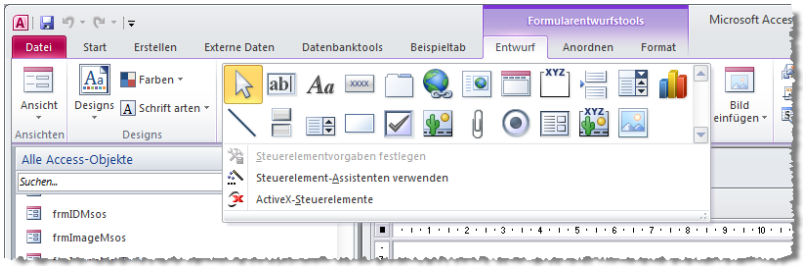


Abbildung 12.6: ... und nach dem Aufklappen der Galerie.

In Access 2007 waren die Steuerelemente über kleinere Schaltflächen erreichbar, die dafür aber zumindest alle gleichzeitig sichtbar waren. Was sich Microsoft auch immer bei dieser Änderung gedacht hat: Es ist ein guter Anlass, die Möglichkeiten der *customUI*-Anpassung mit Access-Bordmitteln zu demonstrieren.

Dazu öffnen Sie die Access-Optionen und wechseln zum Bereich *Menüband anpassen*. Es erwartet Sie dort ein Anblick wie in Abbildung 12.7. Das linke Listenfeld zeigt alle für Anpassungen verfügbaren Befehle an. Sie können diese mit dem darüber befindlichen Kombinationsfeld nach verschiedenen Kriterien filtern. Aber Achtung: Der Eintrag *Alle Befehle* etwa liefert längst nicht alle Befehle – so findet sich der Befehl *Textfeld (Formularsteuerelement)* beispielsweise nur unter *Nicht im Menüband enthaltene Befehle*.

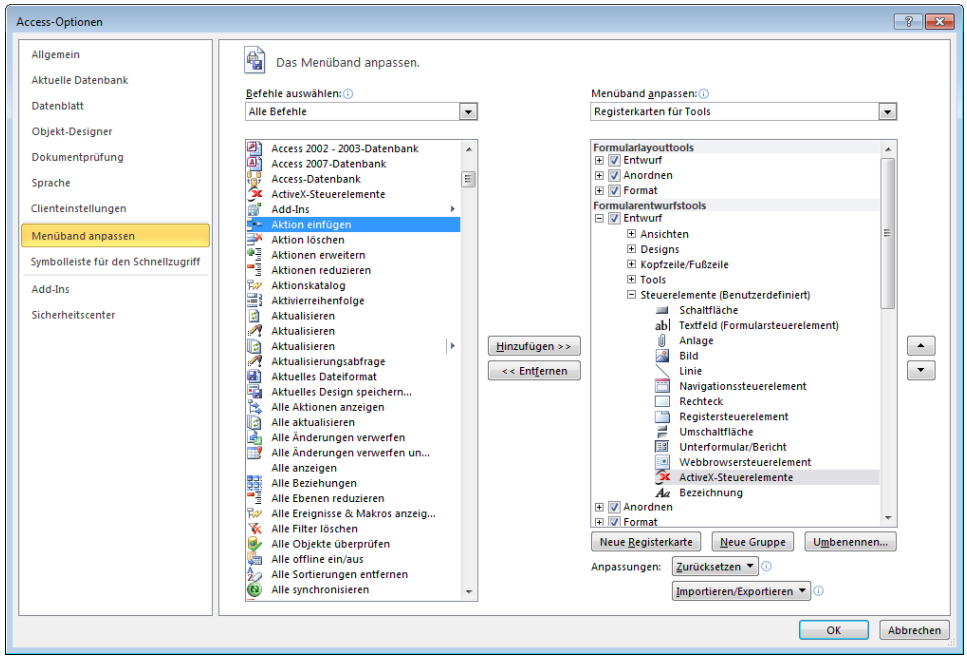


Abbildung 12.7: Anpassen des customUI mit Bordmitteln

Um die Schaltflächen für alle Steuerelemente zusammenzusuchen, müssen Sie sich also ein wenig durch die verschiedenen Listen wühlen.

Wo aber landen die Befehle? Das rechte Listenfeld zeigt die Struktur des Ribbons an, auch hier gibt es verschiedene Filterkriterien. Wählen Sie oben *Registerkarten für Tools* aus und klicken sich dann wie in Abbildung 12.8 zu den Steuerelementen durch. Mit einem Rechtsklick auf *Entfernen* entsorgen Sie die unpraktische Gruppe. Experimentieren Sie ruhig, mit der Schaltfläche *Zurücksetzen* bringen Sie das *customUI* wieder in den Anfangszustand (dies gilt nur für Änderungen, die über diesen Dialog herbeigeführt wurden).

Danach fügen Sie eine benutzerdefinierte Gruppe zum *tab*-Element *Entwurf* hinzu. Aktivieren Sie dazu den Eintrag *Entwurf* und klicken Sie auf *Neue Gruppe*. Benennen Sie die neue Gruppe gleich im Anschluss in *Steuerelemente* um. Nun beginnt die Sucherei: Fügen Sie alle benötigten Steuerelement-Schaltflächen aus dem linken Listenfeld zum rechten Listenfeld hinzu (siehe Abbildung 12.9).

Mit den *Nach oben*- und *Nach unten*-Schaltflächen können Sie die Reihenfolge der Steuerelemente nach Ihren Wünschen ändern. Schließen Sie den Dialog und öffnen Sie ein Formular in der Entwurfsansicht. Und das Ergebnis ist ... fantastisch! Sie erreichen nun alle Steuerelemente mit einem einzigen Mausklick, wie Abbildung 12.10 zeigt.

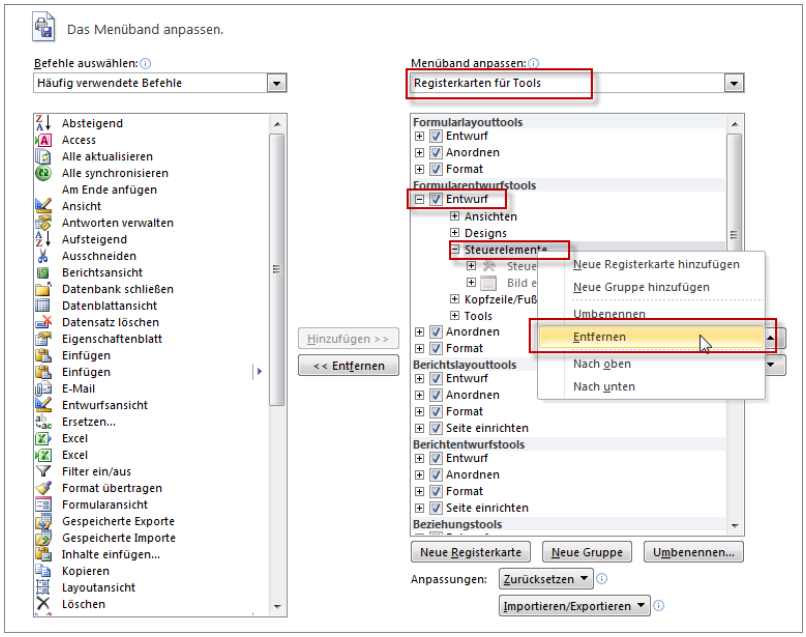


Abbildung 12.8: Löschen der Steuerelemente im Formularentwurf

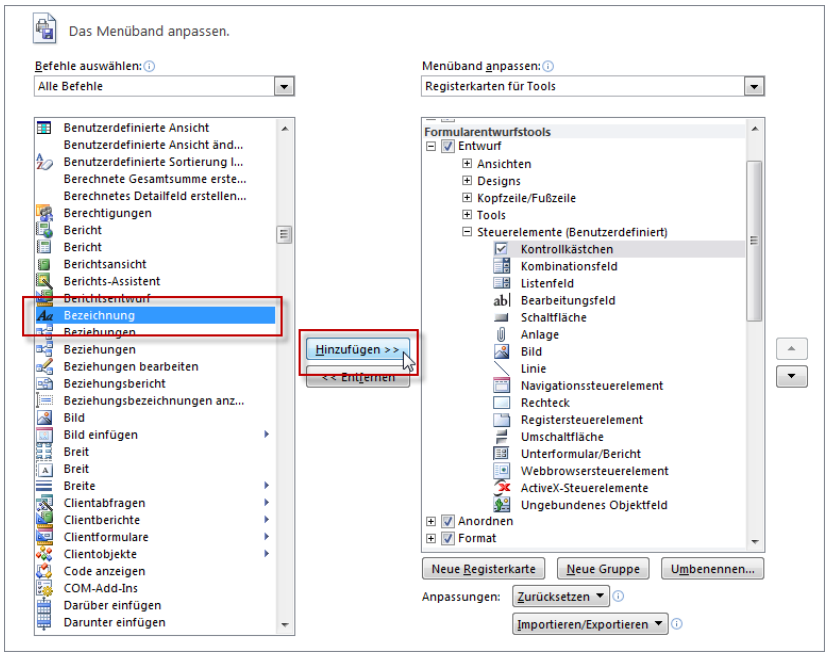


Abbildung 12.9: Hinzufügen von Steuerelementen zu einer benutzerdefinierten Gruppe

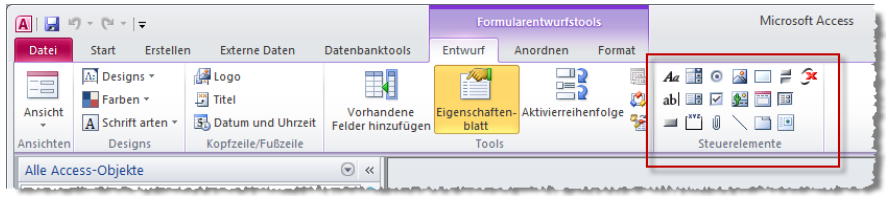


Abbildung 12.10: Die Steuerelemente mit kleinen Icons und auf einen Blick sichtbar

Übrigens müssen Sie die Datenbank nicht erneut öffnen, damit die Änderungen sichtbar werden (außer bei Anpassungen der Schnellzugriffsleiste).

Diese Änderung ist nun immer verfügbar, wenn Sie mit Access arbeiten. Fehlt nur noch eine Möglichkeit, diese Anpassung auf einen anderen Rechner übertragen zu können. Kein Problem: Auch das gelingt mit Access 2010. Klicken Sie unten auf die Schaltfläche *Importieren/Exportieren* und wählen Sie zunächst den Eintrag *Exportieren* aus. Geben Sie einen Dateinamen an und speichern Sie die Änderungen. Anschließend können Sie diese auf anderen Rechnern wieder importieren. Die exportierte Datei sieht wie folgt aus:

```
<mso:cmd app="Access" dt="0" />
  <mso:customUI xmlns:mso="http://schemas.microsoft.com/office/2009/07/customui">
    <mso:ribbon>
      <mso:qat/>
      <mso:contextualTabs>
        <mso:tabSet idMso="TabSetFormTools">
          <mso:tab idQ="mso:TabFormToolsDesign">
            <mso:group id="mso_c1.1A400E2" label="Steuerelemente" autoScale="true">
              <mso:control idQ="mso:FormControlLabel" visible="true"/>
              <mso:control idQ="mso:FormControlEditBox" visible="true"/>
              <mso:control idQ="mso:FormControlButton" visible="true"/>
              <mso:control idQ="mso:FormControlComboBox" visible="true"/>
              ... weitere Steuerelemente
            </mso:group>
            <mso:group idQ="mso:GroupControlsAccess" visible="false"/>
          </mso:tab>
        </mso:tabSet>
      </mso:contextualTabs>
    </mso:ribbon>
  </mso:customUI>
```

Der Aufbau sieht genauso aus wie der einer benutzerdefinierten Anpassung der Benutzeroberfläche – es gibt lediglich ein paar kleine Unterschiede wie etwa das *mso:* vor jedem Elementnamen.

12.4 Symbolleiste für den Schnellzugriff

Auf ähnliche Art passen Sie die Schnellzugriffsleiste an, also die Leiste mit den kleinen Symbolen oben links neben dem Access-Symbol. Die Schnellzugriffsleiste ergänzt die eigentliche Ribbon-Leiste und enthält im Auslieferungszustand drei Einträge – einen zum Speichern, einen zum Wiederholen und einen zum Rückgängigmachen von Aktionen. Weitere Einträge fügen Sie auf verschiedene Weise hinzu:

» Über das mit der nebenan liegenden Schaltfläche zu öffnende Menü: Damit können Sie die vorhandenen Einträge abwählen und andere hinzufügen (siehe Abbildung 12.11).

» Beliebige Elemente aus den vorhandenen Ribbons fügen Sie der Schnellzugriffsleiste über den Eintrag *Zu Symbolleiste für den Schnellzugriff hinzufügen* des Kontextmenüs des jeweiligen Elements hinzu (siehe Abbildung 12.12).

» Im Bereich *Symbolleiste für den Schnellzugriff* des Dialogs *Access-Optionen* können Sie alle in den Ribbons vorhandenen Befehle in der Übersicht anzeigen und der Schnellstartleiste hinzufügen. Dies macht Sinn, wenn man sich etwa eine immer sichtbare *Drucken*-Schaltfläche auf den Schirm zaubern möchte. Den passenden Dialog (siehe Abbildung 12.13) öffnen Sie entweder über den herkömmlichen Weg (*Datei|Access-Optionen*), über den Eintrag *Weitere Befehle...* des Schnellzugriffsleisten-Menüs oder den Eintrag *Symbolleiste für den Schnellzugriff anpassen...* des Kontextmenüs eines Ribbon-Steuerelements.

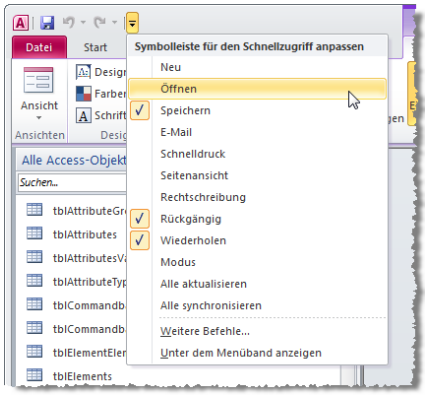


Abbildung 12.11: Die Symbolleiste für den Schnellzugriff und das Menü für ihre Anpassung

Schnellzugriffsleiste positionieren

Die standardmäßig neben dem Access-Symbol befindliche Schnellzugriffsleiste können Sie auch unterhalb des Ribbons platzieren. Dazu wählen Sie etwa den Eintrag *Unter dem*

Menüband anzeigen des Menüs der Schnellzugriffsleiste. Mit dieser Einstellung vergrößert sich der Ribbon-Bereich allerdings um einen zusätzlichen Balken.

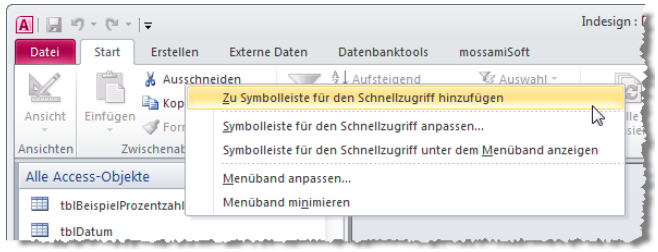


Abbildung 12.12: Per Kontextmenü können Sie beliebige Befehle zur Schnellzugriffsleiste hinzufügen

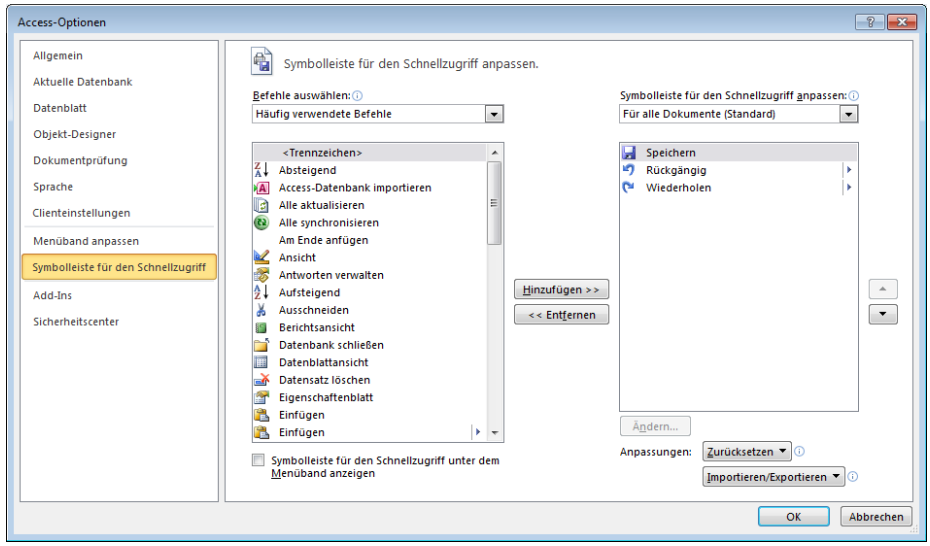


Abbildung 12.13: Dialog zum Anpassen der Schnellzugriffsleiste

Anwendungsspezifische Schnellzugriffsleiste

Sie können der Schnellzugriffsleiste für jede Datenbankanwendung ein eigenes Gesicht verpassen. Dazu klappen Sie im Dialog zum Anpassen der Schnellzugriffsleiste das Kombinationsfeld auf der rechten Seite auf und wählen dort die aktuell geöffnete Datenbank aus.

Access zeigt nun eine leere Liste an, die Sie mit den gewünschten Befehlen füllen können. Alle Befehle, die Sie hier hinzufügen, zeigt Access zusätzlich zu den für alle Datenbanken ausgewählten Befehlen an.

Wenn Sie also nur datenbankspezifische Einträge anzeigen wollten, müssten Sie alle allgemein sichtbaren Einträge der Schnellzugriffsleiste entfernen. Damit wäre allerdings nicht sichergestellt, dass auch der Anwender nur diese Einträge sieht – das hängt von der auf seinem Rechner vorliegenden Konfiguration der Schnellzugriffsleiste ab (die Konfiguration ist in der Registry gespeichert und kann per VBA über die *SetOption*-Methode eingestellt werden).

Um dies zu gewährleisten, kommen Sie um die Definition eines eigenen Ribbons nicht herum.

12.5 Eigene Ribbon-Anpassung erstellen

Bevor Sie loslegen, sollten Sie im Dialog *Verweise* des VBA-Editors (*Extras/Verweise*) einen Verweis auf die Bibliothek *Microsoft Office 14.0 Object Library* anlegen. Anderenfalls kann es bei den folgenden Beispielen zu Fehlermeldungen kommen.

Außerdem beugen wir schon jetzt einem weit verbreiteten Problem vor: Das mühselig programmierte Ribbon erscheint nicht, und eine Fehlermeldung gibt es auch nicht. Dies ist der Fall, wenn Sie die Option *Fehler des Benutzeroberflächen-Add-Ins anzeigen* in den Access-Optionen nicht aktiviert haben.

Prüfen Sie diese Einstellung, die standardmäßig nicht eingeschaltet ist, und aktivieren Sie diese gegebenenfalls (siehe Abbildung 12.14).

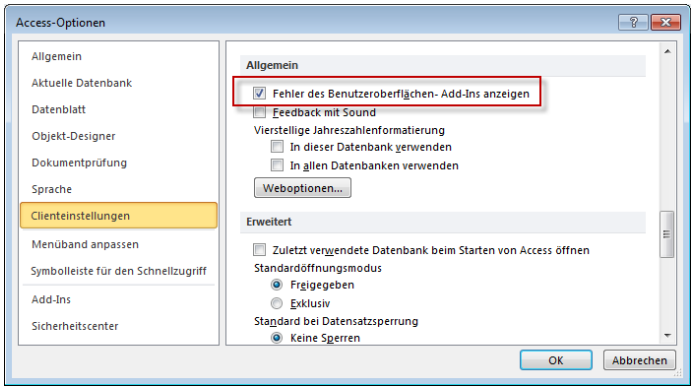


Abbildung 12.14: Diese Einstellung entscheidet, ob Fehler in der *customUI*-Definition angezeigt werden oder stillschweigend übergangen werden.

Sollten Sie diese Option aktiviert haben, liefert Access entsprechende Fehlermeldungen, wenn die XML-Datei nicht der in der Schemadatei vorgegebenen Form entspricht (siehe Abbildung 12.15).

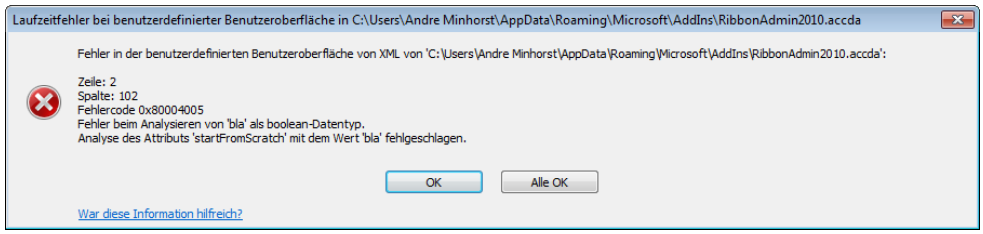


Abbildung 12.15: Fehlermeldung bei ungültigem XML-Dokument

12.5.1 Elemente einer customUI-Anpassung

Im Beispiel von oben haben Sie bereits gesehen, dass die *customUI*-Anpassung durch hierarchisch angeordnete Elemente wie *ribbon*, *tabs*, *tab*, *group* oder *button* realisiert wird. Jedes dieser Elemente besitzt wiederum Attribute, die sich entweder auf das Aussehen oder auf das Verhalten auswirken.

Oben haben Sie bereits das Attribut *label* kennen gelernt: Es legt fest, welche Bezeichnung zum jeweiligen Element angezeigt wird. Es gibt viele weitere Attribute für das Aussehen, mit denen Sie die Größe, das Icon und weitere Einstellungen vornehmen.

Außerdem besitzen die meisten *customUI*-Elemente Attribute, die den Ereigniseigenschaften von Formularen und den enthaltenen Steuerelementen entsprechen. Die Eigenschaft *onAction* eines *button*-Elements etwa erwartet als Wert die Angabe des Namens der VBA-Funktion, die beim Auslösen des *onAction*-Ereignisses aufgerufen werden soll.

ÜBERSICHTSTABELLEN

Übersichten über alle Ribbon-Elemente finden Sie im *.pdf*-Format im Download zu diesem Buch unter *www.access-entwicklerbuch.de/2010/download*.

12.5.2 Die Datei customUI14.xsd

Damit Access mit dem von Ihnen zusammengestellten XML-Dokument auch etwas anfangen kann, prüft es dieses zunächst gegen eine sogenannte Schemadatei namens *customUI14.xsd*. Diese können Sie im Internet herunterladen, aber Sie finden sie auch im Download zu diesem Kapitel. Access braucht sie ohnehin nicht, da das Schema zusätzlich fest in Access verdrahtet ist.

Sie könnten es aber gebrauchen, wenn Sie die XML-Dokumente mit einer Anwendung wie Visual Studio oder XML Notepad 2007 bearbeiten möchten. Wenn Sie das Schema

dann beispielsweise unter einem Pfad wie *C:\Program Files (x86)\Microsoft Visual Studio 10.0\xml\Schemas\1033* ablegen, kann Visual Studio das Schema beim Erstellen eines XML-Dokuments, das auf dieses Schema verweist, für die Bereitstellung der *IntelliSense*-Funktion verwenden.

Die Schemadatei legt in unserem Fall beispielsweise fest, dass alle weiteren Elemente in einem *customUI*-Element enthalten sein müssen.

Der öffnende Teil des Elements enthält einen Hinweis auf die Schema-Datei, die zur Validierung herangezogen werden soll:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    ...
</customUI>
```

Sie können die im *customUI*-Element angegebene Internetadresse schlicht und einfach als Zeichenkette betrachten, die mit dem in der Schema-Datei enthaltenen Ausdruck verglichen wird.

Die *NameSpace*-Bezeichnung hat sich in der finalen Version gegenüber der Beta-Version geändert – nur für den Fall, dass Sie bereits mit der Beta gearbeitet haben und sich wundern, dass die *customUI*-Anpassung nicht mehr funktioniert.

Der *NameSpace* des *customUI* von Access 2007 funktioniert unter Access 2010 immer noch. Andersherum ist dies allerdings nicht der Fall.

customUI

Unterhalb des *customUI*-Elements können Sie eines oder mehrere der folgenden vier Elemente platzieren – je nachdem, welche Bereiche Sie anpassen möchten:

» *backstage*: Einstellen des Backstage-Bereichs

» *ribbon*: Anpassen der eigentlichen Menüleiste, also des Ribbons

» *commands*: Anpassen der durch die Steuerelemente des *customUI* ausgelösten Befehle. Sie können hiermit beispielsweise festlegen, dass die *Speichern*-Schaltfläche eine benutzerdefinierte Funktion auslöst.

» *contextMenus*: Anpassen eingebauter Kontextmenüs etwa durch Hinzufügen eingebauter oder benutzerdefinierter Steuerelemente

backstage

Das *backstage*-Element enthält einen eigenen Bereich der Benutzeroberfläche, nämlich den Backstage-Bereich. Das Buch widmet diesem Bereich ein eigenes Kapitel unter »Backstage« ab Seite 777.

ribbon

Unterhalb des *ribbon*-Elements können Sie drei weitere Bereiche anlegen, um diese anzupassen. Es handelt sich um die folgenden:

» *qat* (Schnellzugriffsleiste): Kleine Leiste rechts neben dem Access-Symbol und standardmäßig über der Ribbon-Leiste, enthält Befehle, die schnell zugänglich sein sollen, und lässt sich per Benutzeroberfläche anpassen (Kontextmenüeintrag *Symbolleiste für den Schnellzugriff anpassen*)

» *tabs*: Übergeordnetes Element der einzelnen *tab*-Elemente. Diese machen das »eigentliche« Ribbon aus, können über die »Registerreiter« ausgewählt werden und enthalten in Gruppen aufgeteilte Steuerelemente

» *contexualTabs*: Übergeordnetes Element solcher *tab*-Elemente, die in Zusammenhang mit bestimmten Objekten eingeblendet werden – etwa das Entwurf-Tab beim Anzeigen eines Formulars in der Entwurfsansicht. Der Vorteil gegenüber herkömmlichen Tabs ist, dass *contextualTabs* farblich hervorgehoben werden und beim Öffnen des Formulars oder Berichts aktiviert werden.

Gegenüber Access 2007 fällt unter Access 2010 das Element *officeMenu* weg. Es wird komplett ersetzt durch das *backstage*-Element, das jedoch in der XML-Struktur eine Ebene höher angesiedelt ist.

commands

Das dritte Element, das Sie unterhalb von *customUI* anlegen können, heißt *commands*. Dieser Bereich enthält ein oder mehrere *command*-Objekte.

Die *command*-Objekte dienen nicht dazu, neue Objekt hinzuzufügen, sondern Sie können damit das *enabled*-Attribut eingebauter Steuerelemente einstellen oder sogar neue Funktionen für eingebaute Elemente definieren. Dazu suchen Sie zunächst die *idMso* des Steuerelements des Ribbons oder Backstage-Bereichs aus

contextMenus

Dieses Element ist in der Version 2010 noch nicht unter Access verfügbar, sondern nur etwa unter Word oder Excel. Dort können Sie auch Kontextmenüs per CustomUI-Anpassung verändern.

12.6 Struktur und Steuerelemente des Ribbons

Nachfolgend lernen Sie die grundlegende Struktur des Ribbons kennen, danach kümmern wir uns um die einzelnen Steuerelemente.

12.6.1 Das ribbon-Element

Dieses Element ist das Hauptelement einer Ribbon-Anpassung. Neben dem Aufnehmen der untergeordneten *tab*-Elemente hat es eine weitere wichtige Aufgabe: Es stellt mit der Eigenschaft *startFromScratch* eine Möglichkeit bereit, die eingebauten Ribbon-Elemente auszublenden. Dazu stellen Sie diese Eigenschaft auf den Wert *true* ein:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon startFromScratch="true"/>
</customUI>
```

Diese *customUI*-Definition würde übrigens schlicht alle Ribbons ausblenden. Backstage, Schnellstartleiste und Kontextmenüs bleiben erhalten.

12.6.2 Das tabs-Element

Das *tabs*-Element hat keine Attribute. Es dient lediglich dazu, die enthaltenen *tab*-Elemente zusammenzufassen.

12.6.3 Das tab-Element

Wenn Sie Registerkartenreiter zum Ribbon hinzufügen möchten, brauchen Sie jeweils ein *tab*-Element. Die *tab*-Elemente landen innerhalb eines öffnenden und eines schließenden *tabs*-Elements. Sie müssen für jedes *tab*-Element die Attribute *id* und *label* festlegen. Die folgenden beiden *tab*-Elemente sorgen für ein Ribbon wie in Abbildung 12.16.

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon>
    <tabs>
      <tab id="tab1" label="Erstes Beispieltab"/>
      <tab id="tab2" label="Zweites Beispieltab"/>
    </tabs>
  </ribbon>
</customUI>
```

Abbildung 12.16: Zwei *tab*-Elemente

12.6.4 Das group-Element

Innerhalb der *tab*-Elemente nehmen Sie mit *group*-Elementen weitere Unterteilungen vor. Genau genommen brauchen Sie mindestens ein *group*-Element, um weitere Steuerelemente auf dem Ribbon platzieren zu können. Das folgende Beispiel fügt zwei Gruppen zum ersten Tab hinzu (siehe Abbildung 12.17):

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon>
    <tabs>
      <tab id="tab1" label="Erstes Beispieltab">
        <group id="grp1" label="Erste Gruppe"/>
        <group id="grp2" label="Zweite Gruppe"/>
      </tab>
      <tab id="tab2" label="Zweites Beispieltab"/>
    </tabs>
  </ribbon>
</customUI>
```

Abbildung 12.17: Ein *tab*-Element mit zwei *group*-Elementen

Das *group*-Element besitzt in Access 2010 gegenüber Access 2007 ein wichtiges neues Attribut: Mit *autoScale* legen Sie fest, ob die enthaltenen Elemente an den verfügbaren Platz angepasst werden.

Außerdem können Sie nun mit dem Attribut *centerVertically* festlegen, dass die in einer Spalte der Gruppe enthaltenen Steuerelemente vertikal zentriert werden. Dies sieht dann beispielsweise wie in Abbildung 12.18 aus.

Abbildung 12.18: Vertikal zentrierte Steuerelemente in einer Gruppe

Flexible Gruppen

Unter Access 2007 wurden einzelne Elemente von Gruppen bereits schrittweise verkleinert, wenn der Platz im Ribbon eng wurde. Dies ist nun in Access 2010 auch in benutzerdefinierten Gruppen möglich. Dazu arbeiten Sie das Attribut *autoScale* mit dem Wert *True* in das *group*-Element ein. Die folgenden Abbildungen zeigen, wie sich die Gruppe mit ihren Elementen bei schwindendem Platzbedarf verändert. Schließlich offenbart sich auch, wofür das bereits seit Access 2007 vorhandene *image*-Attribut des *group*-Elements gut ist: Es wird angezeigt, wenn der Platz nur noch zur Anzeige einer einzigen Schaltfläche zum Aufklappen des Menüs ausreicht.

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui"
loadImage="loadImage">
  <ribbon>
    <tabs>
      <tab idMso="TabHomeAccess" label="Start [mit flexibler Gruppe]">
        <group id="grp1" image="beer_glass" insertAfterMso="GroupSortAndFilter"
            label="Flexible Gruppe" autoScale="true">
          <button id="btn1" image="beer_bottle" label="Schaltfläche 1" size="large"/>
          <button id="btn2" image="beer_glass" label="Schaltfläche 2" size="large"/>
          <button id="btn3" image="cocktail" label="Schaltfläche 3" size="large"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Listing 12.1: Beispiel für eine Gruppe mit flexibler Größe

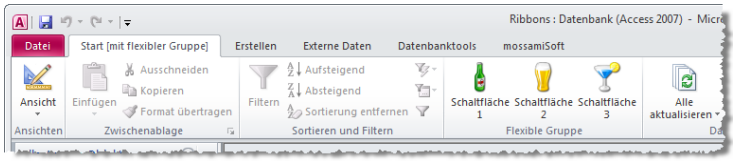


Abbildung 12.19: Schaltflächen mit großen Bildern bei voller Ribbon-Breite, ...

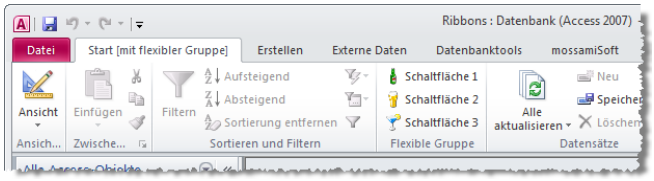


Abbildung 12.20: ... bei etwas knapper werdendem Platz, ...

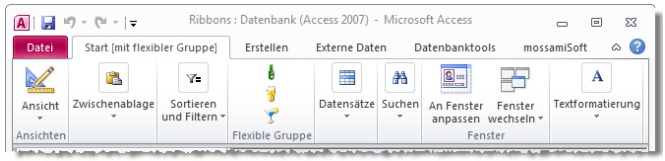


Abbildung 12.21: ... nur noch als Icon und ...

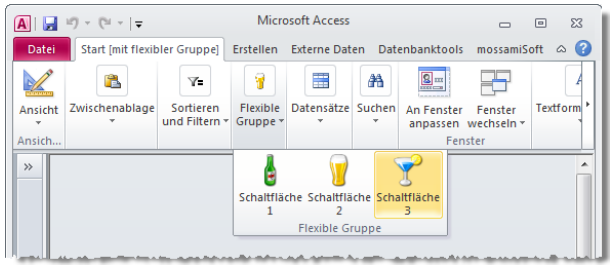


Abbildung 12.22: ... als Menü, das beim Anklicken wieder die komplette Gruppe anzeigt

12.6.5 Das button-Element

Das meistverwendete Element dürfte das *button*-Element sein. Es kann mit oder ohne Bild (*image*, *imageMso*) und klein oder groß (*size*) angezeigt werden.

button-Elemente werden normalerweise innerhalb eines *group*-Elements angelegt (es gibt noch andere Möglichkeiten als Bestandteil weiterer Steuerelemente – dazu später mehr). Im einfachsten Fall fügen Sie einfach nur ein *button*-Element zu einer Gruppe hinzu und legen seine Attribute *id* und *label* fest:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon>
    <tabs>
      <tab id="tab" label="Schaltflächenbeispiele">
        <group id="grp1" label="Einfache Schaltflächen">
          <button id="cmd1" label="Einfache Schaltfläche"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Dieses Beispiel zeigt, wie Sie das *button*-Element innerhalb der übergeordneten Elemente einfügen. Die folgenden Beispiele verzichten auf diese Elemente und zeigen nur noch die *button*-Elemente mit ihren Attributen.

Kleine und große Schaltflächen

Schaltflächen gibt es in der kleinen (normalen) und in einer großen Variante. Drei kleine Schaltflächen nehmen übereinander den Platz einer großen Schaltfläche ein. Damit dies in Abbildung 12.23 deutlich wird, haben wir der Schaltfläche mit dem Attribut *size=large* über *imageMso* ein eingebautes Icon hinzugefügt:

```
<button id="btn1" label="Schaltfläche 1"/>
<button id="btn2" label="Schaltfläche 2"/>
<button id="btn3" label="Schaltfläche 3"/>
<button id="btnGrosseSchaltflaeche" imageMso="CreateForm" label="Große Schaltfläche"
size="large"/>
```

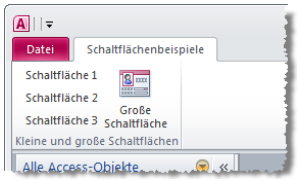


Abbildung 12.23: Kleine und große Schaltflächen im Ribbon

An die *imageMso* für das Bild eines eingebauten Steuerelements gelangen Sie auf die gleiche Weise wie an die *idMso* (siehe »Eingebaute Elemente in benutzerdefinierten Ribbons« ab Seite 762).

12.6.6 Schaltfläche mit Funktion versehen

Bisher löst ein Klick auf die Beispielschaltfläche noch keine Funktion aus, aber das holen Sie nun nach. Erweitern Sie die Definition des *button*-Elements aus dem obigen CustomUI-XML-Dokument beispielsweise für die große Schaltfläche *btnGrosseSchaltflaeche* wie folgt:

```
<button id="btn4" imageMso="CreateForm" label="Große Schaltfläche"
onAction="OnActionCommand" size="large"/>
```

Das *onAction*-Attribut enthält den Aufruf einer öffentlichen VBA-Routine – eine sogenannte Callback-Funktion –, die Sie in einem Standardmodul anlegen. Erstellen Sie also ein neues Standardmodul namens *mdlRibbon* und fügen Sie die folgende Routine hinzu:

```
Sub OnActionCommand(control As IRibbonControl)
    MsgBox "Sie haben die Schaltfläche '" & control.id & "' angeklickt."
End Sub
```

Listing 12.2: Diese Routine wird beim Klick auf die passende Schaltfläche ausgelöst

Die Funktion soll schlicht und einfach ein Meldungsfenster anzeigen, um die einwandfreie Funktion des Aufrufs zu bestätigen – und gleichzeitig noch den Namen des Steuerelements anzuzeigen. Diesen liefert die Eigenschaft *id* des Parameters *control*. Um dies zu testen, müssen Sie die Datenbankanwendung erneut schließen und wieder öffnen und dann auf die neue Schaltfläche klicken.

Möglicherweise erscheint nun eine Meldung wie *Der von Ihnen eingegebene Ausdruck enthält den Namen einer Funktion, die von Microsoft Office Access nicht gefunden werden kann*. Das bedeutet nicht, dass Sie etwas falsch gemacht haben, allerdings liefert diese Meldung auch nicht unbedingt hilfreiche Informationen.

Eine oder mehrere OnAction-Prozeduren

Da Access beim Aufruf einer Callback-Funktion meist einen Verweis auf das aufrufende Steuerelement in Form eines Parameters bereitstellt, brauchen Sie nicht für jedes Steuerelement eine eigene Callback-Funktion anzulegen.

Sie können auch nur eine einzige Callback-Funktion etwa mit dem Namen *OnAction* anlegen und dann innerhalb einer *Select Case*-Anweisung auf das entsprechende Steuerelement reagieren:

```
Public Function Beispielfunktion(ctl As IRibbonControl)
    Select Case ctl.Id
        Case "btnBeispielschaltflaeche1"
            'tu was
        Case "btnBeispielschaltflaeche2"
            'tu was anderes
        Case Else
            MsgBox "Unbekannte Schaltfläche!"
    End Select
    MsgBox "Sie haben die Schaltfläche '" & ctl.Id & "' angeklickt."
End Function
```

Listing 12.3: Auswerten des aufrufenden Steuerelements in einer Select Case-Anweisung

12.7 customUI und VBA

Neben dem *button*-Element gibt es noch viele weitere Steuerelemente im *customUI*. Diese werden später detailliert beschrieben.

Die folgenden Abschnitte zeigen zunächst, wie Sie auf Ereignisse im *customUI* reagieren können, wie Sie die Steuerelemente mit Bildern ausstatten und wie Sie die Eigenschaften von Steuerelementen zur Laufzeit dynamisch anpassen können.

12.7.1 Callback-Funktionen

Callback-Funktionen werden, soweit angegeben, vom Ribbon beim Anlegen oder bei bestimmten Aktionen aufgerufen. Es gibt zwei Arten von Callback-Funktionen: solche, die Werte für Attribute zurückliefern, und jene, die als Reaktion auf Benutzeraktionen aufgerufen werden.

12.7.2 Die get...-Attribute

Jedes Element hat mehrere Attribute, die mit *get...* beginnen. Beim *button*-Element sind dies beispielsweise *getDescription*, *getEnabled*, *getImage*, *getKeytip*, *getLabel*, *getScreenTip*, *getShowImage*, *getShowLabel*, *getSize*, *getSupertip* und *getVisible*. All diesen Attributen können Sie Namen von Callback-Funktionen zuweisen, die beim Erzeugen des Ribbons die passenden Attributwerte ermitteln und zurückgeben. Ihr Einsatz ist dann sinnvoll, wenn Sie beim Erstellen des Ribbon-XML-Dokuments noch nicht wissen, wie der Inhalt eines der zu füllenden Attribute aussieht – sonst könnten Sie ja den passenden Wert auch einfach als Wert des Attributs eintragen.

Wenn Sie etwa dem Attribut *label* erst beim Anlegen des Ribbons (beim Öffnen der Anwendung oder, wenn das Ribbon einem Formular oder Bericht zugeordnet ist, beim Öffnen dieser Objekte) einen bestimmten Wert zuordnen möchten, legen Sie mit *getLabel* eine für diesen Zweck angelegte VBA-Routine namens *GetLabel* fest.

Das Beispiel ist nicht so abwegig und könnte beispielsweise beim Erstellen mehrsprachiger Anwendungen interessant sein. Der Kopf dieser Routine muss einer bestimmten Syntax folgen; eine Zusammenstellung der Syntax aller möglichen Funktionen finden Sie im PDF-Format im Download zu diesem Buch (*Ribbon-Referenz.pdf*). Die Definition einer Schaltfläche im XML-Dokument für das Ribbon sieht etwa wie folgt aus (*get...*-Attribut hervorgehoben):

```
<button id="btnBeispielschaltflaeche" onAction="Beispielfunktion"  image="ribbon.png"
getLabel="GetLabel"/>
```

Damit sich das Attribut beim Laden des Ribbons überhaupt bemerkbar macht, müssen Sie eine passende Routine in einem Standardmodul anlegen. So stellen Sie zumindst schon einmal sicher, dass Access entweder einen Fehler meldet oder, wenn Sie alles richtig gemacht haben, das Attribut mit dem entsprechenden Wert füllt. Eine Routine für das Zurückgeben eines Wertes für das im XML-Element angegebene Attribut sieht so aus:

```
Public Sub GetLabel(ctl As IRibbonControl, ByRef label)
    label = "Beispielbeschriftung"
End Sub
```

Listing 12.4: Diese Callback-Routine liefert den String *Beispielbeschriftung* an das aufrufende Ribbon zurück

Vielleicht wundert es Sie, dass hier von einer Funktion die Rede ist, die aufgerufene Routine aber als Sub-Prozedur ausgeführt ist. Tatsächlich handelt es sich bei den meisten Callback-Routinen um solche Sub-Prozeduren, die eine von der Prozedur zu füllende *ByRef*-Platzhaltervariable bereitstellen.

Sonderfall LoadImage

Etwas anders funktioniert die Routine *LoadImage*. Sie wird wie die in den *get...*-Attributen angegebenen Funktionen beim Anlegen eines Ribbons aufgerufen, ersetzt aber unter Umständen eine ganze Reihe notwendiger *getImage* und *getItemImage*-Attribute:

Sie liest alle in Steuerelementen und deren Unterelementen wie etwa *item*-Elementen enthaltenen *image*-Attribute aus und ruft die unter *loadImages* (*customUI*-Tag) angegebene Callback-Funktion für jedes Image einmal auf. Diese Routine gibt dann passende Objekte mit Verweisen auf die entsprechenden Bilder zurück.

12.7.3 Ereigniseigenschaften

Ribbons bieten Ereigniseigenschaften, wie sie von der VBA-Programmierung von Formularen und Berichten bekannt sind – zumindest, was den Auslöser angeht. Dabei handelt es sich nämlich entweder um einen Klick auf ein Steuerelement, das Drücken einer mit dem Attribut *keytip* angegebenen Tastenkombination oder die Änderung seines Inhalts. Außerdem gibt es noch eine Ereigniseigenschaft, die beim Laden des Ribbons ausgelöst wird:

» *onAction*: Verfügbar für *button*-, *checkBox*-, *dropDown*-, *gallery*- und *toggleButton*-Elemente, wird bei Aktionen wie etwa einem Mausklick oder der Auswahl eines der Einträge ausgelöst.

» *onChange*: Verfügbar für *comboBox*- und *editBox*, wird beim Ändern des Inhalts beziehungsweise der Auswahl eines neuen Eintrags ausgewählt.

» *onLoad*: Wird beim Laden des Ribbons ausgelöst.

Auch für die Ereigniseigenschaften gibt es jeweils eine vorgegebene Syntax, die immer einen Verweis auf das aktuelle Steuerelement und auf den aktuellen Wert übergibt. Auch diese Syntax-Beschreibungen finden Sie in den Übersichtstabellen im Download zu diesem Buch.

12.7.4 Umgang mit Callback-Funktionen

Wenn Sie einmal ein Ribbon mit mehr als nur einem *tab*-Element mit wenigen Unterelementen und passenden Callback-Funktionen bestücken, bekommen Sie möglicherweise Probleme bei der Vergabe der Namen für die Callback-Funktionen. Die Beispiele

verwenden ja meist Routinennamen, die dem Namen der Ereigniseigenschaft bis auf den großgeschriebenen Anfangsbuchstaben gleichen.

Wenn Sie auch so vorgehen möchten, müssen Sie berücksichtigen, dass es auch einmal mehr als ein Steuerelement geben kann, für das Sie etwa eine Routine namens *OnAction* anlegen möchten.

In diesem Fall haben Sie zwei Möglichkeiten:

» Sie verwenden für jede Callback-Routine jedes Elements einen eindeutigen Namen.

» Sie verwenden für jede Ereignisart eine Routine, die jeweils das auslösende Steuerelement auswertet – etwa mit einem *Select Case* über den per Parameter übergebenen Steuerelementnamen.

Callback-Routinen mit eindeutigem Namen

Im ersten Fall weisen Sie den zu erstellenden Routinen Namen zu, die eindeutig sind und dennoch dem Element zugeordnet werden können (sonst entsteht schnell ein sehr unübersichtlicher Berg von Callback-Funktionen).

So können Sie etwa das *label*- oder *id*-Attribut des Steuerelements integrieren und die Bezeichnung *btnBeispiel_onChange* verwenden. Und wenn verschiedene *group*- oder *tab*-Elemente Steuerelemente gleichen Namens enthalten, bauen Sie eben auch noch das *label*- oder *id*-Attribut der übergeordneten Elemente mit ein.

Im Extremfall würde eine Callback-Routine dann beispielsweise *tabMain_grpDateien_btnOeffnen_onAction* heißen.

Eine Callback-Routine für alle Elemente

Bei der zweiten empfehlenswerten Variante legen Sie tatsächlich nur eine Callback-Routine für jedes Ereignis wie etwa *onAction* oder *getDescription* an, deren Name beispielsweise der Attributbezeichnung mit großem Anfangsbuchstaben entspricht (also *OnAction* oder *GetDescription*).

Beim Aufruf einer solchen Funktion wertet diese dann den Parameter *control* aus, der übrigens in allen Callback-Funktionen enthalten ist. Das sieht dann beispielsweise wie im folgenden Listing aus:

```
Public Sub OnAction(ctl As IRibbonControl)
    Select Case ctl.id
        Case "btnBeispielschaltflaeche1"
            'tu was
        Case "btnBeispielschaltflaeche2"
            'tu was anderes
```

```
        Case Else
            MsgBox "Unbekannte Schaltfläche!"
    End Select
End Sub
```

Listing 12.5: Diese Routine wertet aus, von welchem Steuerelement sie aufgerufen wurde, und reagiert mit der für dieses Element vorgesehenen Aktion

Leider kommen Sie damit nicht allzu weit: Wie Sie der Tabelle *Ereigniseigenschaften der customUI-Elemente* aus dem Download entnehmen können, besitzen Callback-Funktionen für gleichnamige Ereigniseigenschaften durchaus nicht immer die gleiche Syntax.

Die obige Routine entspricht etwa der Syntax für ein *button*-Element, die Syntax für die *onAction*-Callback-Routine eines *dropDown*-Elements hingegen sieht so aus:

```
Sub OnAction(control As IRibbonControl, selectedId As String, selectedIndex As Integer)
```

Und da Access einen Fehler meldet, wenn eine Callback-Funktion die falschen Parameter enthält (vorausgesetzt, Sie haben wie oben beschrieben die Fehlerbehandlung für Ribbons aktiviert), müssen Sie zumindest für jede Steuerelementart eine Callback-Funktion mit einem eigenen Namen anlegen. Beim *onAction*-Attribut könnte diese etwa *OnButtonAction*, *OnCheckBoxAction*, *OnDropDownAction*, *OnGalleryAction* oder *OnToggleButtonAction* heißen. Die Klasse (genauer: das Interface) *IRibbonControl* ist übrigens in der *Microsoft Office 14.0 Library* definiert, die Sie deshalb den Verweisen des VBA-Projekts hinzufügen müssen. Sie hat die folgenden Eigenschaften:

» *ID*: Gibt die in der XML-Definition für das Steuerelement festgelegte und obligatorische ID zurück.

» *Context*: Gibt einen Objektverweis auf die Anwendung des Ribbons zurück. Bei Access handelt es sich hierbei um das *Application*-Objekt.

» *Tag*: Gibt eine Zeichenfolge zurück, die Sie optional in der XML-Definition für das Steuerelement angegeben haben. Beispiel:

```
<button id="btn1" label="Beispielschaltfläche" tag="Zusatzinfo"/>
```

Fehler in Callback-Routinen

Falls die Deklaration der Callback-Funktion nicht genau den Vorgaben entspricht, weil Sie fälschlicherweise etwa eine Deklaration wie in Listing 12.5 für ein *dropDown*-Element angegeben haben, dann meldet Access nicht etwa einen Syntaxfehler, sondern bemerkt lapidar: »Access kann die Makro- oder Rückrufaktion 'OnAction' nicht ausführen.«. Die gleiche Meldung erscheint aber auch, wenn die Prozedur gar nicht existiert. Sollten Sie also feststellen, dass die betroffene Prozedur nicht fehlt, kön-

nen Sie von einer fehlerhaften Deklaration der Parameter ausgehen. Access setzt das automatische Debugging von VBA bei allen Callback-Funktionen, die Eigenschaften zurückliefern, außer Kraft. Das bedeutet, dass bei Fehlern im VBA-Code der Callback-Routine nicht die übliche VBA-Fehlermeldung erscheint, sondern die Routine stillschweigend verlassen wird – so, als enthielte die Prozedur zu Beginn ein *On Error Resume Next*. Das macht das Debuggen dieser Routinen schwieriger. Wenn Sie vermuten, dass mit Ihrer Callback-Routine etwas nicht stimmt oder sie nicht das erwartete Ergebnis liefert, fügen Sie dem Code ganz vorne die Anweisung *Stop* hinzu. VBA unterbricht dann an dieser Stelle die Ausführung des Codes und lässt Sie im Einzelschritt (F8) die nächsten Code-Zeilen durchlaufen, den Ablauf untersuchen und Variableninhalte einsehen.

12.7.5 Ribbon-Tab per VBA einstellen

In der Access-Community wurde oft nach einer Möglichkeit gefragt, ein bestimmtes *tab*-Element zu aktivieren. Eine Möglichkeit ist das Öffnen eines Formulars mit einem kontextsensitiven *tab*-Element – mehr dazu später unter »Kontextsensitive Ribbons« ab Seite 770.

Microsoft hat die Hilferufe erhört und mit den Methoden *ActivateTab, ActiveTabMso* und *ActiveTabQ* Mittel präsentiert, mit denen Sie ein bestimmtes Tab im aktuellen Ribbon aktivieren können. Die folgende XML-Definition fügt dem Ribbon drei Tabs hinzu:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui" onLoad="OnLoad_
Tabwechsel">
  <ribbon>
    <tabs>
      <tab id="tab1" label="Tab 1"/>
      <tab id="tab2" label="Tab 2"/>
      <tab id="tab3" label="Tab 3"/>
    </tabs>
  </ribbon>
</customUI>
```

Das *customUI*-Element löst beim Laden die Funktion *OnLoad_Tabwechsel* aus, welche die Objektvariable *objRibbon_Tabwechsel* mit einem Verweis auf das *IRibbonUI*-Objekt füllt:

```
Public objRibbon_Tabwechsel As IRibbonUI

Sub onLoad_Tabwechsel(ribbon As IRibbonUI)
    Set objRibbon_Tabwechsel = ribbon
End Sub
```

13

Backstage

Der Backstage-Bereich ersetzt das Office-Menü von Office 2007 und fügt viele weitere mögliche Elemente zu diesem Bereich hinzu. Im Gegensatz zum Office-Menü erstreckt sich der Backstage-Bereich gleich über das komplette Anwendungsfenster. Dem Entwickler bietet der Backstage-Bereich Funktionen an, die in Access 2003 und älter größtenteils im *Datei*-Menü, aber auch im *Extras*-Menü enthalten waren. Auch gegenüber Access 2007 gibt es einige Verbesserungen – so können Sie nun endlich mehr als nur neun Einträge in der Liste der zuletzt verwendeten Datenbanken anzeigen. Und es kommt noch besser: Es gibt sogar die Möglichkeit, oft benutzte Anwendungen ganz oben in der Liste »anzupinnen«. Sie als Leser dieses Buchs interessieren sich aber vermutlich viel mehr dafür, wie Sie den neuen Backstage-Bereich anpassen und wofür sie ihn einsetzen können. Aufbauend auf dem vorherigen Kapitel zum Thema Ribbon lernen Sie

auf den folgenden Seiten die Struktur und die Steuerelemente des Backstage-Bereichs kennen.

BEISPIELDATENBANKEN

Den Download mit dem Beispielen finden Sie unter *www.acciu.de/aeb2010*. Die Datenbank zu diesem Kapitel heißt *Backstage.accdb*.

Vorab einige Ideen für Einsatzbereiche:

» Optionen-Dialog: Während Sie normalerweise vermutlich ein eigenes Formular als Dialog für die Einstellungen Ihrer Datenbank verwendet haben, können Sie dies nun in den Backstage-Bereich auslagern.

» Informationen über die Anwendung: Wenn Sie eine Anwendung für einen Kunden programmieren, kann es sinnvoll sein, wenn dieser Ihnen schnell Informationen wie etwa den Dateipfad von Front- und/oder Backend oder auch die Versionsnummer mitteilen kann. Der Backstage-Bereich ist ein prima Ort, um solche Daten unterzubringen.

» Neben der Versionsnummer können Sie im Backstage-Bereich natürlich auch die Möglichkeit zum Updaten einer Anwendung unterbringen.

13.1 Elemente des Backstage-Bereichs

In den folgenden Abschnitten lernen Sie die Struktur und die verschiedenen Elemente des Backstage-Bereichs kennen.

ÜBERSICHTSTABELLEN

Übersichten über alle Backstage-Elemente finden Sie im *.pdf*-Format im Download zu diesem Buch unter *www.acciu.de/aeb2010*.

13.1.1 Das backstage-Element

Der Backstage-Bereich ist, dies suggeriert zumindest ein Blick auf die Benutzeroberfläche, ein Teil des Ribbons. Zumindest klicken Sie scheinbar auf ein Ribbon-Tab, um diesen Bereich zu öffnen. In der XML-Definition des CustomUI, also der Benutzeroberfläche von Access, stellt sich dies jedoch anders dar: Direkt unterhalb des *customUI*-Elements gibt es gleich vier weitere Bereiche. Einer davon heißt *ribbon* und umfasst die im vorigen

Kapitel beschriebenen Elemente, ein anderer *backstage*. Wenn Sie eine Anpassung des *backstage*-Bereichs vornehmen möchten, fügen Sie daher ein öffnendes und ein schließendes Element unterhalb des *customUI*-Elements hinzu:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <backstage>
  </backstage>
</customUI>
```

Allein für sich ändert dieses Element nichts. Dies ist erst der Fall, wenn Sie die nachfolgend beschriebenen Elemente hinzufügen. Da das *backstage*-Element sich nicht auf die Benutzeroberfläche auswirkt, gibt es auch keine entsprechenden Attribute. Die einzigen Attribute dieses Elements machen sich vielmehr bemerkbar, wenn Sie den Backstage-Bereich öffnen und schließen: Es handelt sich dabei um die Callback-Attribute *onShow* und *onHide*.

Backstage-Bereich ändern

Wenn Sie Änderungen des Backstage-Bereichs in Form einer entsprechenden XML-Definition zusammengestellt haben, wenden Sie diese genau wie eine Ribbon-Anpassung an. Wie dies funktioniert, erfahren Sie unter »Schnellstart« ab Seite 715.

Ereignisse beim Ein- und Ausblenden

Wie Callback-Funktionen grundsätzlich funktionieren, haben Sie ja bereits in »customUI und VBA« auf Seite 734 erfahren.

Wenn Sie per VBA-Prozedur auf das Ein- und Ausblenden des Backstage-Bereichs reagieren möchten, fügen Sie dem Element je nach Bedarf das Attribut *onShow* oder *onHide* hinzu und geben den Namen der aufzurufenden VBA-Prozedur an:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <backstage onShow="OnShow" onHide="OnHide"/>
</customUI>
```

Fügen Sie in ein Standardmodul wie etwa *mdlRibbon* die beiden folgenden Prozeduren ein:

```
Sub OnShow(contextObject As Object)
    MsgBox "OnShow " & contextObject.Name
End Sub

Sub OnHide(contextObject As Object)
    MsgBox "OnHide " & contextObject.Name
End Sub
```

Dies führt beim Aktivieren beziehungsweise Deaktivieren zur Anzeige der entsprechenden Meldungen. Das mit dem Parameter *contextObject* referenzierte Objekt liefert einen Verweis auf das *Application*-Objekt der jeweiligen Anwendung, in diesem Fall also *Access.Application*.

13.1.2 button- und tab-Elemente

Ganz links im Backstage-Bereich finden Sie zwei Arten von Steuerelementen: Schaltflächen und Registerlaschen. Letztere sehen zwar nicht so aus wie die vom Registersteuerelement bekannten Laschen, aber sie funktionieren genau so. Wo ist der Unterschied? Mit den Schaltflächen lösen Sie schlicht die angegebene Funktion aus, mit einem Klick auf eine Registerlasche zeigen Sie rechts die dafür hinterlegte Registerseite an. Wenn Sie diese beiden Elemente mit der Maus überfahren, hebt Access bei einer Schaltfläche nur den angezeigten Text und gegebenenfalls das Icon durch einen farbigen Hintergrund hervor (siehe Abbildung 13.1), bei einer Registerlasche den kompletten Bereich des Eintrags (siehe Abbildung 13.2).

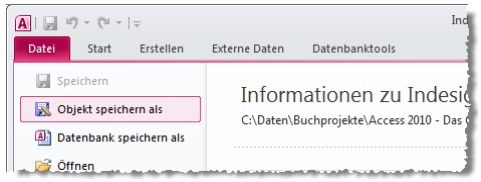


Abbildung 13.1: Schaltfläche für den direkten Aufruf von Funktionen

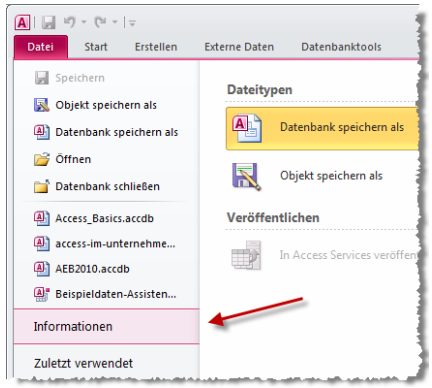


Abbildung 13.2: Registerlasche im Backstage-Bereich

Die XML-Definition des Backstage-Bereichs nehmen Sie grundsätzlich wie bei den übrigen Elementen des *customUI* wie etwa beim Ribbon vor. Sie benötigen eine XML-Datei mit einem *customUI*-Element. Gleich darunter definieren Sie den Backstage-Bereich

mit dem *backstage*-Element und den darin enthaltenen Steuerelementen *button* und *tab*. Die Erweiterung des Backstage-Bereichs um eine Schaltfläche und eine Registerlasche geschieht so:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <backstage>
    <button id="btn1" label="Button 1"/>
    <tab id="tab1" label="Tab 1" />
  </backstage>
</customUI>
```

Das *button*-Element im linken Backstage-Bereich besitzt einige Attribute weniger als das *button*-Element im Ribbon. Dies liegt beispielsweise daran, dass Sie diese Schaltfläche nicht in verschiedenen Größen anzeigen können. Für ein *tab*-Element im Backstage-Bereich können Sie zwei Beschriftungen festlegen: eine für die Registerlasche (*label*) und eine Überschrift für die Registerseite (*title*).

```
<tab id="tab1" title="Beispieltab (title)" label="Beispieltab (label)">
```

Ein Beispiel finden Sie weiter unten in Abbildung 13.3.

13.1.3 firstColumn und secondColumn: Spalten einer Registerseite

Dem *tab*-Element können Sie wiederum zwei Elemente unterordnen – das *firstColumn*- und das *secondColumn*-Element. Diese Elemente besitzen weder eigene Attribute noch ändern sie das Erscheinungsbild des Backstage-Bereichs, wenn Sie keine untergeordneten Elemente hinzufügen.

Sie können jedoch über die Attribute des übergeordneten *tab*-Elements einige Einstellungen vornehmen – zum Beispiel zum Festlegen der Spaltenbreiten:

columnWidthPercent: Gibt an, welchen Anteil der verfügbaren Breite die linke Spalte einnehmen soll (Zahl von 0 bis 100)

firstColumnMinWidth: Gibt die minimale Breite der linken Spalte in Pixel an.

firstColumnMaxWidth: Gibt die maximale Breite der linken Spalte in Pixel an.

secondColumnMinWidth: Gibt die minimale Breite der rechten Spalte in Pixel an.

secondColumnMaxWidth: Gibt die maximale Breite der rechten Spalte in Pixel an.

Dabei ist Folgendes zu beachten: Wenn die verfügbare Breite größer oder gleich der unter *firstColumnMaxWidth* und *secondColumnMaxWidth* angegebenen Breite ist, werden beide Spalten in der maximalen Breite angezeigt. Ist die gesamte Breite kleiner als die

Summe der beiden Spaltenbreiten, werden diese entsprechend dem unter *columnWidthPercent* angegebenen Verhältnis verkleinert.

Die unter *firstColumnMinWidth* und *secondColumnMinWidth* angegebenen Werte begrenzen die Breite des Access-Fensters nach unten. Ein mit diesen Attributen versehenes *tab*-Element wird beispielsweise so definiert:

```
<tab id="tab1" title="Beispieltab (title)" label="Beispieltab (label)"
columnWidthPercent="40" firstColumnMinWidth="100" firstColumnMaxWidth="300"
secondColumnMinWidth="100" secondColumnMaxWidth="300">
...
</tab>
```

Wichtig ist auch, dass Sie auch ein einspaltiges Layout erzeugen können. Dazu brauchen Sie einfach nur das Element *firstColumn* zu verwenden – das Element *secondColumn* lassen Sie weg.

13.1.4 group

Unterhalb von *firstColumn* und *secondColumn* können Sie weitere Elemente anlegen. Mit einem oder mehreren *group*-Elementen teilen Sie die Spalte in mehrere Bereiche auf.

Die *group*-Elemente werden oben angeordnet und nehmen nur den Platz ein, den die enthaltenen Elemente benötigen.

Mehrere Gruppen in einer Spalte

Im folgenden Beispiel soll die linke Spalte drei Gruppen anzeigen, die rechte eine. Das Attribut *style* legt mit den drei Werten *normal*, *warning* und *error* verschiedene Layouts fest (siehe Abbildung 13.3).

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <backstage>
    <tab id="tab1" title="Beispieltab (title)" label="Beispieltab (label)"
        columnWidthPercent="40" firstColumnMinWidth="100" firstColumnMaxWidth="300"
        secondColumnMinWidth="100" secondColumnMaxWidth="300">
      <firstColumn>
        <group id="grp3" label="Gruppe mit style = warning" style="warning"/>
        <group id="grp2" label="Gruppe mit style = normal" style="normal"/>
        <group id="grp1" label="Gruppe mit style = error" style="error"/>
      </firstColumn>
      <secondColumn>
        <group id="grp4" label="Gruppe ohne style"/>
      </secondColumn>
```

```
      </tab>
    </backstage>
  </customUI>
```

Listing 13.1: Code des Beispiels *Tab mit zwei Spalten und Gruppen*

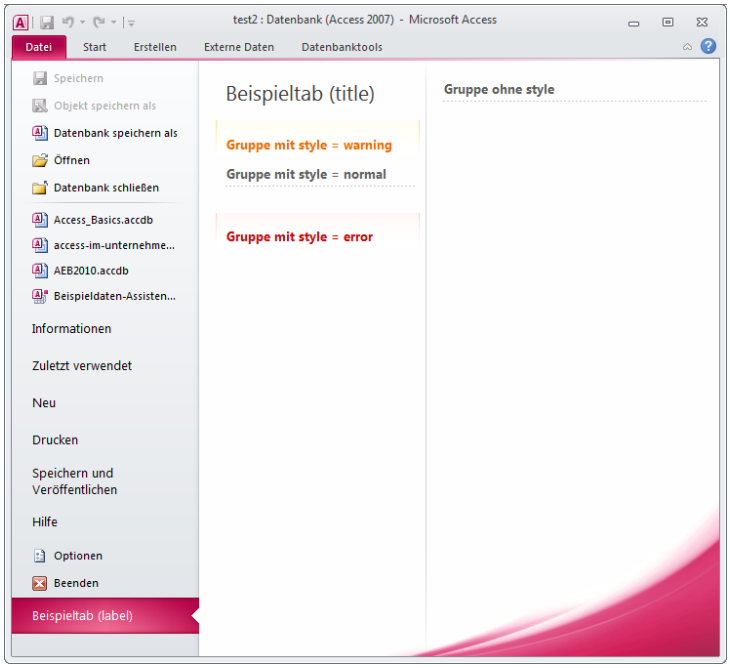


Abbildung 13.3: Backstage-Bereich mit verschiedenen Gruppen

Sie können einer Gruppe außerdem einen weiteren Text mitgeben. Diesen legen Sie mit dem Attribut *helperText* des jeweiligen *group*-Elements fest. Der Text erscheint unterhalb des unter *label* angegebenen Textes (siehe Abbildung 13.4).

Wenn Sie nur einen Text anzeigen möchten, können Sie entweder das Attribut *label* leer lassen oder das Attribut *showLabel* auf *false* einstellen. Den *helperText* geben Sie beispielsweise wie folgt ein:

```
<group id="grp3" label="Gruppe mit style = warning" helperText="Dieser Text steht im
Attribute helperText." style="warning"/>
```

13.1.5 primaryItem, topItems und bottomItems

Unterhalb der *group*-Elemente gibt es wiederum drei Orte, an denen Sie die eigentlichen Steuerelemente unterbringen können.

Abbildung 13.4: *group*-Elemente mit *helperText* (links) und mit *helperText* und ohne *label* (rechts).

primaryItem

Unterhalb von *primaryItem* können Sie nur ein Element unterbringen, und zwar ein *button*- oder ein *menu*-Element:

```
<tab id="tab3" label="Primary-, Top- und Bottom-Items">
  <firstColumn>
    <group id="grp6" label="Gruppe mit primaryItem" style="error">
      <primaryItem>
        <button id="btn8" label="Button im primaryItem"/>
      </primaryItem>
    </group>
  </firstColumn>
</tab>
```

Diese Definition zeigt den Bereich aus Abbildung 13.5 an.

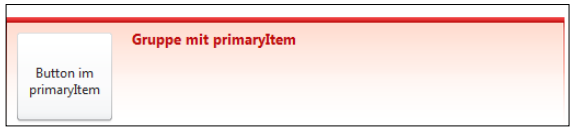


Abbildung 13.5: Ein *button*-Element unterhalb des *primaryItem*-Elements

Statt einer einfachen Schaltfläche können Sie hier auch ein Menü unterbringen, das eines oder mehrere Steuerelemente der Typen *button*, *checkbox*, *toggleButton* oder weitere *menu*-Elemente enthält:

```
<group id="grp10" label="Gruppe mit Menü im primaryItem" style="error">
  <primaryItem>
    <menu id="mnu1" label="Menü">
      <menuGroup id="mnugrp1" label="Menügruppe">
```

```
        <checkBox id="chk1" label="Checkbox im Menü"/>
        <button id="btn11" label="Button im Menü"/>
      </menuGroup>
    </menu>
  </primaryItem>
</group>
```

Der mit dieser Definition festgelegte Bereich sieht wie in Abbildung 13.6 aus. Das *menuGroup*-Element enthält dabei eine Überschrift und kann so mehrere Einträge zusammenfassen.

Abbildung 13.6: Ein Menü unterhalb des *primaryItem*-Elements

Sie können das Label eines *menuGroup*-Elements auch weglassen:

```
<menuGroup id="mnugrp2">
  <button id="btn12" label="Button in Menügruppe ohne Label"/>
</menuGroup>
```

Statt einer farbig hinterlegten Überschrift zeigt der Backstage-Bereich dann eine gestrichelte Trennlinie im Menü an (Abbildung 13.7).

Abbildung 13.7: Menüunterteilung ohne Beschriftung

topItems

Wie die Bezeichnung *topItems* gegenüber *primaryItem* schon sagt, lassen sich hier mehrere Steuerelemente unterbringen und nicht nur eines. Die unterhalb des *topItems*-Elements untergebrachten Steuerelemente erscheinen unter dem *label*- und gegebenenfalls dem *helperText*-Element. Wenn gleichzeitig ein *primaryItem*-Element vorhanden

ist, werden nicht nur *label* und *helperText* der Gruppe, sondern auch die unter *topItems* angegebenen Steuerelemente entsprechend nach rechts verschoben.

Im *topItems*-Bereich können Sie die folgenden Elemente unterbringen: *button*, *checkBox*, *comboBox*, *dropDown*, *editBox*, *groupBox*, *hyperlink*, *imageControl*, *labelControl*, *layoutContainer* und *radioGroup*. Einige dieser Elemente zeigt Abbildung 13.8, einige sind in Access 2010 neu hinzugekommen. Die Beschreibung dieser Elemente und weitere Beispiele finden Sie weiter unten.

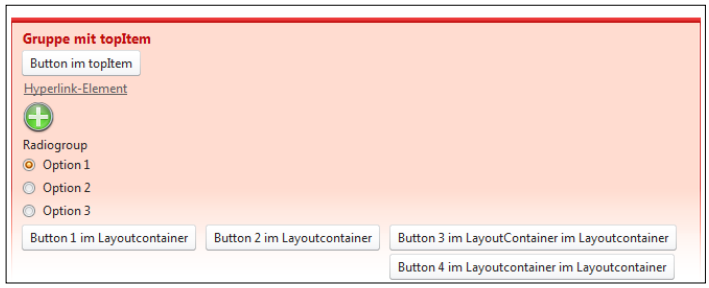


Abbildung 13.8: Steuerelemente im topItems-Bereich

bottomItems

Die unterhalb des Elements *bottomItems* befindlichen Elemente werden linksbündig unter den anderen Elementen angezeigt (siehe Abbildung 13.9):

```
<group id="grp9" label="Gruppe mit Primary- und BottomItem" style="warning">
  <primaryItem>
    <button id="btn21" label="Button im PrimaryItem"/>
  </primaryItem>
  <bottomItems>
    <button id="btn20" label="Button im BottomItem"/>
  </bottomItems>
</group>
```

Abbildung 13.9: *primaryItem*- und *bottomItems*-Elemente mit Schaltflächen

Kommt auch noch ein *topItems*-Bereich hinzu, sieht der Aufbau wie in Abbildung 13.10 aus:

14

Debugging und Fehlerbehandlung

Die in der Kapitelüberschrift aufgeführten Themen sind eng miteinander verknüpft und können über Erfolg oder Scheitern einer Anwendung entscheiden. Grundsätzlich greifen alle drei Mechanismen erst, wenn etwas im Argen liegt – allerdings an unterschiedlichen Stellen. Die klassischen Debugging-Mechanismen von VBA helfen vorrangig bei der Entwicklung einer Anwendung und liefern beispielsweise an Stellen Anhaltspunkte, an denen der Code unerwartete Ergebnisse liefert.

Fehlerbehandlung und Fehlerdokumentation arbeiten sowohl bei der Entwicklung als auch beim Einsatz der Anwendung Hand in Hand.

Die Fehlerbehandlung sorgt dafür, dass eine Anwendung auch beim Auftreten eines Fehlers stabil weiterläuft, während die Fehlerdokumentation für das Notieren von Fehlern sorgt, damit der Entwickler diesen schnell und unkompliziert auf den Grund gehen

kann. Letzteres macht sich im Übrigen sowohl während der Entwicklung gut als auch während des Einsatzes beim Benutzer.

BEISPIELDATENBANKEN

Den Download mit dem Beispielen finden Sie unter dem Link *www.acciu.de/aeb2010*. Die Datenbank zu diesem Kapitel heißt *Debugging.accdb*.

14.1 Fehlerarten

Die bei der Programmierung und beim Einsatz einer Anwendung auftretenden Fehler lassen sich in mehrere Kategorien einteilen. Der Schwierigkeitsgrad beim Auffinden der Fehler und die dazu benötigten Werkzeuge sind je nach Kategorie unterschiedlich.

14.1.1 Syntaxfehler

Syntaxfehler sind vermutlich am leichtesten zu finden, weil die Entwicklungsumgebung sich selbst darum kümmert. Dabei handelt es sich um unvollständige Code-Konstrukte, fehlende beziehungsweise überzählige Klammern oder Anführungszeichen oder die Verwendung von Schlüsselwörtern an Stellen, an denen sie nicht erlaubt sind.

Wenn Sie beispielsweise eine *If Then*-Verzweigung einfügen und vor dem Sprung in die nächste Zeile zwar das Schlüsselwort *If* und die Bedingung, aber nicht das Schlüsselwort *Then* eingegeben haben, teilt Access Ihnen dies unmissverständlich mit (siehe Abbildung 14.1).

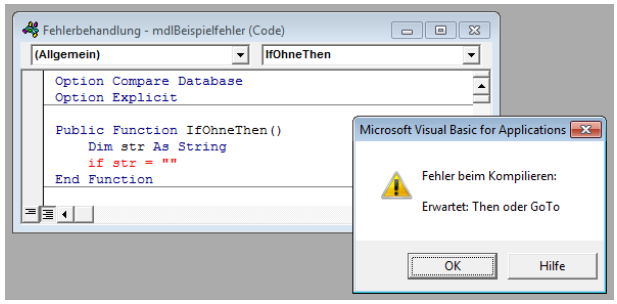


Abbildung 14.1: Syntaxfehler werden an Ort und Stelle geahndet

Andere Syntaxfehler deckt die Entwicklungsumgebung nicht umgehend auf. Meist handelt es sich dabei um fehlende, falsche oder überflüssige Zeilen. Bleiben Sie beim obigen Beispiel: Wenn Sie die erste Zeile der *If Then*-Anweisung vervollständigen,

ist Access zunächst zufrieden, auch, wenn Sie dann an einer anderen Stelle weiterprogrammieren, ohne die abschließende *End If*-Anweisung hinzuzufügen und diese dann vergessen. Darum kümmert sich Access aber später, und zwar spätestens beim Aufrufen der Prozedur. Die Meldung aus Abbildung 14.2 zeigt exakt an, vor welcher Zeile sie spätestens die fehlende *End If*-Anweisung vermisst.

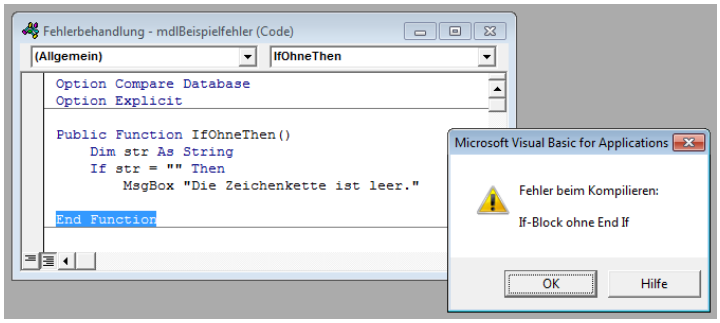


Abbildung 14.2: Unvollständige Code-Konstrukte findet Access erst beim Kompilieren oder Ausführen

Das gleiche Ergebnis erhalten Sie, wenn Sie das VBA-Projekt mit dem Menübefehl *Debuggen/Kompilieren von <Projektname>* kompilieren. Das regelmäßige Kompilieren des VBA-Projekts während der Entwicklung sollten Sie sich zur Angewohnheit machen (etwa jeweils nach Erstellung einer Prozedur). Sie kommen so Fehlern schneller auf die Spur, als wenn der Compiler sich erst beim Testdurchlauf der Anwendung meldet.

Das Ausführen einer Routine erreichen Sie übrigens am schnellsten durch die Betätigung der Taste *F5*, während sich die Einfügemarke innerhalb der zu startenden Routine befindet. Das klappt allerdings auch nur in Routinen ohne Übergabeparameter, und das auch nur in Standardmodulen.

Syntaxprüfung und weitere Optionen

Die Syntaxprüfung während der Eingabe ist eine nützliche Funktion, die Sie auf Fehler aufmerksam macht, während Sie gedanklich noch in der betroffenen Routine stecken. Würden solche Fehler erst nach dem Schreiben einiger Zeilen Code oder gar ganzer Routinen beim nächsten Debuggen oder Test bemerkt, müssten Sie sich zunächst in die jeweiligen Codezeilen hineindenken.

Manch einer wird durch diese Funktion aber in seinem Tatendrang gestört. Die Hinweise auf Syntaxfehler können natürlich auch einmal nervig sein – etwa wenn Sie in einer *If Then*-Anweisung eine Variable verwenden möchten, deren Namen Sie gerade nicht im Kopf haben. Wenn Sie dann schnell nach oben blättern, um den Variablennamen zu ermitteln, macht Ihnen die Syntaxprüfung einen Strich durch die Rechnung.

Deshalb wird es manchen freuen, dass man diese »Live«-Syntaxprüfung abschalten kann, und zwar im *Optionen*-Dialog der VBA-Entwicklungsumgebung (Menüeintrag *Extras/Optionen*, Option *Automatische Syntaxüberprüfung*). Dadurch bleiben allerdings nur die Fehlermeldungen aus, fehlerhafte Zeilen sind aber weiterhin in roter Schrift markiert – Sie verpassen also nichts.

Variablendeklaration erzwingen

Ein weiterer Garant für nicht bemerkte Fehler ist die fehlende Deklaration von Variablen. Im einfachsten Fall weist man einer nicht explizit deklarierten Variablen, die eigentlich Zahlen enthalten sollte, einen Text zu und eckt damit irgendwo an. Damit das nicht passiert, sollten Sie tunlichst alle verwendeten Variablen deklarieren. Da selbst erfahrene Programmierer dies einmal vergessen, können Sie mit der folgenden Zeile im Modulkopf nachhelfen:

```
Option Explicit
```

Access meldet sich anschließend bei jeder Verwendung einer nicht deklarierten Variablen. Damit Sie auch das Setzen dieser Zeile nicht mal vergessen, können Sie im Dialog aus Abbildung 14.3 die Option *Variablendeklaration erforderlich* aktivieren. Access fügt die *Option Explicit*-Anweisung dann automatisch in jedes neue Modul ein.

Es ist grundsätzlich fragwürdig, dass Microsoft diese Einstellung nicht standardmäßig aktiviert.

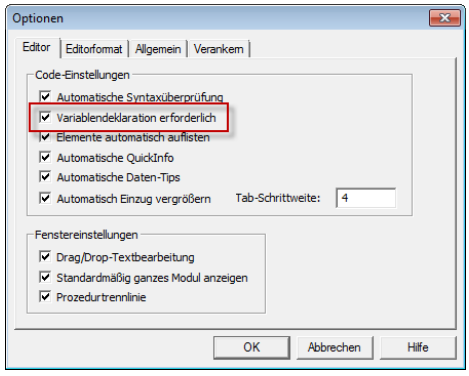


Abbildung 14.3: Optionen für die VBA-Entwicklungsumgebung

14.1.2 Laufzeitfehler

Laufzeitfehler werden nicht durch den Compiler von Access entdeckt. Sie fallen erst bei der Durchführung der entsprechenden Routine auf. Typische Beispiele für Laufzeitfehler sind Divisionen durch 0, Zuweisung von Werten falschen Datentyps (besonders oft in

Zusammenhang mit dem Wert *Null*) oder durch Verweise auf Objekte, die nicht instanziert wurden oder aus anderen Gründen nicht zur Verfügung stehen.

Im Falle eines Laufzeitfehlers zeigt die VBA-Entwicklungsumgebung eine Fehlermeldung an, die eine Fehlernummer und eine Fehlerbeschreibung enthält.

Laufzeitfehler sind natürlich meist unerwünscht. Manchmal lässt sich ein Laufzeitfehler allerdings auch für die Ablaufsteuerung innerhalb einer Routine zweckentfremden oder liefert gar eine performantere oder einfacher zu implementierende Variante.

Ein Beispiel ist das Instanzieren von OLE-Servern: Hier versucht man zunächst, ein bestehenes Objekt zu referenzieren. Ist keine Instanz vorhanden, löst dies einen Fehler aus. Dieser Fehler wird allerdings abgefangen und in der Fehlerbehandlungsroutine wird eine neue Instanz des benötigten Objekts erstellt.

Mehr dazu erfahren Sie unter »Funktionale Fehlerbehandlung« ab Seite 818.

14.1.3 Logische Fehler

Logische Fehler werden gar nicht von Access gemeldet. Sie schlagen sich vielmehr in falschen Ergebnissen bei der Ausführung von Routinen nieder – entweder geben diese unerwartete Werte zurück, speichern falsche Daten in den Tabellen der Datenbank oder machen auf ähnliche Weise auf sich aufmerksam – früher oder später.

Logische Fehler aufdecken

Beim Aufdecken logischer Fehler sind gute Debugging-Möglichkeiten das A und O. Wenn Sie die Möglichkeit haben, Ein- und Ausgangswerte von Funktionen zu prüfen oder gar die Inhalte von Variablen innerhalb der Routinen zu kontrollieren, lassen sich die meisten Fehler mit mehr oder weniger Schritten eingrenzen und schlussendlich auch finden. Die VBA-Entwicklungsumgebung liefert einige sinnvolle Werkzeuge für das Debugging, wie der folgende Abschnitt zeigt.

14.2 Debugging in der VBA-Entwicklungsumgebung

Die VBA-Entwicklungsumgebung bietet einige Werkzeuge zum Debuggen von Code. *»Debugging«* fasst eigentlich alle Tätigkeiten zur Fehlerbehebung zusammen, wozu das Auffinden, Diagnostizieren und Beheben von Fehlern – oder eben »Bugs« – gehört.

Interessant ist in diesem Zusammenhang die Herkunft der Bezeichnung »Bug«: Laut der Internetseite *http://www.waterholes.com/~dennette/1996/hopper/bug.htm* stammt dieser Begriff aus der Zeit des Zweiten Weltkriegs und bezieht sich auf eine Motte, die ein Relais des Computers Mark I bei der Arbeit behinderte.

Diese Motte wurde damals als (toter) Beweis in ein Logbuch eingeklebt, das sich zurzeit in der Smithsonian Institution in Washington befindet.

Die Debugging-Werkzeuge der VBA-Entwicklungsumgebung konzentrieren sich auf das Melden und Anzeigen von Laufzeitfehlern und das Nachverfolgen von Variablen-Werten, wobei dazu unterschiedliche Möglichkeiten zur Verfügung stehen. Nachfolgend lernen Sie die einzelnen Werkzeuge kennen.

14.2.1 Die Debuggen-Symbolleiste

Die Debuggen-Symbolleiste liefert die Möglichkeit, alle nachfolgend beschriebenen Debugging-Tools per Mausklick aufzurufen und den Ablauf von Routinen durch das Starten und Beenden, das Setzen von Haltepunkten oder das schrittweise Durchlaufen zu steuern (siehe Abbildung 14.4).

Abbildung 14.4: Symbolleiste zum Aufrufen der Debugging-Funktionen

14.2.2 Das Direktfenster

Das wichtigste, weil flexibelste Element der VBA-Entwicklungsumgebung ist das Direktfenster. Sie können damit folgende Aktionen durchführen:

» Aufrufen von *Function*- und *Sub*-Prozeduren (mit oder ohne Parameter)

» Aufrufen von Standardfunktionen und Ausgabe der Ergebnisse mit der *Debug.Print*-Anweisung

» Untersuchen von Formularen und Berichten und deren Eigenschaften während der Anzeige in der Formular- beziehungsweise Berichtsansicht durch Ausgeben von Werten mit der *Debug.Print*-Anweisung

» Ausgabe von Werten der Variablen einer gerade angehaltenen Routine sowie Anpassen von Variablen

» Ausgabe von *Debug.Print*-Anweisungen, die Sie an beliebigen Stellen im Quellcode platzieren können

Sollte das Direktfenster einmal nicht sichtbar sein, können Sie es mit der Tastenkombination *Strg + G* sichtbar machen.

Das gilt übrigens auch für die Access-Entwicklungsumgebung: Dort führt diese Tastenkombination zum Öffnen der VBA-Entwicklungsumgebung und zum Aktivieren des Direktfensters.

Das Direktfenster sieht wie in Abbildung 14.5 aus und kann wie eine flexiblere Variante der Eingabeaufforderung verwendet werden. Geben Sie dort einfach die gewünschten Ausdrücke ein und lassen Sie das Ergebnis ausgeben.

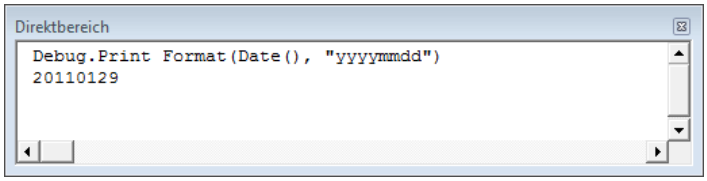


Abbildung 14.5: Testen von Funktionen im Direktfenster

Debug.Print und die Kurzformen

Die vollständige Anweisung mit Angabe des *Debug*-Objekts müssen Sie nur außerhalb des Direktfensters – also innerhalb von Modulen – verwenden. Im Direktfenster reicht die abgekürzte Form *?*.

14.2.3 Haltepunkte

Zum punktuellen Untersuchen des VBA-Codes und aktueller Zustände von Variablen können Sie im VBA-Code Haltepunkte setzen. Das ist zum Beispiel sinnvoll, wenn Sie ahnen, dass an einer bestimmten Stelle im Quellcode etwas nicht so läuft, wie es laufen sollte, und Sie sich die in der Umgebung vorhandenen Variablen einmal näher ansehen möchten.

In solch einem Fall setzen Sie einen Haltepunkt, indem Sie einfach in die graue Leiste links neben dem Quellcode klicken. Auf die gleiche Weise entfernen Sie den Haltepunkt auch wieder. Die zweite Methode zum Setzen eines Haltepunktes ist das Positionieren der Einfügemarke auf der gewünschten Zeile und die Betätigung der Taste *F9* – mit dieser Taste entfernen Sie vorhandene Haltepunkte auch wieder.

Haben Sie wie in Abbildung 14.6 einen Haltepunkt gesetzt, müssen Sie die Routine nur noch starten. Sobald die Abarbeitung an diesem Punkt angelangt ist, wird die Ausführung unterbrochen und die aktuelle Zeile gelb hinterlegt. Sie können dann durch Überfahren mit dem Mauszeiger die Werte der im Code enthaltenen Ausdrücke anzeigen lassen (siehe Abbildung 14.7).

Alternativ können Sie sich den Inhalt von Ausdrücken auch im Direktfenster ausgeben lassen – etwa, wenn Sie den Wert für einen anderen Zweck weiterverwenden möchten. Um den Wert aus Abbildung 14.6 im Direktfenster auszugeben, benutzen Sie den folgenden Ausdruck:

```
Debug.Print tdf.Fields.Count
```

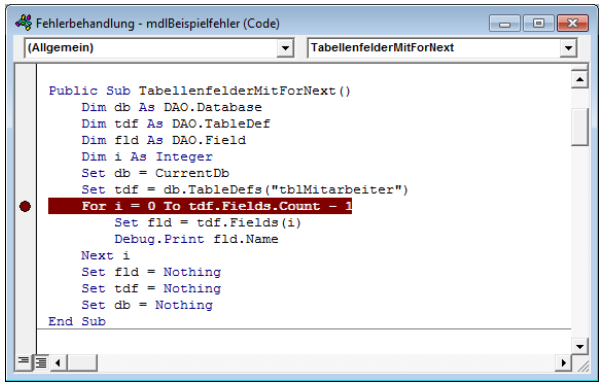


Abbildung 14.6: Ein Haltepunkt in einer VBA-Prozedur

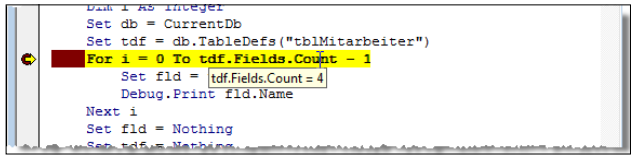


Abbildung 14.7: Anzeige der Werte von Ausdrücken im Haltemodus

Von Anweisung zu Anweisung

Wenn Sie nach dem Erreichen des Haltepunktes Schritt für Schritt durch den Code laufen möchten, verwenden Sie dazu die Schaltfläche *Einzelschritt* (*F8*).

Von Haltepunkt zu Haltepunkt

Wenn Sie mehrere Haltepunkte in der Anwendung gesetzt haben, können Sie die Abarbeitung des Codes nach Erreichen eines Haltepunktes mit der Taste *F5* fortsetzen. Der Code läuft dann bis zum nächsten Haltepunkt weiter, sofern zwischendurch nichts Unvorhergesehenes passiert.

Den Ablauf einer Routine im Haltemodus beeinflussen

Der Haltemodus ist relativ mächtig: Sie können etwa den Zeiger, der die aktuell auszuführende Zeile markiert, verschieben und damit in den Programmablauf eingreifen.

Damit können Sie beispielsweise einen Laufzeitfehler überbrücken, wenn Sie diesen erst später beheben und zunächst die Routine weiter untersuchen möchten, oder auch im Ablauf zurückspringen.

Dies ist allerdings mit Vorsicht zu genießen, da somit beispielsweise auch Zählervariablen wiederholt erhöht werden können.

Haltepunkte fixieren

Die am linken Rand des Code-Moduls markierten Haltepunkte sind nicht lange haltbar – wenn Sie die Access-Anwendung einmal geschlossen haben und wieder öffnen, sind die Haltepunkte verschwunden. Wenn Sie einen Haltepunkt benötigen, der dauerhafter haltbar ist, fügen Sie vor der Zeile, die Sie sonst mit einem Haltepunkt markiert hätten, die Anweisung *Stop* ein. Alternativ verwenden Sie Debug.Assert, wenn der Code nur unter einer bestimmten Bedingung angehalten werden soll:

```
Debug.Assert Not (a=1)
```

Dies hält den Code in dieser Zeile an, wenn *a* gleich *1* ist.

14.2.4 Die Aufrufliste

Manchmal kann es wichtig sein, wenn Sie wissen, über welche Prozeduren Sie zu der aktuellen Routine gelangt sind. Das ist vor allem interessant, wenn Sie einen Haltepunkt in einer Routine platziert haben, die von mehreren anderen Prozeduren aus aufgerufen wird. Die Aufrufliste zeigt nur Informationen an, wenn der Ablauf durch einen Haltepunkt unterbrochen wurde (siehe Abbildung 14.8).

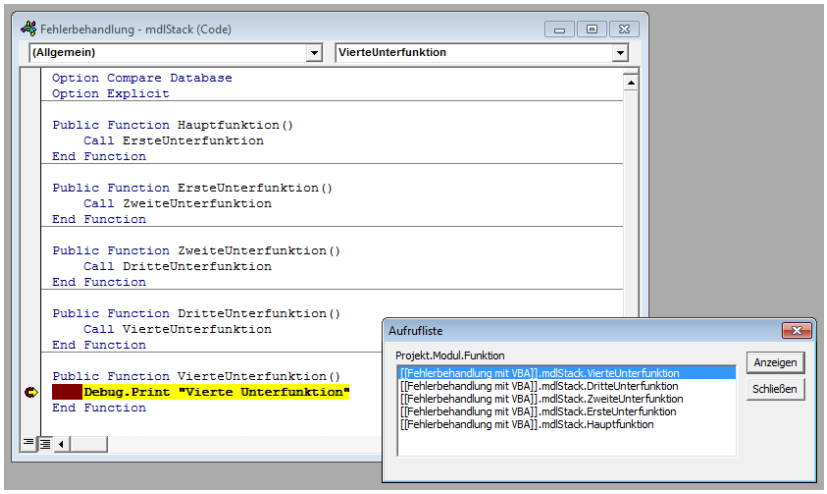


Abbildung 14.8: Die Aufrufliste zeigt die aufgerufenen Routinen in der Hierarchie an

14.2.5 Ausdrücke überwachen

Das Fenster *Überwachungsausdrücke* ermöglicht die Überwachung bestimmter Ausdrücke über den gesamten Ablauf einer Anwendung. Sie zeigen dieses Fenster an, indem Sie den Menüeintrag *Ansicht/Überwachungsfenster* auswählen. Um einen

Ausdruck zur Überwachung anzulegen, können Sie diesen beispielsweise im aktuellen Code-Fenster markieren und dann im Kontextmenü den Eintrag *Überwachung hinzufügen* auswählen.

Der Dialog *Überwachung hinzufügen* (siehe Abbildung 14.9) enthält dann direkt den gewünschten Ausdruck. Hier gibt es nun drei Möglichkeiten:

» Der Ausdruck soll einfach nur überwacht werden.

» Der Ablauf soll unterbrochen werden, wenn der Ausdruck einen bestimmten Wert annimmt. In diesem Fall müssen Sie den Ausdruck noch um das entsprechende Kriterium erweitern, etwa *fld.Name ="MitarbeiterID"*.

» Der Ablauf soll unterbrochen werden, sobald der Wert geändert wurde.

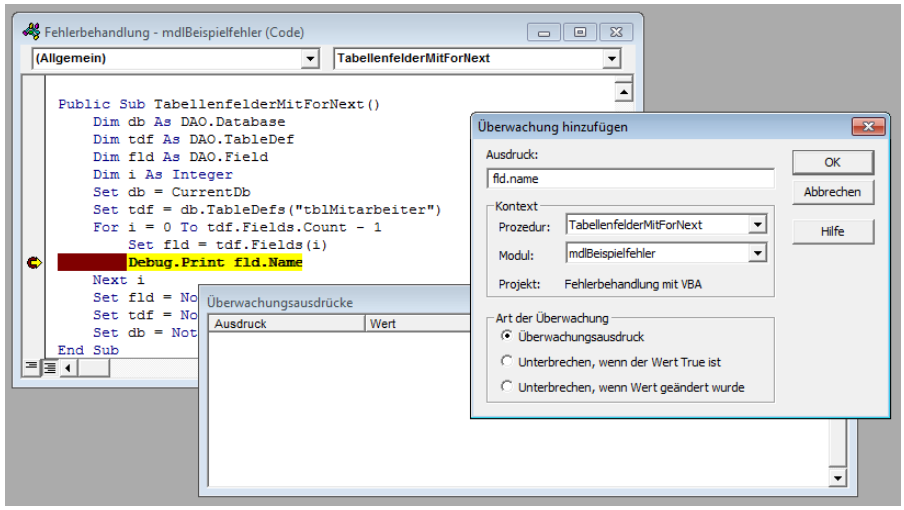


Abbildung 14.9: Anlegen eines neuen Überwachungsausdrucks

Ausdrücke per VBA-Code überwachen

Dass Sie mit der *Debug.Print*-Anweisung beliebige Ausdrücke während des Ablaufs von Routinen ausgeben können, haben Sie bereits weiter oben erfahren. Damit lässt sich die Überwachung von Ausdrücken gut nachbilden, was der ersten Option der Überwachungs-Eigenschaften entsprechen würde.

Auch die zweite Option lässt sich simulieren: Um den Ablauf einer Prozedur zu unterbrechen, wenn ein Ausdruck einen bestimmten Wert enthält, verwenden Sie die *Debug.Assert*-Anweisung.

Als Parameter legen Sie einen Boolschen Ausdruck mit dem gewünschten Ergebnis fest.

14.2.6 Das Lokal-Fenster

Das Lokal-Fenster zeigt direkt alle Variablen und ihre Werte einschließlich der Eigenschaften von Objekten der laufenden Prozedur an (siehe Abbildung 14.10). Um die Eigenschaften von Objekten anzuzeigen, klicken Sie auf das Plus-Symbol (+) vor dem jeweiligen Objekt. Auch dieses Debugging-Tool funktioniert nur in Zusammenhang mit angehaltenem Code.

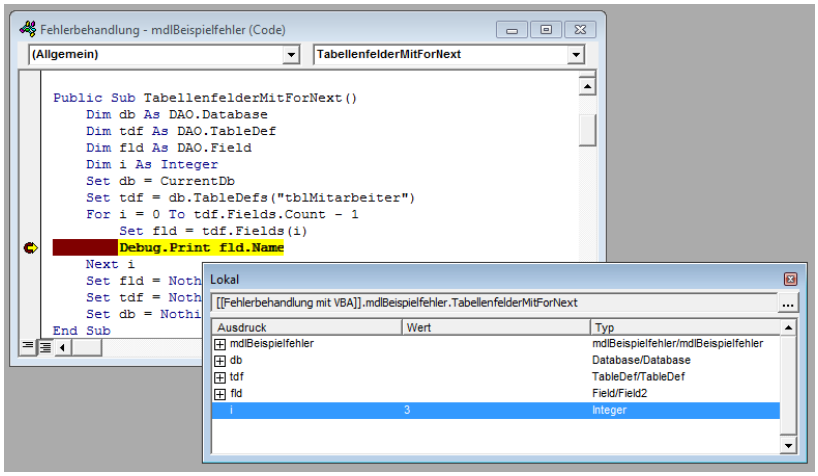


Abbildung 14.10: Das Lokal-Fenster zeigt alle Variablen der laufenden Routine an

14.3 Fehlerbehandlung in VBA

Wenn man von Fehlerbehandlung spricht, bezieht sich dies auf Laufzeitfehler. Andere Fehler wie Syntax- oder Kompilierzeitfehler werden bereits vor dem Start einer Routine gemeldet – hier gibt es auch keine Möglichkeit einer externen Behandlung.

Laufzeitfehler lassen sich unter VBA wie in anderen Programmiersprachen »behandeln«; das heißt, dass Sie mit zusätzlichem VBA-Code auf Fehler reagieren können.

Fehlerbehandlung für benutzerdefinierte Fehlermeldungen

Meist verwendet man eine Fehlerbehandlung dazu, eine benutzerdefinierte Fehlermeldung anzuzeigen, anstatt dem Benutzer der Anwendung die tatsächliche Fehlermeldung mit Blick auf die fehlerhafte Stelle im Code zu präsentieren. Auf diese Weise lassen sich in vielen Fällen aussagekräftigere Meldungen ausgeben. Wenn Sie die Access-Anwendung als *Runtime*-Version weitergeben (mehr dazu in »Weitergabe von Access-Datenbanken« ab Seite 953), gehört eine solche Fehlerbehandlung quasi zum

Pflichtprogramm – erweitert um eine Möglichkeit, Informationen zu den Fehlern in irgendeiner Weise zur Weiterleitung an den Entwickler verfügbar zu machen. Wie das funktioniert, erfahren Sie weiter unten in »Fehlerdokumentation und -übermittlung« auf Seite 823.

Fehlerbehandlung als Vereinfachung

Oft ist es gar nicht geplant, den Benutzer vom Auftreten eines Fehlers in Kenntnis zu setzen – wenn Sie beispielsweise eine Routine verwenden, um eine temporäre Tabelle anzulegen, löschen Sie zuvor eine eventuell vorhandene Tabelle gleichen Namens. Wenn Sie nicht sicher wissen, ob die Tabelle überhaupt vorhanden ist, und keinen Code schreiben wollen, der dies überprüft, sorgen Sie einfach dafür, dass die Tabelle gelöscht wird, wenn diese vorhanden ist, und dass der zu erwartende Fehler, der beim Versuch, eine nicht vorhandene Tabelle zu löschen, auftritt, nicht angezeigt wird – ein Beispiel folgt weiter unten.

Fehlerbehandlung als Lieferant für Bedingungs-Kriterien

Manchmal verwendet man Fehler sogar als eine Art »Verzweigung« – wenn Sie etwa eine Referenz auf ein Word-Objekt benötigen, versuchen Sie zunächst, eine bestehende Instanz anzusprechen.

Ist diese nicht vorhanden, löst dieser Versuch einen Fehler aus. Direkt danach werten Sie aus, ob ein Fehler auftrat, und erzeugen in Abhängigkeit davon eine neue Instanz von Word oder verwenden die existierende Instanz.

14.3.1 Elemente der Fehlerbehandlung

Damit VBA überhaupt irgendetwas anderes macht, als eine Standardfehlermeldung auszugeben, teilen Sie der Routine mit einer *On Error*-Anweisung mit, dass eine Alternative zur Standardfehlermeldung ansteht.

14.3.2 Fehlerbehandlung einleiten

Dabei gibt es folgende Varianten:

» *On Error Resume Next*: Ignoriert sämtliche Fehler und bearbeitet einfach die folgende Zeile.

» *On Error GoTo <Markierung>*: Lässt die Routine im Fehlerfall mit dem hinter *<Markierung>:* befindlichen Code fortfahren.

» *On Error GoTo 0*: Sorgt dafür, dass nachfolgend auftretende Fehler wieder von VBA selbst behandelt werden.

14.3.3 Klassischer Aufbau einer Fehlerbehandlung

Typischerweise ist eine Fehlerbehandlung wie im folgenden Listing aufgebaut. Die Elemente der Fehlerbehandlung sind fett gedruckt. Den Start macht die *On Error*-Anweisung, die angibt, dass die Routine im Falle eines Fehlers an der Markierung *Beispielfehler_Err:* fortgeführt werden soll. Diese Markierungen (auch »Label« genannt) sind an der Stelle der Markierung mit abschließendem Doppelpunkt (:) anzugeben, beim Verweis aber ohne Doppelpunkt.

Zwischen der Markierung *Beispielfehler_Err:* und dem Ende der Routine ist Platz für die eigentliche Fehlerbehandlung. Hier können Sie Informationen zum Fehler auswerten und entsprechende Schritte einleiten – mehr dazu weiter unten. Nach dem Behandeln des Fehlers erfolgt ein erneuter Sprung zur Marke *Beispielfehler_Exit:*. Dort bringen Sie diejenigen Anweisungen unter, die trotz Fehler unbedingt noch ausgeführt werden müssen. Dabei kann es sich um das Schließen von Objekten, die Freigabe von Objektvariablen oder auch das Löschen temporärer Daten handeln.

Wenn die Routine ohne Fehler durchläuft, bleiben die Markierungen bedeutungslos. Das heißt, dass die hinter der Markierung *Beispielfehler_Exit* befindlichen Anweisungen ausgeführt werden, als ob diese Markierung dort gar nicht vorhanden sei. Die *Exit*-Anweisung schließlich sorgt dafür, dass die Routine beim Verlauf ohne Fehler nicht dennoch mit der Fehlerbehandlung endet.

```
Public Function Beispielfehler()
    On Error GoTo Beispielfehler_Err
    …
    'Fehlerbehandlung
Beispielfehler_Exit:
    'Restarbeiten
    …
    Exit Function
Beispielfehler_Err:
    'Fehler auswerten und reagieren
    GoTo Beispielfehler_Exit
End Function
```

Listing 14.1: Grundgerüst einer Fehlerbehandlung

14.3.4 Fehler auswerten

Eine oft gesehene Art der Fehlerauswertung ist die Ausgabe der Fehlerinformationen, etwa durch eine Meldung wie im folgenden Beispiel. Der Fehler durch die Division durch Null führt zur Meldung aus Abbildung 14.11:

15

Performance

Einer der wichtigsten Aspekte für die Akzeptanz einer Anwendung beim Benutzer ist ihre Performance. Eine Anwendung, die Aufgaben nicht zügig in der gewünschten Zeit erledigt, ist von vornherein zum Scheitern verurteilt – die Benutzer verzichten in diesem Fall oft lieber darauf. Deshalb widmet dieses Buch dem Thema Performance ein eigenes Kapitel. Dass dies gerechtfertigt ist, werden Sie im Folgenden sehen: Access birgt in allen Bereichen eine Menge Möglichkeiten für »Performance-Schlucker« und ebenso viel Potenzial für Optimierungen: Dies zieht sich durch alle Bereiche wie Tabellen, Abfragen, Formulare, Berichte und Module.

15.1 Tabellen

Tabellen bilden die Basis einer Datenbankanwendung. Der Aufbau der Tabellen und der Verknüpfungen zwischen den Tabellen entscheidet wesentlich über die Performance der gesamten Anwendung, da alle weiteren Objekte wie Abfragen, gebundene Formulare und Berichte wie auch viele VBA-Prozeduren auf Tabellen zugreifen.

BEISPIELDATENBANKEN

Den Download mit dem Beispielen finden Sie unter dem Link *www.acciu.de/aeb2010*. Die Datenbank zu diesem Kapitel heißt *Performance.accdb*.

15.1.1 Normalisieren des Datenmodells

Das Normalisieren des Datenmodells ist der erste Schritt, um einen schnellen Zugriff auf die in den Tabellen enthaltenen Daten zu ermöglichen. Welche Schritte zur Normalisierung eines Datenmodells notwendig sind, erfahren Sie unter »Normalisierung« ab Seite 69. Für die Performance ist die Normalisierung des Datenmodells wichtig, weil Sie damit beispielsweise die Menge der gespeicherten Daten reduzieren – etwa durch das Extrahieren redundanter Daten in verknüpften Tabellen.

Wenn Sie dabei neue Tabellen erzeugen, versäumen Sie nicht, eine Beziehung zwischen den neu entstandenen Tabellen zu erstellen. Das bringt einen Performance-Gewinn bei der Verknüpfung dieser Tabellen mittels *JOIN* in Abfragen. Falls Sie Beziehungen verwenden, bei denen beim Löschen eines Datensatzes in der Detailtabelle auch alle verknüpften Datensätze der Mastertabelle gelöscht werden sollen, können Sie keine schnellere Methode wählen, als die Löschweitergabe für diese Beziehung zu aktivieren. Diese Einstellung nehmen Sie in den Beziehungseigenschaften im *Beziehungen*-Fenster vor (siehe Abbildung 15.1). Gleiches gilt für die *Aktualisierungsweitergabe*, die Sie im gleichen Dialog aktivieren können.

Eine Suche über mehrere verknüpfte Tabellen, deren Daten zuvor in einer einzigen Tabelle gespeichert waren, kann unter Umständen auch langsamer sein als mit der vorherigen allein stehenden Tabelle. Immerhin kostet das Auflösen von Verknüpfungen innerhalb einer Abfrage ebenso Zeit wie die Suche nach den gewünschten Datensätzen. Wenn Sie tatsächlich einmal eine solche Konstellation vermuten, können Sie je nach Anwendungsfall mit einer *INSERT INTO*-Aktionsabfrage eine temporäre Tabelle aus den verknüpften Tabellen erzeugen, die keine Verknüpfungen mehr aufweist.

In der Regel sind aber durch das Normalisieren des Datenmodells keine Geschwindigkeitseinbußen zu erwarten, die ein nicht normalisiertes Design rechtfertigen.

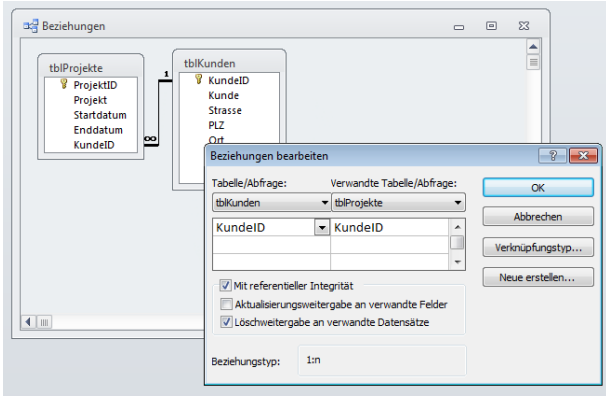


Abbildung 15.1: Aktivieren der Löschweitergabe für eine Beziehung

15.1.2 Indizes

Die richtigen Indizes ermöglichen einen schnellen Zugriff auf die Daten einer Tabelle. Dieser Zugriff erfolgt meist durch Abfragen und bezieht sich oft auf mehr als eine Tabelle. Damit die ACE-Engine die Verknüpfungen schnell verarbeiten kann, sollten alle an einer Verknüpfung beteiligten Felder indiziert sein. Beim Primärschlüsselfeld ist das ohnehin der Fall: Es wird nicht nur indiziert, sondern es ist auch noch eindeutig und lässt keine *Null*-Werte zu. Auch für Fremdschlüsselfelder ist ein Index zu empfehlen. Wenn Sie eine Beziehung zwischen zwei Tabellen etwa im *Beziehungen*-Fenster herstellen (siehe Abbildung 15.2), legt Access automatisch einen Index für das Fremdschlüsselfeld der Detailtabelle an (siehe Abbildung 15.3).

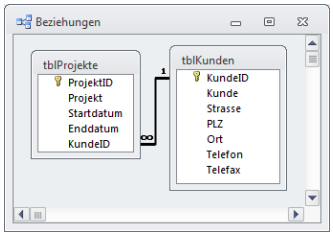


Abbildung 15.2: Das Anlegen einer Beziehung im Beziehungen-Fenster ...

Auch für Felder, die nicht fester Bestandteil einer Beziehung sind, aber gegebenenfalls in einer Abfrage als Schlüsselfeld dienen, legt man einen Index an. Indizieren Sie auch Felder, die als Sortier- oder Filterkriterium in Abfragen dienen. Sparen Sie dabei nur jene Felder aus, die nur wenige unterschiedliche Werte enthalten wie beispielsweise *Ja/Nein*-Felder.

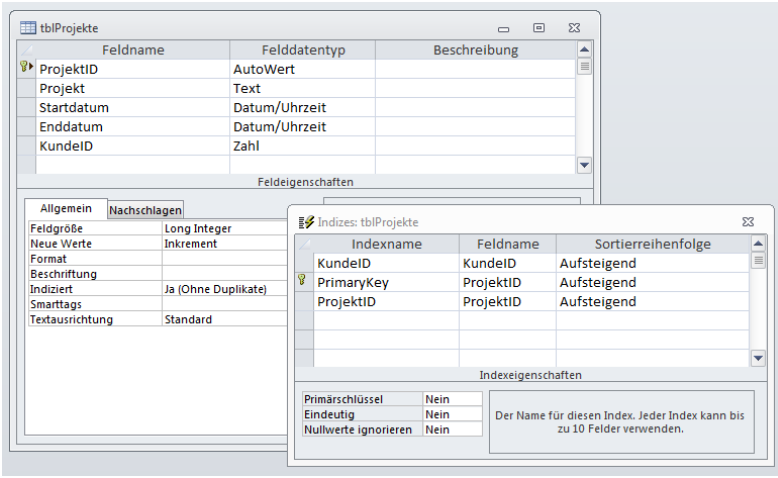


Abbildung 15.3: ... führt zum automatischen Anlegen eines Indexes für das Fremdschlüsselfeld

Kriterium für das Indizieren von Feldern

Ob ein Index abhängig von den Datensätzen sinnvoll ist oder nicht, lässt sich mit der Eigenschaft *DistinctCount* eines Indexes über folgende Funktion berechnen. Die Funktion ermittelt das Verhältnis der enthaltenen verschiedenen Werte zu der Gesamtanzahl der Werte. Dazu übergeben Sie der Routine den Namen der zu prüfenden Tabelle und des Indexes. Ein Aufruf könnte beispielsweise wie folgt aussehen und das angegebene Ergebnis liefern (Beispiel aus der Nordwind-Datenbank):

```
? GetIndexDistinct ("Bestellungen", "Versanddatum")
 0,467469879518072
```

Wenn das Ergebnis zwischen 0,1 und 0,5 liegt, macht der angelegte Index Sinn. Die Funktion sieht folgendermaßen aus:

```
Function GetIndexDistinct(strTabelle As String, strIndex As String) _
    As Double
    Dim db As Database
    Dim tdf As TableDef
    Dim lngDistinct As Long
    Dim lngRecordCount As Long
    Set db = CurrentDb
    Set tdf = dbs.TableDefs(strTabelle)
    lngDistinct = tdf.Indexes(strIndex).DistinctCount
    lngRecordCount = tdf.RecordCount
    GetIndexDistinct = lngDistinct / lngRCnt
```

```
    Set tdf = Nothing
    Set dbs = Nothing
End Function
```

Listing 15.1: Diese Routine prüft, ob ein Feld indiziert werden sollte

Die Routine funktioniert nicht mit verknüpften Tabellen, da die *Recordcount*-Eigenschaft des *TableDef*-Objekts hier den Wert *–1* ausgibt. Und hinterfragen Sie nicht den Sinn dieser Funktion, wenn Sie diese mit einem Primär- oder eindeutigen Schlüssel testen – hier liefert die Funktion logischerweise den Wert *1*.

Manchmal ist weniger mehr

Das Indizieren von Feldern ist allerdings kein Wundermittel, wenn es um Performance-Steigerungen geht. Wenn Sie mehrspaltige Indizes verwenden, binden Sie so wenig Felder wie möglich in den Index ein. Verwenden Sie außerdem eindeutige Indizes, wenn dies möglich ist.

Und zu guter Letzt: Indizieren Sie nicht zu viele Felder! Bedenken Sie, dass eine Datenbankanwendung nicht nur zur Auswahl von Daten gedacht ist, sondern dass damit auch Daten angelegt, bearbeitet und gelöscht werden sollen. Bei all diesen Operationen müssen die Indizes aktualisiert werden, was wiederum zulasten der Performance geht. Als Leitsatz kann gelten: Je größer die beteiligten Tabellen sind und je mehr Datensätze sie enthalten, desto wichtiger sind Indizes beim Mehrbenutzerbetrieb, weil diese die zu übertragende Datenmenge in der Regel reduzieren.

15.1.3 Datentypen

Die Auswahl passender Datentypen beeinflusst nicht nur die Performance, sondern auch den benötigten Speicherplatz der Datenbankanwendung.

Kleinstmögliche Datentypen wählen

Wenn Sie Felder in Tabellen anlegen, wählen Sie den kleinstmöglichen Datentyp aus. Verwenden Sie kein *Memo*-Feld, wenn auch ein Textfeld reicht. Auch Zahlenformate bieten eine Menge Einsparpotenzial. Suchen Sie einfach aus folgender Tabelle den passenden Wertebereich heraus und verwenden Sie den entsprechenden Datentyp. Die Spalte *Speicherbedarf* gibt Ihnen Auskunft darüber, wie sehr die einzelnen Datentypen Speicherplatz und Performance beeinflussen (siehe Tabelle 15.13).

Die landläufige Meinung, dass man Textfelder möglichst klein halten sollte, um nicht unnötig Speicherplatz zu verschwenden, ist nicht richtig. In der Tat können Sie jedes Textfeld mit 255 Zeichen als Feldgröße einrichten; wenn Sie dort aber nur Postleitzahlen mit je sieben Stellen ablegen, werden auch nur sieben Byte belegt.

OLE-Feld versus Anlage-Feld

Der neue Anlage-Felddatentyp ist zwar bezüglich der Handhabung besser als das OLE-Feld aufgestellt, allerdings sollte man sich dennoch überlegen, ob dies performancetechnisch auch die bessere Alternative ist.

Der Grund ist, dass Daten im Anlage-Feld automatisch komprimiert werden, die in OLE-Feldern hingegen nicht.

Wenn Sie also oft in den in einem dieser Felder zu speichernden Daten lesen oder diese oft schreiben, könnte der Einsatz eines OLE-Feldes Vorteile bringen.

Gleiche oder ähnliche Datentypen in Beziehungen

Primärschlüsselfeld und Fremdschlüsselfeld einer Beziehung sollten den gleichen Datentyp besitzen. In einem Fall ist das (zumindest was den Namen betrifft) nicht möglich: Wenn das Primärschlüsselfeld den Datentyp *Autowert* hat, können Sie das Fremdschlüsselfeld natürlich nicht auf den gleichen Datentyp einstellen.

Der Autowert entspricht dem Datentyp *Long* und seltener (zum Beispiel in replizierbaren Datenbanken) dem Typ *Replikations-ID (GUID)*.

15.2 Abfragen

Wenn eine Tabelle nicht gerade genau die Felder enthält, die für die Anzeige in einem Formular oder Bericht oder für die Verwendung in einer VBA-Prozedur benötigt werden, verwenden Sie eine Abfrage, um nur die notwendigen Informationen abzufragen.

15.2.1 Abfragen und die ACE-Engine

Für die Optimierung von Abfragen kann die Kenntnis der bei der Verarbeitung von Abfragen durch die ACE-Engine verwendeten Technik hilfreich sein.

Abfragen entstehen auf unterschiedliche Weise:

» durch Verwendung der Abfrage-Entwurfsansicht

» durch Eingabe eines SQL-Ausdrucks als Datensatzquelle von Formularen und Berichten oder als Datensatzherkunft von Kombinations- und Listenfeldern

» durch Zusammenstellung eines SQL-Ausdrucks in Form eines Strings und anschließende Verwendung in einer VBA-Prozedur

Wie auch immer die Abfrage entstanden ist, verwendet die ACE-Engine letzten Endes den dahinter stehenden SQL-Ausdruck.

Felddatentyp	Speicherbedarf	Wertebereich
Byte	1 Byte	0 bis 255
Boolean	2 Byte	True oder False
Integer	2 Byte	-32.768 bis 32.767
Long (lange Ganzzahl)	4 Byte	-2.147.483.648 bis 2.147.483.647
Single (Gleitkommazahl mit einfacher Genauigkeit)	4 Byte	-3,402823E38 bis -1,401298E-45 für negative Werte; 1,401298E-45 bis 3,402823E38 für positive Werte
Double (Gleitkommazahl mit doppelter Genauigkeit)	8 Byte	-1,79769313486231E308 bis -4,94065645841247E-324 für negative Werte; 4,94065645841247E-324 bis 1,79769313486232E308 für positive Werte
Currency (skalierte Ganzzahl)	8 Byte	-922.337.203.685.477,5808 bis 922.337.203.685.477,5807
Decimal	14 Byte	+/-79.228.162.514.264.337.593.543.950.335 ohne Dezimalzeichen; +/-7,9228162514264337593543950335 mit 28 Nachkommastellen; die kleinste Zahl ungleich Null ist +/-0,0000000000000000000000000001

Tabelle 15.13: Datentypen und ihr Speicherbedarf

Kompilieren einer Abfrage

Die Abfrage wird vor ihrer Ausführung zunächst kompiliert. Vor der Kompilierung ermittelt die ACE-Engine die folgenden Basisinformationen aus der Abfrage:

» Basistabellen: Tabellen, die an der Abfrage beteiligt sind

» Tabellenfelder aus den Ausgabefeldern der Abfrage

» Vorhandene Indizes in den beteiligten Tabellen

» Kriterien

» Verknüpfungsfelder zwischen zwei Tabellen

» Sortierfelder

Optimierung der Abfrage

Anschließend optimiert die ACE-Engine die Abfrage auf Basis dieser Informationen. Dabei erstellt sie unterschiedliche Ausführungspläne und ermittelt daraus die schnellste Variante.

Den größten Einfluss auf die Wahl des Ausführungsplans hat dabei die Analyse der Basistabellen und der Verknüpfungen zwischen diesen Tabellen.

Um den Zugriff auf die Basistabellen zu optimieren, verwendet die ACE-Engine drei Tabellen-Zugriffsstrategien: *Table Scan, Index Range* oder *Rushmore Restriction*. Welche dieser Techniken die ACE-Engine benutzt, hängt von der Größe der jeweiligen Tabelle, der Anzahl der enthaltenen Indizes und der Art und Menge der Kriteriumsfelder ab. Beim *Table Scan* durchsucht die ACE-Engine jeden einzelnen Datensatz für ein Feld der Tabelle. Diese Methode wird angewendet, wenn Kriterienfelder nicht indiziert sind oder wenn die Kriterien die Ergebnismenge nur geringfügig einschränken.

Die *Index Range*-Methode wird verwendet, wenn nur ein indiziertes Feld mit einem Kriterium versehen ist. Die Suche nach den gewünschten Datensätzen erfolgt über den Index dieses Kriterienfeldes.

Die *Rushmore Restriction* kommt zum Zuge, wenn mehr als ein Kriterienfeld indiziert ist. Weitere Informationen finden Sie unter »Abfragen mit Rushmore optimieren« ab Seite 842.

Auch die Beziehungen zwischen den Tabellen spielen eine Rolle bei der Optimierung von Abfragen. Für die Auswertung der Beziehungen werden fünf verschiedene Strategien verwendet, die sich *Nested Iteration Join, Index Join, Lookup Join, Merge Join* und *Index-Merge Join* nennen.

Welche Strategie zum Zuge kommt, hängt von der Beschaffenheit der beteiligten Tabellen, Felder und Indizes, vom Vorkommen von *Null*-Werten und von anderen Faktoren ab.

Weitere Informationen hierzu finden Sie in: *Access 2002 Desktop Developer's Handbook, Litwin, Getz, Gunderloy, Sybex*

Abfragen mit Rushmore optimieren

Rushmore ist eine Technik zur Optimierung von Abfragen, die auf das Vorhandensein von mehreren Restriktionen auf indizierten Feldern abzielt. Die *Rushmore*-Optimierung funktioniert mit den folgenden Operationen:

Index Intersection: Diese Operation wird auf Abfragekriterien folgenden Aussehens angewendet, wobei alle Kriterienfelder indiziert sein müssen:

```
Spalte1 = "Ausdruck1" AND Spalte2 = "Ausdruck2"
```

Der Clou ist, dass die ACE-Engine bei der *Rushmore*-Optimierung nicht mit den eigentlichen Vergleichsoperatoren, sondern mit ihrem Index arbeitet. Dabei sucht die ACE-Engine zunächst nach allen Datensätzen, die die erste Bedingung erfüllen, und anschließend nach allen Datensätzen, die die zweite Bedingung erfüllen, und ermittelt schließlich die Schnittmenge daraus.

Index Union: Die zweite Operation zielt auf durch *OR* verknüpfte Kriterien nach folgendem Schema ab:

```
Spalte1 = "Ausdruck1" OR Spalte2 = "Ausdruck2"
```

Die Abfrage ermittelt hier ebenfalls alle Datensätze, die dem ersten, und alle Datensätze, die dem zweiten Kriterium entsprechen. Allerdings bildet sie hier nicht die Schnittmenge, sondern die Vereinigungsmenge der beiden Ergebnisse. *Rushmore* optimiert obige Abfragen deshalb so gut, weil es »Bitmaps« aus den Index-Werten erstellt und diese indiziert.

Kombiniere, kombiniere ...

Um die Abfrage wirklich optimal ausführen zu können, ermittelt die ACE-Engine den ungefähren Aufwand für die infrage kommenden Tabellen-Zugriffsstrategien in Kombination mit den möglichen Verknüpfungsstrategien und wendet schließlich die am günstigsten erscheinende Variante an.

Ausführung der Abfrage

Wird die Abfrage ausgeführt, tritt noch ein sehr wichtiger Faktor auf: der *Recordset*-Typ. Diese Eigenschaft, die Sie beispielsweise in gespeicherten Abfragen im Eigenschaftsfenster einstellen können (siehe Abbildung 15.4), entscheidet maßgeblich über die Geschwindigkeit der Abfrage.

Eine Abfrage mit dem Recordset-Typ *Dynaset* gibt beispielsweise ein Ergebnis zurück, das lediglich die eindeutigen Schlüsselfelder – soweit vorhanden – der in der Abfrage enthaltenen Tabellen enthält. Es werden nur zusätzliche Daten für die Datensätze in den Speicher geladen, die beispielsweise für die Anzeige im Formular benötigt werden. Abfragen des Recordset-Typs *Snapshot* laden die kompletten Daten – also alle Datensätze mit allen angegebenen Feldern – direkt in den Speicher.

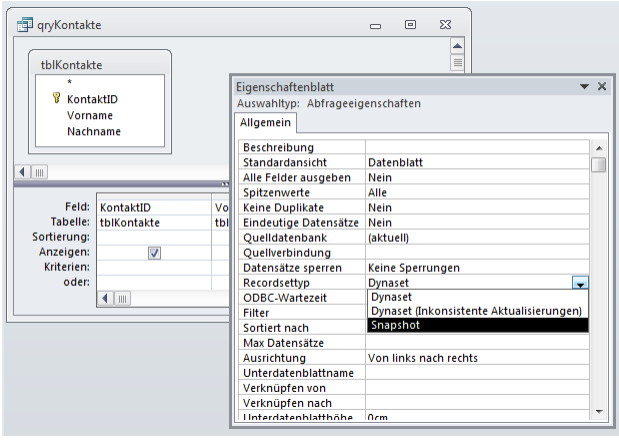


Abbildung 15.4: Einstellen des Recordset-Typs einer Abfrage

Abfragestrategien unter der Lupe

Access bietet die Möglichkeit, die bei einer Abfrage verwendeten Informationen (zumindest teilweise) auszuwerten. Mit der Anpassung eines Registry-Eintrags können Sie dafür sorgen, dass Access Informationen über die Durchführung von Abfragen in einer Textdatei ausgibt.

Den entsprechenden Schlüssel müssen Sie zunächst in der Registry unter *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Office\14.0\Access Connectivity Engine\Engines* hinzufügen. Legen Sie zunächst mit dem Kontextmenüeintrag *Neu|Schlüssel* einen neuen Schlüssel für den Schlüssel *Engines* an und nennen Sie diesen *Debug*. Wählen Sie dann aus dem Kontextmenü des neuen Schlüssels den Eintrag *Neu|Zeichenfolge* aus und nennen Sie den neuen Eintrag *Jetshowplan*.

Stellen Sie dort den Wert auf *On* ein. Anschießend schreibt die ACE-Engine Informationen über die Ausführungspläne der Abfragen in eine Datei namens *showplan.out*. Diese Funktion ist nicht offiziell dokumentiert. Der Ausgabeort der Datei *showplan.out* liegt meist im aktuellen Datenbankverzeichnis, das Sie in den Access-Optionen unter *Häufig verwendet|Standarddatenbankordner* finden.

Die Funktion lässt sich leider nicht auf eine einzelne Datenbank beschränken (es sei denn, Sie aktivieren und deaktivieren diesen Schlüssel per VBA von der entsprechenden Datenbank aus). Da diese Option demnach für alle Datenbanken gilt, wird die Datei *Showplan.out* schnell sehr groß und macht Access langsam; behalten Sie diese daher im Auge oder schalten Sie die Funktion nur bei Bedarf ein. Um die Funktion zu deaktivieren, entfernen Sie den Schlüssel entweder komplett oder stellen den Wert auf *Off* ein (das wirkt sich übrigens erst beim nächsten Start von Access aus).

Das folgende Listing zeigt den Inhalt der Datei *showplan.out* für die Abfrage *Bestellungszusammenfassung* der Nordwind-Datenbank von Access 2007. Der erste Teil enthält die für die Auswahl der Strategie für den Zugriff auf die Tabellen ermittelten Informationen, der zweite Teil die Reihenfolge bei der Abarbeitung der Verknüpfungen.

```
--- Bestellungszusammenfassung ---
- Inputs to Query -
 Table 'Status der Bestellung'
     Using index 'PrimaryKey'
     Having Indexes:
     PrimaryKey 4 entries, 1 page, 4 values
       which has 1 column, fixed, unique, primary-key, no-nulls
 Table 'Bestellungen'
 Table 'Bestelldetails'
 Table 'Status für Bestelldetails'
     Using index 'PrimaryKey'
```

```
    Having Indexes:
    PrimaryKey 6 entries, 1 page, 6 values
      which has 1 column, fixed, unique, primary-key, no-nulls
- End inputs to Query -
 01) Outer Join table 'Bestelldetails' to table 'Status für Bestelldetails'
       using index 'Status für Bestelldetails!PrimaryKey'
       join expression "[Bestelldetails].[Status-Nr]=[Status für Bestelldetails].
 [Status-Nr]"
 02) Group result of '01)'
 03) Outer Join table 'Bestellungen' to result of '02)'
       using temporary index
       join expression "Bestellungen.[Bestell-Nr]=[Gesamtpreis der Bestellung].BestellNr"
 04) Outer Join result of '03)' to table 'Status der Bestellung'
       using index 'Status der Bestellung!PrimaryKey'
       join expression "Bestellungen.[Status-Nr]=[Status der Bestellung].[Status-Nr]"
 05) Sort result of '04)'

--- temp query ---
- Inputs to Query -
 Recordset
- End inputs to Query -

 01) Scan recordset
 02) Restrict rows of table 01)
       using index 'UNKNOWN'
       for expression "[_VersionHistory_F5F8918F-0A3F-4DA9-AE71-184EE5012880]=FK"
 03) Sort result of '02)'

--- Bestellungszusammenfassung ---
- Inputs to Query -
 Table 'Status der Bestellung'
    Using index 'PrimaryKey'
    Having Indexes:
    PrimaryKey 4 entries, 1 page, 4 values
      which has 1 column, fixed, unique, primary-key, no-nulls
 Table 'Bestellungen'
 Table 'Bestelldetails'
 Table 'Status für Bestelldetails'
    Using index 'PrimaryKey'
    Having Indexes:
    PrimaryKey 6 entries, 1 page, 6 values
```

```
        which has 1 column, fixed, unique, primary-key, no-nulls
  - End inputs to Query -

  01) Outer Join table 'Bestelldetails' to table 'Status für Bestelldetails'
        using index 'Status für Bestelldetails!PrimaryKey'
        join expression "[Bestelldetails].[Status-Nr]=[Status für Bestelldetails].
  [Status-Nr]"
  02) Group result of '01)'
  03) Outer Join table 'Bestellungen' to result of '02)'
        using temporary index
        join expression "Bestellungen.[Bestell-Nr]=[Gesamtpreis der Bestellung].BestellNr"
  04) Outer Join result of '03)' to table 'Status der Bestellung'
        using index 'Status der Bestellung!PrimaryKey'
        join expression "Bestellungen.[Status-Nr]=[Status der Bestellung].[Status-Nr]"
  05) Sort result of '04)'
```

Listing 15.2: Inhalt der Datei *showplan.out* nach dem Durchführen einer komplexen Abfrage

Wenn Sie beim Tuning Ihrer Abfragen ins Detail gehen möchten, kann die Ausgabe der ACE-Engine durchaus weiterhelfen. Wenn Sie der Abfrage Bestellungszusammenfassung beispielsweise ein Kriterium für das Feld *Versanddatum* hinzufügen und dieses Feld nicht indiziert ist, erhalten Sie im Ausführungsplan folgende Zeile als ersten Schritt:

```
01) Restrict rows of table Bestellungen
      by scanning
      testing expression "Bestellungen.Versanddatum>#1/1/2006#"
```

Das Schlüsselwort *scanning* bedeutet in diesem Zusammenhang, dass ein *Table Scan* durchgeführt wird – ein Index für dieses Feld ist nicht vorhanden.

Besser wäre es, das Feld *Versanddatum* der zugrunde liegenden Tabelle *Bestellungen* mit einem Index zu versehen – auf diese Weise könnte die *Rushmore*-Technik verwendet werden. Die entsprechende Zeile in der Datei *showplan.out* sieht dann folgendermaßen aus:

```
01) Restrict rows of table Bestellungen
      using rushmore
      for expression "Bestellungen.Versanddatum>#1/1/2006#"
```

15.2.2 Datenbank mit kompilierten Abfragen ausliefern

Die ACE-Engine kompiliert eine Abfrage beim ersten Ausführen dieser Abfrage nach dem Speichern. Das Kompilieren beinhaltet auch das Optimieren, weshalb es sinnvoll sein kann, die Abfrage zu bestimmten Anlässen neu zu kompilieren.

Bei Änderungen neu kompilieren

Wie Sie weiter oben erfahren haben, hängt der Ausführungsplan und damit die Reihenfolge und Geschwindigkeit der einzelnen Schritte bei der Durchführung der Abfrage wesentlich von den Tabellen-Eigenschaften wie Indizes und Beziehungen und von Daten-Eigenschaften wie der Anzahl der Datensätze und der dadurch belegten Speicherseiten ab.

Wenn Sie Änderungen am Entwurf der Tabellen vornehmen, auf denen die zu optimierende Abfrage basiert, sollten Sie die Abfrage neu kompilieren, damit diese den neuen Gegebenheiten entsprechend optimiert wird. Auch die Anzahl der in den Tabellen enthaltenen Datensätze spielt eine Rolle. Versuchen Sie, die Tabellen möglichst mit realen Daten zu füllen, um die Abfragen optimiert ausliefern zu können.

Auslieferung im kompilierten Zustand

Die ACE-Engine kompiliert eine Abfrage erst neu, wenn Sie eine Änderung am Entwurf vorgenommen haben, diese dann speichern und ausführen. Das Kompilieren ist in vielen Fällen aufwändiger als die eigentliche Durchführung der Abfrage. Liefern Sie daher die Datenbank immer mit kompilierten Abfragen aus. So erhält der Benutzer direkt bei der ersten Verwendung der Abfrage optimale Performance.

15.2.3 Gespeicherte Abfragen versus Ad-hoc-Abfragen

Gespeicherte Abfragen haben den Vorteil, dass man sie durch Speichern und Kompilieren optimiert und damit in folgenden Einsätzen auf die dabei gewonnenen Erkenntnisse bezüglich der Performance zugreifen kann. Sogenannte Ad-hoc-Abfragen, die in VBA-Routinen zusammengesetzt und erst bei Bedarf das erste Mal kompiliert und optimiert werden, ziehen in der Regel den Kürzeren, was die Performance angeht – in den meisten Fällen sind daher gespeicherte Abfragen vorzuziehen.

Auch hier gibt es allerdings eine Ausnahme: Weiter oben wurde bereits beschrieben, dass man Abfragen vor der Auslieferung möglichst mit der zu erwartenden Anzahl Datensätze in den betroffenen Tabellen kompiliert und speichert. Das ist aber nicht immer möglich. Wenn die Anzahl der Datensätze völlig anders ausfällt als geplant, ist eine gespeicherte Abfrage möglicherweise weniger performant als eine Abfrage, die vor jeder Ausführung neu erstellt wird.

15.2.4 Abfragen auf Performance trimmen

Das Design von Abfragen bietet jede Menge Fallstricke, wenn es um gute Performance geht. Die folgenden Abschnitte enthalten deshalb den nötigen »Feinschliff« für Ihre Abfragen.

Nur notwendige Felder anzeigen

Je mehr Felder eine Abfrage anzeigt, desto mehr leidet die Performance – das gilt vor allem für Abfragen mit dem Recordset-Typ *Snapshot*. Zeigen Sie daher nur die unbedingt notwendigen Felder an. Prüfen Sie vor allem genau, ob die Abfrage gegebenenfalls Kriterienfelder enthält, die nicht unbedingt angezeigt werden müssen. Diese lassen sich über das entsprechende Kontrollkästchen ausblenden. Auch die Anzahl der Tabellen, aus denen Felder angezeigt werden müssen, spielt eine Rolle. Je mehr Tabellen eines oder mehrere Felder zum Abfrageergebnis beitragen, desto langsamer die Abfrage.

Kurze Feldbezeichnungen und Alias-Namen verwenden

Weitere Vorteile bei der Performance liefern kurze Namen von Tabellenfeldern sowie Alias-Namen für Tabellen, wie folgendes Beispiel zeigt:

```
SELECT t2.[Bestell-Nr], t2.[Artikel-Nr], t1.Artikelname,
t2.Einzelpreis, t2.Anzahl, t2.Rabatt,
CCur(t2.Einzelpreis*[Anzahl]*(1-[Rabatt])/100)*100 AS Endpreis
FROM Artikel AS t1
INNER JOIN Bestelldetails AS t2
ON t1.[Artikel-Nr] = t2.[Artikel-Nr]
ORDER BY t2.[Bestell-Nr];
```

statt

```
SELECT Bestelldetails.[Bestell-Nr], Bestelldetails.[Artikel-Nr],
Artikel.Artikelname, Bestelldetails.Einzelpreis,
Bestelldetails.Anzahl, Bestelldetails.Rabatt,
CCur(Bestelldetails.Einzelpreis*[Anzahl]*(1-[Rabatt])/100)*100 AS Endpreis
FROM Artikel
INNER JOIN Bestelldetails ON Artikel.[Artikel-Nr] =
Bestelldetails.[Artikel-Nr]
ORDER BY Bestelldetails.[Bestell-Nr];
```

Sternchen zählen

Soll eine Abfrage die Anzahl der gefundenen Datensätze zurückgeben, wenden Sie die *Count*-Funktion auf den Ausdruck * (Sternchen) an und nicht auf eines der Felder wie beispielsweise das Primärschlüsselfeld.

Keine Funktionen und berechneten Ausdrücke

Verwenden Sie in Abfragen nach Möglichkeit so wenig eingebaute oder benutzerdefinierte Funktionen oder berechnete Ausdrücke wie möglich. Dazu gehören Funktionen wie *IIf* oder *DLookup* und weitere Domänen-Funktionen. Die ACE kann solche Abfragen

nicht optimieren, weil sich das Ergebnis einer Berechnung statistisch nicht vorhersagen lässt.

Kriterien für Schlüsselfelder variieren

Manchmal verwendet man Kriterien für ein Feld, über das eine Beziehung zu einer anderen Tabelle hergestellt wird. Das Primärschlüsselfeld der beteiligten Mastertabelle hat zwar den gleichen Wert wie das entsprechende Fremdschlüsselfeld der Detailtabelle.

Es kann aber einen Unterschied in der Verarbeitungsgeschwindigkeit machen, auf welches der beiden Felder Sie ein Kriterium anwenden. Bei dieser Konstellation müssen Sie ausprobieren, wo das Kriterium für eine schnellere Abarbeitung sorgt.

15.3 Formulare

Die Möglichkeiten zur Verbesserung der Performance von Formularen lassen sich in zwei Kategorien einteilen: Die einen beeinflussen die Zeit, bis das Formular geöffnet und einsatzbereit ist, und die anderen die Arbeit mit dem Formular selbst.

15.3.1 Formulare offenhalten oder schließen?

Je nachdem, wie oft Sie ein Formular verwenden, sollten Sie in Erwägung ziehen, es nicht jedes Mal zu schließen und neu zu öffnen, sondern das Formular einfach unsichtbar zu machen. Dazu verwenden Sie die Eigenschaft *Sichtbar*, die Sie folgendermaßen in VBA einstellen können:

```
'Ausblenden
 Me.Visible = False
'Einblenden
 Me.Visible = False
```

Andererseits fressen offene Formulare Speicherplatz, sodass Sie diese Vorgehensweise tunlichst nicht mit allen Formularen durchexerzieren sollten. Außerdem besteht die Gefahr, dass Sie einmal mit *Screen.ActiveForm* das aktuelle Formular ermitteln möchten, es sich bei diesem jedoch um ein unsichtbares Formular handelt.

15.3.2 Daten des Formulars

Die im Formular angezeigten Daten stammen aus einer oder mehreren Tabellen, die entweder direkt oder in Form einer Abfrage angesprochen werden. Verwenden Sie eine Abfrage, können Sie entweder auf eine gespeicherte Abfrage zurückgreifen oder geben den entsprechenden SQL-Ausdruck direkt für die Eigenschaft *Datensatzquelle*

ein. Gleiches gilt im Übrigen auch für die Datensatzherkunft von Kombinations- und Listenfeldern.

Hier gibt es einige Regeln:

» Die Datensatzquelle von Formularen beziehungsweise die Datensatzherkunft von Kombinations- und Listenfeldern sollte möglichst eine Abfrage sein, die nur die benötigten Felder und – noch wichtiger – nur die benötigten Datensätze enthält. Daher ist in den meisten Fällen eine Abfrage die beste Wahl.

» Wenn Sie eine Abfrage verwenden, können Sie diese entweder speichern und den Namen der Abfrage als Datensatzquelle oder Datensatzherkunft angeben oder direkt den SQL-Ausdruck der Abfrage in die Eigenschaft *Datensatzquelle* oder *Datensatzherkunft* eintragen.

Es stimmt nicht, dass als Abfrage gespeicherte Datenherkünfte Vorteile bringen, weil diese vorkompiliert und damit optimiert werden können. SQL-Ausdrücke, die Sie für die Eigenschaften *Datensatzquelle* oder *Datensatzherkunft* eintragen, werden beim Ausführen genauso kompiliert wie gespeicherte Abfragen.

Interessant ist auch die Variante, die ins Spiel kommt, wenn Sie ein Formular nur zur Eingabe von Daten verwenden möchten. Öffnen Sie ein solches Formular mit folgender Anweisung, wenn Sie dazu VBA verwenden:

```
DoCmd.OpenForm "<Formularname>", DataMode:=acFormAdd
```

Anderenfalls können Sie die Eigenschaft *Daten eingeben* im Eigenschaftsfenster des Formulars auf den Wert *Ja* einstellen.

15.3.3 Steuerelemente

Je mehr Steuerelemente Sie in einem Formular anlegen, desto schlechter wird die Performance. Aber auch die Anordnung der Steuerelemente spielt eine Rolle: Übereinander angeordnete Steuerelemente sorgen beispielsweise für eine weitere Verschlechterung. Das könnte etwa der Fall sein, wenn Sie ein Textfeld und ein Kombinationsfeld direkt übereinander legen, um je nach Bedarf das eine oder andere anzuzeigen. Und auch die Gestaltung eines assistenten-ähnlichen Formulars, das mehrere Schritte eines Assistenten in einem Formular enthält und die verschiedenen Stufen nacheinander ein- beziehungsweise ausblendet, ist performancetechnisch betrachtet keine optimale Lösung – dann doch lieber verschiedene Formulare, die nacheinander geöffnet werden.

Einfache Steuerelemente verwenden

Es gibt unterschiedlich gewichtete Steuerelemente, was die Performance angeht. Das gilt gerade für die verschiedenen Möglichkeiten zur Auswahl von Informationen. Wenn

Sie eine feste, nicht allzu große Anzahl Optionen zur Auswahl bereitstellen möchten, verwenden Sie dazu eine Optionsgruppe. Diese ist bereits performanter als ein Kombinationsfeld oder ein Listenfeld.

Sollte einmal die Entscheidung zwischen einem Kombinations- oder Listenfeld und einem Unterformular anstehen und sollten keine wesentlichen Punkte für eine der beiden Möglichkeiten sprechen, verwenden Sie ein Kombinations- oder Listenfeld. Und hier ist wiederum das Kombinationsfeld zu bevorzugen, weil der Inhalt in der Regel erst beim Aufklappen ermittelt wird.

Bild-Steuerelement vor!

Wenn Sie Bilder in einem Formular anzeigen möchten, verwenden Sie dazu das Bild-Steuerelement statt des ungebundenen Objekt-Steuerelements.

Generell sollten Sie jedoch nur so verfahren, wenn es unbedingt notwendig ist. In vielen Fällen sollte eine Schaltfläche ausreichen, die ein *Popup*-Formular mit dem gewünschten Bild öffnet.

Schaltflächen durch Hyperlinks ersetzen

Wenn Sie eine Schaltfläche lediglich verwenden, um damit ein weiteres Formular zu öffnen, können Sie dazu auch ein Bezeichnungsfeld mit einen Hyperlink verwenden. Damit sparen Sie die Schaltfläche ein.

Die Zuordnung des entsprechenden Links zu einem Bezeichnungsfeld erfolgt ganz einfach über den Dialog aus Abbildung 15.5.

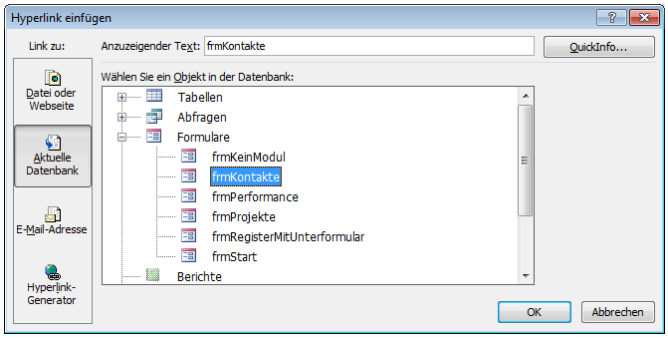


Abbildung 15.5: Setzen eines Hyperlinks auf ein zu öffnendes Formular

Den Hyperlink können Sie natürlich auch manuell eintragen – in diesem Fall würde er folgendermaßen lauten:

```
Form frmKontakte
```

Wie Abbildung 15.5 zeigt, lassen sich auch andere Objekte der Datenbank über einen Hyperlink öffnen.

Kombinationsfelder und Listenfelder

Für Kombinations- und Listenfelder gelten die gleichen Regeln wie für das Formular selbst: Die zugrunde liegende Datenquelle sollte in Form einer gespeicherten Abfrage angegeben werden, die nur die notwendigen Felder und Datensätze enthält. Dabei sollte das gebundene Feld indiziert sein.

Wenn Sie die automatische Ergänzung bei der Eingabe von Einträgen in ein Kombinationsfeld nicht benötigen, lassen Sie diese Funktion weg, indem Sie die Eigenschaft *Automatisch ergänzen* auf den Wert *Nein* einstellen. Das angezeigte Feld im Kombinationsfeld sollte den Datentyp *String* haben, da andere Datentypen sonst erst konvertiert werden müssten.

Sollte der Inhalt eines Kombinations- oder Listenfeldes aus einer Tabelle kommen, auf die über das Netzwerk zugegriffen wird, prüfen Sie, ob sich die in dieser Tabelle gespeicherten Daten oft oder überhaupt ändern. Falls nicht, kopieren Sie die Daten in eine lokale Tabelle und greifen Sie so auf diese Daten zu.

Unterformulare

Unterformular-Steuerelemente enthalten externe Formulare und zählen damit zu den ressourcen-hungrigsten Steuerelementen. Sie zeigen Daten aus einer eigenen Datensatzquelle an, enthalten gegebenenfalls weitere Kombinations- oder Listenfelder und müssen in den meisten Fällen auch noch mit dem Hauptformular synchronisiert werden.

Diese Synchronisation erfolgt über die Eigenschaften *Verknüpfen von* und *Verknüpfen nach* des Unterformular-Steuerelements. Zur Optimierung der Performance sollten Sie die in diesen Eigenschaften angegebenen Felder in den zugrunde liegenden Tabellen indizieren.

Schlechter für die Performance eines Formulars als ein Unterformular sind zwei oder mehr Unterformulare. Wenn diese alle gleichzeitig sichtbar sein sollen, müssen Sie in den sauren Apfel beißen; in manchen Fällen werden die Daten aber auch auf verschiedenen Seiten eines Register-Steuerelements angezeigt.

In diesem Fall gibt es Optimierungspotenzial: Unterformular-Steuerelemente, die kein Unterformular enthalten, brauchen Sie auch nicht mit Daten zu füllen. Auf diese Weise müssen beim Blättern durch die Datensätze des Hauptformulars nicht angezeigte Unterformulare auch nicht mit Daten gefüllt werden.

Um nur im aktuellen Register enthaltene Unterformular-Steuerelemente mit den entsprechenden Formularen zu füllen, gehen Sie folgendermaßen vor:

» Legen Sie die Unterformulare wie gewohnt an, um Größe und Position einzustellen.

» Stellen Sie die Eigenschaft *Herkunftsobjekt* der Unterformular-Steuerelemente auf eine leere Zeichenkette ein.

» Weisen Sie dem Unterformular-Steuerelement, das beim Öffnen des Formulars sichtbar ist, das anzuzeigende Formular zu:

```
Private Sub Form_Open(Cancel As Integer)
    Me!<Unterformularsteuerelement>.SourceObject = _
        "<Formularname>"
End Sub
```

» Sorgen Sie beim Wechsel der Registerseite dafür, dass nicht angezeigte Unterformulare ausgeblendet und sichtbare eingeblendet werden:

```
Private Sub RegisterStr0_Change()
    Select Case Me!RegisterStr0
        Case 0
            Me!frmProjekte1.SourceObject = "<Unterformular1>"
            Me!frmProjekte2.SourceObject = ""
        Case 1
            Me!frmProjekte1.SourceObject = ""
            Me!frmProjekte2.SourceObject = "<Unterformular2>"
    End Select
End Sub
```

15.3.4 VBA in Formularen

Die Programmierung von Formularen erlaubt weitere Möglichkeiten zur Code-Optimierung – oder auch nicht, wie der folgende Abschnitt zeigt.

Formular von Code befreien

Formulare ohne Code werden deutlich schneller geladen. Stellt sich die Frage, was man mit einem Formular ohne VBA-Code alles anstellen kann. Die Antwort lautet: Alles, was auch mit einem Formular mit VBA-Code geht. Sie können ein Formular sogar komplett von seinem Formularmodul befreien und dennoch auf den gewohnten Komfort von VBA zugreifen.

Um das Formular von seinem Modul zu befreien, stellen Sie einfach die Eigenschaft *Enthält Modul* auf den Wert *Nein* ein.

Wenn das Formularmodul zu diesem Zeitpunkt bereits Code enthält, meckert Access natürlich – prüfen Sie also, ob Sie den enthaltenen Code noch benötigen.

16

Objektorientierte Programmierung

Wenn Sie im Internet nach Informationen über den Einsatz objektorientierter Programmierung in Zusammenhang mit Microsoft Access suchen, müssen Sie eine Menge Geduld mitbringen. Das macht sich erst recht bemerkbar, wenn Sie erstmal diejenigen Seiten oder Newsgroup-Beiträge herausfiltern müssen, in denen Entwickler über die Vor- und Nachteile der objektorientierten Entwicklung unter Access beziehungsweise VB/VBA diskutieren oder beleuchten, ob sich VB/VBA überhaupt »objektorientiert« nennen darf. Warum geht dieses Buch also so ausführlich auf objektorientierte Techniken ein, wenn dieses Thema selbst in Entwicklerkreisen anscheinend umstritten ist und sich nur wenige Entwickler diesem Thema widmen? Die Antwort ist einfach: Erstens arbeiten Sie, wenn Sie mit VBA entwickeln, ohnehin schon mehr oder weniger bewusst mit Objekten. Beispiele dafür sind Formulare, Berichte, Steuerelemente, Recordsets oder andere Anwendungen wie Word oder Excel. Dabei greifen Sie beim Programmieren

wie selbstverständlich auf die per *IntelliSense* komfortabel einsetzbaren Methoden und Eigenschaften der Objekte zu. Den Umgang mit Objekten sind Sie also gewohnt – warum sollten Sie nicht von benutzerdefinierten Klassen profitieren?

Und zweitens werden Sie, selbst wenn Sie nur kleine Häppchen Objektorientierung in Form des einen oder anderen Klassenmoduls zum Kapseln einer bestimmten Funktionalität in Ihre Anwendungen einbauen, davon profitieren und möglicherweise schnell mehr davon verwenden wollen.

BEISPIELDATENBANKEN

Den Download mit dem Beispielen finden Sie unter dem Link *www.acciu.de/aeb2010*. Die Datenbank zu diesem Kapitel heißt *Objektorientierung.accdb*.

Beispiel für den Einsatz einer benutzerdefinierten Klasse

Ein gutes Beispiel ist die Klasse *CTimer*, die zusammen mit einem weiteren Modul Funktionen bereitstellt, um einen oder mehrere Timer zu betreiben. Die Aufgabe eines Timers ist dabei, in zuvor festgelegten Intervallen oder zu festgelegten Zeitpunkten Ereignisse auszulösen und damit dafür vorgesehene Routinen zu starten. Damit lässt sich etwa die Anzeige von Daten in einem Formular in festen Intervallen aktualisieren oder das Blinken von Steuerelementen steuern (nur ein Scherz!).

Das Wichtige hierbei ist, dass Sie, wenn Sie auf herkömmlichem Wege einen Timer einsetzen wollten, dazu ein Formular und dessen Ereigniseigenschaft *Bei Zeitgeber* dazu missbrauchen würden – und das selbst dann, wenn der Zweck des Zeitgebers gar nichts mit dem Formular selbst zu tun hat. Außerdem können Sie mit einem Formular nur ein einziges Intervall definieren, und wenn man mal mehrere braucht ...

Also programmiert man sich vielleicht selbst eine passende rudimentäre Funktionalität (es soll ja schnell gehen), die möglicherweise die Intervalle fest im Code verdrahtet. Dabei kann man einen solchen Zeitgeber immer wieder brauchen, warum also nichts Wiederverwendbares schaffen? Etwas, das noch dazu einfach zu handhaben ist? Ohne, dass man sich erst die Namen der Funktionsaufrufe heraussuchen muss? Werfen Sie also einen Blick auf die fertige Klasse *CTimer*. Welche Vorteile bietet sie und wie arbeitet man überhaupt damit? Um diese Klasse einzusetzen, müssen Sie einfach nur das Klassenmodul *CTimer* und das dazugehörende Standardmodul *mdlTimer* in die Zieldatenbank kopieren. Wenn Sie es dann in einem Formularmodul oder einem weiteren Klassenmodul mit dem Schlüsselwort *WithEvents* (dazu später mehr) deklarieren, können Sie für diese Klasse nicht nur Eigenschaften festlegen, sondern auch noch auf Ereignisse reagieren – was ja bei einem Timer ganz sinnvoll ist:

```
Dim WithEvents objTimer1 As CTimer
```

Nehmen wir an, Sie möchten ein Steuerelement eines Formulars alle paar Sekunden aktualisieren – etwa, um darin die Uhrzeit anzuzeigen – und dafür nicht die eingebaute Eigenschaft *Bei Zeitgeber* verwenden (weil Sie diesen möglicherweise für einen anderen Zweck benötigen).

In dem Fall gehen Sie folgendermaßen vor: Fügen Sie einem neuen Formular ein Bezeichnungsfeld namens *lblUhrzeit* hinzu und speichern Sie das Formular unter dem Namen *frmUhrzeit*. Legen Sie für die Ereigniseigenschaft *Beim Laden* des Formulars die folgende Routine an:

```
Private Sub Form_Load()
    Me!lblUhrzeit.Caption = Time
    Set objTimer1 = New CTimer
    With objTimer1
        .TimerID = 1
        .Interval = 1000
        .Repeated = True
    End With
End Sub
```

Listing 16.1: Instanzieren und einstellen der Timer-Klasse

Beachten Sie, dass Sie die Eigenschaften von *objTimer1* mit *IntelliSense*-Unterstützung eingeben können. *TimerID* ist die Nummer des Timers (falls Sie mal mehrere benötigen, verwenden Sie etwa eine fortlaufende Nummerierung), *Intervall* gibt die Zeit bis zum nächsten Auslösen in Millisekunden an und *Repeated* legt fest, ob ein Ereignis wiederholt ausgelöst werden soll. Damit der Timer beim Schließen des Formulars seinen Dienst beendet, legen Sie auch eine Prozedur an, die das Formular beim Entladen aufruft:

```
Private Sub Form_Unload(Cancel As Integer)
    Set objTimer1 = Nothing
End Sub
```

Listing 16.2: Freigeben der Objektvariablen

Und jetzt kommt der Clou: Irgendwie müssen Sie ja noch festlegen, was eigentlich in Intervallen von x Millisekunden passieren soll. Dazu bietet das Objekt ein Ereignis an, für das Sie eine Ereignisprozedur anlegen können! Wählen Sie einfach aus dem rechten Kombinationsfeld im oberen Teil des Codefensters den Eintrag *objTimer1* aus und Access legt automatisch den Rumpf der Prozedur *objTimer_Plop* an. Hier müssen Sie nur noch eintragen, dass Access dem vorhin erstellten Bezeichnungsfeld als Beschriftung die aktuelle Uhrzeit zuweist:

```
Private Sub objTimer_Plop(dtDateTime As Date, ATag As Variant)
    Me!lblUhrzeit.Caption = Time
End Sub
```

Listing 16.3: Ereignis zur richtigen Zeit

Vorteile von Klassen

Dieses Beispiel zeigt bereits einige Vorteile der Verwendung von Klassen:

» Komplizierte Funktionen lassen sich hinter einfachen Methodenaufrufen verstecken.

» Getestete und funktionstüchtige Klassen lassen sich beliebig wieder verwenden.

» Parameter müssen nicht mehr mit dem Funktionsaufruf übergeben werden, sondern können übersichtlich als Eigenschaften des Objekts festgelegt werden.

» Objekte bieten eine komfortable Programmierschnittstelle: IntelliSense verrät Ihnen genau, welche Eigenschaften und Methoden die Instanz eines Klassenmoduls bereitstellt. Dadurch sind Objekte quasi selbst dokumentierend und erlauben einen einfachen Zugriff.

» Klassen können Ereignisse bereitstellen, die durch beliebige Vorgänge ausgelöst werden.

Darüber hinaus gibt es noch einige weitere Vorteile:

» Zusammenhängende Daten, die sonst als Variablen mehr oder weniger öffentlich irgendwo im Quellcode zu finden sind, lassen sich als Eigenschaften eines Klassenmoduls zusammenfassen.

» Methoden, die sich genau auf diese zu einem Klassenmodul zusammengefassten Daten beziehen, lassen sich nur in diesem Zusammenhang aufrufen.

» Die in einem Klassenmodul zusammengefassten Eigenschaften und Methoden können Objekte aus der Realität abbilden.

» Änderungen an einem Klassenmodul, das möglicherweise an vielen Stellen instanziert wird, müssen nur an einer Stelle durchgeführt werden.

» Sie können gleichzeitig mit mehreren Instanzen eines Klassenmoduls arbeiten.

» Objekte können den Datenzugriff kapseln. Damit können Sie etwa Anwendungen erstellen, ohne sich auf eine bestimmte Datenquelle wie Tabellen einer Datenbank, Daten im XML-Format oder eine einfache Textdatei festzulegen.

In den folgenden Abschnitten erfahren Sie, wie Sie eigene Klassen erstellen und diese einsetzen. Das anschließende Kapitel greift diese Kenntnisse auf und vermittelt Ihnen die Grundlagen für den professionellen Einsatz von Objekten in Access.

16.1 Abstrakte Datentypen, Klassen und Objekte

»One way of thinking of a class is as an abstract data type plus inheritance and polymorphism.« (Steve McConnell in *»Code Complete 2«*, Microsoft Press)

Steve McConnell beschreibt *abstrakte Datentypen* im oben genannten Buch als eine Sammlung von Daten und Operationen, die mit diesen Daten arbeiten, wobei die Operationen der Anwendung den Zugriff auf die enthaltenen Daten und deren Änderung erlauben.

Nimmt man das einleitende Zitat hinzu und zieht die fehlenden Möglichkeiten der Vererbung und Polymorphie unter VBA ab, ergibt sich Folgendes: Unter VBA entspricht ein *abstrakter Datentyp* einer Klasse.

Außen vor bleibt dabei die Möglichkeit der Schnittstellenvererbung unter VBA (mehr dazu in »Schnittstellen und Vererbung« ab Seite 918). Ein abstrakter Datentyp ist im Gegensatz zu Basisdatentypen wie String, Integer oder Long oder zusammengesetzten Datentypen (etwa aus Basisdatentypen zusammengesetzten Strukturen) eine Beschreibung einer Schnittstelle zu Daten oder Datenstrukturen und deren Operationen.

Die Betonung liegt dabei auf Beschreibung, denn ein abstrakter Datentyp ist unabhängig von der Implementierung in einer konkreten Programmiersprache. Ein abstrakter Datentyp zeichnet sich außerdem durch folgende Eigenschaften aus:

» Die Kapselung sorgt für das Verbergen der Realisierung der enthaltenen Operationen.

» Die Kapselung verhindert unkontrollierte Zugriffe auf die enthaltenen Daten und sorgt damit für deren Integrität; die Daten können nur über die definierte Schnittstelle geändert werden.

» Der abstrakte Datentyp ist universell einsetzbar und unabhängig von der Implementation.

»Abstrakt« sind abstrakte Datentypen, weil sie reale Objekte modellieren und dabei nur ihre wichtigsten Eigenschaften und Funktionen berücksichtigen. Durch Abstrahieren werden komplexe Strukturen und Zusammenhänge vereinfacht; erst dadurch lassen sich komplizierte Objekte datentechnisch abbilden.

Eine Klasse ist eine Implementierung eines abstrakten Datentyps in einer bestimmten Programmiersprache wie beispielsweise VBA. Die Implementierungsdetails werden dabei in einem Klassenmodul festgelegt. Die Gemeinsamkeiten zwischen dem abstrakten Datentyp und der Klasse beschränken sich dabei auf die fest definierte Schnittstelle, die aus den Methoden und Eigenschaften besteht. Die Implementierung kann und wird vermutlich in jeder Programmiersprache anders aussehen. Die Klasse kann neben den für die Realisierung der Schnittstelle notwendigen Methoden und Eigenschaften natürlich

auch private Variablen, Funktionen und Prozeduren enthalten. Mehr dazu erfahren Sie in »Klassenmodule« ab Seite 891.

16.2 Objekte

Wenn Sie einige Erfahrung mit VBA haben (was wahrscheinlich ist, wenn Sie sich bis hierhin durchgeschlagen haben), sind Sie vermutlich mit der Verwendung von Objekten innerhalb von VBA-Routinen vertraut.

Dennoch finden Sie noch einmal eine Zusammenfassung der dabei verwendeten Techniken und einige Hinweise, wie Sie häufige Fehlerquellen umgehen.

16.2.1 Eingebaute Objekte

Im Urzustand enthält eine Access 2010-Datenbank Verweise auf vier Bibliotheken, die einige für die Programmierung benötigte Objekte zur Verfügung stellen:

» *Visual Basic for Applications*

» *Microsoft Access 14.0 Object Library*

» *OLE Automation*

» *Microsoft Office 14.0 Access database engine Object Library*

Die Methoden, Eigenschaften und Ereignisse dieser Objekte stehen jederzeit zur Verfügung, ihre Verwendung erfolgt ohne vorheriges Instanzieren. Zum größten Teil greift man vermutlich auf diese Objekte zu, ohne dass man sich im Klaren darüber ist, dass es sich auch bei den herkömmlichen Methoden und Eigenschaften um Teile eines Objekts handelt. Ein Beispiel verdeutlicht dies. Mit der folgenden Anweisung können Sie beispielsweise den aktuellen Datenbankbenutzer herausfinden:

```
Debug.Print CurrentUser
```

Der folgende Aufruf belegt, dass diese Eigenschaft zur *Microsoft Access x.y Object Library* gehört:

```
Debug.Print Access.CurrentUser
```

beziehungsweise

```
Debug.Print Access.Application.CurrentUser
```

Letztere Version ist eigentlich korrekter: *Access.CurrentUser* ist nur möglich, weil Application als GlobalMultiUse-Objekt von Access quasi die Standardeigenschaft von Access ist und daher nicht ausgeschrieben werden muss.

MultiUse-Klassen und GlobalMultiUse-Klassen

Vielleicht haben Sie schon einmal festgestellt, dass man manche Objekte unter VBA ohne Weiteres verwenden kann, während man andere erst instanzieren muss. Der Grund ist, dass es tatsächlich unterschiedliche Arten von zugrunde liegenden Klassen gibt: Die sogenannten *GlobalMultiUse*-Klassen brauchen Sie nicht zu instanzieren – dies geschieht im Hintergrund automatisch, sobald Sie eine Methode oder Eigenschaft der Klasse aufrufen.

Die Eigenschaften und Methoden dieser Klassen stehen ohne Ihr Zutun global zur Verfügung. Beispiele sind *DBEngine*, *DoCmd* oder *Application*. Es gibt auch eine Menge *GlobalMultiUse*-Klassen, die Sie niemals bemerken – so sind beispielsweise alle Datums- und Zeitfunktionen Methoden der Klasse *Datetime*. Wenn Sie neugierig geworden sind, zu welcher Klasse die eine oder andere Eigenschaft oder Methode gehört, zeigen Sie einfach den Objektkatalog an und geben als Suchbegriff den gewünschten Ausdruck ein. Beispiele für Klassen, die Sie zunächst instanzieren müssen, sind Ihnen vermutlich ausreichend geläufig – *Database* und *Recordset* etwa dürften Ihnen schon das eine oder andere Mal über den Weg gelaufen sein.

GlobalMultiUse-Klassen können Sie übrigens auch selbst bauen. Im Vorgriff auf die nachfolgenden Erläuterungen zum Thema Klassenprogrammierung zeigen wir kurz, wie das geht. Erstellen Sie ein Klassenmodul wie in Abbildung 16.1 und speichern Sie es mit der folgenden Anweisung in eine Textdatei:

```
SaveAsText acModule, "clsTestGlobalMultiUse", _
    CurrentProject.Path & "\clsTestGlobalMultiUse.cls"
```

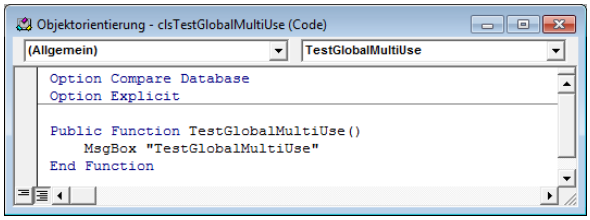


Abbildung 16.1: Beispiel für eine *GlobalMultiUse*-Klasse

Öffnen Sie die gespeicherte Datei dann in einem Texteditor und ändern Sie die Zeile

```
Attribute VB_PredeclaredId = False
```

in diese Zeile:

```
Attribute VB_PredeclaredId = True
```

Danach importieren Sie die geänderte Klasse wieder in die Datenbank:

```
LoadFromText acModule, "clsTestGlobalMultiUse", _
    CurrentProject.Path & "\clsTestGlobalMultiUse.cls"
```

Aktivieren Sie nun mit *Strg + G* das Direktfenster. Wenn Sie nun den Klassennamen eingeben, zeigt IntelliSense verblüffenderweise gleich die einzige öffentliche Funktion dieser Klasse an (siehe Abbildung 16.2).

Mit der folgenden Anweisung lösen Sie diese dann aus:

```
clsTestGlobalMultiUse.TestGlobalMultiUse
```

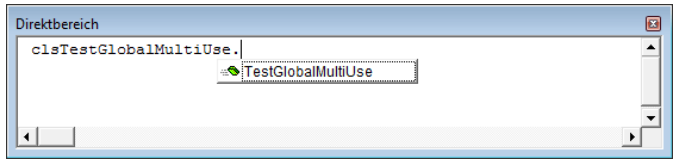


Abbildung 16.2: Methode einer selbst gebauten *GlobalMultiUse*-Klasse

Beispiele für Objekte

Alle Elemente, die in VBA Eigenschaften, Methoden oder Ereignisse bereitstellen, sind Objekte. Manche davon stehen nicht nur für den Zugriff per VBA, sondern direkt in der Benutzeroberfläche zur Verfügung. Dazu gehören beispielsweise:

» Fenster

» Steuerelemente

» Formulare

» Berichte

» Textfelder

» Schaltflächen

Alle genannten Elemente (und natürlich noch mehr) lassen sich auch per VBA ansprechen.

Funktionen zum Instanzieren neuer Objekte

Neben den Methoden und Eigenschaften stellt die Access-Bibliothek wiederum Objekte beziehungsweise Funktionen zur Ermittlung von Verweisen auf die entsprechenden Objekte zur Verfügung.

Wenn Sie beispielsweise *CurrentDb* verwenden, könnte der Eindruck entstehen, dass es sich dabei bereits um ein Objekt handelt. Der Eindruck verstärkt sich, wenn Sie folgendes Beispiel betrachten:

```
Debug.Print Access.CurrentDb.Name
```

Die Anweisung bewirkt die Ausgabe des Verzeichnisses und des Dateinamens der aktuellen Datenbankdatei. Der Zugriff auf die Eigenschaft *Name* erfolgt dabei aber nicht über das »Objekt« *CurrentDb*, sondern über das Objekt, das durch die *CurrentDb*-Methode zurückgeliefert wird. Und dabei handelt es sich wiederum um eine neue Instanz der aktuellen Datenbank.

Man könnte diese »implizite« Instanzierung über Funktionen noch weitertreiben. Die folgende Anweisung gibt die Anzahl Datensätze der angegebenen Tabelle zurück:

```
Debug.Print CurrentDb.OpenRecordset("tblKontakte").RecordCount
```

Diese Anweisung erstellt nicht nur eine neue Instanz der aktuellen Datenbank, sondern innerhalb dieser Instanz auch noch eine neue Datensatzgruppe, um deren Datensatzanzahl auszugeben. Für die Ausgabe zu Testzwecken ist die Verwendung einer solchen Anweisung sicher legitim, innerhalb von Prozeduren sollten Sie diese Anweisungen aber nur für den einfachen Gebrauch einsetzen – hier sind sie dann allerdings auch schneller.

Wenn Sie aber öfter auf ein solches Objekt zugreifen werden, sollten Sie Objektvariablen verwenden und darüber auf die benötigten Eigenschaften und Methoden zugreifen. Die Verwendung der Objektvariablen sieht wie in folgender Routine aus:

```
Public Function DatensaetzeZaehlen()
    'Objektvariablen deklarieren
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    'Objektvariable auf eine bestehende Instanz setzen
    Set db = Access.CurrentDb
    'Neue Instanz eines Objekts erzeugen und
    'per Objektvariable darauf verweisen
    Set rst = db.OpenRecordset("tblKontakte", dbOpenDynaset)
    Debug.Print rst.RecordCount
    '… weitere Aktionen mit dem Recordsetobjekt …
    rst.Close
    'Objektvariablen freigeben
    Set rst = Nothing
    Set db = Nothing
End Function
```

Listing 16.4: Beispiel für das Setzen von Objektvariablen

Wenn Sie die mit *CurrentDB* erzeugte Instanz der aktuellen Datenbank mit einer Objektvariablen des Typs *Database* referenzieren, haben Sie einen entscheidenden

Vorteil: Sie können die Objektvariable anschließend wieder auf den Wert *Nothing* setzen und gehen nicht das Risiko ein, dass die Instanz über den gewünschten Zeitraum hinaus existiert. Gleiches gilt für das im Beispiel erzeugte *Recordset*-Objekt: Dieses können Sie durch die Verwendung einer Objektvariablen erstens schließen (entfernt die Referenz auf die Datensätze daraus) und zweitens können Sie die Objektvariable leeren (gibt belegten Speicherplatz frei).

Auflistungen

Viele Objekte lassen sich über Auflistungen ansprechen. Das *Database*-Objekt enthält beispielsweise eine *TableDefs*-Auflistung, über die man auf alle *TableDef*-Objekte zugreifen kann. Auflistungen stellen je nach Typ unterschiedliche Elemente zur Verfügung – in diesem Fall eine *Count*-Eigenschaft zur Ausgabe der Anzahl der enthaltenen Objekte und die drei Methoden *Append, Delete* und *Refresh*. Auf die Elemente von Auflistungen können Sie meist auf unterschiedliche Art zugreifen. Wenn Sie den Namen des Elements kennen, können Sie ihn in Klammern angeben, um auf das Objekt zuzugreifen:

```
Debug.Print <Database-Objekt>.TableDefs("tblKontakte").RecordCount
```

Eine andere Möglichkeit bietet der Index der jeweiligen Auflistung:

```
Debug.Print CurrentDb.TableDefs(0).RecordCount
```

Die folgenden beiden Prozeduren enthalten Beispiele für die Verwendung von Auflistungen.

```
Public Sub TableDefsAuflisten_I()
    Dim db As DAO.Database
    Dim tdf As TableDef
    Set db = CurrentDb
    For Each tdf In db.TableDefs
        Debug.Print tdf.Name
    Next tdf
    Set db = Nothing
End Sub

Public Sub TableDefsAuflisten_II()
    Dim db As DAO.Database
    Dim intAnzahl As Integer
    Set db = CurrentDb
    intAnzahl = db.TableDefs.count
    For i = 0 To intAnzahl
        Debug.Print db.TableDefs(i).Name - 1
```

```
    Next i
    Set db = Nothing
End Sub
```

Listing 16.5: Beispiele für die Verwendung von Auflistungen

16.2.2 Erzeugen eines Objekts

Wie Sie bereits oben erfahren haben, lassen sich manche Objekte über implizite Methoden wie die *CurrentDB*-Methode oder die *OpenRecordset*-Methode erzeugen. In manchen Fällen ist allerdings eine explizite Instanzierung erforderlich. Die ADODB-Bibliothek erfordert im Vergleich zur DAO-Bibliothek grundsätzlich die explizite Instanzierung über die Anweisung *New*:

```
Public Sub ExpliziteInstanzierung()
    Dim cnn As ADODB.Connection
    Dim rst As ADODB.Recordset
    'Verweis auf existierendes Connection-Objekt
    Set cnn = CurrentProject.Connection
    'explizite Instanzierung
    Set rst = New ADODB.Recordset
    rst.Open "tblKontakte", cnn, adOpenDynamic, adLockOptimistic
    Debug.Print rst.RecordCount
    rst.Close
    Set rst = Nothing
    Set cnn = Nothing
End Sub
```

Listing 16.6: Beispiel für implizite und explizite Instanzierung von Objektvariablen

16.2.3 Zugriff auf die Methoden, Eigenschaften und Ereignisse eines Objekts

Die Methoden und Eigenschaften von Objekten können Sie über den Punkt-Operator ansprechen.

Eigenschaften

Der Zugriff auf die Eigenschaften eines Objekts kann prinzipiell lesend und schreibend erfolgen, aber auch auf eine der beiden Möglichkeiten beschränkt sein. Das Schreiben eines Wertes in eine Eigenschaft erfolgt mit folgender Syntax:

```
<Objekt>.<Eigenschaft> = <Neuer Wert>
```

Beispiel für das Füllen eines Textfeldes im aktuellen Formular (Schlüsselwort *Me*) mit einer neuen Zeichenkette:

```
Me!txtBeispiel.Value = "Text"
```

Da es sich bei der Eigenschaft *Value* um die Default-Eigenschaft von Textfeldern handelt, können Sie diese Anweisung auch abkürzen:

```
Me!txtBeispiel = "Text"
```

Das Lesen einer Eigenschaft eines Objekts sieht ähnlich aus:

```
<Objekt>.<Eigenschaft>
```

Folgende Beispielanweisung liest den Inhalt eines Textfeldes im aktuellen Formular und gibt ihn im Direktfenster aus:

```
Debug.Print Me.txtBeispiel
```

Etwas anders sieht es aus, wenn die Eigenschaft einen Verweis auf ein Objekt enthält.

Das Setzen einer solchen Eigenschaft auf ein Objekt folgt dieser Syntax:

```
Set <Objekt>.<Eigenschaft> = <Objekt>
```

Methoden

Auch Methoden werden über die Punkt-Syntax referenziert:

```
<Objekt>.<Methode>
```

Wie bei herkömmlichen VBA-Routinen verbergen sich auch hinter den Methoden eines Objekts *Function*- und *Sub*-Prozeduren mit oder ohne Parameter.

Grundsätzlich ruft man *Sub*-Methoden mit Parameter ohne Klammern auf:

```
<Objekt>.<Methode> <Parameterliste>
```

Function-Methoden sollen Werte zurückliefern, also muss man die Syntax mit Klammern verwenden:

```
<Variable> = <Objekt>.<Methode>(<Parameterliste>)
```

16.2.4 Lebensdauer eines Objekts

Objekte beginnen ihr Dasein mit der Instanzierung. Wie lange ein Objekt »lebt«, hängt von zwei Faktoren ab – dem Gültigkeitsbereich und einer eventuellen manuellen Zerstörung.

Es gibt folgende Gültigkeitsbereiche:

» Global: Die globale Gültigkeit wird in einem Standardmodul über die Deklaration einer öffentlichen Objektvariablen erreicht. Der Gültigkeitsbereich endet entweder mit der manuellen Zerstörung der Variablen oder mit dem Beenden der Anwendung. Daneben führen unbehandelte Fehler im VBA-Projekt ebenfalls dazu, dass Objekte ihren Inhalt verlieren.

» Modulweit: Die Objektvariable wird in einem Klassenmodul oder Formular-/Berichtsmodul deklariert. Der Gültigkeitsbereich beginnt mit dem Instanzieren eines Objekts auf Basis des Klassenmoduls beziehungsweise mit dem Öffnen eines Formulars oder Berichts oder dem Instanzieren des jeweiligen Moduls.

» Prozedurweit: Die Objektvariable wird innerhalb der Prozedur deklariert und lebt nur so lange, bis die Prozedur abgearbeitet ist.

16.3 Klassenmodule

Wenn Sie in Ihrer Anwendung mit benutzerdefinierten Objekten arbeiten möchten, müssen Sie zunächst entsprechende Klassenmodule anlegen und darin die Eigenschaften und Methoden festlegen, die das Objekt zur Verfügung stellen soll.

16.3.1 Anlegen eines Klassenmoduls

Das Anlegen eines Klassenmoduls mit seinen Eigenschaften und Methoden lernen Sie am Beispiel eines Bankkontos kennen. Ein Klassenmodul legen Sie so an:

» Im Access-Fenster: Betätigen Sie den Ribbon-Eintrag *Erstellen|Andere|Makro/Klassenmodul*.

» Im VBA-Editor verwenden Sie entweder denselben Menüeintrag *Einfügen|Klassenmodul* oder den gleichnamigen Kontextmenüeintrag des Projektexplorers (siehe Abbildung 16.3). Den Projektexplorer aktivieren Sie am schnellsten mit *Strg + R*.

16.3.2 Benennen des Klassenmoduls

Das neue Klassenmodul erhält automatisch den Namen *Klasse1* (außer, dieser Name ist bereits vergeben – dann wird statt der *1* die nächst höhere noch nicht verwendete Zahl angehängt). Diesen Namen ersetzen Sie natürlich durch eine sinnvollere Bezeichnung, die aus dem Präfix *cls* (im Gegensatz zum Präfix *mdl* für Standardmodule) und dem Singular des Namens des beschriebenen Objekts besteht. Im vorliegenden Fall soll ein Objekt mit den Eigenschaften und Methoden eines Kontos erzeugt werden, also vergeben Sie für das entsprechende Klassenmodul den Namen *clsKonto*. Um den Modulnamen zu ändern, müssen Sie das Modul zunächst speichern (*Strg + S*). Im nun erscheinenden

Speichern unter-Dialog tragen Sie dann den gewünschten Namen ein. Zum späteren Ändern des Namens eines Klassenmoduls gibt es zwei Möglichkeiten:

» Markieren Sie im Navigationsbereich den zu ändernden Eintrag und drücken Sie *F2* oder

» ändern Sie den Namen direkt im VBA-Editor, indem Sie das Eigenschaftsfenster aktivieren (*F4*) und dort die Änderung vornehmen.

Beide Varianten setzen das vorherige Speichern des Klassenmoduls voraus.

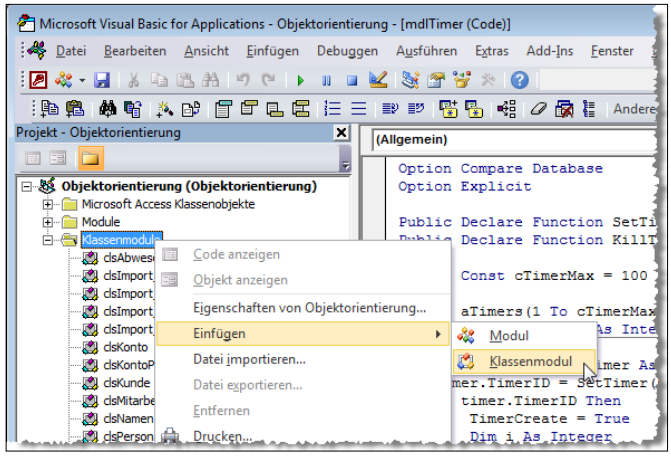


Abbildung 16.3: Einfügen eines neuen Klassenmoduls

16.4 Eigenschaften einer Klasse

Bevor Sie Eigenschaften und Methoden im Klassenmodul anlegen, empfiehlt sich ihre Skizzierung in Form eines Klassendiagramms wie in Abbildung 16.4. Das scheint im vorliegenden Fall vielleicht ein wenig übertrieben, aber es ist sicher kein Fehler, sich vor dem Programmieren noch einmal mit dem Plan auseinanderzusetzen. Die Kontoklasse hat die vier Eigenschaften *Besitzer, Kontonummer, Kontostand* und *Dispositionsrahmen* sowie die beiden Methoden *Einzahlen()* und *Auszahlen()*. Den Besitzer könnte man auch in einer eigenen Klasse modellieren, aber für dieses einfache Beispiel soll sein Name in der Kontoklasse gespeichert werden.

16.4.1 Öffentliche und nicht öffentliche Eigenschaften

Das Klassenmodul enthält für jede Eigenschaft der Klasse eine Variable. Eine Variable, die von außen änderbar sein soll, deklariert man in Standardmodulen als öffentliche

Variable. Der Inhalt des Klassenmoduls *clsKonto* würde dann folgendermaßen aussehen:

```
Option Compare Database
Option Explicit

Public Besitzer As String
Public Kontonummer As Long
Public Kontostand As Currency
Public Dispositionsrahmen As Currency
```

Listing 16.7: Kontoklasse mit öffentlichen Eigenschaften

Abbildung 16.4: Die Kontoklasse im Überblick

Die Eigenschaften des Kontos wären dann von allen Stellen aus les- und schreibbar, von denen man auch auf das Objekt zugreifen könnte.

Sie könnten beispielsweise die folgenden Anweisungen im Testfenster (*Strg + G*) absetzen und damit den Kontostand lesen und auf einen beliebigen Wert ändern (Details über das Instanzieren eines Objekts und den Zugriff auf dessen Eigenschaften finden Sie in »Objekte« ab Seite 884):

```
Set objKonto = New clsKonto
objKonto.Kontostand = 100
Debug.Print objKonto.Kontostand
```

Die *Debug.Print*-Anweisung gibt im Testfenster den Wert *100* als aktuellen Kontostand aus, der zuvor manuell auf diesen Wert eingestellt wurde.

Die Kapselung wird mit der öffentlichen Deklaration der Variablen wie mit einem Vorschlaghammer zerstört.

Das wird gerade im Beispiel des Bankkontos deutlich: Auch sorgfältig festgelegte Geschäftsregeln können nicht greifen, wenn man sie von außen umgehen kann – was hier der Fall ist..

In diesem Beispiel könnte man etwa den Kontostand so verändern, dass er nicht mehr mit dem Dispositionsrahmen vereinbar ist.

16.4.2 Zugriff auf die Eigenschaften einer Klasse kontrollieren

Sie müssen die Eigenschaften also auf irgendeine Weise vor dem direkten Zugriff von außen schützen und damit die Kapselung realisieren. Dazu sind zwei Schritte erforderlich.

» Deklarieren Sie die Variablen der Klasse mit dem Schlüsselwort *Private* und verhindern Sie so den direkten Zugriff von außen.

» Erstellen Sie öffentliche Methoden für den kontrollierten Zugriff auf die privaten Eigenschaften.

Schritt 1 lässt sich leicht in die Tat umsetzen. Das folgende Listing enthält die als *Private* deklarierten Eigenschaften. Außerdem hat jede Eigenschaft ein *m* als Präfix erhalten.

Damit kennzeichnet man die *Member*-Variablen eines Objekts – das sind Variablen, die zwar privat sind, aber über entsprechende Prozeduren gelesen oder geschrieben werden können.

```
Private mBesitzer As clsKunde
Private mKontonummer As Long
Private mKontostand As Currency
Private mDispositionsrahmen As Currency
```

Listing 16.8: Deklaration privater Variablen in einem Klassenmodul

Nachdem die Variablen nun vor dem unkontrollierten Zugriff von außen geschützt sind, können Sie die Möglichkeit zum Lesen und Schreiben der Variablen nach Bedarf freigeben. Dazu verwenden Sie sogenannte Eigenschaftsprozeduren. Es gibt drei unterschiedliche Arten von Eigenschaftsprozeduren:

» Eine *Property Get*-Prozedur gibt je nach dem Variablentyp einen Verweis auf ein Objekt oder den Wert der Variablen zurück.

» Eine *Property Let*-Prozedur schreibt den als Parameter übergebenen Wert in die angegebene Variable. Diese Prozedurart können Sie nur verwenden, wenn die Variable einen konkreten Wert erhält.

» Eine *Property Set*-Prozedur weist der angegebenen Variablen einen Verweis auf das als Parameter übergebene Objekt zu. Diese Prozedurart ist das Pendant für die *Property Let*-Prozedur für Objektvariablen.

Skalare Variable versus Objektvariable

Bei der Verwendung von Eigenschafts-Prozeduren für den Zugriff auf skalare Variablen und Objektvariablen gibt es einige Unterschiede. Der Grund sind Unterschiede zwi-

schen den Datentypen selbst. Damit Sie jeweils den richtigen Fall auswählen, finden Sie nachfolgend eine kurze Erläuterung und Einordnung dieser beiden Variablenarten. Skalare Variablen sind Variablen mit eingebauten Datentypen wie String, Integer oder Long, Aufzählungstypen und benutzerdefinierte Datentypen. Diese Datentypen haben gemein, dass sie konkrete Werte enthalten und je nach Datentyp den entsprechenden Speicherplatz reservieren. Die Zuweisung von Werten an solche Datentypen erfolgt durch Verwendung des Gleichheitszeichens und des entsprechenden Wertes:

```
intZahl = 1
```

Im Gegensatz dazu enthalten Objektvariablen kein Objekt, sondern lediglich einen Verweis darauf. Das bringt einige Besonderheiten mit sich:

» Speicherplatz wird nach der Deklaration nur für den Verweis selbst reserviert (Zeiger des Datentyps *Long*).

» Der Verweis auf das Objekt erfordert die Verwendung des Schlüsselwortes *Set*:

```
Set objKunde = New clsKunde
```

» Es können auch mehrere Objektvariablen auf dasselbe Objekt verweisen:

```
Set objKunde = New clsKunde
Set objPartner = objKunde
```

» Vergleiche zweier Objektvariablen erfolgen über den Operator *Is*:

```
If objKunde Is objPartner Then
    MsgBox "Kunde und Partner sind gleich."
End If
```

» Die Prüfung, ob eine Objektvariable auf ein Objekt verweist, erfolgt über den Vergleich mit dem Operator *Is* und dem Vergleichswert *Nothing*:

```
If objKunde Is Nothing Then
    MsgBox "Kunde ist nicht gesetzt."
End If
```

» Das Aufheben des Verweises erfolgt beim Verlassen des Gültigkeitsbereichs (also etwa beim Beenden der Prozedur, in der der Verweis gesetzt wurde) oder durch explizites Setzen des Verweises auf den Wert *Nothing*.

Tipp: Setzen Sie jede erzeugte Objektvariable per Code wieder auf den Wert *Nothing*. Manchmal benötigt man einen Verweis auf eine Objektvariable längst nicht mehr, obwohl ihr Gültigkeitsbereich noch nicht verlassen wurde.

Zu Gunsten einer ressourcenschonenden Programmierweise sollten Sie eine Objektvariable daher auf den Wert *Nothing* setzen, sobald sie nicht mehr benötigt wird.

17

Sicherheit von Access-Datenbanken

Wenn bei Access-Datenbanken von »Schutz« oder »Sicherheit« die Rede ist, kann dies ganz unterschiedliche Gründe haben. Der Entwickler beispielsweise möchte seinen in langen Nächten produzierten Code vor den Blicken des Kunden verbergen, um sich Exklusivrechte an eventuellen Erweiterungen zu sichern, der Lottospieler schützt die Datenbank zum Ermitteln des sicheren Sechsers per Kennwort vor seinen Kollegen und der Geschäftsführer einer Firma möchte selbst auf alle Daten zugreifen, aber den Mitarbeitern nur die jeweils relevanten Daten zeigen. Eines vorneweg: Access 2010 unterstützt wie bereits Access 2007 in *.accdb*-Dateien nicht mehr das alte Sicherheitssystem von Access, das Sicherheit auf Gruppen- oder Benutzerebene liefern sollte. Die Betonung liegt hier auf »sollte«, denn es war seit langem bekannt, dass dieses Sicherheitssystem leicht ausgehebelt werden konnte. Aus diesem Grund hat sich Microsoft entschlossen, dieses System zum Schutz der Daten komplett zu entsorgen. Schade ist dies für Entwickler,

die das Sicherheitssystem weniger zur Gewährleistung der Sicherheit, als vielmehr zur Benutzer- und Gruppenverwaltung eingesetzt haben (etwa, um je nach Benutzer den Zugriff auf bestimmte Formulare, Funktionen oder Berichte freizugeben).

BEISPIELDATENBANKEN

Den Download mit dem Beispielen finden Sie unter dem Link *www.acciu.de/aeb2010*. Die Datenbanken zu diesem Kapitel liegen in der *.zip*-Datei *Sicherheit.zip*.

In den nächsten Abschnitten finden Sie Näheres zu den folgenden Aufgabenstellungen:

» Verbergen des Codes durch Umwandlung in eine *.accde*-Datenbank

» Schützen des Codes durch ein Kennwort

» Schützen der Datenbank durch Aktivieren des Kennwortschutzes

» Verschlüsseln einer Datenbank

» Kein Sicherheitssystem – was nun?

17.1 Datenbank schützen

Der erste Teil dieses Kapitels beschäftigt sich mit den Maßnahmen, die Sie zum Schutz Ihrer Datenbankdatei treffen können.

17.1.1 Code schützen per .accde-Datenbank

Wer viel Zeit und Hirnschmalz in die Entwicklung einer Anwendung gesteckt hat, möchte diese möglicherweise nicht mit frei zugänglichem Quellcode weitergeben. Access bietet hier einen sehr zuverlässigen Schutz: Dabei wandeln Sie die Datenbank einfach in eine sogenannte *.accde*-Datei um und verhindern so den Zugriff auf den in Formular-, Berichts-, Klassen- und Standardmodulen enthaltenen Quellcode.

Die Umwandlung ist äußerst einfach: Sie zeigen einfach nur den Backstage-Bereich an, wechseln zur Registerseite *Speichern und veröffentlichen* und klicken dann doppelt auf *ACCDE erstellen* (siehe Abbildung 17.1). Dann geben Sie im folgenden Dialog den Namen der zu erstellenden *.accde*-Datei und gegebenenfalls einen alternativen Pfad ein.

Einen kleinen Haken hat die Sache allerdings, wenn Sie mit Access 2010 arbeiten, aber eine Datei im Format von Access 2000, 2002 oder 2003 geöffnet haben. Access bietet dann im Ribbon zwar einen Eintrag an, um eine Datenbank in eine MDE-Datei zu konvertieren, allerdings funktioniert dieser nicht. Stattdessen erscheint die Meldung, dass

Sie die Datenbank für das Konvertieren zunächst in eine *.accdb*-Datenbank umwandeln müssen – das ist nicht ganz konsistent, denn dann erhält man eine *.accde*-Datenbank und keine *.mde*-Datenbank. Wie auch immer: Wenn Sie eine mit einer älteren Access-Version erstellte Datenbank in eine *.mde*-Datenbank umwandeln möchten, brauchen Sie die jeweilige Access-Version. Access 2010 erlaubt wie die Vorgängerversionen nur die Umwandlung von Datenbanken im gleichen Format (in diesem Fall das gemeinsame Format von Access 2007 und 2010). Wenn Sie die Datenbank im Access 2000- oder Access 2002/2003-Format weitergeben möchten, benötigen Sie die entsprechende Access-Version für die Konvertierung in eine *.mde*-Datenbank.

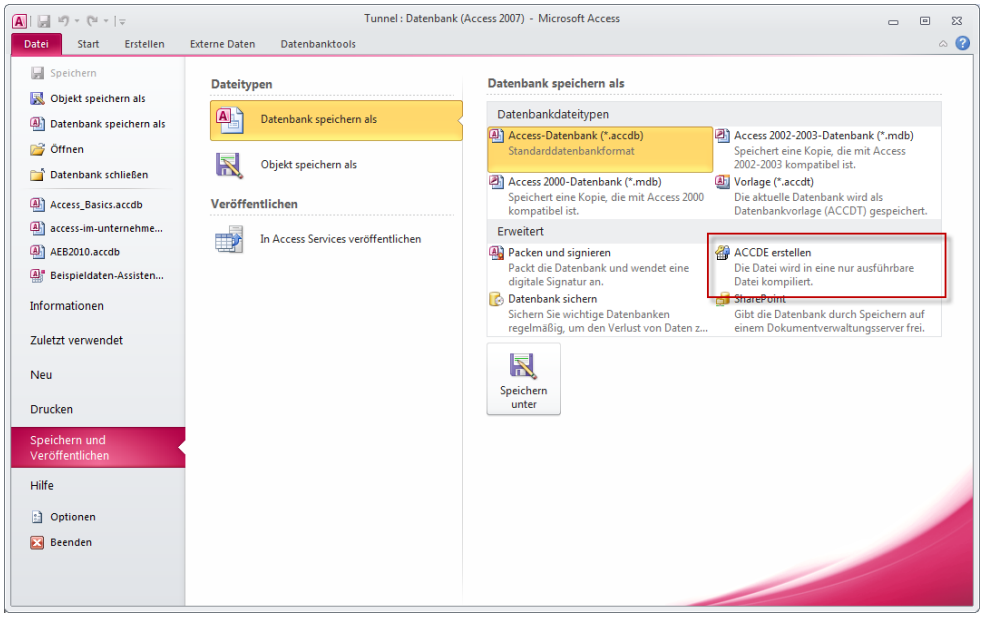


Abbildung 17.1: Erstellen einer .accde-Datenbank

Nach dem Konvertieren ist die Entwurfsansicht von Formularen, Berichten, Seiten, Makros und Modulen gesperrt und außer Makros lässt sich auch keines der genannten Objekte neu anlegen. Lediglich Tabellen und Abfragen entziehen sich der Sperrung der Entwurfsansicht – sie sind offen zugänglich und können auch neu angelegt werden.

Um keine falschen Hoffnungen zu wecken, deaktiviert Access auch alle Möglichkeiten zum Kopieren oder Exportieren (siehe Abbildung 17.2).

Löschen Sie nach der Erstellung der *.accde*-Datenbank auf gar keinen Fall die Originaldatei. Die *.accde*-Datenbank bietet keine reguläre Möglichkeit zur Rückumwandlung in eine normale Datenbankdatei (möglicherweise gibt es in Zukunft Dienstleister, die ein solches Reengineering gegen Entgelt durchführen).

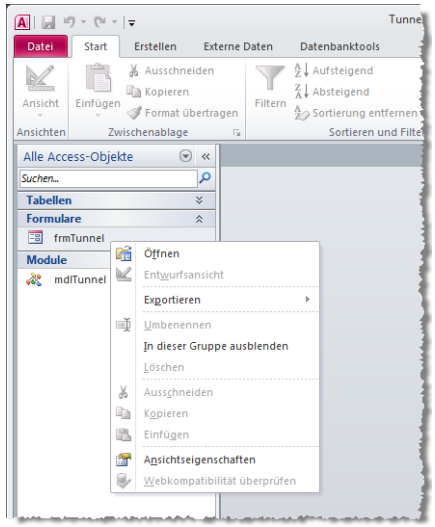


Abbildung 17.2: In .accde-Datenbanken sind die Änderungsmöglichkeiten einiger Objekte stark eingeschränkt

Wenn Sie nur Teile Ihres Codes schützen möchten – etwa die Sammlung Standardfunktionen, die Sie im Laufe der Jahre zusammengetragen haben –, gibt es eine verfeinerte Variante: Speichern Sie die relevanten Objekte in einer separaten Datenbank und binden Sie diese per Verweis in die eigentliche Datenbank ein. Natürlich müssen Sie die separate Datenbank zuvor in eine *.accde*-Datenbank umwandeln. Auf diese Weise kann der Benutzer zumindest die anwendungsspezifischen Objekte wie Formulare und Berichte bearbeiten beziehungsweise eigene Objekte hinzufügen.

17.1.2 Code schützen per Kennwort

Wenn Sie das VBA-Projekt nicht direkt als *.accde*-Datei weitergeben wollen, sondern die Möglichkeit offenhalten möchten, noch einmal Änderungen daran durchzuführen, können Sie dem VBA-Projekt und den enthaltenen Modulen auch ein Kennwort zuweisen. Dazu öffnen Sie im exklusiven Modus den Dialog *<Projektname> – Projekteigenschaften* des VBA-Editors mit dem Menübefehl *Extras|Eigenschaften von <Projektname>*, wobei *<Projektname>* immer für den jeweiligen Projektnamen steht.

Im Dialog wechseln Sie zur Registerseite *Schutz*, haken das Kontrollkästchen *Projekt für die Anzeige sperren* an und geben zweimal das gleiche Kennwort ein (siehe Abbildung 17.3).

Nach dem nächsten Öffnen der Datenbank müssen Sie das Kennwort eingeben, bevor Sie auf die enthaltenen Module zugreifen können.

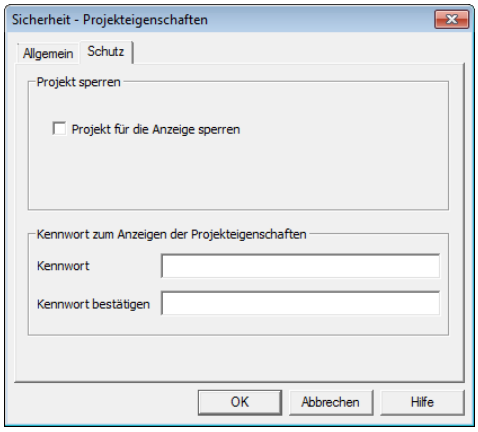


Abbildung 17.3: Dialog zum Anlegen eines Kennworts für ein VBA-Projekt

17.1.3 Einfacher Kennwortschutz mit Verschlüsselung

Für den schnellen Rundumschutz bietet Access einen einfachen Kennwortschutz, der seit der Version 2007 mit einer Verschlüsselung der Datenbank einhergeht. Der Verschlüsselungsalgorithmus wurde mit Access 2010 noch verbessert. Damit verhindern Sie das Öffnen einer Datenbank durch Unbefugte und sorgen gleichzeitig dafür, dass ihr Inhalt nicht mit einem Hex-Editor gelesen werden kann.

Um ein Kennwort für eine Datenbank anzulegen, wechseln Sie wiederum zum Backstage-Bereich der Datenbank. Dort zeigen Sie die Registerseite *Informationen* an und klicken auf die Schaltfläche *Mit Kennwort verschlüsseln* (siehe Abbildung 17.4). Im nun erscheinenden Dialog geben Sie das Kennwort zweimal ein (siehe Abbildung 17.5).

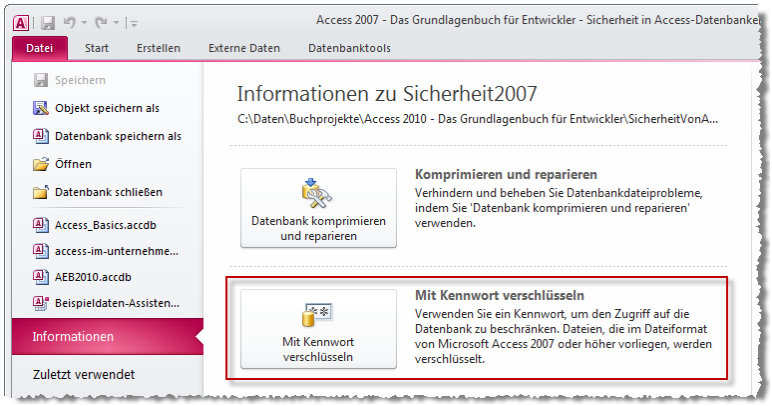


Abbildung 17.4: Verschlüsseln einer Datenbank mit Kennwort

Beim nächsten Öffnen der Datenbank erscheint ein Dialog, der den Benutzer zur Eingabe des Datenbankkennworts auffordert und bei Eingabe eines falschen Kennworts den Zugang verwehrt.

Das Zuweisen eines Datenbankkennworts erfordert, dass die Datenbank im Exklusiv-Modus geöffnet ist.

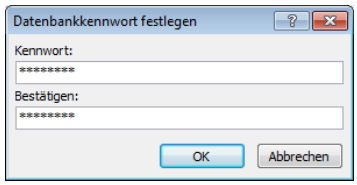


Abbildung 17.5: Zuweisen eines Datenbankkennworts

Beachten Sie, dass bei verschlüsselten Datenbanken beim Zugriff auf die Daten eine Sperrung auf Seitenebene verwendet wird, was sich beim Einsatz in einer Mehrbenutzerumgebung so auswirken kann, dass mehrere Datensätze einer Tabelle gleichzeitig gesperrt werden.

17.1.4 Kein Sicherheitssystem — was nun?

Wer mehr möchte, als einfach nur den Zugriff auf Daten zu verhindern – etwa um verschiedenen Benutzern und Benutzergruppen unterschiedliche Berechtigungen für die einzelnen Datenbankobjekte und die enthaltenen Daten zu gewähren, hat bisher das Sicherheitssystem von Access verwendet.

Dieses ist zumindest insoweit nicht mehr vorhanden, als Sie es seit Access 2007 nicht mehr einrichten können. Datenbankanwendungen, die Sie mit einer älteren Version von Access geschützt haben, werden allerdings auch unter Access 2007 und 2010 unter Einsatz des Sicherheitssystems geöffnet.

Sie brauchen also erstens keine Sorge zu tragen, dass jemand einfach so mit der neuen Access-Version auf die Daten Ihrer Anwendung zugreifen kann, und können zweitens immer noch Datenbanken mit älteren Access-Versionen schützen und diese unter Access 2010 einsetzen.

Der einzige Nachteil ist, dass Sie geschützte Datenbanken nicht in das Datenbankformat von Access 2007/2010 konvertieren können.

Möglicherweise sind Sie noch in Besitz einer älteren Version von Access 2003 und möchten einfach eine *.mdb*-Datenbankanwendung für den Einsatz mit Access 2007 erstellen und diese mit dem Sicherheitssystem schützen. Dann können Sie natürlich wie gehabt vorgehen.

Wie das funktioniert, erfahren Sie am Ende dieses Kapitels. Sie sollten sich jedoch im Klaren darüber sein, dass das Sicherheitssystem von Access nicht wirklich sicher ist und dass jemand, der unbedingt an die Daten einer geschützten Datenbank herankommen möchte, dies auch bewerkstelligen kann.

Benutzer- und gruppenabhängige Benutzeroberfläche

Wenn Sie das Sicherheitssystem von Access nur dazu verwenden, Benutzer und Gruppen zu verwalten, damit diese nach der Anmeldung eine individuell auf die jeweilige Benutzergruppe zugeschnittene Benutzeroberfläche verwenden können, schießen Sie mit Kanonen auf Spatzen.

Sie sollten stattdessen drei Tabellen anlegen, in denen Sie Benutzer, Gruppen und die Zuteilung von Benutzern zu Gruppen vornehmen und dort die benötigten Zugriffsdaten speichern.

Fügen Sie Ihrer Datenbankanwendung einen Anmeldedialog hinzu, mit dem die Benutzer sich identifizieren können. Die Daten über den aktuellen Benutzer speichern Sie im einfachsten Fall in einer globalen Variablen, besser aber in einer Klasse oder einer Konfigurationstabelle.

Für die Steuerung der Benutzeroberfläche in Abhängigkeit vom aktuellen Benutzer müssen Sie ja auch unter Verwendung des Sicherheitssystems entsprechende Funktionen bereitstellen, die den aktuellen Benutzer beziehungsweise die Benutzergruppe auslesen und die entsprechenden Elemente der Benutzeroberfläche bereitstellen.

Daten schützen: Alternativen

Wenn Sie Ihre Daten mit einem Sicherheitssystem schützen wollen, das Funktionen zur Verwaltung von Benutzern und Benutzergruppen bereitstellt, verwenden Sie am besten einen SQL-Server zum Verwalten Ihrer Daten. Eine Vielzahl der Hinweise in der Communitiy führt in Richtung von Microsoft-Produkten wie dem Microsoft SQL Server 2008 oder dem kostenlosen Microsoft SQL Server 2008 Express Edition.

Microsoft empfiehlt, von Access aus per ODBC auf die Daten eines SQL-Servers zuzugreifen, es ist aber auch noch möglich, ein Access-Projekt dafür zu verwenden.

Davon abgesehen muss es aber auch gar nicht unbedingt ein Microsoft-Produkt sein: Wenn Sie schon – wie von Microsoft empfohlen – per ODBC auf den SQL-Server zugreifen und damit auf die Vorzüge der Access-Projekte verzichten, können Sie auch direkt eine Alternative wie MySQL oder einen der anderen teilweise frei erhältlichen SQL-Server verwenden.

Da sich dieses Buch allein auf Access konzentriert, werden wir das Thema SQL-Server gar nicht erst anreißen – es würde ohnehin den Rahmen sprengen.

17.2 Sicherheit beim Umgang mit Access

Access-Anwendungen können potenziell großen Schaden auf dem System des Benutzers anrichten. Befehle, die etwa Dateien löschen, lassen sich sowohl unter VBA als auch eingebettet in SQL-Anweisungen ausführen. Der zweite Teil dieses Kapitels beschäftigt sich mit den Maßnahmen, die Sie und die Benutzer Ihrer Anwendungen treffen können, um dies zu verhindern. Grundsätzlich gibt es dabei die folgenden Quellen für gefährlichen Code:

- VBA-Code
- Makros (es gibt allerdings sichere und unsichere Makrobefehle – die sicheren können Sie unabhängig von den Sicherheitseinstellungen immer ausführen. Im Makro-Editor blenden Sie die unsicheren Befehle mit der Schaltfläche *Alle Aktionen anzeigen* ein. Diese werden dann durch ein Blitz-Symbol in der Liste der Aktionen gekennzeichnet – siehe Abbildung 17.6.)
- ActiveX-Steuerelemente
- SQL-Code
- Aktionsabfragen (die etwa Daten löschen oder verändern können)
- Ausdrücke als Eigenschaften etwa von Formularen oder Steuerelementen

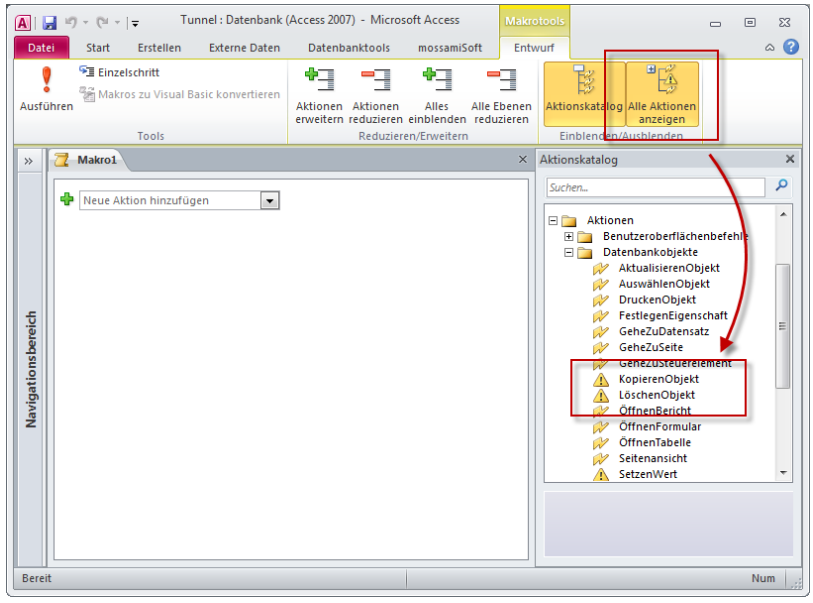


Abbildung 17.6: Sichere und unsichere Makrobefehle

17.2.1 Datenbank öffnen mit Standardeinstellungen

Wenn Sie Access 2010 frisch installiert haben und eine Anwendung öffnen, die VBA-Code enthält, zeigt Access direkt einen Hinweis an (siehe Abbildung 17.7).

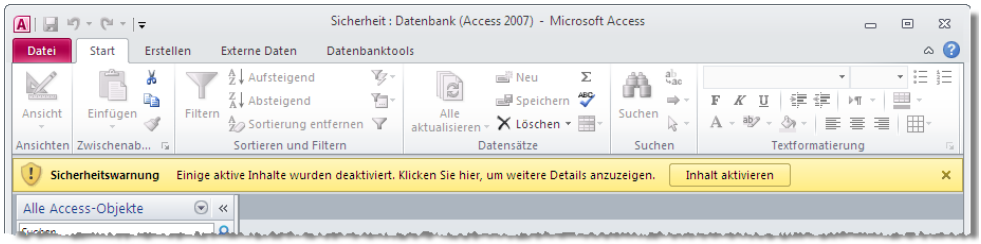


Abbildung 17.7: Access weist auf eine potenziell unsichere Datenbank hin.

Mit einem Klick auf den Link öffnen Sie die Seite *Informationen* im Backstage-Bereich und erfahren, dass die Datenbank VBA-Code enthält, der mit den aktuellen Einstellungen nicht ausgeführt werden kann (siehe Abbildung 17.8).

Abbildung 17.8: Weitere Informationen zu den unsicheren Elementen

Hier haben Sie nun zwei Möglichkeiten (siehe Abbildung 17.9):

» Sie aktivieren alle Inhalte und erklären die Datenbankdatei damit dauerhaft zu einem vertrauenswürdigen Dokument.

» Sie wählen die weiteren Optionen aus. In diesem Fall erscheint der Dialog aus Abbildung 17.10, mit dem Sie entweder alle aktiven Inhalte deaktivieren oder diese für die aktuelle Sitzung aktivieren.

Abbildung 17.9: Aktivieren oder weitere Optionen anzeigen?

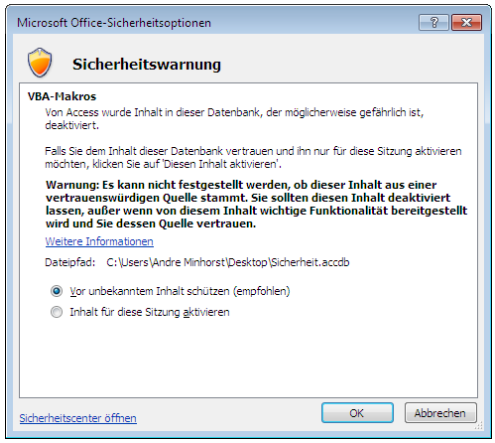


Abbildung 17.10: Weitere Optionen beim Öffnen einer Datenbankdatei

Schlussendlich können Sie sich also entscheiden, ob Sie die Datenbankdatei dauerhaft als unkritisch einstufen möchten, ob Sie dies nur einmal tun möchten oder ob die Datenbankdatei zunächst ohne aktiven Inhalt geöffnet werden soll.

In den folgenden Abschnitten erfahren Sie, welche Auswirkungen die hier durchgeführten Aktionen haben und wie Sie diese manuell im Sicherheitscenter vornehmen können.

17.2.2 Das Sicherheitscenter unter Access 2010

Genau wie unter Access 2007 gibt es einen Dialog, mit dem Sie die sicherheitsrelevanten Einstellungen vornehmen können. Sie öffnen diesen Dialog durch die folgenden Schritte:

» Aktivieren Sie den Backstage-Bereich durch einen Klick auf den Registerreiter *Datei* im Ribbon.

» Klicken Sie links auf *Optionen*.

» Wählen Sie im Optionen-Dialog den Bereich *Sicherheitscenter* aus.

18

Installation, Betrieb und Wartung

Mit der Fertigstellung ist die Arbeit an einer Access-Anwendung noch lange nicht beendet. Die Anwendung muss installiert, komprimiert oder gewartet werden, wobei gerade der erste Schritt manche Schikane birgt. Soll die Anwendung im Mehrbenutzerbetrieb als Einzelplatzlösung eingesetzt werden oder kommt vielleicht sogar Replikation zum Einsatz? Wie sorgen Sie während des Betriebs für das Erstellen von Backups und was hat es mit den oft erwähnten Problemen mit den Verweisen auf sich? Antworten auf diese und weitere Fragen finden Sie in diesem Kapitel.

18.1 Verschiedene Access-Versionen auf demselben Rechner

Normalerweise sollte sich nur eine Access-Version gleichzeitig auf einem Rechner befinden. Das ist zumindest der Optimalfall. In Zeiten der virtuellen Maschinen gibt es auch kaum Gründe, mehr als eine Access-Version einzusetzen. Wer etwa Access 2010 auf der Hauptmaschine verwendet, kann ältere Versionen auf jeweils eine virtuelle Maschine auslagern.

BEISPIELDATENBANKEN

Den Download mit dem Beispielen finden Sie unter dem Link *www.acciu.de/aeb2010*. Die Dateien zu diesem Kapitel liegen in der Datei *InstallationBetriebWartung.zip*.

Es gibt eigentlich nur einen triftigen Grund, mehrere Versionen gleichzeitig zu verwenden: Wenn man Anwendungen von Dritten auf dem Rechner installiert, die verschiedene Access-Versionen voraussetzen. Hier kann es zu Problem kommen, wenn Access 2007 und/oder Access 2010 im Spiel sind – gegebenenfalls kombiniert mit einer älteren Version von Access. Beim wechselnden Einsatz verschiedener Versionen konfiguriert Access nämlich jedesmal das halbe System um, was mitunter nervtötend lange dauern kann.

Wenn Sie etwa einem Kunden eine Access 2010-Datenbank mit der Runtime-Version von Access installieren (mehr zum Thema Runtime weiter unten) und dieser Ihre Anwendung abwechselnd oder gar parallel beispielsweise mit einer Vollversion von Access 2003 einsetzt, wird er nicht gerade begeistert sein – immerhin war ja vor der Installation Ihrer Software alles in Ordnung.

Hier gibt es aktuell nur eine Lösung: Den *Access 2010 Deployment Wizard* von *www.sagekey.com*. Die günstigste Version kostet schlappe 450 $. In Anbetracht dessen, dass die Runtime-Version von Access seit Version 2007 kostenlos ist, sollte ein professioneller Entwickler, der die Weitergabe von Access-Anwendungen mit der Runtime plant, diese Investition einplanen.

Der *Access 2010 Deployment Wizard* liefert zunächst einmal einen Assistenten, der einem beim Zusammenstellen eines Setups (*.exe*-Datei) mit allen notwendigen Einstellungen hilft. Dabei können Sie etwa die zu verwendenden Datenbanken und weitere Dateien auswählen, vorzunehmende Registry-Einstellungen festlegen, das Aussehen des Setups anpassen und vieles mehr. Das so erstellte Setup installiert neben der Runtime und der Datenbankdatei eine weitere Datein namens *StartAccess.exe*. Diese wird durch eine entsprechende Dateiverknüpfung vor dem Aufruf der eigentlichen Datenbankdatei

gestartet und verhindert den Zeitverlust, den Sie sonst beim abwechselnden Starten durch die Rekonfiguration verschiedener Access-Versionen erleiden.

Beim Installieren stellt das mit dem *Access 2010 Deployment Wizard* erstellte Setup außerdem sicher, dass der Benutzer Ihre Anwendung gleich voll einsetzen kann, ohne erst noch die üblichen Sicherheitsdialoge abarbeiten zu müssen.

Starten von Access-Datenbanken unterschiedlicher Versionen

Wenn Sie beispielsweise Access 2003 und Access 2007/2010 auf einem Rechner installiert haben, gibt es beim Aufruf von Datenbankdateien per Doppelklick manchmal das Problem, dass die falsche Version von Access geöffnet wird. Mit Anwendungen im Access 2007-Format haben Sie das Problem nicht mehr, da diese ja mit *.accdb* eine andere Dateiendung haben und somit automatisch die richtige Version geöffnet wird.

Das Verhalten für *.mdb*-Datenbanken bleibt indes unverändert: Wenn Sie zuletzt eine *.accdb*-Datenbank geöffnet hatten, öffnet Access auch die als nächste geöffnete *.mdb*-Datei mit Access 2007/2010. Im umgekehrten Fall tritt die oben erwähnte zeitliche Verzögerung durch die Umkonfigurierung der Registry und gegebenenfalls eine Reparaturinstallation auf.

Grundsätzlich stehen aber beim Öffnen einer *.mdb*-Datenbank unter Access 2007/2010 alle nötigen Elemente der Benutzeroberfläche bereit – wenn Sie also darauf achten, keine Access 2007/2010-spezifischen Funktionen zu verwenden, können Sie *.mdb*-Datenbanken auch mit den neueren Access-Versionen bearbeiten.

Gelegentlich landen aber dennoch Elemente in der *.mdb*-Datei, die unter älteren Access-Versionen zu Problemen führen – speziell in Formularen. Sie können solche Formulare mit dem unter dem folgenden Link vorgestellten Tool bereinigen: *http://mosstools.de/index.php?option=com_content&view=article&id=103&Itemid=84*

18.2 Weitergabe von Access-Datenbanken

Nicht jeder Access-Entwickler erstellt mit Access ausschließlich Anwendungen für den Eigenbedarf. Die meisten entwickeln Anwendungen für Kunden oder – wenn sie in einem Unternehmen arbeiten – auch für die eigenen Mitarbeiter. Wenn Sie Glück haben, verfügt der künftige Benutzer der neuen Access-Anwendung über Microsoft Access in der einen oder anderen Version. Wenn nicht, gibt es zwei Möglichkeiten: Entweder der Benutzer legt sich eine Lizenz zu oder Sie erstellen ihm eine Runtime-Version.

Für die Runtime-Version gibt es zwei Möglichkeiten: Entweder Sie übermitteln dem Anwender direkt die Runtime-Installationsdatei und kopieren die Anwendung auf seinen Rechner oder Sie erstellen ein Setup, das beide Schritte automatisch durchführt.

18.2.1 Die Runtime-Version von Access

Die Runtime-Version ist prinzipiell eine Vollversion von Access ohne die Elemente der Benutzeroberfläche, mit denen die Anwendung selbst bearbeitet werden kann. Der Benutzer wird aber immerhin ohne weitere Kosten für eine eigene Access-Lizenz in der Lage sein, mit der Access-Anwendung zu arbeiten – so, wie dies schon früher bei anderen Office-Dokumenten mit Tools wie dem Word-, Excel- und PowerPoint-Viewer möglich war. Die Runtime-Version von Access 2010 finden Sie unter *www.acciu.de/runtime*.

18.2.2 Der Paket-Assistent von Access

Der für die Erstellung einfacher Setups notwendige Paket-Assistent ist nicht standardmäßig verfügbar. Sie installieren ihn aber wie folgt (Erläuterung am Beispiel von Windows 7):

» Öffnen Sie die Systemsteuerung und klicken Sie doppelt auf den Eintrag *Programme und Funktionen*.

» Klicken Sie mit der rechten Maustaste auf den Eintrag Microsoft Office Professional Plus 2010 und wählen Sie den Eintrag *Ändern* aus.

» Im nun erscheinenden Dialog selektieren Sie die Option *Features hinzufügen oder entfernen*.

» Navigieren Sie zum Eintrag *Microsoft Access|Add-Ins|Paket-Assistent* und wählen Sie *Von 'Arbeitsplatz' ausführen* aus (siehe Abbildung 18.1).

Wenn Sie nun die zu verpackende Anwendung öffnen und im Backstage zum Bereich *Speichern und veröffentlichen* wechseln, finden Sie dort einen neuen Eintrag namens *Lösung packen* vor (siehe Abbildung 18.2). Im Assistenten wählen Sie zunächst das Zielverzeichnis für die Setup-Datei aus oder laden bereits vorhandene Einstellungen. Danach wählen Sie die zu verpackende Access-Datenbank aus, bei der es sich nicht zwangsläufig um die aktuell geöffnete Datenbank handeln muss. Sie wählen den Zielordner aus, geben ein Unterverzeichnis in diesem Ordner an und legen die Bedingungen für die Installation fest:

» Access muss installiert sein (anderenfalls wird die Anwendung nicht installiert).

» Der Benutzer installiert die Runtime selbst. Dies überfordert den Benutzer gegebenenfalls, zumal er die richtige Version installieren muss (32-bit oder 64-bit).

» Sie liefern die Runtime mit. Dies ist die bessere Variante, weil Sie dem Benutzer somit erstens den Download abnehmen und zweitens selbst entscheiden, welche Version installiert wird.

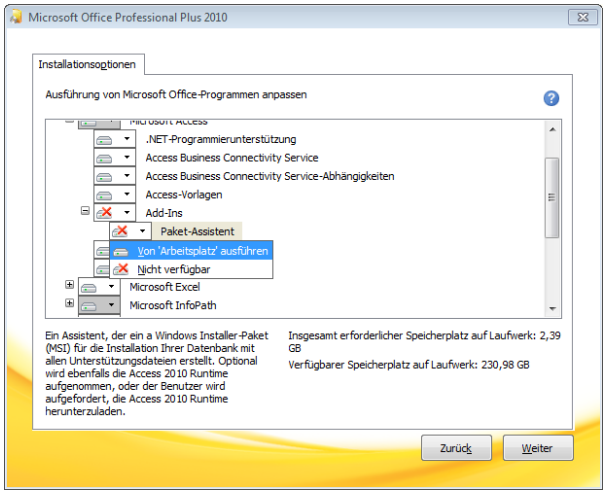


Abbildung 18.1: Installieren des Paket-Assistenten

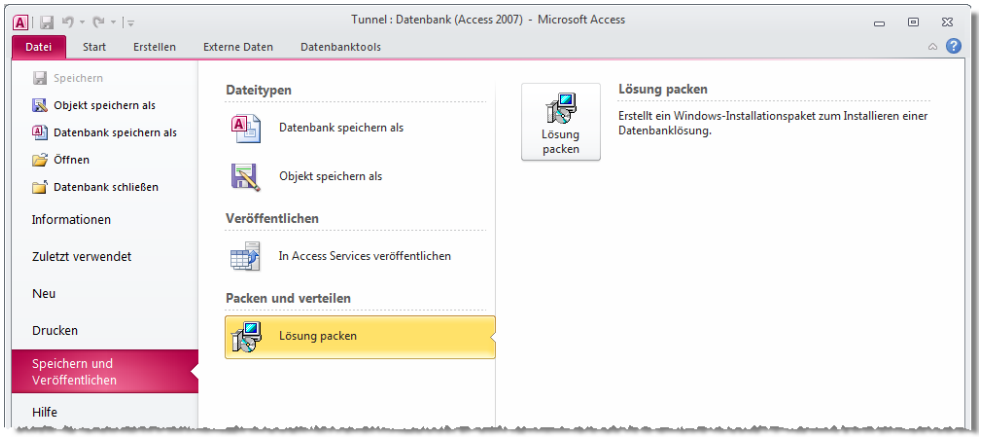


Abbildung 18.2: Erstellen eines Pakets

Unter *Verknüpfungsoptionen* legen Sie fest, wie die Verknüpfung zum Start der Anwendung gestaltet werden soll. Hier gibt es folgende interessante Möglichkeiten:

» Startmakro: Legt den Namen eines Makros fest, das beim Start ausgeführt werden soll

» Wert des VBA-Befehls: Legt weitere Kommandozeilen-Optionen fest (siehe »Access per Verknüpfung öffnen« ab Seite 958)

An dieser Stelle sei die Anmerkung erlaubt, dass Microsoft diesen Dialog ruhig etwas breiter hätte gestalten können, damit man nicht unnötig hin- und herscrollen muss. Vor

allem, weil das rote Sternchen zur Markierung von Pflichtfeldern immer ganz rechts liegt und man sich gelegentlich wundert, warum die *Weiter*-Schaltfläche noch nicht aktiviert ist. In diesem Fall schauen Sie also einfach mal im nicht sichtbaren Bereich nach, ob es noch offene Pflichtfelder gibt ...

Im folgenden Dialog tragen Sie zusätzliche Dateien ein (beispielsweise ein Backend) und legen Registrierungsschlüssel fest, die das Setup beim Ausführen anlegen soll.

Die letzte Dialogseite fragt noch verschiedene Texte ab, die etwa im Setup oder in der Programme-Liste der Systemsteuerung erscheinen.

Mit einem Klick auf *OK* erstellen Sie das Setup und können Ihre Anwendung so weitergeben.

Eine ausführlichere Diskussion der Möglichkeiten und Probleme beim Erstellen von Installationsroutinen für die Runtime-Version von Access ist in diesem Buch aus Platzgründen leider nicht möglich. Die folgenden Tipps sollen aber helfen, Probleme mit der Access-Anwendung selbst zu verhindern.

18.2.3 Benutzerdefinierte Menüs

In Runtimes gibt es keinen Navigationsbereich und auch die eingebauten Ribbon-Tabs und Backstage-Elemente werden nicht angezeigt. Letzteres liegt in der Natur der Sache, denn die Runtime-Version von Access liefert nur die Funktionen, die zum Laufen der Datenbankanwendung selbst (also der Benutzeroberfläche einschließlich Formularen, Berichten und Menüs) und von VBA notwendig sind.

Die wichtigste Konsequenz hieraus ist: Sie müssen selbst Sorge tragen, dass der Benutzer auf irgendeinem Wege die benötigten Formulare und Berichte öffnen beziehungsweise die VBA-Routinen aufrufen kann. Das kann durch ein Formular geschehen, das beim Start der Anwendung angezeigt wird und die möglichen Optionen enthält.

Sinnvoller, da ständig sichtbar, ist eine benutzerdefinierte Anpassung des Ribbons mit Elementen zum Aufrufen von Formularen und Berichten und anderen Funktionen wie dem *Drucken* oder *Beenden* der Datenbankanwendung (weitere Informationen siehe »Ribbon« ab Seite 713).

18.2.4 Fehlerbehandlung

Access-Runtimes sind mit einer äußerst sparsamen Fehlerbehandlung ausgestattet. Das bedeutet, dass im Falle eines Fehlers höchstens eine wenig aussagekräftige Fehlermeldung erscheint (»Die Ausführung dieser Anwendung wurde wegen eines Laufzeitfehlers angehalten. Die Anwendung wird beendet.«) und die Anwendung sich daraufhin rasch verabschiedet.

Das ist nicht nur für den Benutzer unbefriedigend, sondern auch für den Entwickler: Der Benutzer kann dem Entwickler lediglich mitteilen, an welcher Stelle der Fehler aufgetreten ist und was dazu geführt hat.

Es gibt aber weder eine aussagekräftige Fehlermeldung noch Informationen über den fehlerhaften Code. Deshalb sollten Sie gerade in Datenbanken, die Sie als Runtime weitergeben, unbedingt eine umfassende Fehlerbehandlung integrieren. Das gilt natürlich auch für »normale« zur Weitergabe bestimmte Datenbankanwendungen, aber hier ist es besonders wichtig.

Weitere Informationen zum Thema Fehlerbehandlung finden Sie im Kapitel »Debugging und Fehlerbehandlung« ab Seite 803.

18.2.5 Runtime-Simulation

Wer eine Anwendung zur Weitergabe als Runtime-Version entwickelt, muss nicht jedes Mal eine Runtime-Version erstellen und diese neu installieren, um zu prüfen, ob alles läuft wie geplant. Sie können auch einfach die Dateiendung von *.accdb* auf *.accdr* ändern. Es ist also nicht mehr nötig, wie bei älteren Access-Versionen einen Aufruf der *msaccess.exe* mit der gewünschten Datenbank und dem Parameter */runtime* abzusetzen.

Wenn Sie dies ausprobieren, werden Sie schnell feststellen, dass sämtliche Elemente der Benutzeroberfläche mit Ausnahme der Datei-Schaltfläche und einigen Einträgen im Backstage nicht mehr angezeigt werden. Um die Benutzeroberfläche mit Leben zu füllen, erstellen Sie entweder eine passende Ribbon-Definition und weisen diese der Datenbank zu oder stellen ein Übersichtsformular bereit, das beim Öffnen der Datenbank angezeigt wird. Das anzuzeigende Formular können Sie in den Access-Optionen unter *Aktuelle Datenbank|Anwendungsoptionen|Formular anzeigen* einstellen; Informationen zum Erstellen einer geeigneten Ribbon-Definition finden Sie im Kapitel »Ribbon« ab Seite 713.

Die Verwendung einer Datei mit der Endung *.accdr* ersetzt jedoch nicht vollständig den Test mit einer echten Runtime: Vollversion mit *.accdr* und die Runtime-Version reagieren an verschiedenen Stellen unterschiedlich. Eine nur mit der Runtime-Version ausgestattete virtuelle Entwicklermaschine gehört somit zum Pflichtprogramm, wenn Sie Anwendungen auf diese Weise weitergeben möchten.

18.2.6 Weitergabe ohne Runtime

Die Weitergabe von Datenbankanwendungen an Benutzer, die bereits über die benötigte Access-Version verfügen, ist in den meisten Fällen völlig unproblematisch. Am einfachsten ist dies natürlich, wenn Sie nur eine einzige *.accdb*-Datei oder *.accde*-Datei weitergeben müssen – hier spielt es nicht einmal eine Rolle, wo der Benutzer die Anwendung speichert, sofern es keine relativen Referenzen auf andere Dateien gibt.

Einige Anwendungen sind aber etwas aufwändiger: So könnte es beispielsweise sein, dass die Datenbank aus Frontend und Backend besteht und im Frontend noch der neue Standort des Backends angegeben werden muss (weitere Informationen zu Frontend- und Backend-Datenbanken unter »Mehrbenutzerbetrieb mit Access-Datenbanken« ab Seite 962).

Oder die Datenbank verweist auf Bilddateien oder Ähnliches, wozu ein entsprechender Ordner einzurichten ist. Wenn Sie die Installation nicht selbst vor Ort vornehmen, können Sie natürlich eine Dokumentation erstellen, die alle notwendigen Informationen zu den Speicherorten und Verknüpfungen enthält. Das ist aber nicht besonders kundenfreundlich.

Alternativ können Sie ein Setup erstellen, das den Benutzer beim Einrichten der Anwendung unterstützt, indem es beispielsweise das Verzeichnis abfragt, in dem die Dateien installiert werden sollen, oder ob im Startmenü oder auf dem Desktop Verknüpfungen anzulegen sind.

Für diese Aufgaben gibt es sogar kostenlose Tools wie etwa *Inno Setup*. Es bietet praktisch alle Möglichkeiten professioneller Setup-Tools. Sie finden das Tool unter folgender Internetadresse: *http://www.jrsoftware.org*.

In Access-Kreisen hat sich *Inno Setup* einen Namen gemacht, weil es die Erstellung professioneller Runtime-Setups mit Access 97 ermöglicht hat. Für Access 97 gibt es im Internet ausreichend Beispielskripte. Leider ist die Installation von Runtimes neuerer Access-Versionen zu komplex geworden, sodass bisher keine als zuverlässig geltenden Skripte zur Erstellung von Runtimes mit *Inno Setup* und neueren Access-Versionen existieren.

Auch der Windows Installer ist eine interessante Alternative. Einige Informationen finden Sie unter folgender Internetadresse: *http://www.installsite.org*.

Auch der Assistent zum Packen von Lösungen von Access 2010 wartet mit einem passenden Tool auf: Der Verpackungs-Assistent hilft dann beim Erstellen eines MSI-Setups für die Anwendung mit oder ohne Access-Runtime (siehe »Der Paket-Assistent von Access« ab Seite 954).

18.2.7 Access per Verknüpfung öffnen

Wenn Sie eine Access-Anwendung mit einer bestimmten *MSAccess.exe* öffnen möchten oder andere Parameter beim Start übergeben möchten, verwenden Sie die Befehlszeilenparameter von Access in einer entsprechenden Verknüpfung. Eine solche Verknüpfung können Sie als echte Verknüpfung im Windows Explorer anlegen oder aber Sie erstellen eine Textdatei, die Sie dann mit der Dateiendung *.bat* speichern. Ein Doppelklick auf eine solche Datei führt den enthaltenen Befehl aus.

Grundsätzlich benötigen Sie die Angabe der Access-Instanz sowie der zu öffnenden Datei (ohne diese Angabe wird Access allein gestartet). Unter Windows 7 lautet der Speicherort von Access 2010 (32-bit) so:

```
C:\Program Files (x86)\Microsoft Office\Office14\MSAccess.exe
```

Dahinter folgt der Name der Access-Datenbank, also beispielsweise *c:\Beispiel.accdb*.

Außerdem können Sie die folgenden Parameter angeben:

» */runtime*: Öffnet die Datenbank im Runtime-Modus.

» */excl*: Öffnet die Datenbank exklusiv, weitere Instanzen können nur lesend geöffnet werden.

» */ro*: Öffnet die Datenbank schreibgeschützt.

» */user*: Gibt den Namen des Benutzers an – nur interessant bei Verwendung einer Arbeitsgruppen-Informationsdatei in Zusammenhang mit *.mdb*-Datenbanken.

» */pwd*: Gibt das Kennwort für den Benutzer an.

» */x*: Führt das Makro mit dem angegebenen Namen aus (alternativ zum Makros namens *Autoexec*, siehe »Code beim Starten einer Datenbank ausführen« ab Seite 959).

» */wrkgrp*: Gibt an, welche Arbeitsgruppen-Informationsdatei verwendet werden soll.

» */cmd*: Erlaubt die Übergabe eines Parameters, der später im VBA-Code mit der Funktion *Command* ausgelesen werden kann.

18.3 Aktionen beim Starten oder Beenden der Datenbank durchführen

Einige Aktionen wie etwa Komprimieren der Datenbank, Einbinden von Tabellen aus Backend-Datenbanken oder Sichern einer Datenbank sollten regelmäßig durchgeführt werden. Dazu bieten sich der Start oder das Beenden einer Datenbank an. Während das Ausführen von Code oder das Aufrufen eines Formulars beim Starten einer Datenbank recht einfach zu realisieren sind, ist für Aktionen, die beim Beenden der Datenbankanwendung durchgeführt werden sollen, schon ein kleiner Trick notwendig.

18.3.1 Code beim Starten einer Datenbank ausführen

Um eine Routine beim Starten einer Datenbankanwendung auszuführen, benötigen Sie ein Makro namens *Autoexec*. Ein Makro mit diesem Namen wird von Access automatisch

beim Start einer Anwendung ausgeführt – außer natürlich, der Benutzer hält beim Start die *Umschalt*-Taste gedrückt. Das *Autoexec*-Makro aus Abbildung 18.3 ruft die eingebaute Funktion *Meldung* auf. Statt dieser können Sie jede beliebige öffentliche Funktion innerhalb der Datenbank angeben.

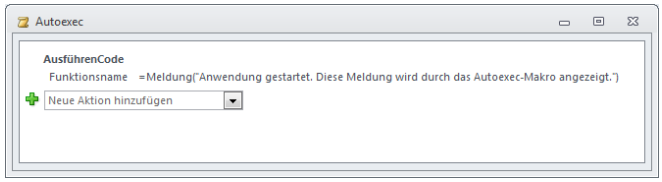


Abbildung 18.3: Dieses Autoexec-Makro zeigt beim Starten einer Anwendung ein Meldungsfenster an

Beachten Sie, dass Ihnen die Sicherheitseinstellungen von Access einen Strich durch die Rechnung machen können: Mehr darüber erfahren Sie in »Das Sicherheitscenter unter Access 2010« ab Seite 936.

18.3.2 Formular beim Starten einer Datenbank anzeigen

Wenn Sie den Benutzer Ihrer Datenbank mit einem *Begrüßungsformular* beglücken möchten, können Sie beim Öffnen dieses Formulars die beim Start der Anwendung auszuführenden Routinen aufrufen. Dazu fügen Sie die gewünschten Aufrufe einfach der Prozedur hinzu, die durch die Ereigniseigenschaft *Beim Anzeigen* ausgelöst wird. Dass Access das Formular beim Starten der Anwendung anzeigen soll, teilen Sie ihm in den Access-Optionen unter *Aktuelle Datenbank|Anwendungsoptionen|Formular anzeigen* mit.

18.3.3 Aktion beim Schließen einer Datenbank ausführen

Access bietet nur Ereignisse für Formulare, Berichte, Steuerelemente und andere Objekte, aber keine globalen Ereignisse. Während dieser Mangel sich beim Start einer Datenbank mit den oben genannten und für diesen Zweck vorgesehenen Methoden umgehen lässt, ist für das Ausführen von Aktionen beim Schließen einer Datenbank ein wenig Fantasie gefordert.

Der Clou ist, dass Formulare ein Ereignis bieten, das beim Entladen eines Formulars ausgelöst wird, und dass Access alle Formulare, die beim Schließen der Anwendung noch geöffnet sind (ob sichtbar oder unsichtbar), entlädt. Das heißt: Sie brauchen nur dafür zu sorgen, dass die beim Beenden der Datenbank auszuführende Routine in der Ereignisprozedur *Beim Entladen* eines Formulars enthalten ist und dass dieses Formular beim Schließen der Anwendung sicher noch geöffnet ist. Das lässt sich einfach realisieren: Öffnen Sie beim Starten der Anwendung ein solches Formular und machen Sie es

direkt unsichtbar. Sofern der Benutzer dieses Formular nicht explizit öffnet und dann wieder schließt beziehungsweise in der Entwurfsansicht anzeigt, ist das Formular beim Schließen der Anwendung noch geöffnet.

Damit der Benutzer keinen Unsinn mit dem benötigten Formular anstellt, zeigen Sie ihm einfach den Navigationsbereich nicht – der sollte übrigens in professionellen Anwendungen ohnehin nie zu sehen sein (Einstellung in den Access-Optionen unter *Aktuelle Datenbank|Navigationsbereich anzeigen*).

Sie können natürlich auch das Start-Formular für das Auslösen einer Prozedur beim Schließen der Datenbank verwenden. Das Formular aus Abbildung 18.4 ist in der Beispieldatenbank *InstallationBetriebWartung.accdb* als beim Start anzuzeigendes Formular angegeben.

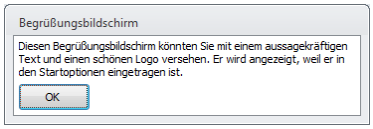


Abbildung 18.4: Startformular mit Pep

Die Schaltfläche *OK* löst die folgende Routine aus und macht das Formular damit praktisch unsichtbar:

```
Private Sub cmdOK_Click()
    Me.Visible = False
End Sub
```

Listing 18.1: Hokuspokus – Formular verschwindibus

Alternativ können Sie ein Formular auch direkt unsichtbar öffnen. Dazu verwenden Sie wiederum das *Autoexec*-Makro und legen dort einen Eintrag mit der Aktion *ÖffnenFormular* mit dem gewünschten Formularnamen und dem Fenstermodus *Ausgeblendet* an.

Es verharrt dann im unsichtbaren Zustand, bis die Anwendung geschlossen wird. Dies löst nämlich das Ereignis *Beim Entladen* des Formulars aus, was in der Ausführung der folgenden Routine resultiert. In diesem Fall zeigt die Routine einfach nur ein Meldungsfenster an; Sie können dort aber beliebige Funktionen etwa zum Erstellen einer Sicherheitskopie anlegen:

```
Private Sub Form_Unload(Cancel As Integer)
    MsgBox "Das Formular frmStart wird nun entladen."
End Sub
```

Listing 18.2: Das Formular wird beim Schließen der Anwendung entladen und das löst diese Prozedur aus

18.4 Datenbanken komprimieren und reparieren

Datenbanken wachsen mit der Zeit – zumindest wenn Sie regelmäßig Daten anlegen. Falls Sie Daten im gleichen Maße löschen, sollte der Umfang der Datenbank eigentlich abnehmen – was aber nicht der Fall ist. Der Grund ist, dass Access gelöschte Daten erst beim Komprimieren einer Datenbank endgültig aus der Datenbank entfernt. Dabei ist mit Komprimieren nicht das Packen in ein anderes Format wie *.zip*, *.tar*, *.arj* oder Ähnliches mit einem externen Komprimierungsprogramm gemeint, sondern die Verwendung der Access-eigenen Funktion. Diese lösen Sie mit dem Ribbon-Eintrag *Datenbanktools|Datenbank komprimieren und reparieren* oder den Backstage-Eintrag *Informationen|Datenbank komprimieren und reparieren* aus.

Das Komprimieren löscht nicht nur alle noch vorhandenen Datenfragmente, sondern initialisiert auch die *Autowert*-Felder der in der Datenbank enthaltenen Tabellen. Das bedeutet, dass der Zähler beim nächsten Aufruf beim Folgewert des größten enthaltenen Wertes ansetzt. Leere Tabellen, die zwischenzeitlich einmal Daten enthalten haben, werden somit – rein autowert-technisch betrachtet – in einen jungfräulichen Zustand zurückversetzt. Zwischenräume in Tabellen mit bestehenden Daten werden dadurch allerdings nicht gefüllt. Gleichzeitig ordnet Access auch die Indizes der Tabellen neu an, sodass mit Abfragen schneller darauf zugegriffen werden kann. Den seit einigen Access-Versionen mit dem Reparieren zusammengefassten Komprimiervorgang können Sie automatisch beim Beenden einer Datenbank durchführen lassen. Dazu aktivieren Sie einfach die Access-Option *Aktuelle Datenbank|Anwendungsoptionen|Beim Schließen komprimieren*.

18.5 Mehrbenutzerbetrieb mit Access-Datenbanken

Der Mehrbenutzerbetrieb von reinen Access-Datenbanken setzt voraus, dass Sie die enthaltenen Daten auf der einen und die Benutzeroberfläche und die Anwendungslogik auf der anderen Seite in einzelnen Access-Dateien speichern. Das hört sich wilder an, als es ist – selbst wenn Sie nicht die Dienste des Assistenten zur Datenbankaufteilung in Anspruch nehmen, ist eine Datenbank im Nu gegliedert. Die aufgeteilten Datenbanken finden Sie im Download zu diesem Kapitel (siehe vorne).

18.5.1 Aufteilen einer Access-Datenbank

Dazu sind nur die folgenden drei Schritte erforderlich:

» Importieren aller Tabellen in eine neue, leere Datenbank

» Löschen der Tabellen aus der ursprünglichen Datenbank

Index

Symbole

A

B

C

D

E

F

G

H

I

K

L

M

N

Q

R

S

T

U

V

W

Z