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

Access [basics]-News

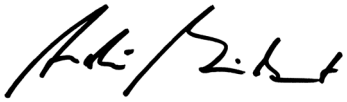
Mit der neuen Ausgabe von **Access [basics]** stellen wir die Beispieldateien von einer großen Datenbank auf eine Datenbank je Artikel um. Die Beispieldatenbank ist doch recht unübersichtlich geworden, eine einzelne Datei zu jedem Artikel scheint mir praktischer zu sein. Ich werde in der nächsten Zeit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

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

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

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

Und nun: Viel Spaß beim Lesen der neuen Ausgabe!



André Minhorst

In dieser Ausgabe:

Impressum

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

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

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

Autor: André Minhorst

ISSN: 2190-8761

ISSN: 2190-8761

Werte zu Kombinationsfeldern hinzufügen

Kombinationsfelder bieten meist Daten aus Lookup-Tabellen zur Auswahl an. Das bedeutet, dass Sie damit etwa die Anrede oder den Titel einer Person festlegen können, wobei Anreden und Titel in separaten Tabellen gespeichert sind. Das ist schon hilfreich. Noch praktischer wäre es allerdings, wenn Sie neue Einträge für die separaten Tabellen direkt über das Kombinationsfeld eintragen könnten. Wie dies funktioniert, zeigt dieser Artikel.

Beispieldatenbank

Die Beispiele zu diesem Artikel finden Sie in der Datenbank **1105_KombifelderErweitern.mdb**.

Lookup-Tabellen

Wie Sie Lookuptabellen anlegen, haben Sie bereits in **Anreden und Co. mit Wertlisten oder Lookup-Tabellen verwalten** erfahren. Der Benutzer kann dort über ein Kombinationsfeld auf einfache Weise beispielsweise einen der Werte **Herr** oder **Frau** als **Anrede** für eine Person auswählen.Das ist natürlich kein besonders gutes Beispiel, wenn es darum geht, die Daten einer Lookup-Tabelle zu erweitern. Interessanter sind Kandidaten wie Abteilungen, Funktionen, Kategorien et cetera.

Im Rahmen dieses Artikels kümmern wir uns um die Artikelkategorien der Südsturm-Datenbank und wollen diese über die Benutzeroberfläche erweitern. Dabei interessieren uns besonders die beiden Tabellen **tblArtikel** und **tblKategorien**. Die Tabelle **tblArtikel** enthält ein Feld namens **KategorieID**, mit dem der Primärschlüsselwert der passenden Kategorie der Tabelle **tblKategorien** festgelegt werden kann (siehe Bild 1).

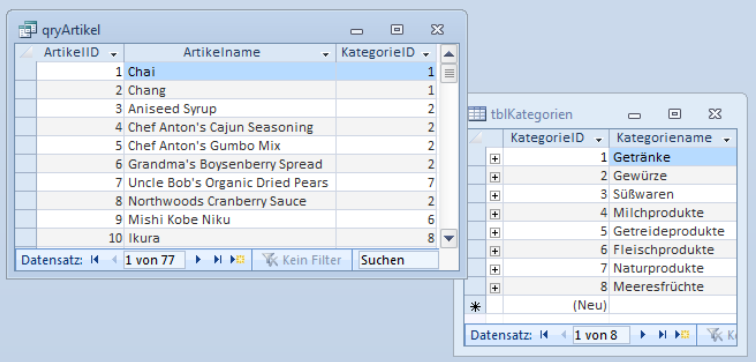


Bild 1: Artikel und Kategorien werden über das Fremdschlüsselfeld **KategorieID** verknüpft

Damit der Benutzer die Kategorie einfach per Nachschlagefeld beziehungsweise Kombinationsfeld auswählen kann, wurde das Feld **KategorieID** wie in →*[basics]* **Nachschlagefeld für eine 1:n-Beziehung einrichten** beschrieben in ein Nachschlagefeld umgewandelt.

Formular zur Anzeige von Artikeln und Kategorien

Das Formular aus Bild 2 heißt **frmArtikelUndKategorien** und verwendet die Tabelle **tblArtikel** als Datenherkunft. Um das Beispiel einfach zu halten, haben wir einfach nur die drei Felder **ArtikelID**, **Artikelname** und **KategorieID** in den Detailbereich der Entwurfsansicht des Formulars gezogen. Dadurch, dass das Feld **KategorieID** bereits im Tabellenentwurf als Nachschlagefeld definiert wurde, wird es auch im Formular gleich als Kombinationsfeld ausgeführt. Wenn Sie nun in die Formularansicht wechseln, können Sie vorhandene Artikel bearbeiten oder neue Artikel eingeben und dabei die Kategorie bequem per Kombinationsfeld auswählen (siehe Bild 3).

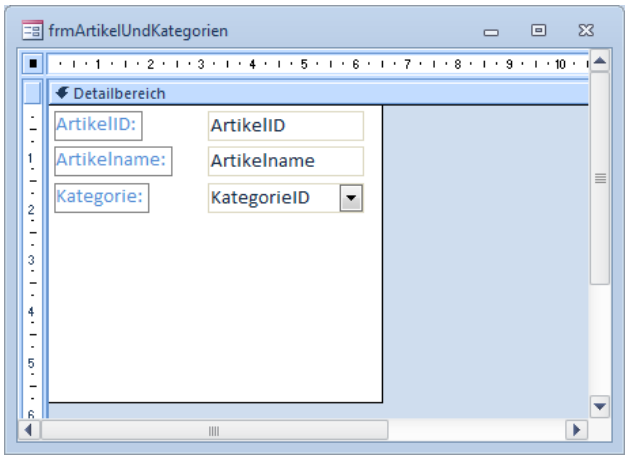


Bild 2: Das Beispielformular in der Entwurfsansicht

Beim Anlegen der Steuerelemente, die an die Felder **ArtikelID**, **Artikelname** und **KategorieID** gebunden sind, vergibt Access eben diese Bezeichnungen als Steuerelementnamen. Damit wir später im Artikel besser zwischen den Feldern und Steuerelementen unterscheiden können, erweitern Sie die Namen der Steuerelemente um entsprechende Präfixe, also

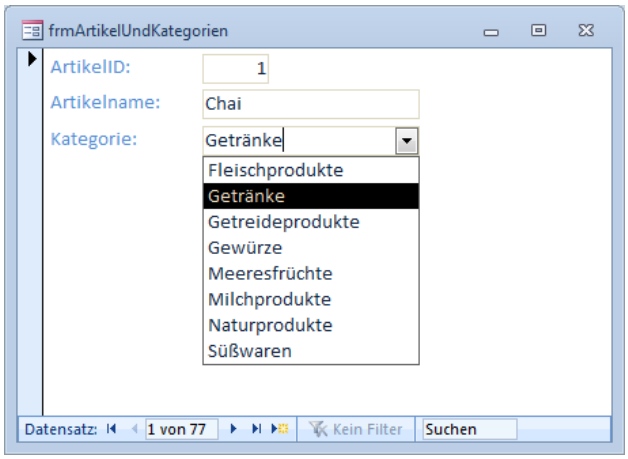


Bild 3: Kombinationsfeldeinträge auswählen

txtArtikelID, **txtArtikelname** und **cboKategorieID**. Mehr zu Präfixen erfahren Sie unter **Konventionen**.

Neue Kategorien eingeben

Was aber, wenn Sie nun einen Artikel anlegen, der sich in keine der vorhandenen Kategorien einsortieren lässt? Ganz klar: Der Benutzer öffnet die Tabelle **tblKategorien**, trägt die neue Kategorie ein und kann diese dann nach dem Aktualisieren des Formulars **frmArtikelUndKategorien** auswählen ... oder doch nicht? Nein! Der Benutzer soll auf gar keinen Fall direkt auf die Tabellen einer Anwendung zugreifen. Sie als Entwickler dürfen natürlich in Ausnahmefällen direkt Daten in Tabellen bearbeiten, aber dem Benutzer sollten Sie geeignete Formulare zur Verfügung stellen.

Es gibt nun zwei Möglichkeiten zur Eingabe von Kategorien:

- Sie legen ein neues Formular an, das an die Tabelle **tblKategorien** gebunden ist und erlauben darüber das Eingeben und Bearbeiten der Kategorien. Dieses Formular können Sie beispielsweise über eine Schaltfläche rechts vom Kombinationsfeld öffnen.

- Sie erlauben dem Benutzer, neben den vorhanden Kategorien auch neue Kategorien direkt in das Kombinationsfeld einzugeben. Diese werden dann, gegebenenfalls nach Rückfrage, in der Tabelle **tblKategorien** gespeichert.

Beide Varianten haben Vorteile: Die erste erlaubt gleichzeitig, die bestehenden Kategorien zu bearbeiten oder auch mal gleich mehrere neue Kategorien einzugeben. Die zweite ist direkter: Der Benutzer braucht nur den Kategorietext einzugeben und diese wird direkt für den aktuellen Artikel übernommen. In diesem Artikel betrachten wir die zweite Methode, die erste schauen wir uns in einem weiteren Artikel namens **Meldungsfenster anzeigen und auswerten** an.

Kombinationsfeld aufbohren

Ob der Benutzer neue Werte in ein Kombinationsfeld eingeben kann, hängt in erster Linie von der Eigenschaft **Nur Listeneinträge** ab (siehe Bild 4). Nun könnten Sie auf die Idee kommen, dass wir diese Eigenschaft für unsere Zwecke, nämlich im laufenden Betrieb direkt neue Kategorien in das Kombinationsfeld einzugeben, auf den Wert **Nein** einstellen müssen. Der Versuch scheitert: Die Eigenschaft darf laut

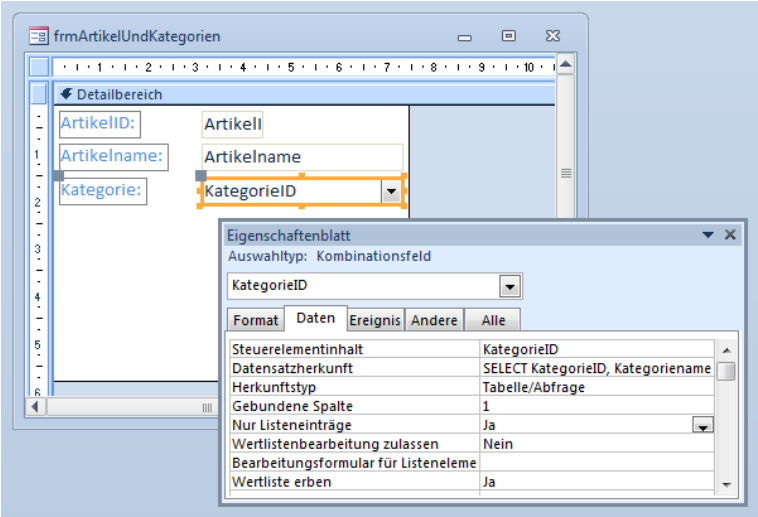


Bild 4: Die Eigenschaft **Nur Listeneinträge**

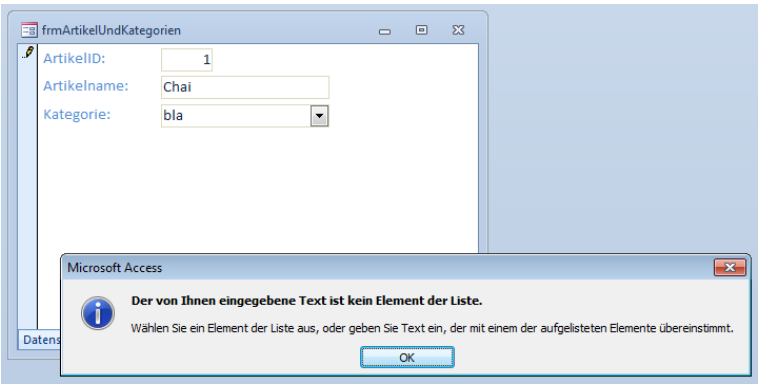


Bild 5: Diese Fehlermeldung erscheint, wenn die Eigenschaft **Nur Listeneinträge** den Wert **Ja** enthält.

der dabei erscheinenden Meldung nur auf **Nein** eingestellt werden, wenn die erste sichtbare Spalte die gebundene Spalte ist.

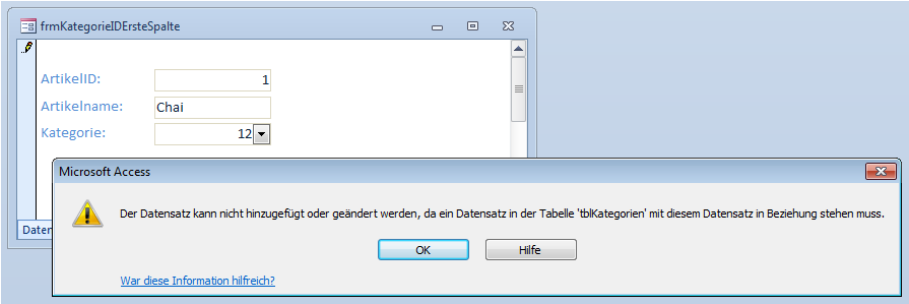


Bild 6: Fehler beim Einfügen eines neuen Wertes in ein Kombinationsfeld, das durch eine Beziehung mit referentieller Integrität nur vorhandene Werte oder **Null** aufnehmen kann.

In unserem Fall enthält die erste Spalte der Datensatzherkunft des Kombinationsfeldes das Feld **KategorieID**, angezeigt wird aber die zweite Spalte **Kategoriename**. Die Eigenschaft **Nur Listeneinträge** könnten Sie nur auf **Nein** einstellen, wenn die **KategorieID** als erstes Feld im Kombinationsfeld angezeigt würde.

Das lässt sich ändern, indem Sie die Eigenschaft **Spaltenbreiten** auf einen Wert wie **1cm** ändern – die Daten des Feldes **KategorieID** werden dann auf einer Breite von einem Zentimeter angezeigt, das Feld **Kategoriename** nimmt den verbleibenden Platz der aufgeklappten Liste ein (im Steuerelement selbst wird nun nur noch die **KategorieID** angezeigt).

Nun lässt sich die Eigenschaft **Nur Listeneinträge** auf den Wert **Nein** einstellen. Ist es nun auch möglich, neue Werte für die gebundene Spalte des Steuerelements, also für die **KategorieID**, einzugeben? Nein: Das Feld **KategorieID** ist als Fremdschlüsselfeld zum gleichnamigen Feld der Tabelle **tblKategorien** ausgelegt und gleichzeitig gibt es eine mit referentieller Integrität festgelegte Beziehung zwischen diesen beiden Feldern.

Dies besagt, dass das Fremdschlüsselfeld **KategorieID** nur den Wert **Null** oder einen der Werte des Feldes **KategorieID** der Tabelle **tblKategorien** aufnehmen darf. Sprich: Sie können nun zwar theoretisch Werte in das Kombinationsfeld **cboKategorieID** eingeben, aber dies löst einen Fehler aus, weil die entsprechende Beziehung nur vorhandene Werte oder **Null** als Wert des zugrunde liegenden Feldes erlaubt.

Wenn Sie dieses Verhalten reproduzieren möchten, schauen Sie sich das Formular **frmKategorieID-ErsteSpalte** an (siehe auch Bild 6).

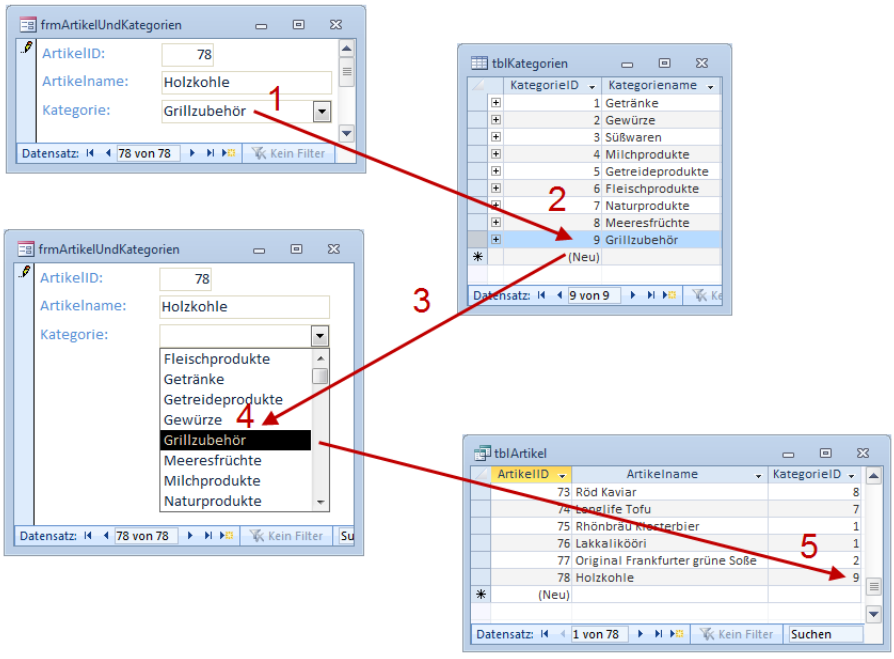


Bild 7: Ablauf beim Hinzufügen eines Eintrag zu einer Lookup-Tabelle per Kombinationsfeld

Listeneinträge und mehr

Es gibt jedoch eine Konstellation, die es erlaubt, Listeneinträge auszuwählen und auch neue Einträge zu einem Feld hinzuzufügen. Dies wäre beispielsweise der Fall, wenn die Kategorie nicht aus einer separaten Tabelle ausgewählt wird, sondern einfach nur in ein Textfeld der Tabelle **tblArtikel** eingetragen würde.

Diese Variante beleuchtet der Beitrag **Kombinationsfeld als Eingabehilfe für Textfelder** in Ausgabe 6/2011 von **Access [basics]**.

Neue Kategorie per Kombinationsfeld

Aus den bisherigen Erkenntnissen folgern wir: Eine direkte Eingabe einer neuen Kategorie ist nicht so einfach möglich. Es sind dazu vielmehr die folgenden Schritte zu erledigen, die auch in Bild 7 dargestellt werden:

- (1) Eingeben des neuen Wertes in das Kombinationsfeld
- (2) Eintragen eines neuen Datensatzes mit der gewünschten Kategorie in die Tabelle **tblKategorien**
- (3) Aktualisieren der Datensatzherkunft des Kombinationsfeldes **cboKategorieID**, damit die neue Kategorie hier ausgewählt werden kann (4)
- (5) Automatisches Ermitteln der **KategorieID** der neuen Kategorie und Eintragen dieses Wertes in das Fremdschlüsselfeld **KategorieID** der Tabelle **tblArtikel**

Den ersten Schritt erledigt der Benutzer, die übrigen sollen per Code automatisch durchgeführt werden. Zunächst benötigen Sie dabei eine Möglichkeit, überhaupt zu erkennen, dass der Benutzer einen nicht vorhandenen Eintrag eingegeben hat. Praktischerweise bietet das Kombinationsfeld dazu ein passendes Ereignis, nämlich **Bei Nicht in Liste**. Wählen Sie für die entsprechende Eigenschaft den Wert **[Ereignisprozedur]** aus und klicken Sie auf die Schaltfläche mit den drei Punkten (siehe Bild 8).

Die nun im VBA-Editor erscheinende Prozedur ergänzen Sie wie folgt:

```
Private Sub KategorieID_NotInList(NewData As _
                    String, Response As Integer)
    Dim lngKategorieID As Long
    Dim db As DAO.Database
    Set db = CurrentDb
    db.Execute "INSERT INTO tblKategorien _
      (Kategoriename) VALUES('" & NewData & "')", _
                                    dbFailOnError
    lngKategorieID = db.OpenRecordset _
                ("SELECT @@IDENTITY").Fields(0)
    Me!cboKategorieID.Undo
    Me!cboKategorieID.Requery
    Me!cboKategorieID = lngKategorieID
    Response = acDataErrAdded
    Set db = Nothing
```

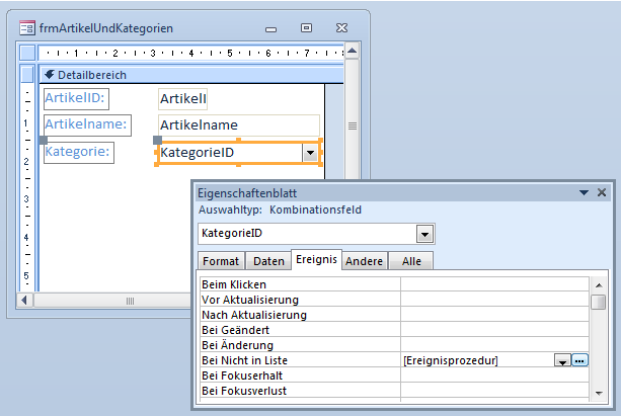


Bild 8: Dieses Ereignis wird ausgelöst, wenn der Benutzer einen neuen Eintrag in das Kombinationsfeld einträgt.

```
End Sub
```

Dies sieht auf den ersten Blick möglicherweise kompliziert aus, aber wir schauen uns die einzelnen Anweisungen nun genau an.

Zunächst einmal ist der Prozedurkopf bemerkenswert. Er enthält zwei Parameter. Der erste heißt **NewData** und liefert der Prozedur genau den Wert, den der Benutzer neu in das Kombinationsfeld eingetragen hat.

Wenn der Benutzer also wie in der Abbildung den Wert **Grillzubehör** einträgt, können Sie über die Parametervariable **NewData** auf diesen Wert zugreifen.

Der zweite Parameter heißt **Response** und enthält einen Wert, der festlegt, wie Access auf die Eingabe eines neuen Ausdrucks im Kombinationsfeld reagieren soll. Standardmäßig ist diese Parametervariable mit dem Wert **1** gefüllt, was der Konstanten **acDataErrDisplay** entspricht – auf gut deutsch: Zeige eine Fehlermeldung an (in diesem Fall **Der von Ihnen eingegebene Text ist kein Element der Liste.**). Wenn Sie den Wert dieses Parameters innerhalb der Ereignisprozedur **NotInList** nicht ändern, gibt Access die Meldung aus Bild 5 aus.

Im Folgenden soll der eingegebene Ausdruck jedoch gespeichert und dem Feld **KategorieID** zugewiesen werden – die Fehlermeldung würde den Benutzer da nur verwirren. Damit diese nicht erscheint, müssen Sie irgendwo innerhalb der Ereignisprozedur **NotInList** die Variable **Response** auf den Wert **2** beziehungsweise **acDataErrAdded** festlegen. Eigentlich reicht **2**, aber mit **acDataErrAdded** ist der Code besser lesbar. Access verwendet intern eine Liste von Konstanten wie **acDataErrDisplay** oder **acData-**

ErrAdded und weist diese automatisch den entsprechenden Zahlenwerten zu.

Die benötigte Anweisung zum Unterbinden der Fehlermeldung sieht also wie folgt aus:

```
Response = acDataErrAdded
```

Die Hauptarbeit ist jedoch das Speichern des neuen Wertes in der Tabelle **tblKategorien** und das Einstellen des Kombinationsfeldes **KategorieID** auf den neuen Wert.

Die folgenden drei Zeilen legen den neuen Datensatz in der Tabelle **tblKategorien** an. Dies wird durch eine Anfügeabfrageabfrage erledigt, die wir bereits in **Aktionsabfragen - Teil I: Anfügeabfragen** beschrieben haben. Unsere Anfügeabfrage legt einen neuen Datensatz in der Tabelle **tblKategorien** an, wobei das Feld **Kategoriename** mit dem Wert gefüllt wird, den der Parameter **NewData** uns übergibt:

```
Dim db As DAO.Database
Set db = CurrentDb
db.Execute "INSERT INTO tblKategorien
      (Kategoriename) VALUES('" & NewData & "')", _
                                    dbFailOnError
```

Für den Ausdruck **Grillzubehör** sieht die Abfrage also beispielsweise so aus:

```
INSERT INTO tblKategorien(Kategoriename)
VALUES('Grillzubehör')
```

Wie das Ausführen von Aktionsabfragen per VBA genau funktioniert, erfahren Sie übrigens unter **Aktionsabfragen per VBA ausführen**.

Danach folgt die Aufgabe, den Wert des Feldes **KategorieID** für den neuen Datensatz zu ermitteln. Diesen benötigen wir ja, um das gleichnamige Fremdschlüsselfeld der Tabelle **tblArtikel** zu füllen und so dem Artikel gleich die neue Kategorie zuzuweisen.

Das Primärschlüsselfeld **KategorieID** der Tabelle **tblKategorien** ist als Autowertfeld ausgelegt. Für diese gibt es eine spezielle Abfrage, die den zuletzt hinzugefügten Wert innerhalb der aktuellen Sitzung mit der Anwendung ermittelt. Den Wert speichern wir in der Variablen **lngKategorieID**, die so deklariert wird:

```
Dim lngKategorieID As Long
```

Die Abfrage zum Ermitteln dieses Wertes rufen Sie wie folgt auf, wobei der einzige zurückgelieferte Wert des Ergebnisses gleich in der Variablen **lngKategorieID** gespeichert wird:

```
lngKategorieID = db.OpenRecordset _
                    ("SELECT @@IDENTITY").Fields(0)
```

Die **OpenRecordset**-Anweisung wurde in **Access [basics]** bislang noch nicht erläutert. Nehmen Sie diese eine Zeile als gegeben hin, wir kommen in einer späteren Ausgabe darauf zurück.

Danach folgen noch einige Schritte, die das Ergebnis abrunden. Wir haben nun die Tabelle **tblKategorien** um den neuen Datensatz erweitert.

Das Kombinationsfeld **cboKategorieID** zeigt den neuen Eintrag aber noch nicht an, sondern nur die bisher schon vorhandenen Einträge. Also wenden Sie die **Requery**-Methode auf dieses Steuerelement an, damit seine Datensatzherkunft aktualisiert wird.

Dies würde allerdings einen Fehler auslösen, weil Access ein Kombinationsfeld nur aktualisieren kann, wenn eine bereits vorgenommene Änderung am Inhalt gespeichert wurde. Dies können wir nicht tun, also verwerfen wir die Änderung – das Eintragen der neuen Kategorie – einfach mit der **Undo**-Methode des Kombinationsfeldes:

```
Me!cboKategorieID.Undo
```

Dieses Zurücksetzen können wir deshalb so sorglos ausführen, weil wir den neu eingegebenen Wert ja bereits in der Tabelle **tblKategorien** gespeichert haben und den Wert des Feldes **KategorieID** für den entsprechenden Datensatz kennen. Danach aktualisieren wir schließlich die Datensatzherkunft des Kombinationsfeldes:

```
Me!cboKategorieID.Requery
```

Nun fehlt nur noch ein letzter Schritt – das Zuweisen des neuen Eintrags der Tabelle **tblKategorien** zum Kombinationsfeld **cboKategorieID**. Dies erledigt die folgende Anweisung:

```
Me!cboKategorieID = lngKategorieID
```

Fertig – die neue Kategorie wird kommentarlos in die Tabelle **tblKategorien** eingetragen und als Wert des Kombinationsfeldes ausgewählt.

Rückfrage vor dem Anlegen

Gelegentlich kann es hilfreich sein, nicht alle Aktionen des Benutzers kommentarlos hinzunehmen. Im Beispiel dieses Artikels kommen so etwa schnell einige neue Einträge in der Tabelle **tblKategorien** zusammen. Der Haken dabei ist: Wenn ein Benutzer das Kombinationsfeld nicht aufklappt und den gewünschten Eintrag auswählt, sondern schnell den gewünschten Eintrag eintippt, kann es vorkommen, dass dieser sich vertippt.

Das Resultat ist, dass die Tabelle **tblKategorien** dann nicht nur einen Eintrag wie **Getränke**, sondern auch noch **Getraenke** enthält. Und wenn der Benutzer dann später weitere Artikel beiden Kategorien zuweist, kann keine vernünftige Auswertung mehr nach den Kategorien mehr erfolgen.

Deshalb erweitern wir die Lösung noch um eine Rückfrage vor dem Speichern der neuen Kategorie, die wie in Bild 9 aussehen soll.

Dies erledigen Sie durch eine **MsgBox**-Funktion. Die **MsgBox**-Funktion erwartet Parameter etwa für den anzuzeigenden Text, die Titelleiste und die benötigten Schaltflächen. In unserem Fall soll die Titelleiste den Text **Neue Kategorie** anzeigen. Die Meldung selbst soll **'<Kategoriename>' als neue Kategorie anlegen?** lauten.

Dazu erweitern wir den Code um einige Zeilen. Die erte stellt die Meldung zusammen und speichert sie in einer Variablen, die zuvor deklariert wird:

```
Dim strMeldung As String
```

Für die Meldung verketten wir ein Hochkomma, den Inhalt von **NewData** und den Rest des Ausdrucks mit dem **&**-Operator:

```
strMeldung = "'" & NewData & "' als neue ⤵
                              Kategorie anlegen?"
```

Die **MsgBox**-Funktion liefert je nach gedrückter Schaltfläche verschiedene Ergebnisse zurück (siehe auch **VBA-Befehle: MsgBox**). Wenn dieses Ergebnis der **Ja**-Schaltfläche entspricht (**vbYes**), erhält die Variable **bolAnlegen** den Wert **True**:

```
Dim bolAnlegen As Boolean
bolAnlegen = MsgBox(strMeldung, vbYesNo, ⤵
                    "Neue Kategorie") = vbYes
```

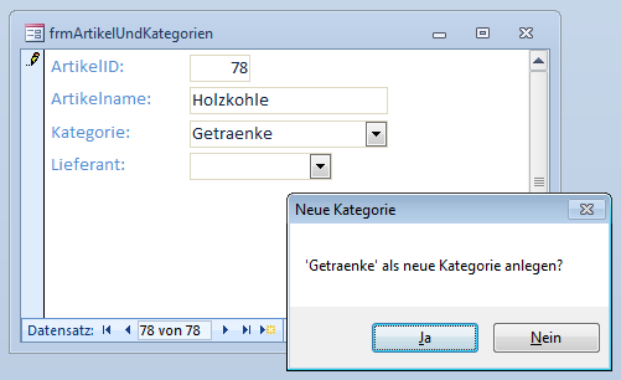


Bild 9: Rückfrage vor dem Anlegen eines neuen Datensatzes

Die folgende **If...Then**-Bedingung (siehe **VBA-Grundlagen – Teil III: If...Then- und Select Case-Bedingungen**) prüft, ob **bolAnlegen** den Wert **True** enthält. Falls ja, wird die Kategorie wie oben beschrieben angelegt.

Falls nicht, geschieht nichts – außer, dass Sie dem Parameter **Response** die Konstante **acDataErrContinue** zuweisen. Dies bedeutet, dass weder ein Fehler ausgelöst noch ein neuer Wert zum Kombinationsfeld hinzugefügt wurde – das Kombinationsfeld wird dadurch schlicht aufgeklappt angezeigt:

```
If bolAnlegen = True Then
    ... Kategorie anlegen
    Response = acDataErrAdded
Else
    Response = acDataErrContinue
End If
```

Einsatz in eigenen Anwendungen

Wenn Sie die hier vorgestellte Technik in eigenen Formularen einsetzen möchten, brauchen Sie nur einige Änderungen an der vorgestellten Prozedur vorzunehmen. Im Modul **mdlKombinationsfeldErweitern** haben wir für Sie eine Rohfassung der notwendigen Prozedur gespeichert. Die zu ersetzenden Elemente sind jeweils in spitze Klammern eingefasst, zum Beispiel so:

```
Private Sub <Kombinatonsfeldname>_NotInList ⤵
            (NewData As String, Response As Integer)
```

Legen Sie also einfach die Prozedur für das Ereignis **Bei Nicht in Liste** an, ersetzen Sie diese durch die aus dem angegebenen Modul und füllen Sie die Platzhalter – fertig!

Aktionsabfragen per VBA ausführen

Aktionsabfragen lassen sich bequem mit der Entwurfsansicht für Abfragen zusammenstellen. Sie können damit Daten an Tabellen anfügen, bestehende Daten ändern oder löschen und sogar gleich die passende Tabelle zum Einfügen von Daten erstellen. Wie aber soll der Benutzer eine solche Abfrage aufrufen, ohne doppelt auf den entsprechenden Eintrag im Datenbankfenster oder im Navigationsbereich zu klicken? Dieser Artikel zeigt, wie Sie dies realisieren.

Beispieldatenbank

Die Beispiele zu diesem Artikel finden Sie in der Datenbank **1105_AktionsabfragenPerVBA.mdb**.

Aktionsabfragen

Wenn Sie eine Aktionsabfrage erstellen, gehen Sie wie in **Aktionsabfragen – Teil II: Löschabfragen** oder **Aktionsabfragen – Teil I: Anfügeabfragen** beschrieben vor. Weitere Artikel zu Aktualisierungsabfragen oder Tabellenerstellungsabfragen folgen in späteren Ausgaben.

Das Resultat beim Erstellen einer Aktualisierungsabfrage mit der Abfrageentwurfsansicht ist eine gespeicherte Abfrage. Diese führen Sie per Doppelklick auf den entsprechenden Eintrag im Datenbankfenster oder im Navigationsbereich aus.

Ein Beispiel hierfür ist die Löschabfrage aus Bild 1. Sie ist unter dem Namen **qryArtikelLoeschen** gespeichert und bildet die Grundlage für das folgende Beispiel.

Die Abfrage löscht genau einen Datensatz, und zwar den mit dem Wert **1** im Feld **ArtikelID**. Das gilt natürlich nur, wenn ein solcher Datensatz überhaupt vorhanden ist. Sonst bewirkt der Aufruf dieser Abfrage gar nichts.

Die Execute-Methode

Nun folgt der interessante Teil: Access bietet eine ganze Reihe von VBA-Objekten an, die wiederum praktische Methoden etwa für den Zugriff auf Daten offerieren. In unserem Fall geht es um das Objekt, dass die aktuelle Datenbank repräsentiert.

Es hat den Datentyp **Database** und wird als Objektvariable deklariert. Das alles ist für Einsteiger vielleicht noch etwas unverständlich – das ist aber kein Problem. Manche Dinge kann man zu Beginn einfach als gegeben hinnehmen. In diesem Fall geht es um die folgenden Codezeilen:

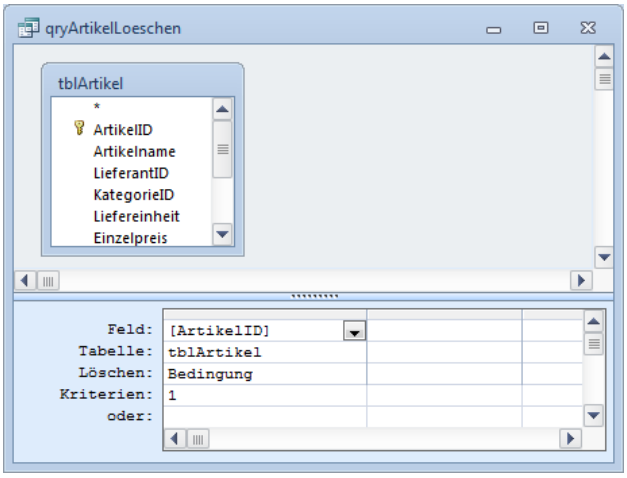


Bild 1: Entwurfsansicht einer Abfrage zum Löschen des Artikels mit dem Wert **1** im Feld **ArtikelID**.

```
Dim db As Database
Set db = CurrentDb
```

Die erste Zeile deklariert **db** als Objektvariable des Typs **Database**, die zweite füllt diese Variable mit dem Wert der Funktion **CurrentDb**. **CurrentDb** wiederum liefert einen Verweis auf die aktuelle Datenbank.

Dank dieser beiden Zeilen können Sie nachfolgend über die Variable **db** bestimmte Aktionen mit der Datenbank durchführen.

In unserem Fall soll eine Aktionsabfrage durchgeführt werden, genau genommen eine Löschabfrage.

Der dazu benötigte Befehl des mit **db** referenzierten Objekts heißt **Execute**. **Execute** erwartet als Parameter zumindest eine Variable oder Zeichenkette, die folgendes enthält:

- den Namen der Aktionsabfrage, wie Sie ihn auch im Datenbankfenster oder im Navigationsbereich finden (hier also beispielsweise **qryArtikelLoeschen**)
- den SQL-Code der Aktionsabfrage (wie Sie diesen ermitteln, erfahren Sie gleich).

Am einfachsten ist es natürlich, wenn Sie eine **DELETE**-Abfrage per Abfrageentwurf zusammenstellen und diese direkt per VBA aufrufen möchten. Die nötigen Anweisungen sehen dann so aus:

```
Private Sub EinfacherAufruf()
    Dim db As DAO.Database
    Set db = CurrentDb
    db.Execute "qryArtikelLoeschen"
    Set db = Nothing
End Sub
```

Wenn Sie das Beispiel nachvollziehen möchten, legen Sie ein neues Standardmodul an und fügen Sie den obigen Code dort ein (→*[basics]* **VBA-Modul anlegen**).

Wenn Sie als Parameter der **Execute**-Methode einfach nur den Namen einer Aktionsabfrage angeben, wird diese einfach aufgerufen. Allerdings müssen bereits alle Einstellungen in der Abfrage selbst vorgenommen werden – zum Beispiel, ob alle Datensätze der Tabelle gelöscht werden sollen oder nur einer. Im obigen Beispiel wird die Abfrage **qryArtikelLoeschen** ausgelöst, was den Artikel mit dem Wert **1** im Feld **ArtikelID** löscht.

Die Abfrage führen Sie beispielsweise aus, indem Sie die Einfügemarke irgendwo innerhalb der Prozedur platzieren und dann die Taste **F5** betätigen oder den Menübefehl **Ausführen|Sub/Userform ausführen** aufrufen.

Für den Anfang prüfen Sie nun per Sichtkontrolle, ob der Datensatz tatsächlich gelöscht wurde – und zwar, indem Sie die Tabelle **tblArtikel** öffnen und schauen, ob der Datensatz noch vorhanden ist.

Achtung: Keine Rückfrage!

Die **Execute**-Anweisung führt die angegebene Aktionsabfrage ohne Rückfrage aus. Wenn Sie also etwa einen Artikel erst löschen möchten, wenn der Benutzer dem explizit zugestimmt hat, müssen Sie die **Execute**-Anweisung noch in eine **If...Then**-Bedingung einfassen, die das Ergebnis einer **MsgBox**-Funktion prüft. Dies kann etwas allgemein formuliert etwa so aussehen:

```
If MsgBox("Datensatz löschen?", vbYesNo) = vbYes Then
    Set db = CurrentDb
    db.Execute "qryArtikelLoeschen"
End If
```

Aktionsabfrage als SQL-Ausdruck angeben

Manchmal kann es hilfreich sein, die Aktionsabfrage nicht als Abfrage zu speichern, sondern direkt auszuführen. Das bedeutet, dass Sie der **Execute**-Anweisung nicht den Namen der gespeicherten Abfrage, sondern einen entsprechenden SQL-Ausdruck angeben.

Fehlende SQL-Kenntnisse sind hier kein Problem: Access bietet die Möglichkeit, Abfragen in der SQL-Ansicht anzuzeigen. Und aus dieser können Sie den gewünschten SQL-Ausdruck einfach herauskopieren und in den VBA-Code einfügen.

Im Falle der Aktionsabfrage unseres Beispiels sieht das so aus:

- Öffnen Sie die Abfrage in der Entwurfsansicht.
- Klicken Sie mit der rechten Maustaste auf die Titelzeile und wählen Sie den Eintrag **SQL-Ansicht** aus (siehe Bild 2).

Fertig! Der SQL-Ausdruck zur aktuell geöffneten Aktionsabfrage wird nun wie in Bild 3 angezeigt.

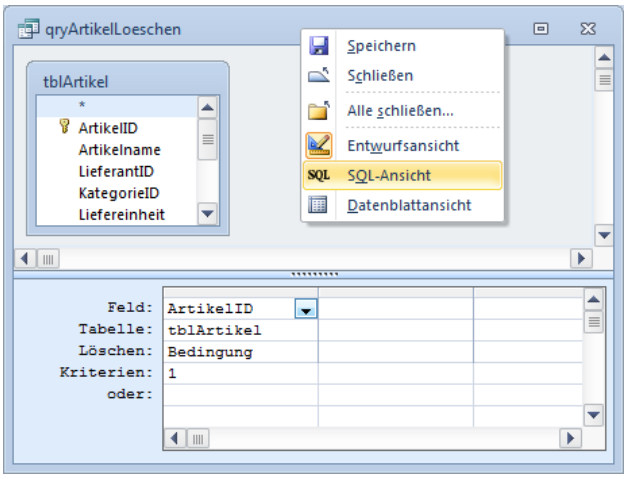


Bild 2: SQL-Ansicht einer Abfrage aktivieren

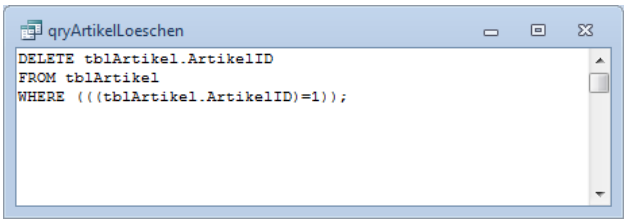


Bild 3: SQL-Ansicht einer Abfrage

Um den Code etwas übersichtlicher zu gestalten, fügen Sie nun zunächst eine Variable ein, die den SQL-Ausdruck aufnimmt:

```
Dim strSQL As String
```

Diesen weisen Sie wie in der folgenden Anweisung zu:

```
strSQL = "DELETE tblArtikel.ArtikelID FROM tblArtikel WHERE (((tblArtikel.ArtikelID)=1));"
```

Der besseren Übersicht halber lässt sich der SQL-Ausdruck auch noch ein wenig vereinfachen:

```
strSQL = "DELETE FROM tblArtikel WHERE ArtikelID=1;"
```

Schließlich verwendet die **Execute**-Methode den in **strSQL** gespeicherten Ausdruck als Parameter:

```
db.Execute strSQL
```

Flexible Aktionsabfragen

Mit einem per VBA erstelltem SQL-Ausdruck können Sie die Abfrage gleich ein wenig flexibler machen: Der SQL-Ausdruck lässt sich nämlich leicht ändern. Im folgenden soll nun nicht mehr unbedingt der Datenatz mit dem Wert **1** im Feld **ArtikelID** gelöscht werden, sondern ein Datensatz, den der Benutzer zuvor über eine **InputBox** eingibt.

Im folgenden Beispiel wird zusätzlich eine Variable namens **lngArtikelID** deklariert. Dieser weist eine **InputBox**-Funktion den vom Benutzer eingegebenen Wert zu. Schließlich wird der benötigte SQL-Ausdruck aus der Zeichenkette **DELETE FROM tblArtikel WHERE ArtikelID =** und dem Inhalt der Variablen **lngArtikelID** zusammengesetzt:

```
Public Sub ArtikelLoeschenSQLMitInputBox()
    Dim db As DAO.Database
    Dim strSQL As String
    Dim lngArtikelID As Long
    lngArtikelID = InputBox("Geben Sie die ID des zu löschenden Artikels ein.")
    Set db = CurrentDb
    strSQL = "DELETE FROM tblArtikel WHERE ArtikelID=" & lngArtikelID
    db.Execute strSQL
    Set db = Nothing
End Sub
```

Einsatz von Parameterabfragen

Es gibt noch eine weitere Möglichkeit, Abfragen flexibel zusammenzustellen – in sogenannten Parameterabfragen. Auf diese Technik gehen wir in einer späteren Ausgabe ein.

War die Aktionsabfrage erfolgreich?

Dem Benutzer werden Sie diese Vorgehensweise natürlich nicht zumuten wollen. Die Prüfung, ob die Aktionsabfrage funktioniert hat, nehmen Sie per VBA vor. Im Falle der Löschabfrage **qryArtikelLoeschen** soll beispielsweise ein Artikel gelöscht werden, also betrifft die Abfrage genau einen Datensatz.

Beim Löschen könnten Sie beispielsweise zweimal die Domänenfunktion **DCount** aufrufen (siehe **Access-Funktionen – Teil I: Domänenfunktionen**): einmal vor und einmal nach dem Löschen. Wenn das Ergebnis des zweiten Aufrufs um eins kleiner als das des ersten Aufrufs ist, war das Löschen offensichtlich erfolgreich. Dies könnte (hier auszugsweise) so aussehen:

```
Dim lngArtikelVorher As Long
Dim lngArtikelNachher As Long
...
lngArtikelVorher = DCount("*", "tblArtikel")
db.Execute "qryArtikelLoeschen"
lngArtikelNachher = DCount("*", "tblArtikel")
If lngArtikelVorher - 1 = lngArtikelNachher Then
    MsgBox "Artikel gelöscht!"
Else
    MsgBox "Artikel nicht gelöscht!"
End If
...
```

Die beiden Variablen **lngArtikelVorher** und **lngArtikelNachher** speichern die Anzahl der Artikel vor und nach dem Aufrufen der Löschabfrage. Wenn **lngArtikelVorher - 1** dem Wert von **lngArtikelNachher** entspricht, wurde offensichtlich genau ein Datensatz gelöscht. In diesem Fall wird die Meldung **Artikel gelöscht!** ausgegeben.

Die RecordsAffected-Eigenschaft

Diese Vorgehensweise hat jedoch einen entscheidenden Nachteil: Es kann sein, dass genau zum Zeitpunkt des Löschens ein anderer Benutzer ebenfalls einen Artikel löscht oder einen hinzufügt. Das Ergebnis würde dann nicht mehr stimmen.

Glücklicherweise bietet das **Database**-Objekt eine Eigenschaft, mit der Sie ermitteln können, wieviele Datensätze durch die zuletzt ausgeführte Execute-Anweisung betroffen waren. Diese Eigenschaft heißt **RecordsAffected**. Sie können den Wert dieser Eigenschaft beispielsweise gleich nach dem Ausführen der **Execute**-Anweisung in einem Meldungsfenster ausgeben:

```
Public Sub ArtikelLoeschenMitPruefungII()
    Dim db As DAO.Database
    Set db = CurrentDb
    db.Execute "qryArtikelLoeschen"
    MsgBox db.RecordsAffected
    Set db = Nothing
End Sub
```

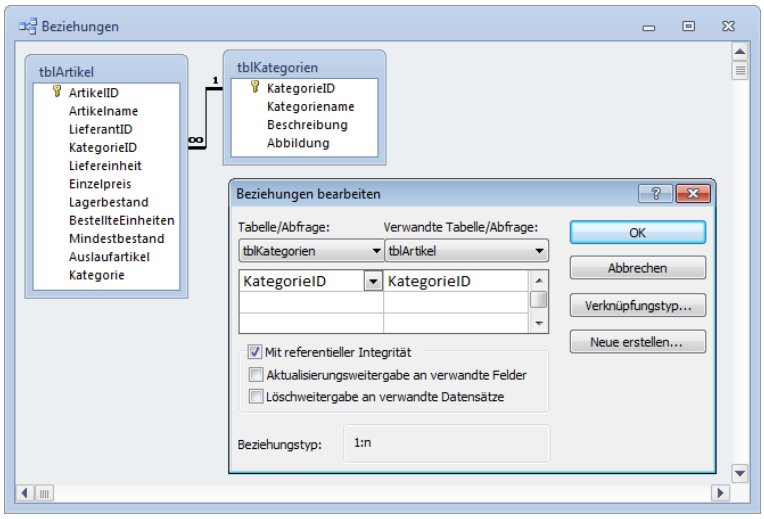


Bild 4: Wenn Artikel und Kategorien wie hier mit referentieller Integrität verknüpft sind, kann keine Kategorie gelöscht werden, wenn diese bereits einem Artikel zugeordnet wurde.

Natürlich lässt sich auch noch ein informativer Text anzeigen. Die **MsgBox**-Anweisung sieht dann so aus:

```
MsgBox "Es wurden " & db.RecordsAffected & " Datensätze gelöscht."
```

Fehler in Aktionsabfragen bemerken

Die **Execute**-Anweisung führt die angegebene Aktionsabfrage nicht nur sofort aus, ohne den Benutzer zu fragen, ob dies auch tatsächlich geschehen soll. Sie gibt standardmäßig auch kein Feedback, wenn beim Ausführen der Aktionsabfrage etwas schiefgeht.

Ein Beispiel hierfür ist das Löschen eines Eintrags der Tabelle **tblKategorien**, wenn die entsprechende Kategorie bereits für einen oder mehrere Artikel aus **tblArtikel** ausgewählt wurde (siehe Bild 4). Wenn Sie dennoch versuchen, einen solchen Datensatz zu löschen, erscheint eine Meldung wie in Bild 5.

Die folgende Prozedur soll die Kategorie mit dem Wert **1** im Feld **KategorieID** löschen:

```
Public Sub KategorieLoeschenMitFehler()
    Dim db As DAO.Database
    Dim strSQL As String
    Set db = CurrentDb
    strSQL = "DELETE FROM tblKategorien WHERE _
                                    KategorieID = 1"
    db.Execute strSQL
    Set db = Nothing
End Sub
```

Die Kategorie kann nicht gelöscht werden. Da die **Execute**-Anweisung aber standardmäßig keine Fehler meldet, läuft die Prozedur ohne Meldung und ohne Ergebnis durch.

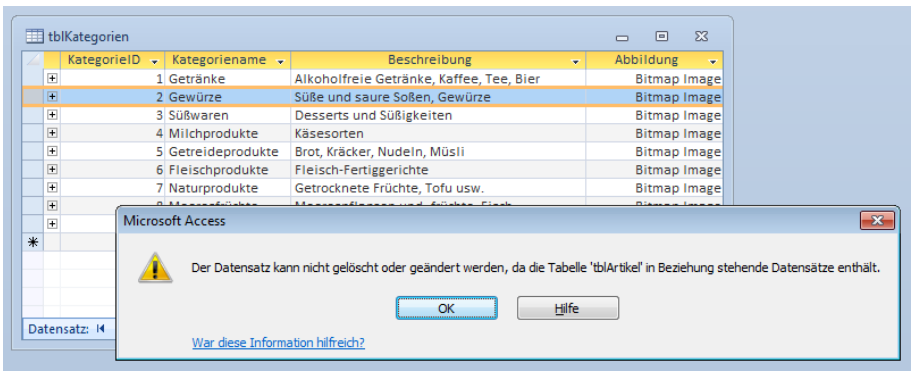


Bild 5: Fehlermeldung beim Löschen eines Datensatzes

Dies ändern Sie, indem Sie einen zweiten Parameter zur **Execute**-Anweisung hinzufügen:

```
db.Execute strSQL, _
dbFailOnError
```

Wenn Sie die Prozedur nun aufrufen, erscheint eine entsprechende Fehlermeldung (siehe Bild 6). An dieser Stelle wagen wir einen kleinen

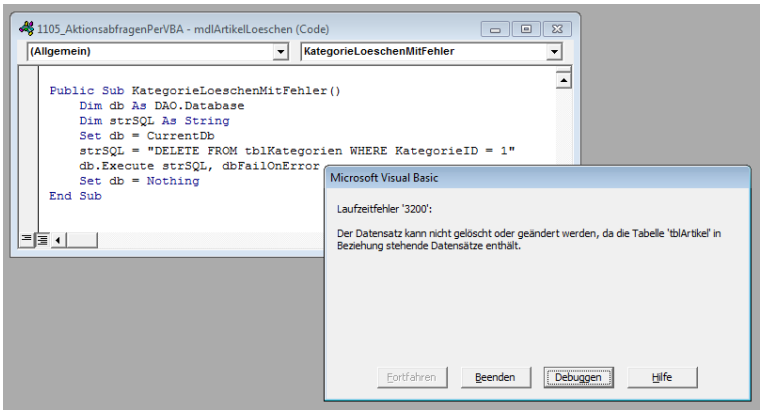


Bild 6: Mit dem Parameter **dbFailOnError** führt auch die **Execute**-Anweisung zu einer Fehlermeldung.

Zwischen diesen beiden **On Error ...**-Anweisungen folgt zunächst die **Execute**-Anweisung, die ja in bestimmten Fällen einen Fehler auslöst – zum Beispiel, wenn der zu löschende Datensatz wie hier bereits in einer verknüpften Tabelle als Fremdschlüsselwert ausgewählt wurde.

Wie Sie Bild 7 entnehmen können, heißt die Fehlernummer dieses Fehlers **3200**. Um eine speziell auf diesen Fehler zugeschnittene Meldung anzeigen zu können, fragen wir mit einer **If...Then**-Bedingung ab, ob der Fehler die Nummer **3200** hat.

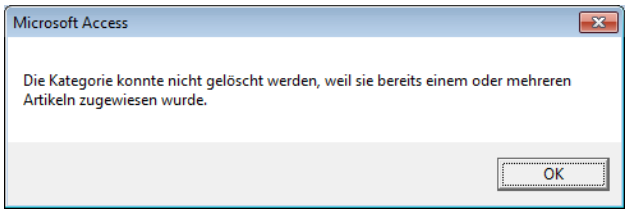


Bild 7: Benutzerdefinierte Fehlermeldung

Ausblick auf ein Thema, das in einer späteren Ausgabe detailliert erläutert wird: das Behandeln von Fehlern im VBA-Code.

Ziel dieser Maßnahme ist es, die aufgetauchte Fehlermeldung zu unterbinden und eine benutzerdefinierte Fehlermeldung anzuzeigen – und zwar eine, die auch der Benutzer versteht.

Dazu bauen Sie um die **Execute**-Anweisung herum einige weitere Anweisungen ein:

```
On Error Resume Next
db.Execute strSQL, dbFailOnError
If Err.Number = 3200 Then
    MsgBox "Die Kategorie konnte nicht gelöscht ⤷
            werden, weil sie bereits einem oder ⤷
             mehreren Artikeln zugewiesen wurde."
End If
On Error GoTo 0
```

Die erste Anweisung **On Error Resume Next** sorgt dafür, dass in den nachfolgenden Zeilen auftauchende Fehler nicht durch eine entsprechende Fehlermeldung gemeldet werden.

Dies wird in diesem Fall erst durch die Anweisung **On Error Goto 0** wieder aufgehoben.

Diese Nummer liefert, auch dazu in einen späteren Artikel mehr, der Ausdruck **Err.Number**. Hat dieser den Wert **3200**, soll in einem Meldungsfenster der Text **Die Kategorie konnte nicht gelöscht werden, weil sie bereits einem oder mehreren Artikeln zugewiesen wurde** angezeigt werden.

Diese Fehlerbehandlung ist allerdings zu rudimentär: Access würde nun nämlich nur auf den Fehler **3200** reagieren, alle anderen Fehler blieben unbehandelt. Daher ändern wir die Fehlerbehandlung der Ordnung halber noch wie folgt:

```
Select Case Err.Number
    Case 3200
        MsgBox "Die Kategorie konnte nicht ⤷
          gelöscht werden, weil sie bereits einem ⤷
          oder mehreren Artikeln zugewiesen wurde."
    Case 0
    Case Else
        MsgBox "Fehler " & Err.Number & " " ⤷
                                & Err.Description
End Select
```

Dieses Konstrukt gibt bei Fehler **3200** die gewünschte Meldung aus. Hat **Err.Number** den Wert **0**, ist kein Fehler aufgetreten – es geschieht dann nichts. Und in allen anderen Fällen wird eine Meldung mit der Fehlernummer und der Fehlerbeschreibung ausgegeben.

Im Artikel **Daten per Kombinationsfeld auswählen und löschen** finden Sie Beispiele für den Praxiseinsatz von Aktionsabfragen.

Daten per Kombinationsfeld auswählen und löschen

Wenn Sie den Benutzern Ihrer Datenbank die Gelegenheit bieten möchten, Datensätze gezielt zu löschen, lässt sich das ganz leicht etwa mit einem Kombinationsfeld und einer Schaltfläche erreichen. Mit dem Kombinationsfeld wählt der Benutzer den zu löschenden Datensatz aus und die Schaltfläche startet den eigentlichen Löschvorgang.

Beispieldatenbank

Die Beispiele zu diesem Artikel finden Sie in der Datenbank **1105_DatenPerKombifeldLoeschen.mdb**.

Beispiel: Artikel löschen

Das Beispiel zu diesem Artikel soll ein Formular enthalten, in dem der Benutzer einen Artikel auswählt und diesen mit einem Klick auf eine Schaltfläche löscht. Die Auswahl des zu löschenden Artikels soll der Einfachheit halber über ein Kombinationsfeld erfolgen, dass die Felder **ArtikelID** und den **Artikelname** der Tabelle **tblArtikel** als Datensatzherkunft enthält.

Zum Nachvollziehen des Beispiels erstellen Sie ein neues Formular namens **frmArtikelLoeschen** und fügen ein Kombinationsfeld und eine Schaltfläche wie in Bild 1 hinzu. Das Kombinationsfeld nennen Sie **cboArtikelID** und die Schaltfläche **cmdArtikelLoeschen**. Damit das Kombinationsfeld die Artikel aus der Tabelle **tblArtikel** zur Auswahl anbietet, klicken Sie in den Eigenschaften des Kombinationsfeldes auf **Datensatzherkunft** und dann auf die rechts erscheinende Schaltfläche mit den drei Punkten. Es öffnet sich der Abfrageeditor. Dort fügen Sie aus der Liste der Tabellen den Eintrag **tblArtikel** hinzu und ziehen die beiden Felder **ArtikelID** und **Artikelname** in das Entwurfsraster. Stellen Sie außerdem die Sortierung für das Feld **Artikelname** auf **Aufsteigend** ein.

Schließen Sie den Abfrageentwurf und prüfen Sie, ob Access den folgenden Ausdruck in die Eigenschaft Datensatzherkunft übernommen hat – dies ist der SQL-Code der Abfrage:

```
SELECT tblArtikel.ArtikelID, tblArtikel.Artikelname
FROM tblArtikel;
```

Stellen Sie außerdem die beiden Eigenschaften **Spaltenanzahl** und **Spaltenbreiten** auf die Werte **2** und **0cm** ein. Auf diese Weise wird die erste Spalte mit den Werten des Feldes **ArtikelID** mit der Breite **0** angezeigt und somit ausgeblendet, das zweite Feld **Artikelname** nimmt den verbleibenden Platz ein. Nach einem Wechseln in die Formularansicht können Sie

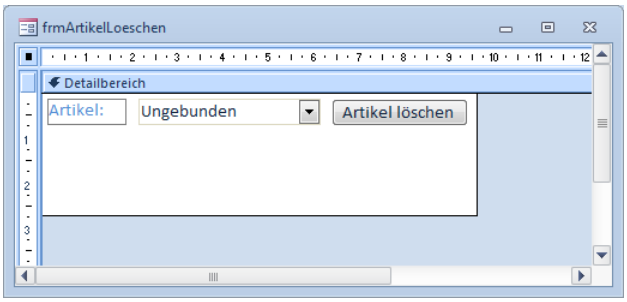


Bild 1: Entwurfsansicht des Formulars zum Löschen von Artikeln

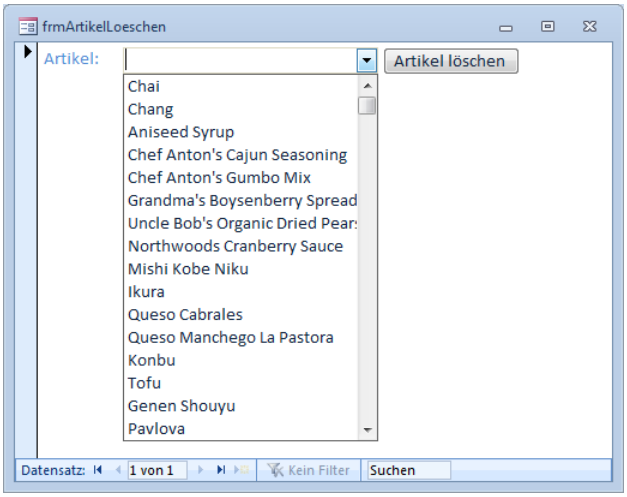


Bild 2: Auswahl eines Artikels per Kombinationsfeld

das Auswählen des zu löschenden Artikels bereits testen (siehe Bild 2).

Löschen per Mausklick

Fehlt noch die Schaltfläche **cmdLoeschen**: Diese soll nach dem Anklicken den aktuell ausgewählten Datensatz aus der Tabelle **tblArtikel** löschen. Dazu legen Sie zunächst eine leere Ereignisprozedur für das Ereignis **Beim Klicken** an. Wählen Sie dazu in der Entwurfsansicht des Formulars den Eintrag **[Ereignisprozedur]** für die Eigenschaft **Beim Klicken** der Schaltfläche aus und klicken Sie dann auf die Schaltfläche mit den drei Punkten. Der VBA-Editor zeigt nun die leere Prozedur an:

```
Private Sub cmdArtikelLoeschen_Click()
End Sub
```

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

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

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

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

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

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

Finetuning

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

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

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

```
lngArtikelID = Me!cboArtikelID
```

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

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

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

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

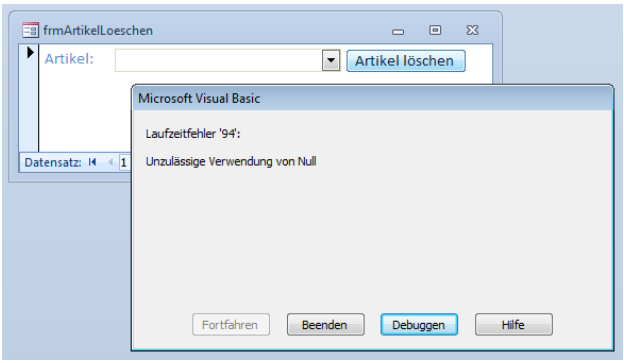


Bild 3: Wurde kein Datensatz ausgewählt, führt ein Klick auf die Schaltfläche zu diesem Fehler.

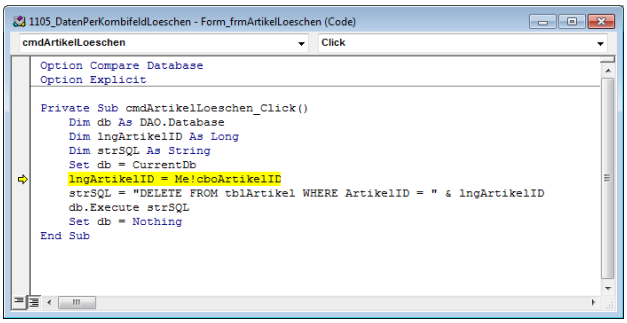


Bild 4: Nach einem Klick auf die **Debuggen**-Schaltfläche zeigt der VBA-Editor die fehlerhafte Zeile an.

Meldungsfenster anzeigen und auswerten

Das Meldungsfenster ist ein wichtiges und einfach zu bediendendes Element zur Interaktion mit dem Benutzer. Einfache Meldungsfenster zeigen einfach nur Meldungen an und werden durch einen Klick auf die OK-Schaltfläche wieder geschlossen. Sie können damit jedoch auch einfache Fragen stellen, die der Benutzer per Mausklick auf die unterschiedlichen Schaltflächen beantworten kann.

Beispieldatenbank

Die Beispiele zu diesem Artikel finden Sie in der Datenbank **1105_Meldungsfenster.mdb**.

Einfaches Meldungsfenster

Meldungsfenster zeigen Sie mit der **MsgBox**-Anweisung an. Auch wenn diese Anweisung mehrere Parameter anbietet, reicht der Einsatz des ersten Parameters in vielen Fällen schon aus. Dabei soll schlicht und einfach ein Text angezeigt werden:

```
MsgBox "Hallo"
```

Sie können diese Anweisung zum Ausprobieren einfach im Direktfenster des VBA-Editors eingeben. Dieses öffnen Sie am schnellsten mit der Tastenkombination **Strg + G**. Die besondere Eigenart des Meldungsfensters ist, dass die aufrufende Anwendung nicht fortgeführt wird, bis der Benutzer das Meldungsfenster wieder geschlossen hat.

Parameter der MsgBox-Anweisung

Das Meldungsfenster kann natürlich nicht nur einfache Texte anzeigen. Sie können auch die Titelzeile festlegen, eines von vier Icons auswählen und ein Set von Schaltflächen festlegen.

Dies sind die vier Parameter des **MsgBox**-Anweisung:

- **prompt**: Die soeben bereits verwendete Meldung.
- **buttons**: Gibt an, welche Schaltflächenkombination das Meldungsfenster anzeigen soll.

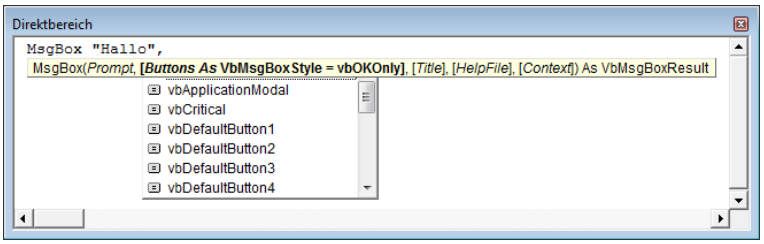


Bild 2: Der zweite Parameter offenbart mehr Optionen als sein Name besagt.

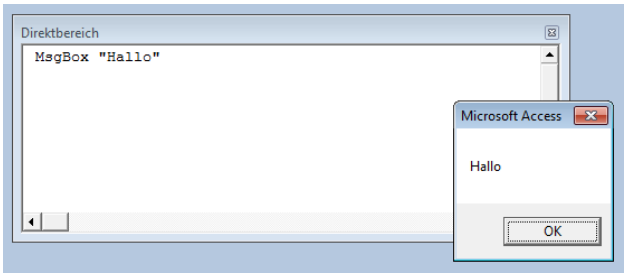


Bild 1: Die einfachste Variante eines Meldungsfensters

- **title**: Legt den Text in der Titelzeile des Meldungsfensters fest.

Die beiden übrigen Parameter **helpfile** und **context** werden in der Praxis so gut wie gar nicht eingesetzt und deshalb hier nicht beschrieben.

Meldungstext

Der Meldungstext kann ca. 1.024 Zeichen lang sein. Zeilenumbrüche erhalten Sie, indem Sie den Text auf zwei durch Anführungszeichen eingefasste Zeichenketten aufteilen und die Konstante **vbCrLf** wie folgt integrieren:

```
MsgBox "Erste Zeile" & vbcrlf & "Zweite Zeile"
```

Schaltflächen und Co.

Wenn Sie im Direktbereich mit der Eingabe der Parameter der **MsgBox**-Funktion experimentieren, stoßen Sie schnell darauf, dass der zweite Parameter offensichtlich nicht nur für die Darstellung der Schaltflächen zuständig ist. In der Tat bietet IntelliSense eine ganze Reihe von Werten für diesen Parameter an (siehe Bild 2) – und nur einige scheinen tatsächlich für Schaltflächenkombinationen zu stehen. Dabei handelt es sich um die folgenden Werte:

- **vbAbortRetryIgnore**: Zeigt die Schaltflächen **Abbrechen**, **Wiederholen** und **Ignorieren** an.
- **vbOKCancel**: Zeigt die Schaltflächen **OK** und **Abbrechen** an.

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

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

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

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

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

Icon des Meldungsfensters

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

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

Das Aussehen der vier Icons entnehmen Sie Bild 4.

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

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

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

Standardschaltfläche festlegen

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

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

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

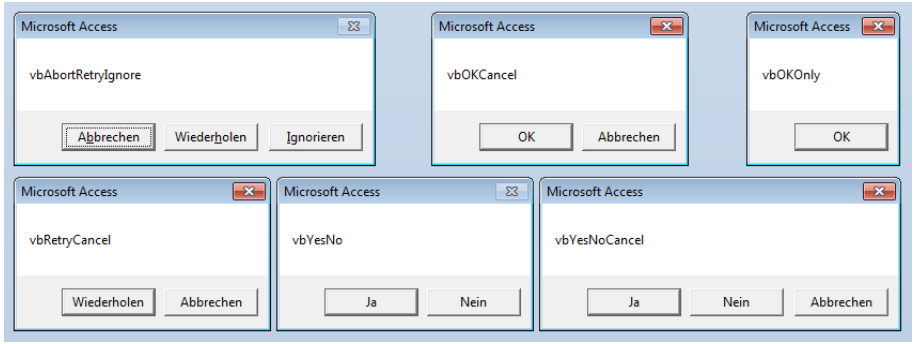


Bild 3: Alle möglichen Schaltflächen-Kombinationen der **MsgBox**-Anweisung

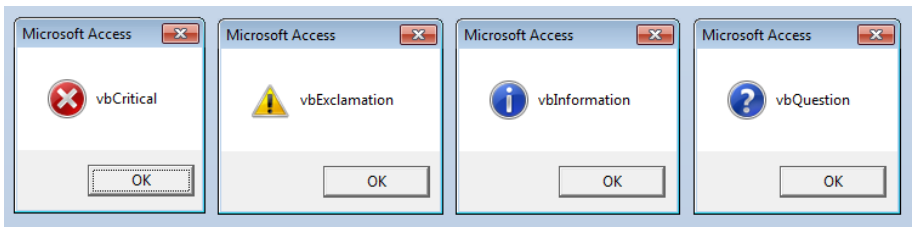


Bild 4: Die verschiedenen Icons im Meldungsfenster

angezeigtem Meldungsfenster den Fokus auf eine andere Schaltfläche verschieben.

Den Wert für die Standardschaltfläche fügen Sie etwa wie im folgenden Beispiel zum Parameter **buttons** hinzu:

```
MsgBox "Standard-Schaltfläche: Abbrechen", _
                    vbOKCancel Or vbDefaultButton2
```

Standardmäßig stellt Access die linke Schaltfläche als Standardschaltfläche ein.

Weitere Werte für den Parameter buttons

Die übrigen Werte dienen beispielsweise dem Anzeigen eines Hilfe-Buttons, dem rechtsbündigen Ausrichten der Überschrift oder beider Texte oder sind unter Access nicht wirksam.

Hier die für übliche Anwendungen sinnvollen Werte:

- **vbMsgBoxHelpButton**: Zeigt eine Hilfe-Schaltfläche an.
- **vbMsgBoxRight**: Zentriert Titel und Text rechts.

Titelzeile angeben

Der dritte Parameter namens **title** nimmt einen Text entgegen, der in der Titelzeile des Meldungsfensters angezeigt wird. Wenn Sie diesen Parameter nicht angeben, erscheint unter Access der Ausdruck **Microsoft Access** in der Titelzeile.

Benutzereingaben in Meldungsfenstern auswerten

MsgBox ist eine Funktion: Es liefert einen Wert der Enumeration **VBMsgBoxResult** zurück. Hinter dieser Enumeration verbergen sich Integer-Zahlenwerte, für die jeweils eine Konstante hinterlegt ist. Es gibt die folgenden Konstanten:

- **vbOK** (**1**): OK
- **vbCancel** (**2**): Abbrechen
- **vbAbort** (**3**): Abbruch
- **vbRetry** (**4**): Wiederholen
- **vbIgnore** (**5**): Ignorieren
- **vbYes** (**6**): Ja
- **vbNo** (**7**): Nein

Die MsgBox-Funktion gibt einen dieser Werte zurück, wenn Sie diese als Funktion aufrufen. Dies können Sie leicht im Direktfenster des VBA-Editors testen. Geben Sie dort beispielsweise folgende Anweisung ein:

```
Debug.Print MsgBox("Ja oder Nein?", vbYesNo)
```

Dies zeigt ein Meldungsfenster mit den beiden Schaltflächen **Ja** und **Nein** an. Klicken Sie auf **Ja**, gibt die **Debug.Print**-Anweisung als Ergebnis der **MsgBox**-Funktion den Wert **6** aus, für **Nein** den Wert **7**.

Diese Rückgabewerte können Sie natürlich auch innerhalb von Prozeduren auswerten. Im folgenden Beispiel wird die Antwort zunächst in einer **Integer**-Variablen namens **intAntwort** gespeichert.

Dann prüft eine **Select Case**-Bedingung, ob die Variable den Wert **vbOK** (**1**) oder **vbCancel** (**2**) aufweist und zeigt jeweils ein entsprechendes Meldungsfenster an:

```
Public Sub MeldungsfensterMitRueckgabewert()
    Dim intAntwort As Integer
    intAntwort = MsgBox("Klicken Sie auf eine _
                        Schaltfläche.", vbYesNo, _
                        "MsgBox mit Rückgabewert")
    Select Case intAntwort
        Case vbOK
            MsgBox "OK."
        Case vbCancel
            MsgBox "Abbrechen."
    End Select
End Sub
```

Daten in der Endlosansicht von Formularen anzeigen

Nicht immer soll ein Formular nur einen Datensatz gleichzeitig anzeigen. Manchmal wollen Sie dem Benutzer auch eine Liste mehrerer Datensätze einer Tabelle präsentieren. Dazu gibt es mehrere Möglichkeiten, zum Beispiel die Endlosansicht. Dieser Artikel zeigt Ihnen, wie Sie die Daten einer Tabelle übersichtlich in einem Formular darstellen.

Beispieldatenbank

Die Beispiele zu diesem Artikel finden Sie in der Datenbank **1105_Endlosformular.mdb**.

Varianten für die Anzeige mehrerer Datensätze

Grundsätzlich gibt es mehrere Möglichkeiten, Daten in Listenform in einem Formular anzuzeigen:

- Datenblattansicht in Formularen
- Endlosansicht in Formularen
- Listenfeld-Steuerelement
- nicht in Access eingebaute Steuerelemente wie etwa das ListView-Steuerelement

Die Datenblattansicht haben Sie bereits in einem anderen Zusammenhang kennengelernt: nämlich zur Anzeige von verknüpften Daten in Unterformularen (**Formulare für die Dateneingabe – Teil II: Formulare für Daten aus 1:n-Beziehungen**). Die Anzeige von Daten in Listenfeldern haben wir in **Steuerelemente – Teil II: Das Listenfeld** vorgestellt. Im vorliegenden Artikel schauen wir uns nun an, welche Möglichkeiten die Endlosansicht von Formularen bietet.

Artikel im Endlosformular

Für dieses Beispiel sollen die Datensätze einer Tabelle namens **tblArtikel** in der Endlosansicht eines Formulars angezeigt werden. Dazu führen Sie zunächst die folgenden Schritte durch:

- Legen Sie ein neues, leeres Formular an.
- Stellen Sie die Eigenschaft **Datenherkunft** auf **tblArtikel** ein.
- Ändern Sie die Eigenschaft **Standardansicht** auf **Endlosformular** ein (siehe Bild 1).

Das waren die Vorbereitungen. Nun geht es an den größten Teil der Arbeit: das Einfügen und Ausrichten der Steuerelemente. Dazu müssen Sie grundsätzlich wissen, dass ein Formular in der Endlosansicht den Detailbereich und die darin enthaltenen Steuerelemente für jeden Datensatz einmal anzeigt. Wieviele Datensätze gleichzeitig sichtbar sind, ohne dass Sie scrollen müssen, hängt von mehreren Faktoren ab. Der erste ist die Höhe des für die Anzeige der Detailbereiche für die einzelnen Datensätze verfügbare Platz, der zweite die Höhe des Detailbereichs für einen Datensatz.

Die Höhe für die Detailbereiche entspricht der Höhe des Formulars minus der Höhe von Formularkopf und Formularfuß, sofern diese beiden Bereiche eingeblendet sind. Um den Formularkopf und -fuß einzublenden, klicken Sie mit der rechten Maustaste auf die Titelzeile des Detailbereichs (siehe Bild 2). Danach erscheinen die beiden Bereiche über- und unterhalb des Detailbereichs (siehe Bild 3). Damit Sie das Zusammenspiel der einzelnen Bereiche genauer kennenlernen, erledigen Sie nun die folgenden Dinge:

- Fügen Sie im Formularkopf ein Beschriftungsfeld mit dem Text **Artikelübersicht** ein (die Schriftgröße können Sie gern auf einen größeren Wert einstellen).
- Legen Sie im Formularfuß eine Schaltfläche mit der Beschriftung **OK** und dem Namen **cmdOK** an.

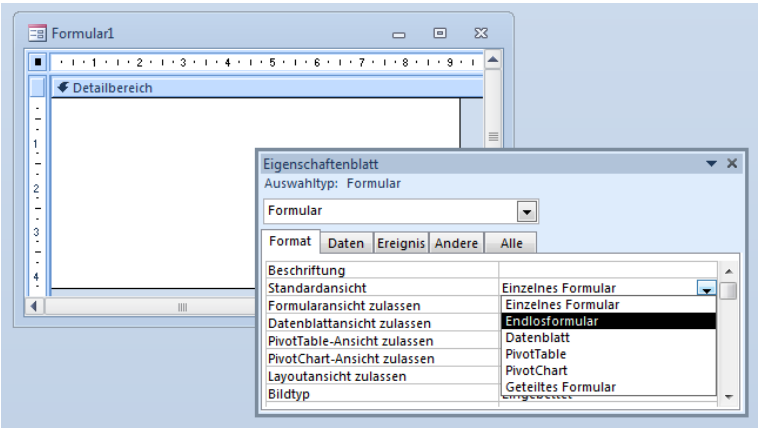


Bild 1: Die verschiedenen Icons im Meldungsfenster

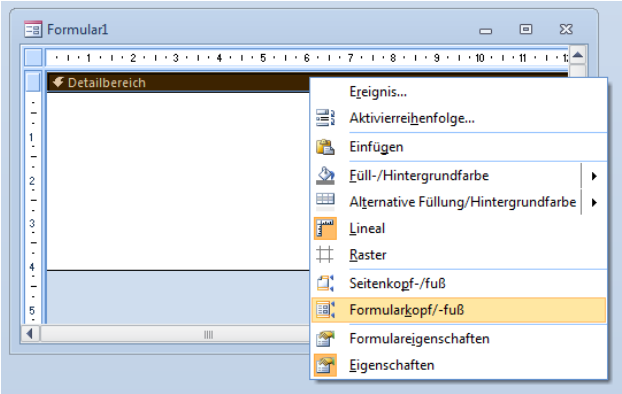


Bild 2: Einblenden von Formularkopf und -fuß

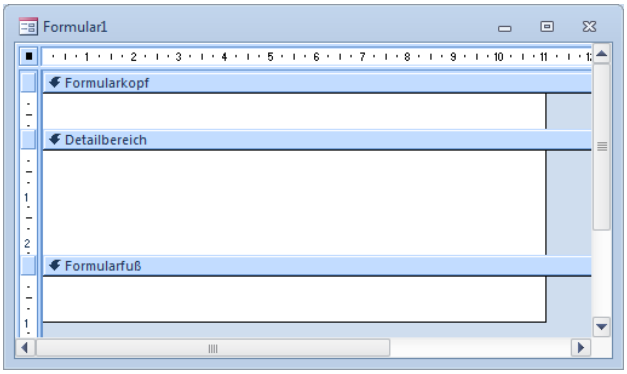


Bild 3: Formular mit Formularkopf und -fuß

Fügen Sie im Detailbereich vorerst zwei Felder der in der Datenherkunft angegebenen Tabelle hinzu. Dazu aktivieren Sie am einfachsten die Feldliste mit der Schaltfläche mit der Tastenkombination **Alt + F8** und ziehen die zwei Felder **ArtikelID** und **Artikel** in den Detailbereich (siehe Bild 4). Nach einem Wechsel in die Formularansicht sieht das Formular wie in Bild 5 aus. Die Inhalte von Formularkopf und Formularfuß werden oben und unten angezeigt, dazwischen erscheint der Detailbereich so oft, wie es der verbleibende Platz zulässt.

Wenn Sie einen Datensatz etwa zum Löschen markieren möchten, klicken Sie wie in der Datenblattansicht auf den Datensatzmarkierer links und betätigen die **Entfernen**-Taste. Zwischen den Datensätzen können Sie wie gewohnt mit den Navigationsschaltflächen unten navigieren.

Hier tritt auch gleich der entscheidende Unterschied zur Datenblattansicht zutage: Sie können die Steuerelemente zur Anzeige der Feldinhalte des Datensatzes beliebig innerhalb des Detailbereichs anordnen. Und es wird noch besser: Sie können sogar Schaltflächen im Detailbereich unterbringen, die für jeden einzelnen Datensatz angezeigt werden. Damit können Sie dem Benutzer eine Löschen-Schaltfläche anbieten, mit der ein Datensatz ohne vorheriges Selektieren mit dem Datensatzmarkierer gelöscht werden kann. Übrigens erreichen Sie die wechselnde Hintergrundfarbe nur unter Access 2007 und jünger, ältere Access-Versionen bieten dieses Feature noch nicht an.

Daten in Tabellenform im Endlosformular

Wenn Sie die übrigen Felder der Tabelle **tblArtikel** zum Detailbereich der Entwurfsansicht hinzufügen, wird es dort etwas unübersichtlich. Das ist logisch, denn die Beschriftungsfelder und die eigentlichen Steuerelemente zur Anzeige der Feldinhalte nehmen Platz weg. Zumindest die Beschriftungsfelder können wir aber viel besser anordnen – nämlich im Formularkopf.

Dummerweise lassen sich die Beschriftungsfelder, beispielsweise das mit der Beschriftung **ArtikelID**,

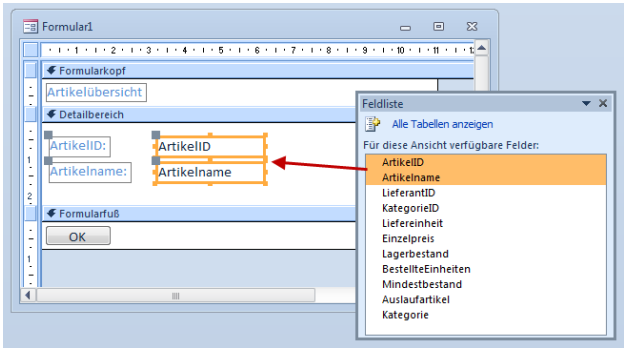


Bild 4: Hinzufügen der Felder zum Detailbereich der Entwurfsansicht

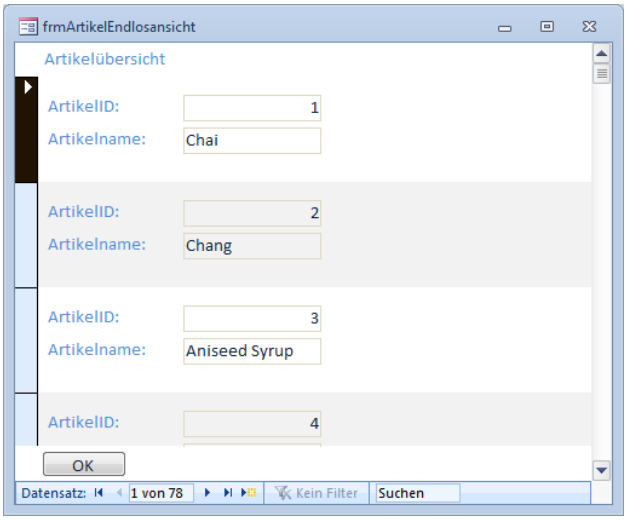


Bild 5: Ein Formular mit mehreren Datensätzen in der Endlosansicht

Bild 6: Aufbau der Steuerelemente für eine Listenansicht im Endlosformular

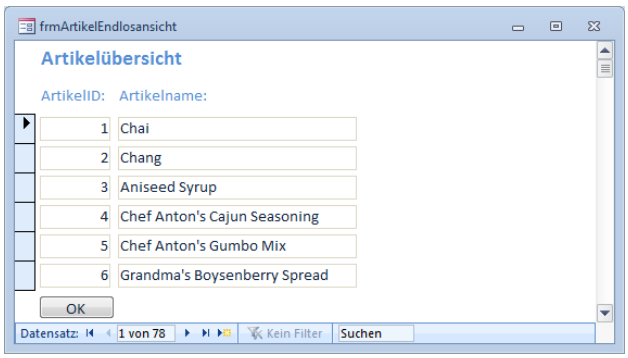


Bild 7: Die Artikel in der Formularansicht des Endlosformulars

nicht ohne das dazugehörige gebundene Steuerelement in den Formularkopf verschieben. Es gibt jedoch einen Trick, wie Sie es doch schaffen: Markieren Sie eines oder mehrere Beschriftungsfelder. Schneiden Sie diese mit der Tastenkombination **Strg + X** aus, klicken Sie in den Formularkopf-Bereich und fügen Sie die Steuerelemente dort mit **Strg + V** wieder ein.

Danach ist ein wenig Feinarbeit nötig, um die Steuerelemente anzuordnen. Tipps zu dieser Aufgabe finden Sie in →*[basics]* **Steuerelemente ausrichten und skalieren**. Anschließend könnte das Formular in der Entwurfsansicht wie in Bild 6 aussehen. Ein Wechsel in die Formularansicht liefert das Formular aus Bild 7.

Wenn Sie die Steuerelemente angeordnet haben, optimieren Sie noch die Höhe des Detailbereichs. Dies erledigen Sie, indem Sie entweder die Eigenschaft Höhe auf den gewünschten Wert einstellen oder indem Sie die Maus genau am unteren Rand des Detailbereichs platzieren und diesen nach dem Erscheinen des Icons aus Bild 8 nach oben oder unten verschieben.

Löschen von Datensätzen im Endlosformular

Wie bereits erwähnt, können Sie dem Detailbereich im Endlosformular nicht nur an die Felder der Datenherkunft gebundene Steuerelemente hinzufügen, sondern auch Schaltflächen und andere Steuerelemente. In diesem Fall soll eine Schaltfläche das Löschen von Einträgen vereinfachen. Dies ist wahrscheinlich die intuitivste Variante: Der Benutzer braucht noch nicht einmal den zu löschenden Datensatz zu markieren, sondern klickt einfach nur auf die **Löschen**-Schaltfläche neben dem betroffenen Datensatz.

Diese Schaltfläche fügen Sie etwa wie in Bild 9 zum Detailbereich hinzu. Sie erhält den Namen **cmdLoeschen** und die Beschriftung **Löschen** sowie den Wert **[Ereignisprozedur]** für die Ereigniseigenschaft **Beim Klicken**. Nach einem Klick auf die Schaltfläche neben der Eigenschaft **Beim Klicken** öffnet sich der VBA-Editor und zeigt den leeren Prozedurrumpf an.

Diesem fügen Sie nun eine Anweisung hinzu, die genau den Datensatz löscht, neben dem sich die

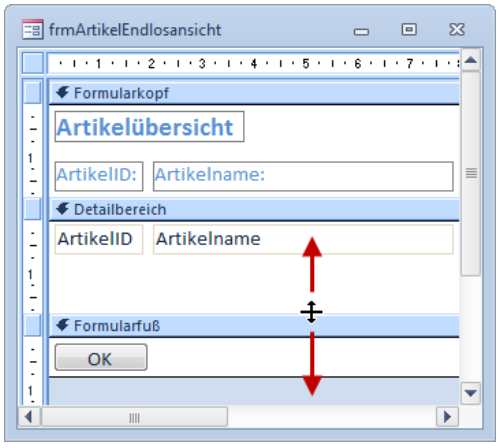


Bild 8: Höhe des Detailbereichs einstellen

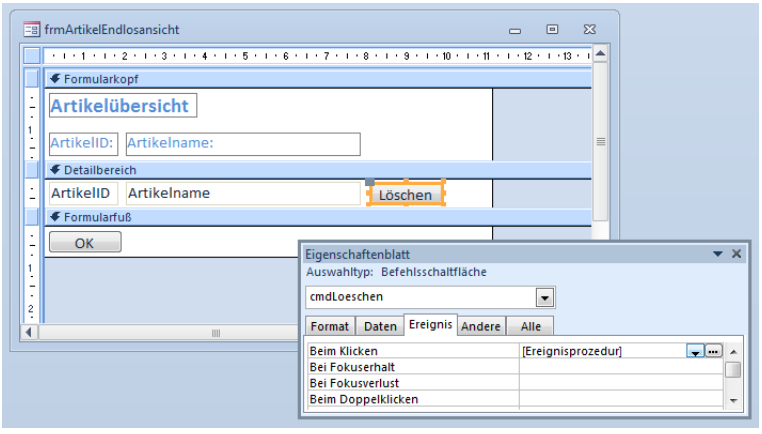


Bild 9: Hinzufügen einer Löschen-Schaltfläche zur Detailansicht

Löschen-Schaltfläche befindet. Wer glaubt, Sie müssten nun noch ermitteln, welcher Datensatz aktuell markiert ist, irrt sich: Durch den Klick auf die **Löschen**-Schaltfläche wird der Fokus automatisch auf den Datensatz verschoben, in dessen Detailbereich sich die Schaltfläche befindet. Es reicht dann der Aufruf der allgemein zum Löschen des aktuellen Datensatzes verwendeten Anweisung. Diese sieht im Kontext der Ereignisprozedur wie folgt aus:

```
Private Sub cmdLoeschen_Click()
    Application.RunCommand acCmdDeleteRecord
End Sub
```

Hier verwenden Sie prinzipiell die **RunCommand**-Methode des **Application**-Objekts, also einen Befehl von Access selbst (**Application** ist gleichbedeutend mit der aktuell geöffneten Office-Anwendung). Der Parameterwert **acCmdDeleteRecord** gibt an, dass der aktuelle Datensatz gelöscht werden soll. Wenn Sie diese Anweisung selbst eingeben, werden Sie feststellen, dass es noch viele weitere Parameterwerte für den **RunCommand**-Befehl gibt.

Optischer Feinschliff

Den Datensatzmarkierer, also die Kästchen links neben dem Detailbereich, benötigen Sie prinzipiell nur, wenn Sie einen Datensatz zum Löschen markieren möchten. Da Sie nun eine Löschen-Schaltfläche zum Datensatz hinzugefügt haben, können Sie den Datensatzmarkierer durch Einstellen der gleichnamigen Eigenschaft auf **Nein** ausblenden. Damit entledigen Sie sich nicht nur eines unnötigen Details, sondern sorgen gleichzeitig dafür, dass die Elemente des Formulars nun linksbündig dargestellt werden. Bild 10 zeigt, wie das Formular mit und ohne Datensatzmarkierer aussieht. Außerdem können Sie noch horizontale Linien in das Formular einfügen, die beispielsweise als Unterstrich für die Spaltenüberschriften dienen oder die den Formularfuß optisch vom Detailbereich trennt (siehe Bild 11).

Bild 10: Endlosformular mit und ohne Datensatzmarkierer

Eine solche Linie fügen Sie hinzu, indem Sie diese in der Toolbox mit den Steuerelementen markieren und dann einfach in den Bereich klicken, dem Sie die Linie hinzufügen möchten. Dadurch wird eine Linie angelegt, die zwar wahrscheinlich zu kurz ist und sich an der falschen Position befindet. Das ist aber kein Problem: Mit den in →*[basics]* **Steuerelemente ausrichten und skalieren** beschriebenen Methoden bringen Sie das Linie-Steuerelement schnell in Form.

Sie können die Linie auch gleich im ersten Schritt positionieren: Dazu drücken Sie die linke Maustaste nach dem Markieren des Linie-Steuerelements in der Toolbox zuerst am Startpunkt herunter, bewegen den Mauszeiger dann zum Zielpunkt und lassen schließlich die Maustaste los. Nach Bedarf können Sie natürlich auch Trennlinien zwischen die einzelnen Datensätze einfügen. Dazu legen Sie einfach eine Linie oben oder unten im Detailbereich an.

OK-Schaltfläche

Die **OK**-Schaltfläche soll das Formular schließen. Dazu legen Sie für das Ereignis **Beim Klicken** der Schaltfläche **cmdOK** die folgende Ereignisprozedur an:

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

Bild 11: Endlosformular mit Trennlinien