A1 VBA in Formularen und Berichten

Dieser letzte Teil soll die praktische Anwendung des bisher Gelernten zeigen. Im Idealfall sollte hier nun die Entwicklung einer kleinen Access-Anwendung vorgestellt werden, in der ein Großteil des bisher behandelten Stoffs vertieft wird. Dazu fehlen dem Autor aber leider 150 bis 200 Seiten Platz ...

Deshalb muss die Vertiefung an ausgewählten grundsätzlichen Beispielen erfolgen, die Sie anschließend nach entsprechenden Anpassungen in Ihre eigenen Datenbanken übernehmen können. Außerdem lernen Sie bei diesen Beispielen – hoffentlich – das zur Lösung solcher Probleme grundsätzlich notwendige methodische Vorgehen kennen. Dazu zählt vor allem auch, dass Sie auf die kleinen, aber entscheidenden Nebensächlichkeiten achten.

Die Verwendung von VBA in Formularen bedeutet in der praktischen Arbeit in erster Line die Behandlung von Ereignissen und Eigenschaften – das ist Ihnen mit Sicherheit bereits klar.

Suchen im Formular

Eine Datenbankanwendung sollte eigentlich die Möglichkeit bieten, schnell einen bestimmten Datensatz auszuwählen und zum aktuellen Datensatz zu machen. Neben verschiedenen anderen Lösungsmöglichkeiten kann dies auch über die Auswahl des betreffenden Datensatzes aus einem Listenfeld realisiert werden. Wir betrachten zunächst die fertige Lösung:

Das Ergebnis

In dem Formular ist eine Schaltfläche *btnSuche* mit der Beschriftung *Suche...* vorhanden.

Tipp

Die drei Pünktchen ... am Ende der Schaltflächenbeschriftung weisen den Anwender auf einen nachfolgenden Dialog hin.

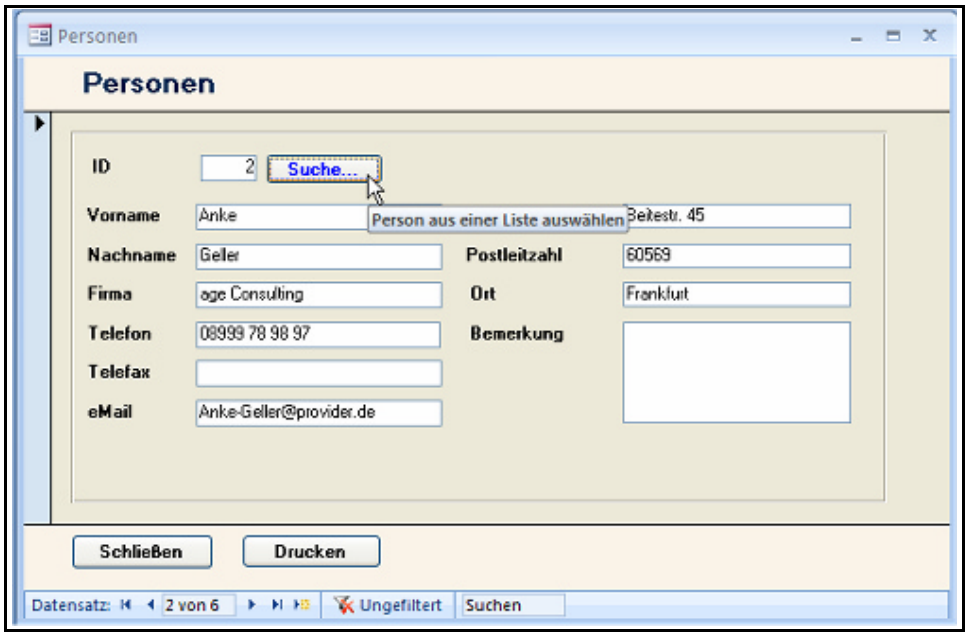

Abb. 1: Suchschaltfläche im Formular

Nach einem Klick auf diese Schaltfläche wird ein bis dahin unsichtbares Listenfeld sichtbar geschaltet und angezeigt. Nach der Auswahl eines Eintrags wird der entsprechende Datensatz zum aktuellen Datensatz im Formular gemacht.

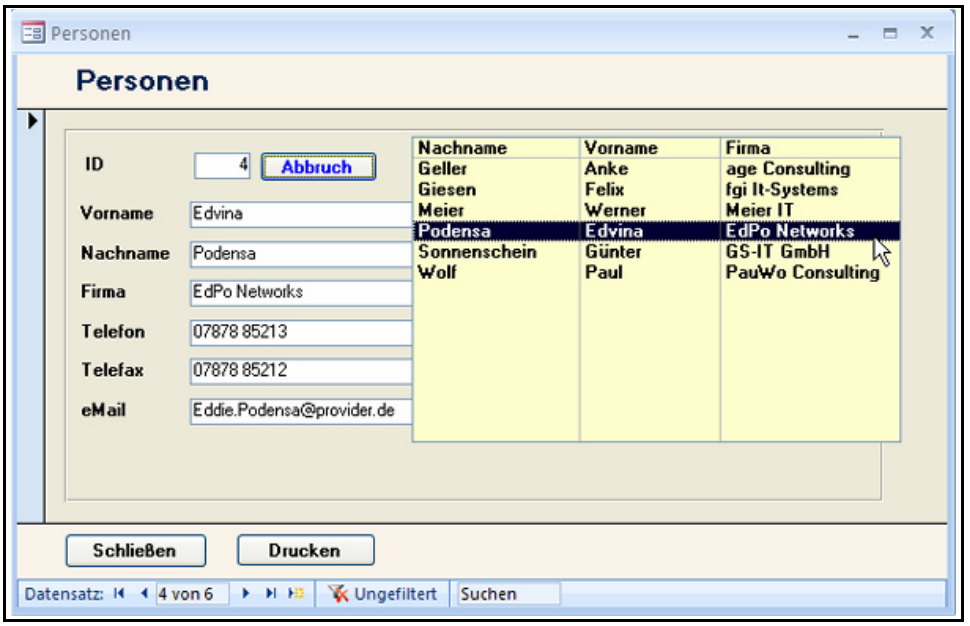

Abb. 2: Suche über ein Listenfeld

Hintergrund

Der wichtigste Punkt bei der Realisierung einer solchen Funktionalität ist es, bestimmte Eigenschaften zum richtigen Zeitpunkt zu verändern. Deshalb geht es bei diesem Beispiel auch weniger um die Vorstellung einer von vielen möglichen Suchmöglichkeiten in einem Formular, sondern vielmehr darum, die Veränderung von Eigenschaftseinstellungen zur Laufzeit per VBA zu demonstrieren.

Vorüberlegungen

Bei der Umsetzung hin zu der gezeigten Funktionalität sind zwei Steuerelemente zu behandeln:

- die Schaltfläche *btnSuche*
- das Listenfeld *lstSuche*

Wir beginnen mit der Schaltfläche.

Vor der Erstellung der Schaltfläche beziehungsweise der Ereignisprozedur dafür sind einige Überlegungen über den gewünschten beziehungsweise tatsächlichen Ablauf der Verarbeitung notwendig:

Beim Klick auf die Schaltfläche soll das Listenfeld angezeigt und dort der gewünschte Datensatz durch Anklicken angesprungen werden. Daraufhin kann auch das Listenfeld wieder geschlossen werden.

So weit, so gut. Doch was, wenn der Anwender es sich anders überlegt und gar nicht mehr zu einem anderen Datensatz wechseln will? Ihn einfach zu irgendeinem Datensatz wechseln lassen, nur um das Listenfeld wieder verschwinden lassen zu können? Eine solche Lösung wäre zwar ohne Weiteres vertretbar, ist jedoch nicht so recht zufriedenstellend.

Eleganter wäre es nämlich, in einem solchen Fall das Listenfeld durch einen zweiten Klick auf die Schaltfläche *btnSuche* wieder unsichtbar zu schalten. Dazu sollte die Schaltfläche allerdings auch eine andere Beschriftung als *Suche* haben. Auch der *ToolTip*-Text, der bei der Berührung der Schaltfläche mit der Maus angezeigt wird, sollt dann ein anderer sein.

Sie erkennen, dass schon bei der Lösung eines solchen recht simplen Problems der Teufel im Detail stecken kann.

Die Schaltfläche „btnSuche“

1. Öffnen Sie das Formular in der Entwurfsansicht.
2. Fügen Sie eine Befehlsschaltfläche hinzu und zeigen Sie das Dialogfeld *Eigenschaftenblatt* an.
3. Stellen Sie die Eigenschaften für die Schaltfläche wie in der folgenden Tabelle ein:

Eigenschaft	**Einstellung**
Name	btnSuche

Beschriftung	Suche...
SteuerelementTip-Text	Person aus einer Liste auswählen

Tab. 1: Eigenschafteneinstellungen für die Schaltfläche *btnSuche*

4. Erstellen Sie für das Ereignis *Beim Klicken* der Schaltfläche folgende Ereignisprozedur:

Ereignisprozedur

```
Private Sub btnSuche_Click()

01   If Me!lstSuche.Visible = False Then

02     DoCmd.RunCommand acCmdSaveRecord

03     With Me

04         lstSuche.Requery

05         btnSuche.ControlTipText = "Liste
ausblenden"

06         lstSuche.Visible = True

07         btnSuche.Caption = "Abbruch"

08     End With

09   Else

010    With Me

11         btnSuche.ControlTipText = "Person aus
einer Liste auswählen"

12         lstSuche.Visible = False

13         btnSuche.Caption = "Suche..."

14     End With

15   End If

End Sub
```

5. Kompilieren Sie die Prozedur und speichern Sie das Modul.

Die Ereignisprozedur

Zunächst überprüft die Ereignisprozedur in einer *If*-Anweisung (Zeile 01) den Wert der Eigenschaft `Visible` des Listenfelds auf *True* oder *False*. Falls sie das Ergebnis *False* zurückgibt, ist das Listenfeld unsichtbar. Das heißt, der Anwender will einen Datensatz aussuchen.

Anwender will suchen

Die Prozedur speichert zunächst den aktuellen Datensatz (Zeile 02) und fragt mit der *Requery*-Methode die Datensatzherkunft des Listenfelds erneut ab (Zeile 03). Damit ist sichergestellt, dass im Listenfeld alle aktuellen Datensätze angezeigt und zur Auswahl bereitgestellt werden. Dann wird das Listenfeld sichtbar geschaltet (Zeile 05). Quasi als Nebenarbeiten werden noch die Beschriftung der Schaltfläche *btnSuche* (Zeile 05) und der *ControlTip*-Text (Zeile 07) eingestellt. Dies bereitet den Fall vor, dass der Anwender die Suche abbrechen will.

Anwender bricht Suche ab

Wenn der Anwender nicht suchen will und auf die Schaltfläche *btnSuche* mit der Beschriftung *Abbruch* klickt, ist das Listenfeld sichtbar, und die *If*-Anweisung in Zeile 01 verzweigt über das Schlüsselwort *Else* (Zeile 09) in den *True*-Teil (ab Zeile 10). Das Listenfeld wird unsichtbar geschaltet und die Schaltfläche *btnSuche* erhält wieder die Beschriftung *Suche...* (Zeilen 12 und 13).

Das Listenfeld „lstSuche“

Das Listenfeld *lstSuche* ist als ungebundenes Listenfeld im Formular *frmPersonen* eingefügt. Die Anordnung ist auf *Vordergrund* eingestellt.

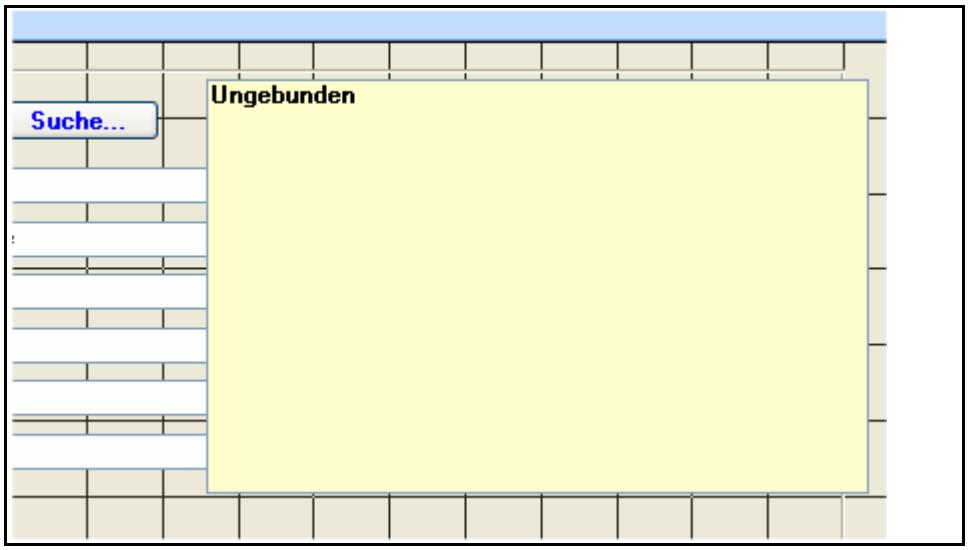

Abb. 3: Listenfeld im Formular

Als Eigenschaft *Datensatzherkunft* oder *RecordSource* für das Listenfeld dient die folgende SQL-Anweisung:

```
SELECT PersonID, Nachname, Vorname, Firma FROM
tblPersonen ORDER BY Nachname, Vorname;
```

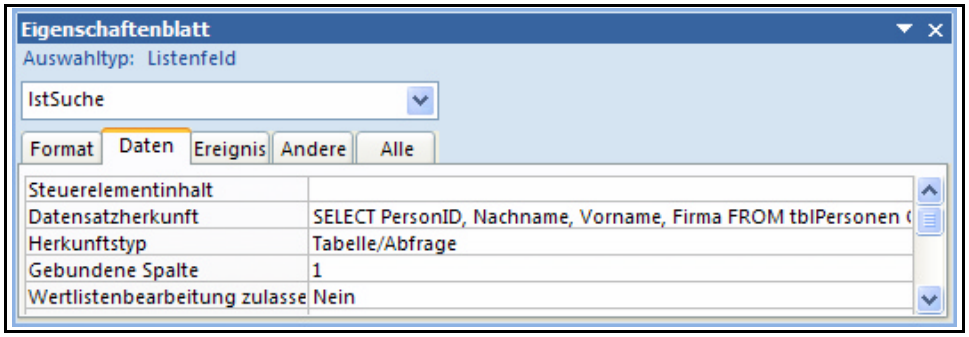

Abb. 4: Datensatzherkunft des Listenfelds

Die weiteren notwendigen Eigenschafteneinstellungen für das Listenfeld entnehmen Sie bitte der folgenden Tabelle:

Eigenschaft	Einstellung
Gebundene Spalte	1
Spaltenanzahl	4
Spaltenüberschriften	Ja
Spaltenbreiten	0cm;3cm;2,501cm;3cm
Sichtbar	Nein
In Reihenfolge	Nein

Tab. 2: Eigenschafteneinstellungen für das Listenfeld *lstSuche*

Die Verknüpfung des Listenfelds zum Formular erfolgt über das Primärschlüsselfeld *PersonID*. Da dessen Eigenschaft *Spaltenbreite* – Gebundene Spalte 1 – hier auf den Wert *0 cm* eingestellt ist, ist es also nicht sichtbar.

Der Suchvorgang

Der Suchvorgang wird durch das Anklicken einer Zeile im Listenfeld auslöst. Dabei wird das Ereignis

- *Nach Aktualisierung* oder *AfterUpdate*

ausgelöst und eine Ereignisprozedur ausgeführt.

Ereignis After Update

Dieses Ereignis bietet unter anderem die Möglichkeit, den im Listenfeld vorhanden Wert auszuwerten. Das ist bei dieser Verarbeitung der Fall.

Die Ereignisprozedur „AfterUpdate“

VBA-Code

In der Prozedur für das Ereignis *AfterUpdate* wird der mit dem Listenfeld übereinstimmende Datensatz im Formular gesucht und dort zum aktuellen

Datensatz gemacht. Anschließend werden noch einige Abschlussarbeiten erledigt.

Der VBA-Code dieser Prozedur ist im folgenden Listing abgedruckt:

```
Private Sub lstSuche_AfterUpdate()

01   Dim rs As DAO.Recordset

02   Set rs = Me.RecordsetClone

03     rs.FindFirst "PersonID = " & Me!lstSuche

04     Me.Bookmark = rs.Bookmark

05   Set rs = Nothing

' Restarbeiten

06     With Me

07       btnSuche.ControlTipText = "Person aus einer
Liste auswählen"

08       Vorname.SetFocus

09       lstSuche.Visible = False

10       btnSuche.Caption = "Auswahl..."

11     End With

End Sub
```

Beschreibung

Diese Prozedur ist faktisch in die eigentliche Suche (Zeilen 01 bis 05) und die Ausführung von Restarbeiten aufgeteilt (Zeilen 06 bis 10). Wir beginnen mit dem ersten Teil.

Suche

Die Prozedur verwendet die Eigenschaft *RecordsetClone* des *Form*-Objekts. Diese stellt eine exakte Kopie der Datensatzgruppe dar, die dem Formular zugrunde liegt. Eine solche geklonte Datensatzgruppe kann genauso behandelt werden wie die Datensatzgruppen, die Sie im letzten Kapitel kennengelernt haben.

Sie erkennen dies daran, dass nach Deklaration (Zeile 01) und Zuweisung (Zeile 02) in Zeile 03 die Methode *FindFirst* auf diese Datensatzgruppe angewendet wird. Damit und mit der *Bookmark*-Eigenschaft wird die im Listenfeld enthaltene *PersonID* im Formular *frmPersonen* gesucht (Zeile 04).

Restarbeiten

Zum Abschluss werden in der Prozedur dann noch innerhalb einer *With*-Anweisung der *ControlTip*-Text geändert (Zeile 07), der Fokus auf ein anderes Steuerelement im Formular gesetzt (Zeile 08), das Listenfeld wieder unsichtbar geschaltet (Zeile 09) und die Beschriftung der Schaltfläche *btnSuche* angepasst (Zeile 10) – typische Restarbeiten also.

Aktuellen Datensatz drucken

Eine oftmals gestellte Anforderung an die Funktionalität von Formularen ist, den Ausdruck des aktuellen Datensatzes zu ermöglichen. Grundvoraussetzung dafür ist, dass ein entsprechender Bericht vorhanden ist, dessen Datensatzherkunft mit der des Formulars übereinstimmt.

Alles Weitere kann vom Formular aus mit Hilfe einer Schaltfläche und einer kleinen Ereignisprozedur erledigt werden.

1. Fügen Sie in den Formularfuß eine Befehlsschaltfläche mit der Beschriftung *Drucken* ein.
2. Stellen Sie die Eigenschaft *Name* der Schaltfläche mit *btnDruck* ein.
3. Erstellen Sie dann für das Ereignis *Beim Klicken* der Schaltfläche folgende Ereignisprozedur:

Ereignisprozedur

```
Private Sub btnDruck_Click()

   ' Eventuelle Datensatzänderungen speichern

01    DoCmd.RunCommand acCmdSaveRecord

   ' Bericht in der Vorschau öffnen

02    DoCmd.OpenReport "repPersonen", acViewPreview,
, _

                                "PersonID = " &
Me!PersonID

End Sub
```

4. Kompilieren Sie die Prozedur und speichern Sie das Modul.
5. Schließen Sie die Entwurfsansicht.

Beschreibung

In Zeile 01 werden alle eventuellen Datensatzänderungen gespeichert. Ohne diese Zeile wäre folgendes Szenario denkbar:

Sie nehmen Änderungen an einem Datensatz vor. Diese Änderungen werden aber erst beim Datensatzwechsel oder dem Schließen des Formulars gespeichert. Wenn Sie also den Bericht aufrufen, werden die Änderungen am Datensatz nicht mit ausgedruckt.

In Zeile 02 wird der Bericht mit der Methode *DoCmd.OpenReport* geöffnet. Die Konstante *acViewPreview* legt die Ansicht *Seitenansicht* fest, und als Öffnungsargument wird der Wert des Primärschlüsselfelds des aktuellen Datensatzes übergeben.

Bei einem Klick auf die Schaltfläche *btnDruck* haben Sie das gewünschte Ergebnis vor sich auf dem Bildschirm.

Abb. 5: Der Bericht in der Seitenansicht

Formular filtern

Access stellt in Formularen verschiedene Möglichkeiten zur Verfügung, Daten zu filtern. Vor diesem Hintergrund betrachtet, wäre das folgende Beispiel also überflüssig:

Das Ergebnis

Information

In einem Artikelformular – Datensatzherkunft die Tabelle *tblArtikel* – sollen die Artikel mit Hilfe eines Kombinationsfelds nach den einzelnen Kategorien gefiltert werden.

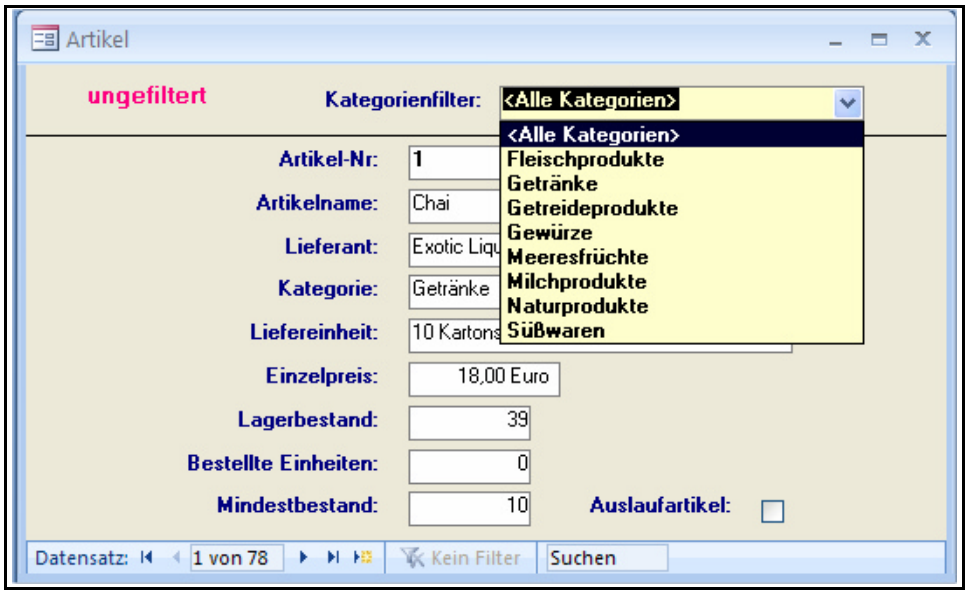

Abb. 6: Artikelformular

Konkret bedeutet dies: Wenn in dem Kombinationsfeld eine bestimmte Kategorie ausgewählt wird, werden in dem Formular nur die Artikel angezeigt, die zu dieser Kategorie gehören. Außerdem besteht noch de Möglichkeit, in den Kombinationsfeld die Anzeige aller Artikel auszuwählen. Als nette Spielerei wird in dem Formular noch der aktuelle Status der Filterung in einem Bezeichnungsfeld angezeigt: *ungefiltert* beziehungsweise *gefiltert*.

Realisiert wird diese Funktionalität dadurch, dass abhängig von der Auswahl im Kombinationsfeld die Datensatzherkunft des Formulars entsprechend geändert wird. Und dies ist auch der Grund, warum die Darstellung eines solchen Beispiels Sinn macht:

um die Datensatzherkunft als Reaktion auf ein Ereignis zur Laufzeit zu ändern.

Das Kombinationsfeld

Das Herzstück der gesamten Funktionalität zur Umsetzung der Lösung sind das Kombinationsfeld und die dazugehörige Ereignisprozedur. Dabei sind natürlich

auch das Datenmodell oder die Beziehungen zwischen den Tabellen der kleinen Anwendung von Bedeutung.

Beziehungen

Die Tabelle *tblKategorien*, in der die Kategorien gespeichert sind, ist *1:n* mit der Artikeltabelle *tblArtikel* verknüpft.

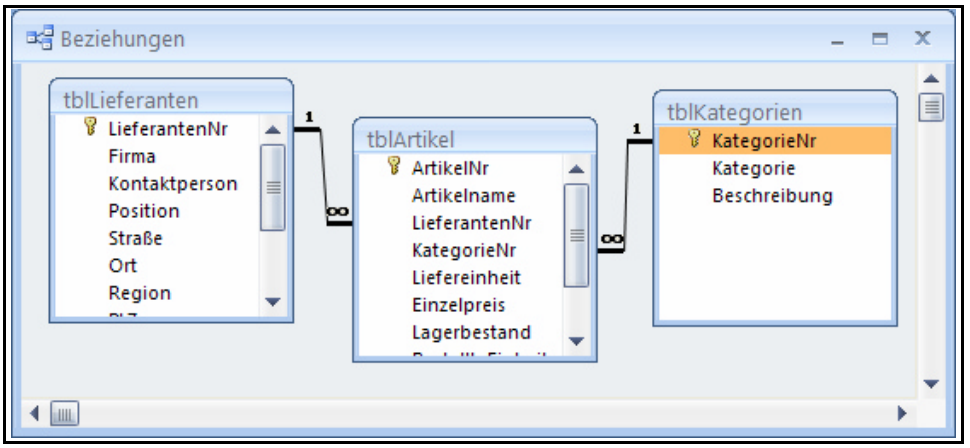

Abb. 7: Die Beziehungen

Auf dieser Grundlage kann nun das Kombinationsfeld erstellt werden.

Kombinationsfeld erstellen

1. Öffnen Sie das Formular in der Entwurfsansicht.
2. Fügen Sie in den Formularkopf das Kombinationsfeld *cboFilterKategorie* ein.
3. Stellen Sie die Eigenschaften für das Kombinationsfeld wie in der folgenden Tabelle ein:

Eigenschaft	**Einstellung**
Spaltenanzahl	2
Gebundene Spalte	1
Spaltenbreiten	0cm;4cm

Tab. 3: Eigenschafteneinstellungen für das Kombinationsfeld

Es fehlt jetzt noch die Einstellung der in diesem Zusammenhang wichtigsten Eigenschaft: der Datensatzherkunft.

Datensatzherkunft

Information

Damit das Kombinationsfeld die gewünschte Funktionalität bietet, muss die Datensatzherkunft eine UNION-SELECT-Anweisung sein. UNION-Abfragen bieten die Möglichkeit, entweder die Daten mehrerer gleichartig aufgebauter Tabellen in einer Abfrage zu vereinen oder Werte mit in die Abfrage aufzunehmen, die gar nicht in der zugrunde liegenden Tabelle vorhanden sind.

Tipp

Dieser Abfragentyp kann nicht im Abfragenentwurf erstellt werden, sondern muss direkt in die SQL-Ansicht der Abfrage eingegeben werden.

Wir benötigen allerdings nur den SQL-Code der Abfrage, der direkt im Eigenschaftenblatt in das Feld *Datensatzherkunft* eingegeben werden kann:

UNION-Abfrage

```
SELECT [KategorieNr], [Kategorie] FROM tblKategorien
UNION SELECT 0 AS [KategorieNr], '<Alle Kategorien>'
AS Kategorie FROM tblKategorien ORDER BY [Kategorie];
```

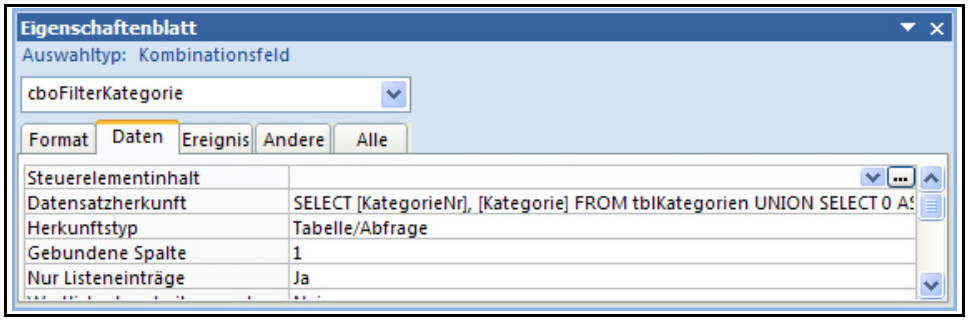

Abb. 8: SQL-Anweisung als Datensatzherkunft

Als gespeicherte Abfrage schaut das Abfrageergebnis dieser Abfrage wie in der folgenden Abbildung aus. Dabei wird auch die Wirkungsweise der *UNION*-Abfrage deutlich:

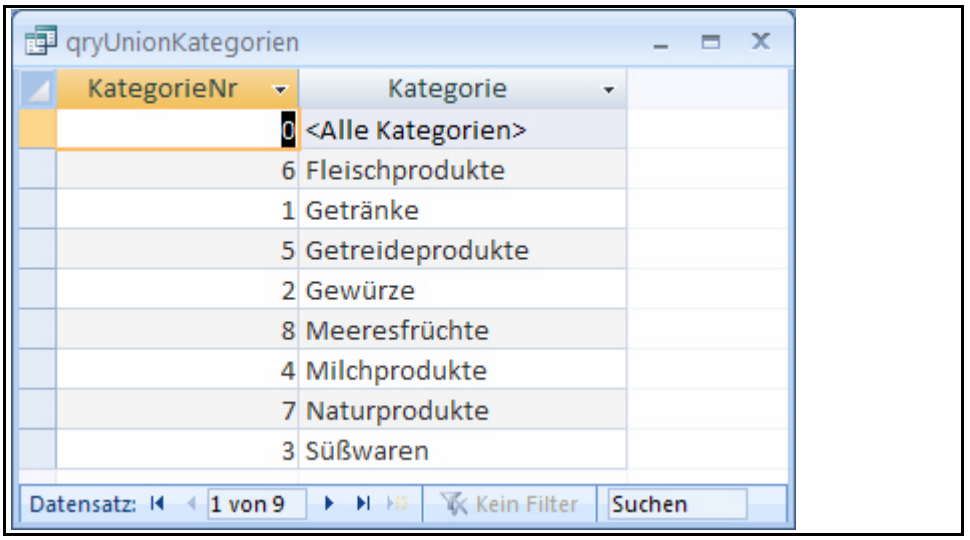

Abb. 9: Abfrageergebnis

Zusätzlich zu den bereits in der Tabelle vorhandenen Kategorien wird eine neue, in der Tabelle nicht vorhandene Kategorie *0* mit dem Namen *<Alle Kategorien>* erzeugt.

Auf dieser Basis kann nun das Kombinationsfeld von einer Ereignisprozedur ausgewertet werden.

Die Ereignisprozedur

Für die Ereignisprozedur benötigen wir das Ereignis *Nach Aktualisierung* oder *AfterUpdate* des Kombinationsfelds.

Ereignis AfterUpdate

Das Ereignis *AfterUpdate* für ein Kombinationsfeld bietet auch die Möglichkeit, den dort aktuellen Wert weiter zu verarbeiten. Genau dies geschieht in der folgenden Ereignisprozedur.

4. Erstellen Sie für das Ereignis *AfterUpdate* des Kombinationsfelds folgende Ereignisprozedur:

Ereignisprozedur

```
Private Sub cboFilterKategorie_AfterUpdate()

01   Dim strSQL As String

02   Dim IntKategorieNr As Integer

03   IntKategorieNr = Me!cboFilterKategorie.Value

04     If IntKategorieNr = 0 Then

05       strSQL = "SELECT * FROM tblArtikel"

06       Me.RecordSource = strSQL

07       Me.Requery

08       Me!lblFilterStatus.Caption = "ungefiltert"

09     Else

10       strSQL = "SELECT * FROM tblArtikel WHERE
KategorieNr=" & IntKategorieNr

11       Me.RecordSource = strSQL

12       Me.Requery

13       Me!lblFilterStatus.Caption = "gefiltert"

    End If

End Sub
```

5. Kompilieren Sie die Prozedur und speichern Sie das Modul.
6. Schließen Sie die Entwurfsansicht.

Beschreibung

In der Ereignisprozedur müssen zwei mögliche Zustände behandelt werden:

- Es sollen alle Datensätze angezeigt werden.
- Es sollen die Datensätze einer bestimmten Kategorie angezeigt werden.

Also benötigen wir für diese Fallunterscheidung eine *If*-Anweisung. Zuvor jedoch müssen zwei notwendige Variablen deklariert werden.

Die Variable *strSQL* in Zeile 01 speichert den String für eine SQL-Anweisung, und in der Variablen *intKategorieNr* in Zeile 02 wird der Zahlenwert der ausgewählten Kategorie aus dem Kombinationsfeld gespeichert.

Mit dieser Zuweisung beginnt auch die Verarbeitung in Zeile 03. Die *If*-Anweisung (Zeile 04) überprüft, ob dieser Wert *0* ist und alle Datensätze angezeigt werden sollen.

Im *True*-Fall wird in der Variablen *strSQL* eine SQL-Anweisung gespeichert, die alle Datensätze der Tabelle *tblArtikel* übergibt (Zeile 05). Diese SQL-Anweisung wird dann in Zeile 06 der Eigenschaft *Recordsource* des Formulars übergeben und in Zeile 07 mit der Methode *Requery* abgefragt. Das war's im Wesentlichen. In Zeile 08 wird dann noch die Beschriftung des Statusfelds auf *ungefiltert* eingestellt, doch das ist rein nebensächlich.

Falls die *If*-Anweisung in Zeile 04 *False* zurückliefert, ist auch der Wert der Variablen *intKategorieNr* logischerweise ungleich *0*. Der Anwender hat also eine Kategorie ausgewählt, und es wird in den *Else*-Teil (Zeile 08) verzweigt.

Dieser unterscheidet sich in der Hauptsache durch die Zusammensetzung der SQL-Anweisung in Zeile 10. Diese verfügt jetzt über eine *WHERE*-Klausel, in der die zurückgegebenen Datensätze anhand des Wertes der *KategorieNr* aus dem Kombinationsfeld eingeschränkt werden.

Kombinationsfeld-Kosmetik

Zum Abschluss dieses Abschnitts noch eine Kleinigkeit zu dem eben besprochenen Kombinationsfeld. In diesem Kombinationsfeld wird nach dem Öffnen des Formulars der Eintrag <*Alle Datensätze*> angezeigt.

Abb. 10: Öffnen des Formulars: Eintrag im Kombinationsfeld

Eine solche Anzeige ist aber nicht so ohne Weiteres zu haben, denn standardmäßig zeigt ein solches Kombinationsfeld gar keinen Eintrag an.

Abb. 11: Öffnen des Formulars: Leeres Kombinationsfeld

Das schaut allerdings nicht sonderlich gut aus, wie Sie mir sicherlich zustimmen. Um dies zu ändern, ist ein kleiner Eingriff notwendig, der sich allerdings auf eine einzige Zeile VBA-Code beschränkt.

Kombinationsfelder – wie auch Listenfelder – verfügen unter anderem über die Eigenschaft *ItemData*. Mit Hilfe dieser Eigenschaft kann auf die Zeilennummern eines Kombinationsfelds zugegriffen werden, wobei die erste Zeile die Nummer *0* hat. Damit können Sie entweder die Zeilen durchlaufen oder die entsprechenden Daten ausgeben.

1. Öffnen Sie das Formular in der Entwurfsansicht.
2. Zeigen Sie das Eigenschaftenblatt für das Formular an.

Erstellen Sie für das Ereignis *Beim Laden* des Formulars folgende Ereignisprozedur:

Formularereignis Load

Das Ereignis *Beim Laden* oder *Load* für ein Formular tritt dann ein, wenn das Formular einschließlich Steuerelementen zwar geöffnet, aber noch nicht sichtbar ist. Es eignet sich unter anderem für das Setzen von Werten für Steuerelemente.

```
Private Sub Form_Load()

    Me!cboFilterKategorie =
Me!cboFilterKategorie.ItemData(0)

End Sub
```

3. Kompilieren Sie die Prozedur und speichern Sie das Modul.
4. Schließen Sie die Entwurfsansicht.

Beschreibung

In der Prozedur wird beim Laden des Formulars dem Kombinationsfeld *cboFilterKategorie* der Wert der ersten Zeile des gleichen Kombinationsfelds zugewiesen und angezeigt.

Anlage- und Änderungsdatum im Formular

Information

An Datenbankanwendungen wird oftmals die Anforderung gestellt, das Anlegen beziehungsweise Bearbeiten von Datensätzen nachvollziehen zu können – also zu protokollieren.

Da eine solche Funktionalität bei Access standardmäßig nicht vorhanden ist, muss zu deren Realisierung selbst Hand angelegt werden. Wir beginnen mit dem Anlegen der beiden zusätzlichen Datenfelder in der betreffenden Tabelle.

Tabellenfelder anlegen

1. Öffnen Sie die Tabelle in der Entwurfsansicht.
2. Legen Sie ein neues Feld *AnlageDatum* mit dem Felddatentyp *Datum/Uhrzeit* an.

3. Wählen Sie dann im Bereich *Feldeigenschaften* als *Format* für dieses Feld den Wert *Standarddatum* aus.

Feldeigenschaften

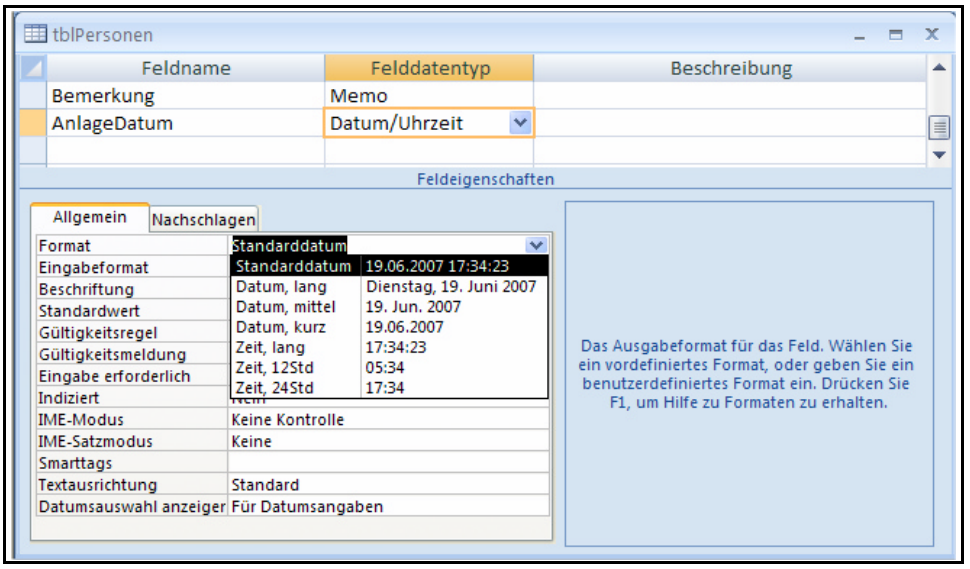

Abb. 12: Feldeigenschaften

4. Legen Sie anschließend noch ein weiteres Feld *AenderungsDatum* mit identischen Feldeigenschaften an.

5. Speichern Sie Änderungen am Tabellenentwurf und verlassen Sie die Entwurfsansicht der Tabelle.

Formular ändern

Im zweiten Arbeitsschritt müssen die beiden neuen Felder in das Formular eingefügt werden. Das sollte für Sie kein Problem darstellen.

Da es sich bei den beiden neuen Feldern um reine Anzeigefelder handelt, stellen Sie die Eigenschaften der beiden Felder wie in der folgenden Tabelle beschrieben ein:

Eigenschaften

Eigenschaft	Einstellung
Datumsauswahl anzeigen	Nie
Aktiviert	Nein
Gesperrt	Ja
In Reihenfolge	Nein

Tab. 4: Eigenschafteneinstellungen für die Anzeigefelder

Tipp

Um die Felder auch optisch als reine Anzeigefelder zu kennzeichnen, wählen Sie am besten auch noch eine andere Schrift- oder Hintergrundfarbe für sie aus.

Anschließend sollte das Formular im Entwurf wie in der folgenden Abbildung ausschauen:

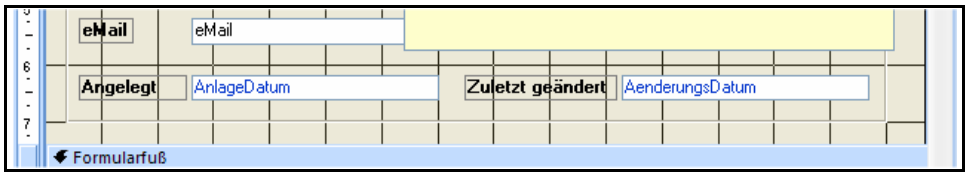

Abb. 13: Die neu eingefügten Felder im Formularentwurf

Jetzt fehlt nur noch eine passende Ereignisprozedur, die beim Anlegen und Ändern eines Datensatzes die Zeitangaben in die entsprechenden Tabellenfelder schreibt.

Die Ereignisprozedur

Für unsere Ereignisprozedur benötigen wir das Ereignis *Vor Aktualisierung* oder *BeforeUpdate* für das Formular.

Ereignis BeforeUpdate

Dieses Ereignis *BeforeUpdate* tritt nach dem Auslösen des Speicherns für den Datensatz, aber vor dem eigentlichen Speichern auf. Es kann unter anderem für das Setzen von Feldwerten verwendet werden.

6. Aktivieren Sie das Eigenschaftenblatt für das Formular und klicken Sie dort am besten auf das Register *Ereignisse*.
7. Wählen Sie das Feld *Vor Aktualisierung* aus und erstellen Sie die folgende Ereignisprozedur:

Ereignisprozedur

```
Private Sub Form_BeforeUpdate(Cancel As Integer)

01   If Me.NewRecord = True Then

02     Me!AnlageDatum = Now

03   Else

04     Me!AenderungsDatum = Now

05   End If

End Sub
```

8. Kompilieren Sie die Prozedur und speichern Sie das Modul.
9. Schließen Sie die Entwurfsansicht.

Eigenschaft NewRecord

Mit der Eigenschaft *NewRecord* können Sie bestimmen, ob der aktuelle Datensatz ein neuer Datensatz ist oder nicht. Bei einem neuen Datensatz hat die Eigenschaft *NewRecord* den Wert *True*, unabhängig davon, ob der Datensatz bereits bearbeitet wurde oder nicht.

Beschreibung

In Zeile 01 fragt die *If*-Anweisung den Wert der Eigenschaften *NewRecord* ab. Im *True*-Fall handelt es sich um einen neuen Datensatz, und dem Feld *AnlageDatum* wird der Wert der Funktion *Now* zugewiesen (Zeile 02).

Durch die Verwendung der Funktion *Now* wird zusätzlich zum Datum auch die Uhrzeit in das Feld geschrieben. Falls Sie dies für überflüssig erachten, verwenden Sie hier einfach die Funktion *Date*, die lediglich das Datum zurückgibt.

Im *False*-Fall – also kein neuer Datensatz – verzweigt die *If*-Anweisung zum *Else*-Teil (Zeile 03) und aktualisiert dort in Zeile 04 das Feld *AenderungsDatum* mit der Funktion *Now*.

Anschließend schaut das Formular nach Datensatzänderungen wie in der folgenden Abbildung aus:

Abb. 14: Fertiges Formular

Erweiterung der Funktionalität

Eine naheliegende und leicht zu realisierende Erweiterung dieser Funktionalität wäre, jeweils in einem zweiten Feld den Benutzernamen des aktuell an der Access-Datenbank angemeldeten Benutzers zu erfassen. Dies macht allerdings nur in einer Mehrbenutzerumgebung Sinn.

Falls eine solche nicht eingerichtet ist, sind Sie beim Starten von Access automatisch als Benutzer *Admin* angemeldet. Sie können dies leicht durch den Aufruf der *CurrentUser*-Methode des *Application*-Objekts im Direktfenster nachvollziehen:

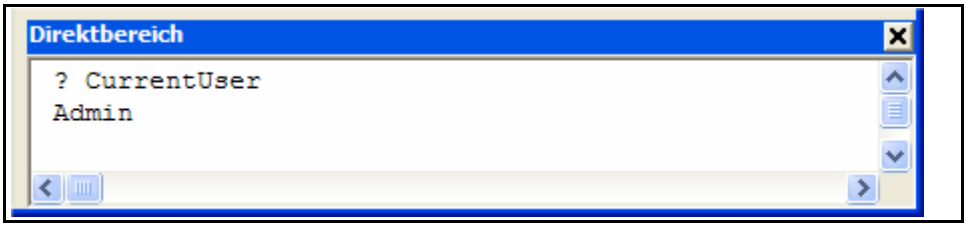

Abb. 15: Ausgabe des aktuellen Benutzers im Direktfenster

Eine entsprechende Erweiterung der eben vorgestellten Prozedur für zwei weitere Textfelder *AngelegtVon* beziehungsweise *GeaendertVon* sähe wie folgt aus:

```
...

Me!AngelegtVon = CurrentUser

...

Me!GeaendertVon = CurrentUser

...
```

Manueller AutoWert

Sie kennen mit Sicherheit den Felddatentyp *AutoWert*, der in Tabellen in der Regel für Primärschlüsselfelder verwendet wird und die Anzahl der Datensätze automatisch hochzählt.

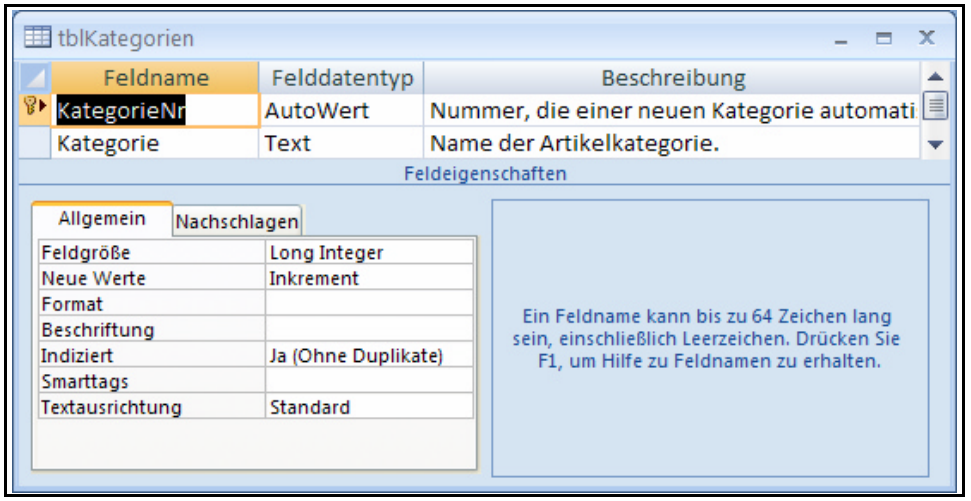

Abb. 16: Tabelle in der Entwurfsansicht

Information

In den allermeisten Fällen ist die Funktionalität, die ein solches *AutoWert*-Feld zur Verfügung stellt, vollkommen ausreichend. Es gibt allerdings auch Fälle; wo dies eben nicht zutrifft:

- Das *AutoWert*-Feld soll einen ganz bestimmten Startwert haben, der sich auch bei einer Komprimierung der Datenbank nicht verändert.
- Das *AutoWert*-Feld soll neben dem Zähler noch eine Zeichenkette enthalten, um beispielsweise einen Produktcode oder ein anderes Kürzel im AutoWert unterzubringen.

Manueller AutoWert

In einem solchen Fall benötigen Sie als Ersatz eine Funktionalität, die man als manuellen AutoWert bezeichnen könnte. Ein solcher manueller *AutoWert* kann mit Hilfe einer passenden VBA-Funktion realisiert werden. Bei einer solchen Lösung geht es also im Grunde genommen darum, fehlende Möglichkeiten von Access zu erweitern.

Die in dem folgenden Beispiel vorgestellte Funktion zur Erstellung eines manuellen AutoWerts ist recht flexibel und bietet folgende Möglichkeiten:

- Der *AutoWert* kann entweder ein reiner Zahlenwert sein oder zusätzlich noch eine beliebige Zeichenkette enthalten.
- Der *AutoWert* kann auf einen beliebigen festen Startwert eingestellt werden.

Tabelle, Formular und Modul

Information

In der Tabelle einer solchen Lösung benötigen wir zunächst ein passendes Primärschlüsselfeld. Bei einer Zeichenkette im *AutoWert* muss der Felddatentyp des betreffenden Feldes natürlich logischerweise *Text* sein. In allen anderen Fällen können Sie auch den Datentyp *Zahl* verwenden.

Tabellenfelder

1. Erstellen Sie eine neue Tabelle mit den drei Textfeldern *TableID*, *TableField_1* und *TableField_2*.

2. Weisen Sie dem Textfeld *TableID* das Attribut *Primärschlüssel* zu.

3. Speichern Sie die Tabelle unter dem Namen *tblAutoWert*.

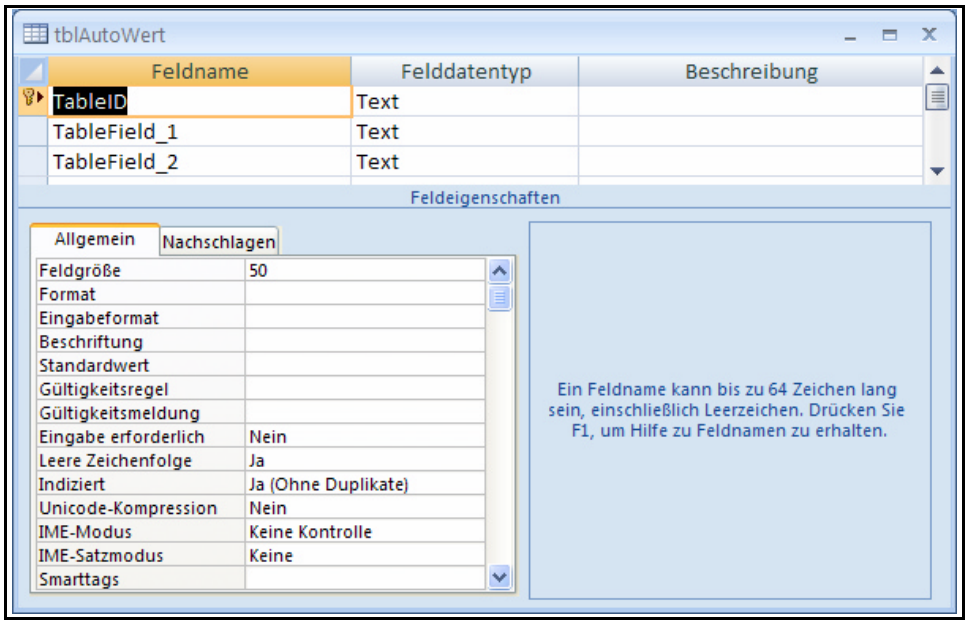

Abb. 17: Die Tabelle *tblAutoWert* in der Entwurfsansicht

Das Formular

4. Erstellen Sie anschließend mit Hilfe des Assistenten auf Grundlage der Tabelle ein Formular und speichern Sie es unter dem Namen *frmAutoWert* ab.

5. Erstellen Sie abschließend noch ein neues Standardmodul und speichern Sie es unter dem Namen *basAutoWert* ab.

Damit sind die vorbereitenden Arbeiten abgeschlossen.

Die Funktion „AutoWertM“

6. Erstellen Sie im Modul *basAutoWert* die im folgenden Listing abgedruckte Funktion *AutoWertM*.

Tipp

In dem folgenden Listing wurde aus Gründen der Übersichtlichkeit die Fehlerbehandlung weggelassen.

```
01 Public Function AutoWertM(strFieldForm As String,
_
    strFieldTable As String, strTableName As String,
_
    Optional strPraefix As String, Optional lngStart
As Long = 1)

02   Dim lngNewID As Long

03   Dim frm As Form

04   Dim ctl As Control

04   Set frm = Screen.ActiveForm

06   Set ctl = frm(strFieldForm)

07     If IsNull(ctl) Then

08       If (IsNull(DMax("[" & strFieldTable & "]",
strTableName))) Then

09         lngNewID =

10       Else

11         lngNewID = DMax("[" & strFieldTable & "]",
strTableName)

12           If Len(Trim(Nz(strPraefix))) > 0 Then

13             lngNewID = Right(lngNewID,
(Len(lngNewID) - Len(Trim(Nz(strPraefix)))))

14           End If
```

```
15              lngNewID = lngNewID + 1
16          End If
17            If Len(Trim(Nz(strPraefix))) > 0 Then
18              ctl = strPraefix & lngNewID
19            Else
20              ctl = lngNewID
21            End If
22      End If
23 End Function
```

Beschreibung der Funktion

Die Beschreibung der Funktion muss sich hier aus Platzgründen auf die wesentlichen Merkmale beschränken.

Funktionsparameter

Wir beginnen mit den Übergabeparametern der Funktion in Zeile 01, die in der folgenden Tabelle beschrieben werden:

Parameter	**Beschreibung**
strFieldForm	Formularfeld für den *AutoWert*
strFieldTable	Tabellenfeld für den *AutoWert*
strTableName	Tabellenname
Optional strPraefix	Optionaler Präfix des *AutoWerts*
Optional lngStart	Optionaler Startwert des *AutoWerts*. Voreingestellt ist der Wert 1

Tab. 5: Übergabeparameter der Funktion

Tipp

Mit Hilfe der Parameterübergabe beim Aufruf können Sie die Funktion ohne irgendeine Anpassung in jeder beliebigen Tabellen-/Formularkombination verwenden.

Beschreibung

In Zeile 08 wird mit Hilfe der eingebauten Funktion *DMax* der Wert des benutzerdefinierten *AutoWert*-Feldes in der Tabelle ausgelesen. Wenn die Funktion *Null* zurückgibt, dann handelt es sich um den ersten Datensatz. Dieser bekommt dann – wenn kein anderer Startwert beim Funktionsaufruf übergeben wurde – den Wert *1*.

Falls bereits Daten vorhanden sind, ermittelt die Funktion *DMax* den maximalen Wert und erhöht diesen dann um den Wert 1 (Zeile 11).

In den Zeilen 12 und 13 wird bei einem als Zeichenkette übergebenen Präfix mit Hilfe der eingebauten VBA-Funktionen *Len*, *Trim* und *Nz* der numerische Teil entfernt.

In den Zeilen 17 bis 21 wird dann innerhalb einer *If*-Anweisung – mit oder ohne Zeichenkettenanteil –der neue *AutoWert* an das Feld im Formular übergeben und so auch in der Tabelle gespeichert.

Der Funktionsaufruf

Information

Der Aufruf der Funktion erfolgt durch das Formularereignis *BeforeUpdate*. Das liegt auf der Hand und bedeutet, dass der neu angelegte Datensatz seinen *AutoWert* erhält, bevor er gespeichert wird.

Je nachdem, ob der *AutoWert* nur eine Zahl, ein Präfix oder zusätzlich noch einen Startwert enthält, variieren die Ausdrücke in der Ereignisprozedur *BeforeUpdate* leicht voneinander.

Beispiel

- Reiner Zahlenwert:

```
Private Sub Form_BeforeUpdate(Cancel As Integer)

  Dim Aufruf

  Aufruf = AutoZaehlerM("TableID", "strTableID",
"tblAutoWert")

End Sub
```

- Präfix *ADR_* plus Zahlenwert mit dem Startwert *100*:

```
Aufruf = AutoZaehlerM("TableID", "strTableID",
"tblAutoWert", "ADR_", 100)
```

Ein solcher *AutoWert* schaut dann im Formular wie in der folgenden Abbildung aus:

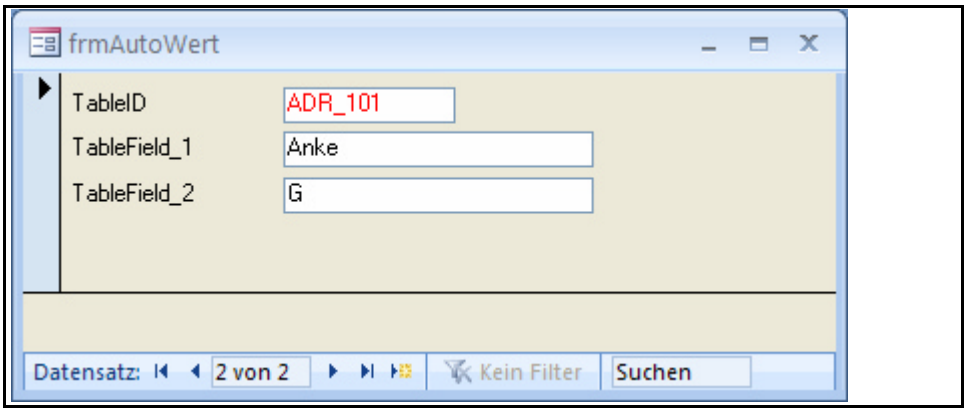

Abb. 18: AutoWert mit Präfix und Zahl

- Reiner Zahlenwert mit dem Startwert 100:

```
Aufruf = AutoZaehlerM("TableID", "strTableID",
"tblAutoWert", , 100)
```

Datensatzänderungen bestätigen

Information

Datensatzänderungen in Access-Formularen werden standardmäßig beim Wechsel des Datensatzes beziehungsweise beim Schließen des Formulars kommentarlos gespeichert. Dieses Verhalten verunsichert manche Anwender. Sie befürchten nämlich, auch unbeabsichtigte Datensatzänderungen zu speichern.

In einem solchen Fall erstellen Sie einfach eine kleine Ereignisprozedur für das Formularereignis *BeforeUpdate*. Dort können Sie mit wenigen Zeilen VBA-Code erreichen, dass das Ändern eines Datensatzes von der ausdrücklichen Bestätigung durch den Anwender abhängig gemacht wird.

Ereignisprozedur

```
Private Sub Form_BeforeUpdate(Cancel As Integer)

01   If Me.Dirty Then

02     If MsgBox("Der Datensatz wurde geändert." &
vbCrLf & _

        "Soll die Änderung übernommen werden?", _

        vbYesNo + vbQuestion + vbDefaultButton2, _

        "Datensatzänderung") = vbYes Then

03     Else

04       Cancel = True

05       DoCmd.RunCommand acCmdUndo

06       Exit Sub

07     End If

08  End If

End Sub
```

Beschreibung

In der Prozedur erkennen Sie zwei ineinander verschachtelte *If*-Anweisungen. Die äußere *If*-Anweisung (Zeile 01) wertet die Eigenschaft *Dirty* des Formulars aus.

Eigenschaft Dirty

Diese Eigenschaft hat den Wert *True*, wenn der Datensatz seit dem letzten Speichern verändert wurde. Im *False*-Fall wird die Prozedur verlassen.

Im *True*-Fall wertet eine zweite *If*-Anweisung (Zeile 02) den Rückgabewert der *MsgBox*-Funktion aus. Wenn der Anwender auf die Schaltfläche *Ja* klickt, wird die Prozedur ebenfalls verlassen, und die Datensatzänderungen werden gespeichert.

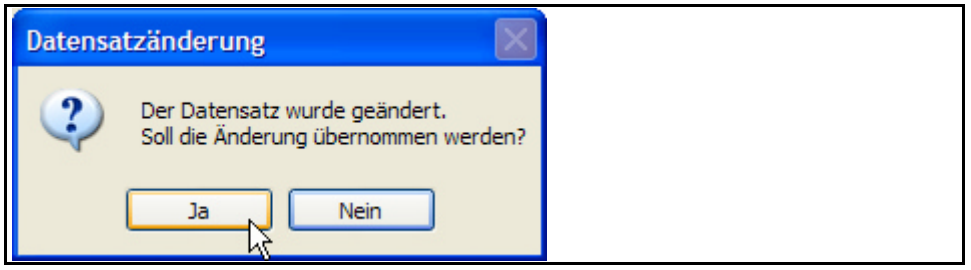

Abb. 19: Bestätigung einer Datensatzänderung

Die Konstante *vbDefaultButton2* macht übrigens die zweite Schaltfläche *Nein* zur standardmäßig aktivierten Schaltfläche. Das heißt, wenn der Anwender aus Versehen beim Anzeigen des Meldungsfensters die Taste Enter drückt, wird keine Datensatzänderung gespeichert.

Damit sind wir aber bereits beim *False*-Teil dieser *If*-Anweisung angekommen (Zeilen 03 bis 06). Zunächst wird in Zeile 04 die Konstante *Cancel* auf *True* gesetzt und dadurch der Speichervorgang abgebrochen.

RunCommand-Methode

Anschließend wird in Zeile 05 die *RunCommand*-Methode mit der Konstanten *acCmdUndo* ausgeführt. Dadurch werden alle Änderungen am Datensatz rückgängig gemacht. Direkt danach wird die Prozedur verlassen (Zeile 06).

VBA in Berichten

Zum Abschluss dieses Kapitels finden Sie noch zwei Beispiele zur Verwendung von VBA in Berichten.

Bericht ohne Daten

Information

Falls die Datensatzquelle eines Berichts keine Daten übergibt, werden leere Berichte gedruckt. Dies können Sie mit Hilfe des Ereignisses *Bei Ohne Daten* oder *NoData* und einer entsprechenden Ereignisprozedur leicht abfangen.

Ereignisprozedur

```
Private Sub Report_NoData(Cancel As Integer)

01   MsgBox "Dieser Bericht enthält keine Daten!", _

  vbExclamation Or vbOKOnly, "Keine Daten vorhanden"

02   Cancel = True

End Sub
```

Beschreibung

Beim Auslösen des Ereignisses *NoData* wird der Anwender durch ein Meldungsfenster entsprechend informiert (Zeile 01). Anschließend wird in Zeile 02 die Konstante *Cancel* auf *True* gesetzt und dadurch der ganze Vorgang abgebrochen.

Datensätze im Bericht seitenweise nummerieren

Information

Bestimmte Berichte – beispielsweise Listen – sind für den Leser übersichtlicher, wenn die dort gedruckten Datensätze seitenweise durchnummeriert sind. Eine solche Funktionalität erreichen Sie mit einem Textfeld als Zählerfeld im Detailbereich und einigen Zeilen VBA-Code.

Textfeld

1. Öffnen Sie den Bericht in der Entwurfsansicht.

2. Fügen Sie in den Detailbereich ein Textfeld ein und nennen Sie es *txtCount.*

3. Löschen Sie das zu dem eingefügten Textfeld gehörende Bezeichnungsfeld.

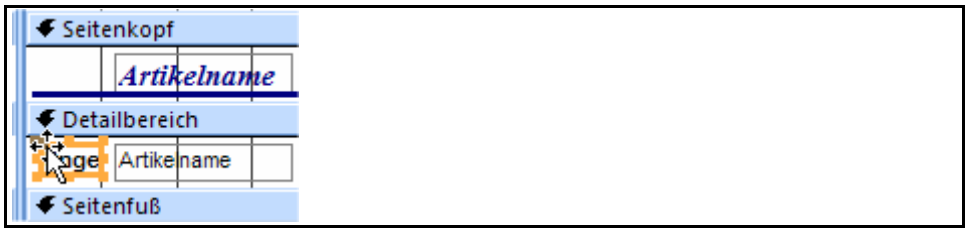

Abb. 20: Textfeld im Detailbereich

4. Zeigen Sie das Eigenschaftenblatt für den Detailbereich an und legen Sie für das Ereignis *Beim Drucken* oder *Print* die folgende Ereignisprozedur an:

Ereignisprozedur 1

```
Private Sub Detailbereich_Print(Cancel As Integer,
PrintCount As Integer)

  If PrintCount = 1 Then

    txtCount = txtCount + 1

  End If

End Sub
```

Beschreibung

Die Eigenschaft *PrintCount* gibt als Zahlenwert zurück, wie oft die *OnPrint*-Eigenschaft des betreffenden Bereichs ausgewertet wurde.

Information

Damit die Datensatzzählung auf jeder Berichtsseite von vorne beginnt, benötigen wir noch eine zweite Ereignisprozedur für das Ereignis *Beim Drucken* des Seitenkopfbereichs.

5. Zeigen Sie das Eigenschaftenblatt für den Seitenkopfbereich an und legen Sie für das Ereignis *Print* die folgende Ereignisprozedur an:

Ereignisprozedur 2

```
Private Sub Seitenkopfbereich_Print(Cancel As
Integer, PrintCount As Integer)

  txtCount = 0

End Sub
```

6. Kompilieren und speichern Sie das Modul.
7. Schließen Sie die Entwurfsansicht.