Lookup-Daten löschen

Wenn Sie mit Lookup-Daten arbeiten und dem Benutzer erlauben, Daten in die Lookup-Tabelle wie etwa eine Tabelle zum Verwalten von Kategorien einzugeben, sollten Sie auch eine Löschen-Funktion für diese Daten bereitstellen. Benutzer sind nämlich schnell übereifrig und legen ähnliche und somit unter Umständen redundante Daten an. Wenn dann fleißig Datensätzen den redundanten Lookupwerten zugeordnet wurden, ist guter Rat teuer: Wie die Dubletten entfernen, und was geschieht mit den bereits verknüpfte Daten? Dieser Artikel bringt Lichts ins Dunkel.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1802_LookupLoeschen.accdb**.

Ausgangslage

Wir beschäftigen uns in diesem Artikel mit den beiden Tabellen **tblArtikel** und **tblKategorien** (siehe Bild 1). Zu jedem Artikel kann der Benutzer eine Kategorie auswählen. Um das flexibel zu gestalten, soll der Benutzer eigene Kategorien eingeben können – und zwar, indem er diese einfach in das Kombinationsfeld zur Auswahl der Kategorien einträgt und eine kurze Bestätigung nach Eingabe eines neuen Eintrags bestätigt. Dabei geschieht es dann, dass der Benutzer eine neue Kategorie anlegt – wie etwa **Bier** –, aber übersieht, dass es ja bereits eine Kategorie namens **Getränke** gibt, in die der neue Artikel **Pilsener Quellbräu** aufgenommen werden kann. In diesem Fall soll der Benutzer später seinen Fehler korrigieren können, indem er die falsch angelegte Kategorie wieder löscht. Sollten jedoch bereits Datensätze der Tabelle **tblArtikel** mit den zu löschenden

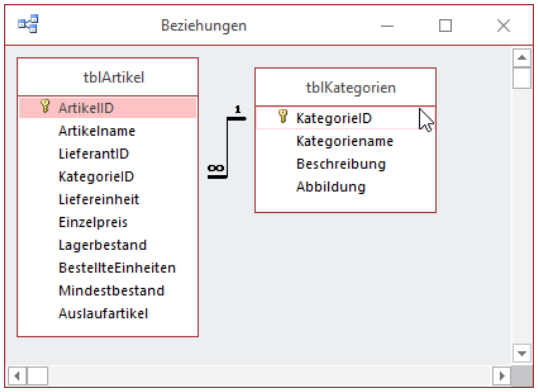

Bild 1: Tabellen der Beispieldatenbank

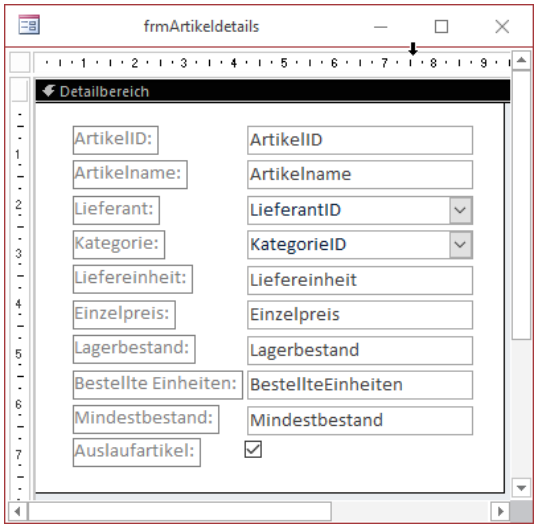

Bild 2: Entwurf des Formulars

Datensätzen aus der Tabelle **tblKategorien** angelegt sein, gelingt dies nicht so einfach: Wir haben nämlich referenzielle Integrität für die Beziehung zwischen den beiden Tabellen definiert, und zwar ohne Löschweitergabe. Das heißt, dass wir einen Datensatz aus der Tabelle **tblKategorien**, der mit mindestens einem Datensatz der Tabelle **tblArtikel** verknüpft ist, nicht einfach löschen können. Oder andersherum formuliert: Wir können nur leere Kategorien löschen.

Damit hat der Benutzer eine Aufgabe vor der Brust: Er muss nämlich alle Datensätze der Tabelle **tblArtikel** ausfindig machen, die mit der zu löschenden Kategorie verknüpft sind und diese mit einer anderen Kategorie verknüpfen und kann erst dann die überflüssige Kategorie löschen. Dies wollen wir im folgenden etwas einfacher gestalten.

Beispielformular

Ausgangspunkt dafür ist das Formular **tblArtikeldetails**, das im Entwurf wie in Bild 2 aussieht und über

die Eigenschaft **Datenherkunft** an die Tabelle **tblArtikel** gebunden wird. Da das Fremdschlüsselfeld **KategorieID** gleich im Tabellenentwurf als Nachschlagefeld ausgelegt wurde, erscheint es nach dem Hineinziehen der Felder aus der Feldliste in den Detailbereich gleich als Kombinationsfeld (übrigens genau wie das Feld **LieferantID**).

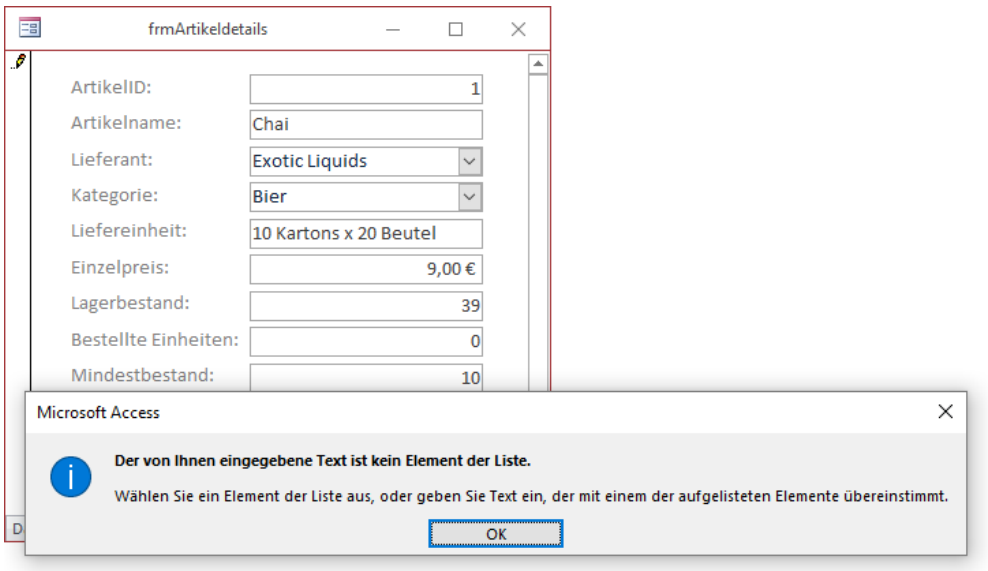

Bild 3: Im Standardzustand bewirkt das Einfügen einer neuen Kategorie diesen Fehler.

Wenn wir dann allerdings in die Formularansicht wechseln, erhalten wir die Fehlermeldung aus Bild 3. Kein Wunder: Der Eintrag ist in der Liste nicht bekannt.

Neuen Eintrag hinzufügen

Das ändern wir durch die Ereignisprozedur **Bei nicht in Liste** des Kombinationsfeldes, das wir zuvor in **cboKategorieID** umbenennen. Die Prozedur sieht wie in Listing 1 aus und liefert mit dem Parameter **NewData** den Wert des vom Benutzer neu hinzugefügten Eintrags. Die Prozedur fragt den Benutzer, ob die angegebene Kategorie neu in die Liste der Kategorien aufgenommen werden soll. Falls ja, führt sie eine **INSERT INTO**-Abfrage aus, welche einen entsprechenden neuen Datensatz zur Tabelle **tblKategorien** hinzufügt. In diesem Fall stellen wir den Wert des Rückgabeparameters **Response** auf **acDataErrAdded** ein, was Access mitteilt, dass ein neuer Eintrag hinzugefügt wurde.

Danach haben wir also eine neue Kategorie, die gegebenenfalls überflüssig ist, weil sie schon durch eine andere Kategorie abgedeckt wird, und der Benutzer fügt fleißig neue Artikel hinzu und weist sie dieser Kategorie zu. Nun benötigen wir Elemente für die Benutzeroberfläche, mit denen wir die Einträge der Tabelle **tblKategorien** verwalten können und dies es auch erlauben, den Artikeln mit einer zu löschenden Kategorie auf einfache Weise die richtige Kategorie zuzuweisen. Wir aber erledigen wir das auf effiziente Weise?

```
Private Sub cboKategorieID_NotInList(NewData As String, Response As Integer)
    Dim intResult As VbMsgBoxResult
    Dim db As DAO.Database
    intResult = MsgBox("Möchten Sie die Kategorie '" & NewData & "' hinzufügen?", vbYesNo, "Neue Kategorie")
    If intResult = vbYes Then
        Set db = CurrentDb
        db.Execute "INSERT INTO tblKategorien(Kategoriename) VALUES('" & NewData & "')", dbFailOnError
        Response = acDataErrAdded
    End If
End Sub
```

Listing 1: Hinzufügen einer neuen Kategorie zur Tabelle **tblKategorien**

Die übersichtlichste Methode ist die Anzeige eines neuen Formulars, dass die zu löschende Kategorie samt der verknüpfte Datensätze der Tabelle **tblArtikel** anzeigt. Außerdem sollte dieses Formular noch die Möglichkeit bieten, vor dem Löschen eine alternative Kategorie für die betroffenen Artikel auszuwählen – und das gegebenenfalls noch nach vorheriger Selektion der Artikel, um diese auch mehreren verschiedenen Kategorien zuordnen zu können.

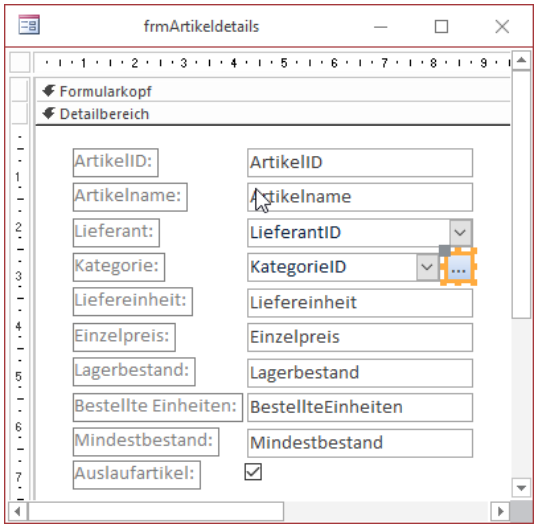

Bild 4: Schaltfläche zum Bearbeiten der Kategorien

in Bild 5 aus. Das obere Kombinationsfeld soll die Kategorie anzeigen, die im Formular **frmArtikeldetails** ausgewählt war, als der Benutzer auf die Schaltfläche mit den drei Punkten geklickt hat.

Dazu weisen wir dem Kombinationsfeld als Datensatzherkunft zunächst eine Abfrage auf Basis der Tabelle **tblKategorien** zu, die wie folgt lautet:

```
SELECT KategorieID, Kategoriename FROM tblKategorien ORDER BY Kategoriename;
```

Aufruf des Formulars zum Bearbeiten der Kategorien

Damit der Benutzer dieses Formular aufrufen kann, fügen wir dem Formular **frmArtikeldetails** eine Schaltfläche neben dem Kombinationsfeld **cboKategorieID** hinzu (siehe Bild 4). Die Schaltfläche heißt **cmdKategorieBearbeiten** und soll die folgende Ereignisprozedur auslösen:

```
Private Sub cmdKategorieBearbeiten_Click()
    DoCmd.OpenForm _
        "frmKategorienZuweisen", _
        WindowMode:=acDialog, _
        OpenArgs:=Me!cboKategorieID
End Sub
```

Die Prozedur öffnet das Formular **frmKategorienZuweisen** als modalen Dialog und übergibt die ID der aktuell im Kombinationsfeld **cboKategorieID** ausgewählten Kategorie als Öffnungsargument an das geöffnete Formular.

Formular zum Zuordnen und Löschen von Kategorien

Das Formular, mit dem wir überflüssige Kategorien loswerden wollen, heißt **frmKategorienZuweisen** und sieht im Entwurf wie

Außerdem stellen wir die Eigenschaft **Spaltenanzahl** auf den Wert **2** und **Spaltenbreiten** auf **0cm** ein, damit nur die zweite Spalte mit der Bezeichnung der Kategorie im Kombinationsfeld angezeigt wird.

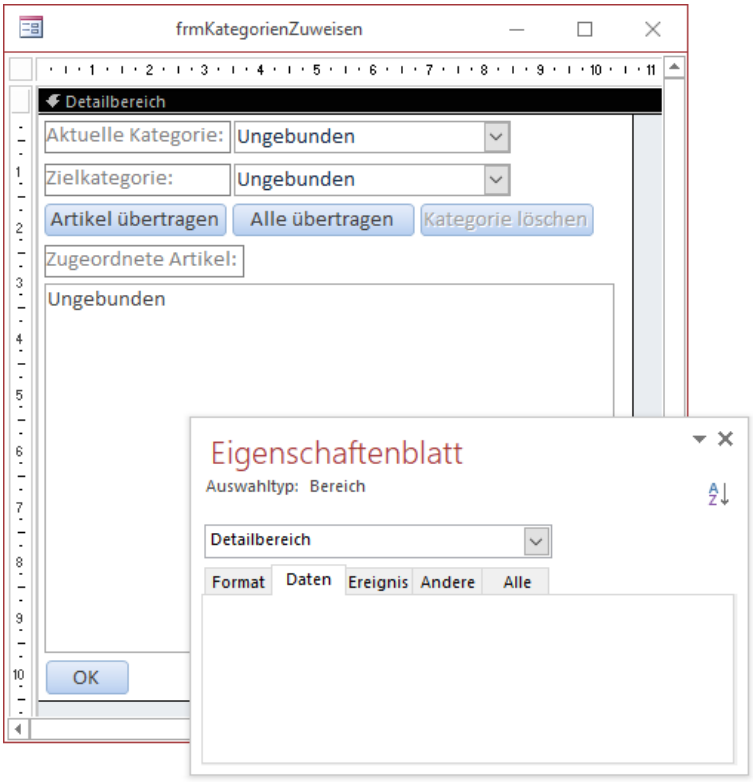

Bild 5: Formular zum Zuweisen und Löschen von Kategorien

Damit die Kategorie, deren **KategorieID** als Öffnungsargument des Formulars übergeben wurde, im Kombinationsfeld erscheint, bestücken wir die durch das Ereignis **Beim Laden** des Formulars ausgelöste Ereignisprozedur wie folgt:

```
Private Sub Form_Load()
    Me!cboAktuelleKategorieID = Me.OpenArgs
End Sub
```

Da das Formular nicht an eine Datenherkunft gebunden wird, stellen wir die Eigenschaften **Navigationsleiste**, **Datensatzmarkierer**, **Trennlinien** und **Bildlaufleisten** auf den Wert **Nein** und **Automatisch zentrieren** auf **Ja** ein. Wenn Sie das Formular nun für eine der Kategorien öffnen, sollte dies wie in Bild 6 aussehen.

Das zweite Kombinationsfeld statten wir fast mit der gleichen Datensatzherkunft wie das erste Kombinationsfeld aus – mit einem Unterschied: Die im ersten Kombinationsfeld angezeigte Kategorie soll dort nicht erhalten sein. Es macht ja auch keinen Sinn, die Artikel einer Kategorie der gleichen Kategorie zuzuweisen.

Da wir aber zur Entwurfszeit noch nicht wissen, welcher Datensatz im ersten Kombinationsfeld ausgewählt wird, stellen wir die Datensatzherkunft des zweiten Kombinationsfeldes auch erst beim Laden des Formulars ein. Dazu erweitern wir die Prozedur **Form_Load** wie in Listing 2.

Die erste Anweisung kennen Sie bereits, die zweite stellt eine SQL-Anweisung zusammen, die alle Datensätze der Tabelle **tblKategorien** außer der mit **Me.OpenArgs** gelieferten Kategorie enthält. Für die

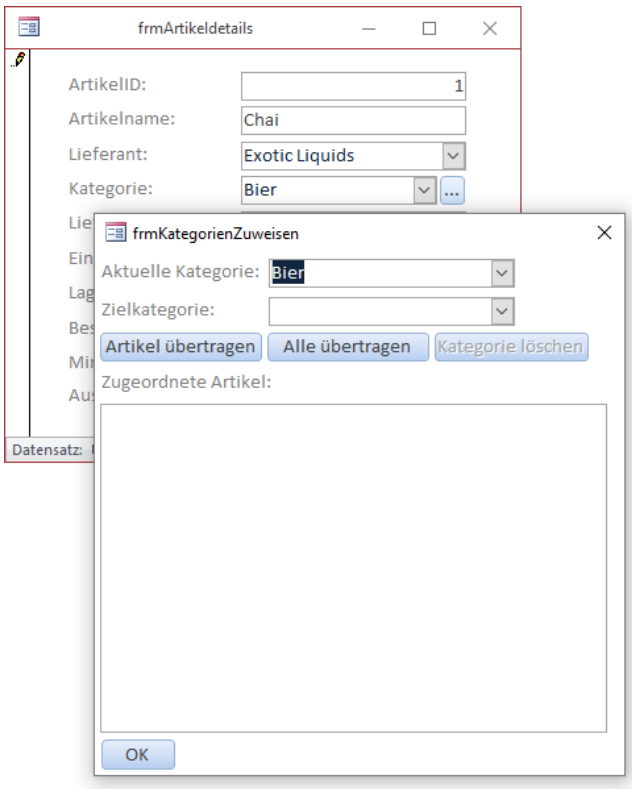

Bild 6: Formular mit der zu bearbeitenden Kategorie

Kategorie mit der ID 9 sieht die Abfrag beispielsweise so aus:

```
SELECT KategorieID, Kategoriename
FROM tblKategorien
WHERE KategorieID NOT IN (9)
ORDER BY Kategoriename
```

Die dritte Anweisung stellt das Kombinationsfeld **cboZielkategorieID** auf den ersten Eintrag der Datensatzherkunft ein. Auch für das Kombinationsfeld **cboZielkategorieID** stellen wie die Eigenschaften **Spaltenanzahl** auf **2** und **Spaltenbreiten** auf **0cm** ein.

```
Private Sub Form_Load()
    Me!cboAktuelleKategorieID = Me.OpenArgs
    Me!cboZielkategorieID.RowSource = "SELECT KategorieID, Kategoriename FROM tblKategorien WHERE KategorieID NOT IN (" _
        & Me.OpenArgs & ") ORDER BY Kategoriename"
    Me!cboZielkategorieID = Me.cboZielkategorieID.ItemData(0)
End Sub
```

Listing 2: Aktionen beim Laden des Formulars **frmKategorienZuweisen**

Füllen des Listenfeldes

Nun wollen wir noch das Listenfeld namens **lstZugeordneteArtikel** mit den Datensätzen der Tabelle **tblArtikel** bestücken, die aktuell mit der im ersten Kombinationsfeld **cboAktuelleKategorieID** angezeigten Kategorie verknüpft sind. Das erledigen wir initial ebenfalls in der Prozedur **Form_Load**. Das Listenfeld soll nur die Artikelnamen anzeigen, also stellen wir die Eigenschaft **Spaltenanzahl** auf **2** und die Eigenschaft **Spaltenbreiten** auf **0cm** ein. Der Prozedur **Form_Load** fügen wir am Ende die folgende Zeile hinzu:

```
Me!lstZugeordneteArtikel.RowSource = _
        "SELECT ArtikelID, Artikelname FROM tblArtikel _
              WHERE KategorieID = " & Me.OpenArgs & " _
                                  ORDER BY Artikelname"
```

Wenn wir das Formular nun über die Schaltfläche **cmdKategorieBearbeiten** des Formulars **frmArtikeldetails** öffnen, sieht dieses wie in Bild 7 aus.

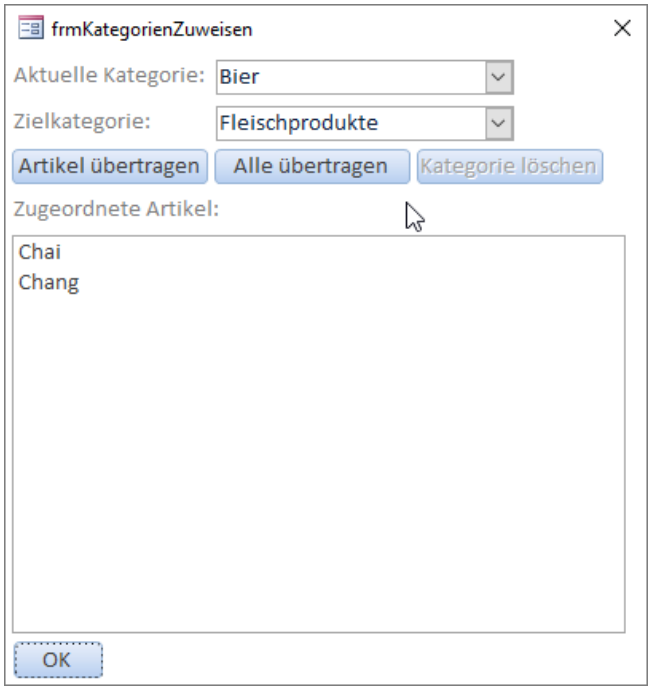

Bild 7: Alle notwendigen Daten sind nun verfügbar.

Daten aktualisieren

Nun könnte es sein, dass der Benutzer noch andere Kategorien als die aktuell im oberen Kombinationsfeld dargestellte entfernen möchte. Dann müssen wir auch das zweite Kombinationsfeld und das Listenfeld an die neue Auswahl anpassen. Also legen wir eine Prozedur an, die durch das Ereignis **Nach Aktualisierung** des Kombinationsfeldes **cboAktuelleKategorieID** ausgelöst wird. Nach dem Aufruf dieser Prozedur sollte das Formular dann etwa wie in Bild 8 aussehen.

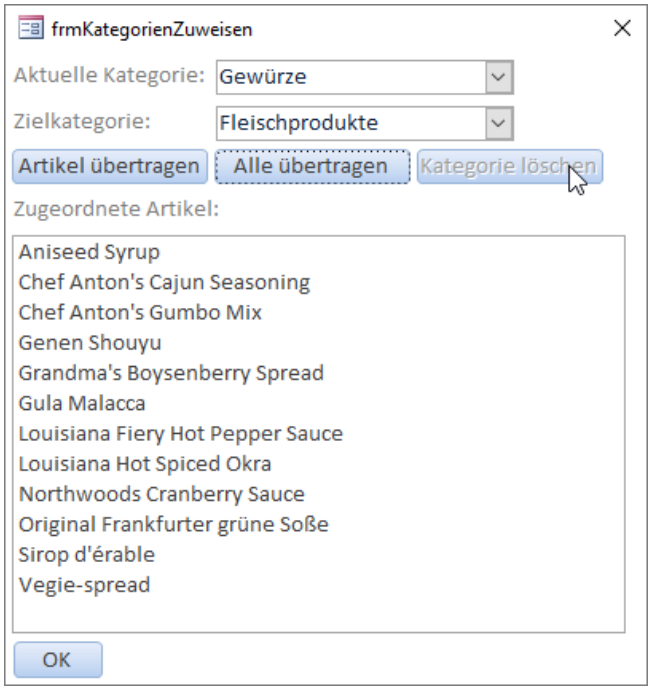

Bild 8: Formular nach dem Wechsel zu einer anderen Kategorie

Die Prozedur ermittelt zunächst den Primärschlüsselwert der im ersten Kombinationsfeld angezeigten Kategorie und speichert diese in der Variablen **lngKategorie**. Dann führt sie drei Anweisungen aus, die den hinteren drei Anweisungen der Ereignisprozedur **Form_Load** sehr ähneln.

Der einzige Unterschied ist, dass hier kein Bezug auf die mit dem Öffnungsargument übermittelte Kategorie mehr genommen wird, sondern auf die durch den Benutzer im Kombinationsfeld **cboAktuelleKategorieID** ausgewählte Kategorie. Dementsprechend zeigt das zweite Kombinationsfeld nun alle Datensätze der Tabelle **tblKategorien** an, die nicht im Kombinationsfeld **cboAktuelleKategorieID** ausgewählt sind und das Listenfeld liefert nun alle Artikel, die zu der aktuell

```
Private Sub cboAktuelleKategorieID_AfterUpdate()
    Dim lngAktuelleKategorieID As Long
    lngAktuelleKategorieID = Me!cboAktuelleKategorieID
    Me!cboZielkategorieID.RowSource = "SELECT KategorieID, Kategoriename FROM tblKategorien WHERE KategorieID NOT IN (" _
        & lngAktuelleKategorieID & ") ORDER BY Kategoriename"
    Me!cboZielkategorieID = Me!cboZielkategorieID.ItemData(0)
    Me!lstZugeordneteArtikel.RowSource = "SELECT ArtikelID, Artikelname FROM tblArtikel WHERE KategorieID = " _
        & lngAktuelleKategorieID & " ORDER BY Artikelname"
End Sub
```

Listing 3: Aktualisierung des ersten Kombinationsfeldes

```
Private Sub SteuerelementeAktualisieren(lngAktuelleKategorieID As Long)
    Me!cboZielkategorieID.RowSource = "SELECT KategorieID, Kategoriename FROM tblKategorien WHERE KategorieID NOT IN (" _
        & lngAktuelleKategorieID & ") ORDER BY Kategoriename"
    Me!cboZielkategorieID = Me!cboZielkategorieID.ItemData(0)
    Me!lstZugeordneteArtikel.RowSource = "SELECT ArtikelID, Artikelname FROM tblArtikel WHERE KategorieID = " _
        & lngAktuelleKategorieID & " ORDER BY Artikelname"
End Sub
```

Listing 4: Ausgelagerte Anweisungen zur Aktualisierung der Steuerelemente

im Kombinationsfeld **cboAktuelleKategorieID** angezeigten Kategorie gehören (siehe Listing 3).

Bevor wir weitergehen, prüfen wir, ob wir nicht schon dieser Stelle eine Vereinfachung des Codes vornehmen können. Das ist in der Tat der Fall: Die drei hinteren Zeilen der Prozeduren **Form_Load** und **cboAktuelleKategorie_AfterUpdate** unterscheiden sich nur durch einen Parameter.

Also vereinfachen wir diesen Code, indem wir die drei Zeilen in eine neue Prozedur packen, diese mit einem Parameter versehen und einen Aufruf dieser Prozedur in die ursprünglichen Routinen einbauen. Die Prozedur **SteuerelementeAktualisieren** sieht nun wie in Listing 4 aus. Damit können wir die Prozedur **Form_Load** wie folgt anpassen:

```
Private Sub Form_Load()
    Me!cboAktuelleKategorieID = Me.OpenArgs
    SteuerelementeAktualisieren Me.OpenArgs
End Sub
```

Und auch die Prozedur **cboAktuelleKategorieID_AfterUpdate** können wir erheblich schlanker machen:

```
Private Sub cboAktuelleKategorieID_AfterUpdate()
    Dim lngAktuelleKategorieID As Long
    lngAktuelleKategorieID = Me!cboAktuelleKategorieID
    SteuerelementeAktualisieren lngAktuelleKategorieID
End Sub
```

Auswahl der zu übertragenden Artikel

Die der Kategorie im Kombinationsfeld **cboAktuelleKategorieID** zugewiesenen Artikel sollen vielleicht nicht alle in die gleiche neue Kategorie übertragen werden, sondern in verschiedene Kategorien. Deshalb wollen wir dem Benutzer die Mehrfachauswahl im Listenfeld ermöglichen und stellen dazu die Eigenschaft **Mehrfachauswahl** auf **Erweitert** ein.

Dies sorgt dafür, dass er die Einträge im Listenfeld nun etwa wie in der Dateiliste im Windows Explorer selektieren kann – also bei gedrückter Umschalttaste Bereiche auswählen oder bei gedrückter **Strg**-Taste einzelne Einträge aus- oder abwählen (siehe Bild 9).

Damit haben wir alle Vorbereitungen getroffen, um endlich die Funktion für die Schaltflächen mit der Beschriftung **Artikel übertragen** und **Alle übertragen** zu definieren.

Artikel übertragen

Um die markierten Artikel per Mausklick auf die Schaltfläche **cmdArtikelUebertragen** von der Kategorie aus dem Kombinationsfeld **cboAktuelleKategorieID** zur Kategorie aus **cboZielkategorieID** zu übertragen, legen wir für das **Beim Klicken**-Ereignis die Prozedur aus Listing 5 an.

Diese ermittelt zunächst den Primärschlüsselwert der Zielkategorie aus dem Kombinationsfeld **cboZielkategorieID** und speichert diese in der Variablen **lngZielkategorieID**. Danach prüft sie, ob der Benutzer überhaupt einen Eintrag im Listenfeld ausgewählt hat.

Falls nicht, erscheint eine entsprechende Meldung und die Prozedur wird beendet, anderenfalls geht es wie folgt weiter:

Die Prozedur durchläuft in einer **For Each**-Schleife über alle Einträge der Auflistung **ItemsSelected** des Listenfeldes **lstZugeordneteArtikel** die markierten Einträge. Die Auflistung **ItemsSelected** liefert dabei

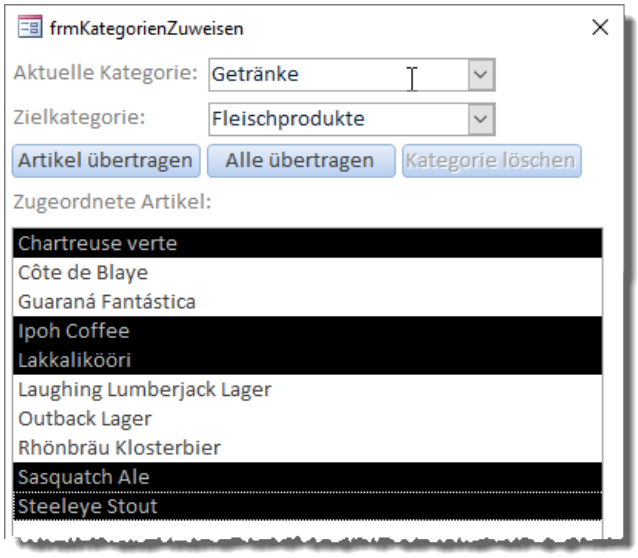

Bild 9: Markierung der zu übertragenen Artikel

die 0-basierten Indexwerte der markierten Einträge. Über die **ItemData**-Eigenschaft können wir dazu den Wert der gebundenen Spalte, in diesem Fall **ArtikelID**, des Listenfeldeintrags ermitteln. Diesen speichern wir in der Variablen **lngArtikelID**.

```
Private Sub cmdArtikelUebertragen_Click()
    Dim db As DAO.Database
    Dim var As Variant
    Dim lngArtikelID As Long
    Dim lngZielkategorieID As Long
    Set db = CurrentDb
    lngZielkategorieID = Me!cboZielkategorieID
    If Not Me!lstZugeordneteArtikel.ItemsSelected.Count = 0 Then
        For Each var In Me!lstZugeordneteArtikel.ItemsSelected
            lngArtikelID = Me.lstZugeordneteArtikel.ItemData(var)
            db.Execute "UPDATE tblArtikel SET KategorieID = " & lngZielkategorieID & " WHERE ArtikelID = " & lngArtikelID, _
                dbFailOnError
        Next var
        Me!lstZugeordneteArtikel.Requery
        If Me!lstZugeordneteArtikel.ListCount = 0 Then
            Me!cmdKategorieLoeschen.Enabled = True
        End If
    Else
        MsgBox "Sie müssen mindestens einen Artikel auswählen."
    End If
End Sub
```

Listing 5: Artikel von einer Kategorie zur anderen übertragen

```
Private Sub cmdAlleUebertragen_Click()
    Dim db As DAO.Database
    Dim lngAktuelleKategorieID As Long
    Dim lngZielkategorieID As Long
    Set db = CurrentDb
    lngAktuelleKategorieID = Me!cboAktuelleKategorieID
    lngZielkategorieID = Me!cboZielkategorieID
    db.Execute "UPDATE tblArtikel SET KategorieID = " & lngZielkategorieID & " WHERE KategorieID = " & lngAktuelleKategorieID,
dbFailOnError
    Me!lstZugeordneteArtikel.Requery
End Sub
```

Listing 6: Alle Artikel einer Kategorie zu einer anderen Kategorie übertragen

Die Werte der beiden Variablen **lngZielkategorieID** und **lngArtikelID** nutzen wir in der folgenden Zeile, um eine **UPDATE**-Aktionsabfrage zusammenzustellen, welche den Datensatz der Tabelle **tblArtikel** mit dem Primärschlüsselwert aus der Variablen **lngArtikelID** bearbeitet und dabei den Wert des Feldes **KategorieID** auf den Wert aus der Variablen **lngZielkategorieID** einstellt. Die abschließende Anweisung aktualisiert noch den Inhalt des Listenfeldes, damit die von dieser Kategorie entfernten Artikel dort nicht mehr aufgelistet werden.

Alle Artikel übertragen

Die Schaltfläche **cmdAlleUebertragen** soll alle in der aktuell mit **cboAktuelleKategorieID** enthaltenen Artikel zu der in **cboZielkategorieID** angegebenen Kategorie hinzufügen. Dazu legen wir die Prozedur aus Listing 6 an. Die Prozedur ermittelt den Primärschlüsselwert der aktuellen Kategorie aus dem Kombinationsfeld **cboAktuelleKategorieID** und die der Zielkategorie aus **cboZielKategorieID** und speichert diese in zwei dafür vorbereiteten Variablen. Die folgende SQL-Anweisung aktualisiert alle Artikel, deren Feld **KategorieID** mit dem Wert aus **lngAktuelleKategorieID** übereinstimmt und legt für dieses Feld den Wert aus **lngZielkategorieID** fest. Anschließend aktualisiert sie noch den Inhalt des Listenfeldes **lstZugeordneteArtikel**.

Kategorie löschen

Die Schaltfläche **cmdKategorieLoeschen** haben wir gleich im Entwurf deaktiviert, indem wir die Eigenschaft **Aktiviert** auf **Nein** eingestellt haben. Der Hintergrund ist, dass diese Schaltfläche nur aktiviert werden soll, wenn die im Kombinationsfeld **cboAktuelleKategorieID** angezeigte Kategorie keinen zugeordneten Artikel mehr in der Tabelle **tblArtikel** enthält.

Dies wollen wir nun noch programmieren. Dazu erweitern wir zunächst die Prozedur **cmdAlleUebertragen_Click** um die folgende Anweisung, die wir am Ende einsetzen:

```
Me!cmdKategorieLoeschen.Enabled = True
```

Nach dem Ausführen dieser Prozedur wissen wir auf jeden Fall, dass es keine verknüpften Datensätze für diese Kategorie mehr in der Tabelle **tblArtikel** gibt, sodass wir die Schaltfläche aktivieren können.

Was aber, wenn der Benutzer die letzten Artikel einer Kategorie zu einer anderen Kategorie überträgt, dazu aber nicht die Schaltfläche **cmdAlleUebertragen** verwendet, sondern die restlichen Einträge im Listenfeld markiert und dann auf **cmdArtikelUebertragen** klickt?

In diesem Fall müssen wir am Ende der Prozedur **cmdArtikelUebertragen_Click** prüfen, ob der aktuellen Kategorie noch Artikel zugeordnet sind und gegebenenfalls die Schaltfläche **cmdKategorieLoeschen** aktivieren. Das erledigen wir durch Anhängen der folgenden Zeilen an die Prozedur **cmdArtikelUebertragen_Click**:

```
If Me!lstZugeordneteArtikel.ListCount = 0 Then
    Me!cmdKategorieLoeschen.Enabled = True
End If
```

Damit sieht das Formular dann nach dem vollständigen Leeren des Listenfeldes wie in Bild 10 aus.

Bild 10: Aktivierte Schaltfläche zum Löschen einer Kategorie

Kategorie löschen

Das Löschen einer Kategorie erfolgt über die Schaltfläche mit der Beschriftung **Kategorie löschen**. Sie ermittelt die ID der zu löschenden Kategorie aus dem Kombinationsfeld **cboAktuelleKategorieID** und trägt diese in die Variable **lngAktuelleKategorieID** ein.

Dann führt sie eine **DELETE**-Aktionsabfrage aus, welche den Datensatz aus der Tabelle **tblKategorien** löscht, deren Feld **KategorieID** den Wert aus **lngAktuelleKategorieID** aufweist.

Anschließend aktualisiert sie den Inhalt von **cboAktuelleKategorieID**, denn der vorherige Eintrag wurde ja soeben gelöscht, und wählt den ersten Eintrag der Liste aus. Außerdem ruft sie die Prozedur **SteuerelementeAktualisieren** auf, die dann wieder das zweite Kombinationsfeld sowie das Listenfeld aktualisiert.

Schaltfläche zum Löschen von Kategorien wieder deaktivieren

In diesem Fall soll auch die Schaltfläche **cmdKategorieLoeschen** geprüft werden. Enthält die aktuelle Kategorie noch zugeordnete Artikel, soll die Schaltfläche **cmdKategorieLoeschen** wieder deaktiviert werden. Das können wir gleich in die Prozedur **SteuerelementeAktivieren** aufnehmen. Dieser fügen wir die folgenden Anweisungen hinzu:

```
If Me!lstZugeordneteArtikel.ListCount = 0 Then
    Me!cmdKategorieLoeschen.Enabled = True
Else
    Me!cmdKategorieLoeschen.Enabled = False
End If
```

Das können wir allerdings auch als Einzeiler formulieren:

```
Me!cmdKategorieLoeschen.Enabled = _
                Me!lstZugeordneteArtikel.ListCount = 0
```

Kategorie in aufrufendem Formular anpassen

Wenn wir das Formular **frmKategorienZuweisen** vom Formular **frmArtikeldetails** aus aufgerufen und die Kategorie des derzeit im aufrufenden Formular angezeigten Artikels angepasst haben, zeigt da Kombinationsfeld **cboKategorieID** nach dem Schließen des Formulars **frmKategorienZuweisen** immer noch den

```
Private Sub cmdKategorieLoeschen_Click()
    Dim db As DAO.Database
    Dim lngAktuelleKategorieID As Long
    Set db = CurrentDb
    lngAktuelleKategorieID = Me!cboAktuelleKategorieID
    db.Execute "DELETE FROM tblKategorien WHERE KategorieID = " & lngAktuelleKategorieID, dbFailOnError
    Me!cboAktuelleKategorieID.Requery
    Me!cboAktuelleKategorieID = Me!cboAktuelleKategorieID.ItemData(0)
    SteuerelementeAktualisieren Me!cboAktuelleKategorieID
End Sub
```

Listing 7: Löschen einer Kategorie

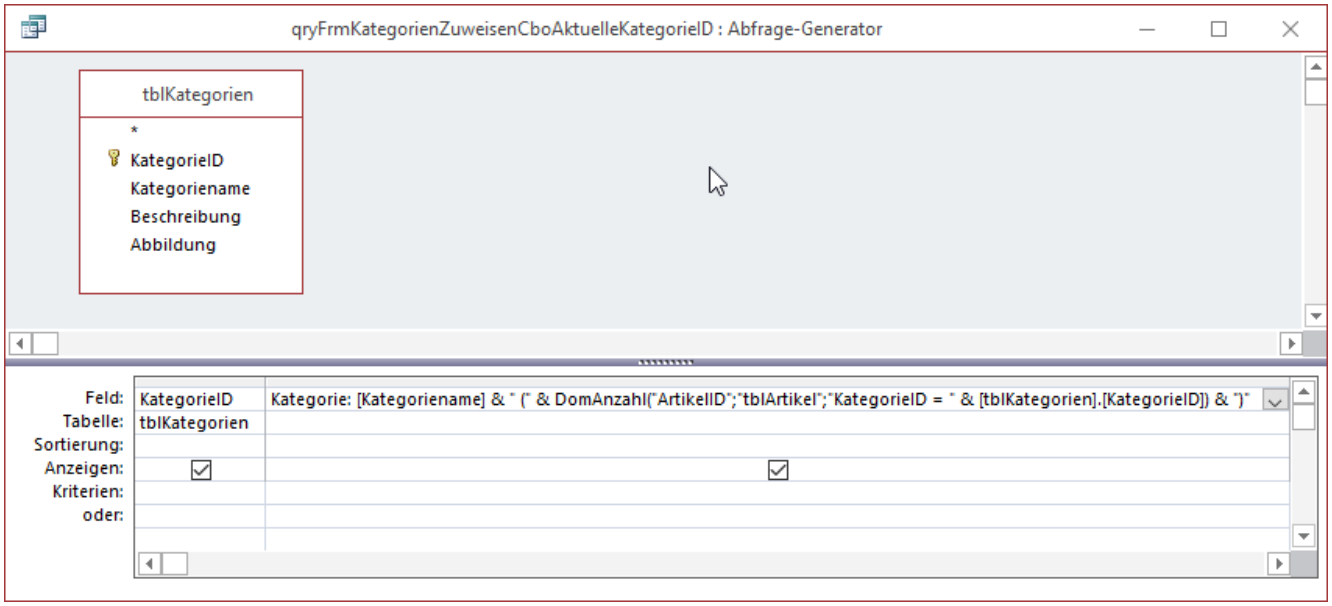

Bild 11: Abfrage, welche die Anzahl der Artikel je Kategorie mitliefert

zuvor ausgewählten Eintrag an. Dies wollen wir noch ändern, indem wir die aufrufende Prozedur wie folgt ergänzen:

```
Private Sub cmdKategorieBearbeiten_Click()
    ...
    Me!cboKategorieID.Requery
End Sub
```

Finetuning

Nachdem die Grundfunktionen der Lösung stehen, wollen wir noch an den Feinheiten arbeiten. Wenn

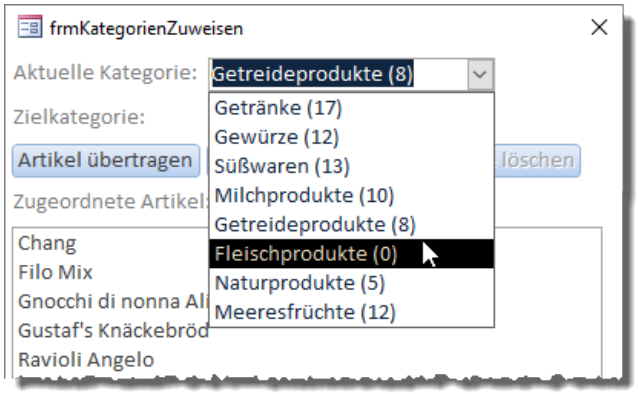

Bild 12: Anzeige der Kategorien mit der Anzahl der enthaltenen Artikel-Datensätze

der Benutzer sich die Kategorien allgemein ansehen möchte, um zu prüfen, welche vielleicht keine Einträge enthalten und diese zu löschen, muss er alle Kategorien durchsehen.

Es wäre wesentlich schöner, wenn der Benutzer im Kombinationsfeld **cboAktuelleKategorieID** in Klammern hinter der Bezeichnung der Kategorie noch die Anzahl der enthaltenen Artikel-Datensätze je Kategorie anzeigen würde. Dazu hinterlegen wir für die Eigenschaft **Datensatzherkunft** des Kombinationsfeldes **cboAktuelleKategorieID** die Abfrage aus Bild 11.

Hier legen wir statt **Kategoriename** ein neues berechnetes Feld namens **Kategorie** an, das per **DCount**-Funktion die Anzahl der Artikel der aktuellen Kategorie ermittelt. Das Ergebnis in der Formularansicht des Formulars **frmKategorienZuweisen** sieht schließlich wie in Bild 12 aus.

Zusammenfassung und Ausblick

Damit haben wir eine komfortable Möglichkeit geschaffen, die Lookup-Daten des Feldes **KategorieID** zu verwalten und versehentlich angelegte oder nicht mehr benötigte Kategorien zu löschen.