

ACCESS

IM UNTERNEHMEN

UNION-ABFRAGEN

Stellen Sie die Daten aus verschiedenen Tabellen in einer Abfrage zusammen (ab S. 21)



In diesem Heft:

RÜCKGÄNGIG IN MEMOFELDERN

Stellen Sie den Zustand vor einer Änderung in Memofeldern wieder her.

SEITE 29

KOMBINATIONSFELDER ERWEITERN

Fügen Sie Einträge zur Aufnahme neuer Daten zu Kombinationsfeldern hinzu.

SEITE 35

DATENMAKROS VERWALTEN

Verwalten Sie Datenmakros mit unserer Lösung komfortabler als mit Bordmitteln.

SEITE 2

Tabellen-Union

Access und andere Datenbanken auf SQL-Basis bieten viele Möglichkeiten, die Daten verschiedener Tabellen zusammenzuführen und diese entsprechend ihrer Verknüpfungen und verschiedener Kriterien und Sortierungen auszugeben. Eine besondere Art, die Daten von zwei oder mehr Tabellen zusammenzuführen, ist die UNION-Abfrage. Damit kombinieren wir nicht die Daten verschiedener Tabellen, sondern wir bilden praktisch die Vereinigungsmenge dieser Tabellen.



Um dieses Thema geht es im Beitrag **UNION-Abfragen** ab Seite 21. Wir stellen Ihnen die grundlegenden Möglichkeiten vor und zeigen, welche Tricks und Kniffe Sie benötigen, um das gewünschte Ergebnis zu erhalten.

Im Beitrag **Kombinationsfeld mit Extraeinträgen** kommen wir ab Seite 35 dann auch direkt auf diese Technik zurück. Dort nutzen wir **UNION**-Abfragen, um zu den Datensätzen von Kombinationsfeldern aus der eigentlichen Datenherkunft noch weitere Einträge hinzuzufügen. Der erste dieser Einträge dient dazu, gleich beim Anlegen eines neuen Datensatzes den Text **<Auswählen>** anzuzeigen. So weiß der Benutzer direkt, dass hier etwas zu tun ist! Der zweite Wert, den wir per **UNION**-Abfrage hinzufügen, heißt **<Neuer Eintrag>**. Wenn der Benutzer diesen auswählt, soll beispielsweise ein Dialog zum Anlegen eines neuen Kunden erscheinen, wenn das Kombinationsfeld die Auswahl von Kunden ermöglicht. Auf diese Weise sparen wir uns eine gesonderte Schaltfläche zum Anlegen eines neuen Datensatzes in der Tabelle, die den Inhalt des Kombinationsfeldes liefert.

In der vorherigen Ausgabe haben wir uns bereits um das Thema Datenmakros gekümmert. In der aktuellen Ausgabe gehen wir noch einen Schritt weiter und bieten eine einfach zu bedienende Möglichkeit, um sich einen Überblick über die aktuell für die verschiedenen Tabellen der Datenbank definierten Datenmakros zu verschaffen und diese zu bearbeiten. Das ist mit Bordmitteln nicht möglich – hier müssen Sie jede Tabelle einzeln öffnen und die enthaltenen Datenmakros über spezielle Menüeinträge öffnen. Wie Sie das einfacher erreichen, zeigen wir im Beitrag **Datenmakros verwalten** ab Seite 2.

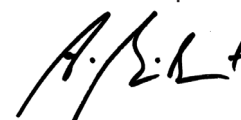
Aus dem Nähkästchen stammt die Idee zum Beitrag **Von Version zu Version** (ab Seite 50). Ich musste eine Webanwendung von einer alten auf eine neue Version migrieren, was kompliziert war, weil sich das Datenmodell geändert hatte. Mit speziell programmierten Prozeduren habe ich die Datenmodelle der beiden Versionen hinsichtlich der Gemeinsamkeiten abgeglichen und so die Grundlage für das Zusammenstellen von SQL-Anweisungen zum Durchführen der Migration geschaffen – allerdings nicht ohne ein wenig abschließender Handarbeit.

Dabei hat die Lösung aus dem Beitrag **Kopier- und Löschreihenfolge in MySQL** (ab Seite 43) mir weitere Arbeit abgenommen, denn die dort beschriebenen Routinen haben die zu migrierenden Tabellen in der richtigen Reihenfolge zum Löschen und Kopieren der Daten ermittelt.

Der Beitrag **Rückgängig in Memofeldern** zeigt ab Seite 29, wie Sie die Daten in einem Memofeld nach unbeabsichtigten Änderungen wiederherstellen können und so Datenverlust durch Fehleingaben vorbeugen können.

Und auch bei unserer Ticketverwaltung geht es weiter: Bevor es in der kommenden Ausgabe auf die Zielgerade geht, haben wir unter **Ticketverwaltung, Teil V** ab Seite 61 noch einige Feinheiten optimiert.

Und nun: Viel Spaß beim Lesen!



Ihr André Minhorst

Datenmakros verwalten

Datenmakros dienen dazu, Aktionen automatisch beim Ändern der Daten einer Tabelle auszulösen – nämlich beim Anlegen, Bearbeiten oder Löschen eines Datensatzes. Diese legt man in der Regel über die Benutzeroberfläche an. Leider kann man damit immer nur sehen, welche Datenmakros für die aktuelle im Entwurf oder in der Datenblattansicht angezeigte Tabelle zur Verfügung stehen. Dieser Beitrag zeigt, wie Sie sich einen besseren Überblick über die vorhandenen Datenmakros verschaffen, diese anzeigen und bearbeiten und sogar neue Datenmakros anlegen können – und das auch noch parametrisiert für gleichartige Datenmakros und mehrere Tabellen gleichzeitig.

Genau genommen sind wir durch die Lösung des Beitrags **Zugriffsrechte mit Datenmakros** (www.access-im-unternehmen.de/1193) auf die Idee zu diesem Beitrag gekommen. Dort haben wir die Tabellen der Beispieldatenbank mit einigen Datenmakros ausgestattet, die dafür sorgen, dass die Daten der Tabelle nur durch Benutzer mit entsprechenden Zugriffsrechten geändert werden dürfen.

Es wäre etwas mühselig, diese Datenmakros bei großen Datenbankanwendungen für jede Tabelle einzelnen hinzuzufügen. Daher haben wir dort schon gezeigt, wie Sie per VBA-Funktion ein Datenmakro zu der im Parameter der Funktion genannten Tabelle hinzufügen können. Aber wir wollen etwas grundlegender starten.

Datenmakros erkunden

Die erste Frage ist: Wie können wir überhaupt auf die bereits in den Tabellen der Anwendung vorhandenen Datenmakros zugreifen? Gibt es eine

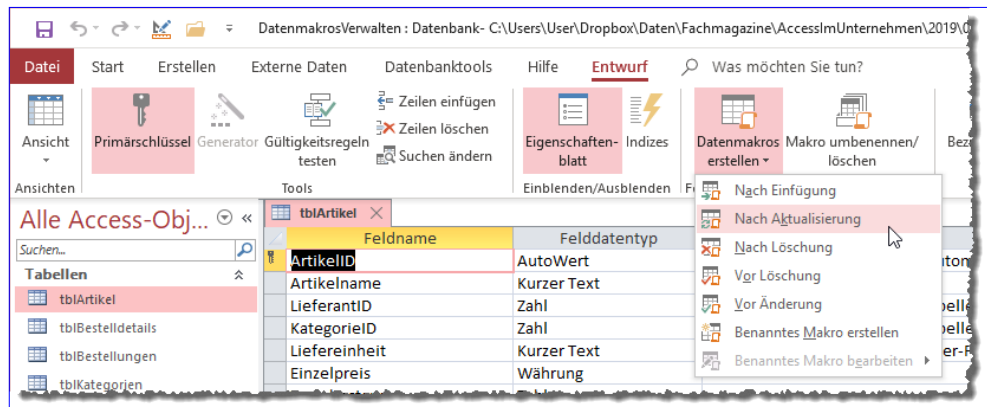


Bild 1: Datenmakros über die Entwurfsansicht verwalten

Art VBA-Auflistung? Oder auf welche Technik können wir zugreifen, um die vorhandenen Datenmakros zu analysieren?

Datenmakros per Benutzeroberfläche

Schauen wir uns erst einmal an, wie wir über die Benutzeroberfläche auf die Datenmakros zugreifen. Der erste Weg ist über die Entwurfsansicht einer Tabelle. Dann finden Sie im Ribbon-Tab **Entwurf** unter **Feld-, Datensatz- und Tabellenereignisse** den Befehl **Datenmakros erstellen**. Wenn man diese Schaltfläche anklickt, öffnet sich eine Liste aller möglichen Datenmakros. In Bild 1 sind beispielsweise noch keine Datenmakros angelegt.

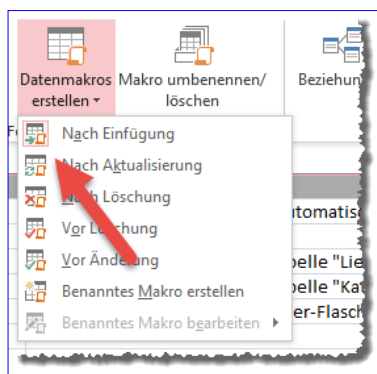


Bild 2: Kennzeichnung eines vorhandenen Datenmakros

Wenn wir hier das Makro **Nach Einfügung** anlegen, wird das Icon mit einem roten Rand versehen (siehe Bild 2).

Datenmakros per Datenblatt

In der Datenblattsicht ist es noch übersichtlicher (siehe Bild 3). Hier finden Sie im Ribbon-Tab **Tabelle** einige Schaltflächen, mit denen Sie Datenmakros anlegen oder bearbeiten können. Wenn das Datenmakro bereits vorhanden ist, hinterlegt Access die Schaltfläche mit einem Hintergrund wie hier beim Datenmakro **Nach Einfügung**.

Wenn Sie auf eine der Schaltflächen klicken, öffnet Access die Entwurfsansicht für das Erstellen und Bearbeiten von Datenmakros. Diese steht aber in diesem Beitrag nicht im Mittelpunkt – daher schauen wir uns diese nun nicht im Detail an.

Per VBA auf die Datenmakros zugreifen

Nun möchten wir herausfinden, wie wir per VBA auf die Datenmakros einer Tabelle beziehungsweise der kompletten Datenbank zugreifen können. Welches Werkzeug im VBA-Editor hilft uns weiter, etwas über die Befehle in Zusammenhang mit Makros herauszufinden? Der Objektkatalog.

Diesen öffnen wir und suchen nach dem Schlüsselwort **Macro**. Hier finden wir zunächst einige **acCmd...**-Konstanten, die letztlich zum Ausführen der Ribbonbefehle

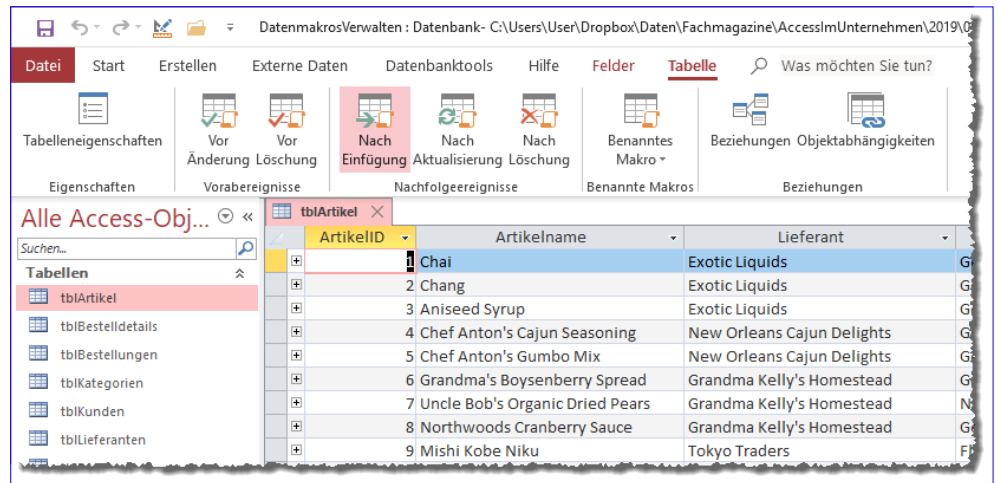


Bild 3: Datenmakros über die Datenblattsicht verwalten

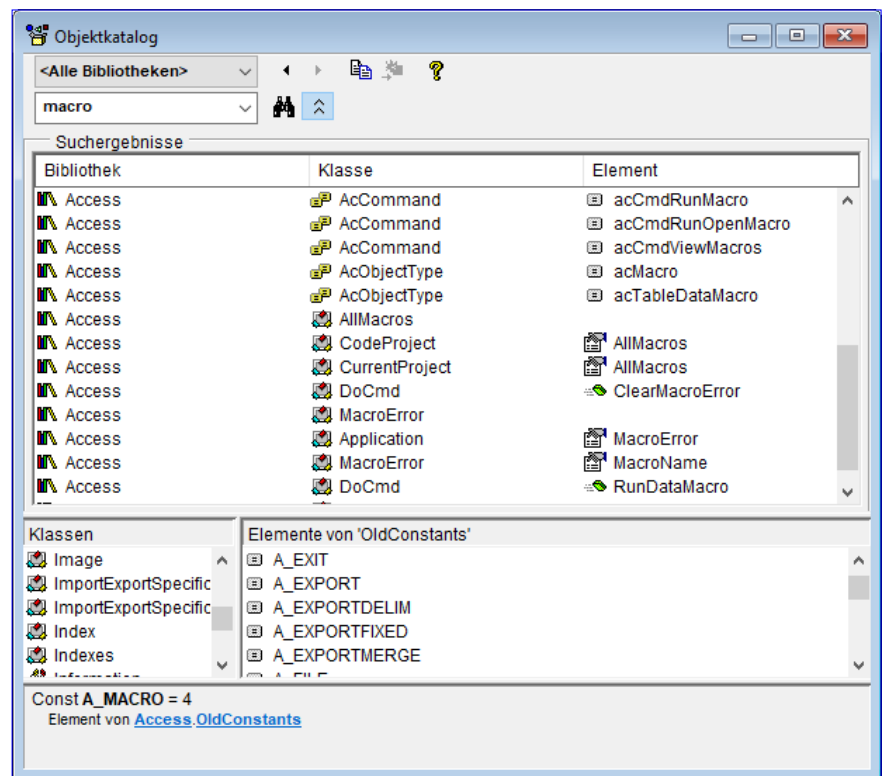


Bild 4: Befehle, welche die Zeichenkette **Macro** enthalten

per VBA dienen. Außerdem entdecken wir allerdings auch ein paar weitere Elemente, die interessant sein könnten: besonders die Auflistung **AllMacros** (siehe Bild 4).

Wir schauen uns einmal in einer kleinen Prozedur an, welches Ergebnis diese Auflistung liefert:

```
Public Sub AlleMakros()
    Dim mac As Object
    For Each mac In CurrentProject.AllMacros
        Debug.Print TypeName(mac), mac.Name
    Next mac
End Sub
```

Dies liefert allerdings kein Ergebnis, wenn Sie nur Datenmakros angelegt haben. Die Auflistung dient allein dem Zugriff auf die herkömmlichen Makros, die auch im Navigationsbereich unter **Makros** angezeigt werden.

Es gibt also in der Tat keine Möglichkeit, per VBA direkt auf die Datenmakros zuzugreifen. Also gehen wir einen kleinen Umweg.

Umweg über SaveAsText

Wir wissen, dass Datenmakros in Zusammenhang mit Tabellen gespeichert werden. Weiterhin wissen wir, dass es eine Möglichkeit gibt, Access-Objekte über die Anweisung **SaveAsText** auf der Festplatte zu speichern und etwa über einen Texteditor auf diese Dateien zuzugreifen. Also schauen wir uns diesen Befehl einmal genauer an. Dies gelingt etwa über die IntelliSense-Funktion im Direktbereich des VBA-Editors. Dort finden wir dann ganz unten in der Liste eine interessante Konstante, nämlich **acTableDataMacro** (siehe Bild 5).

Auch für diese Konstante geben wir als zweiten Parameter den Namen des zu speichernden Objekts an sowie als dritten Parameter den Dateinamen, unter dem das Objekt gespeichert werden soll. Nach kurzem Experimentieren

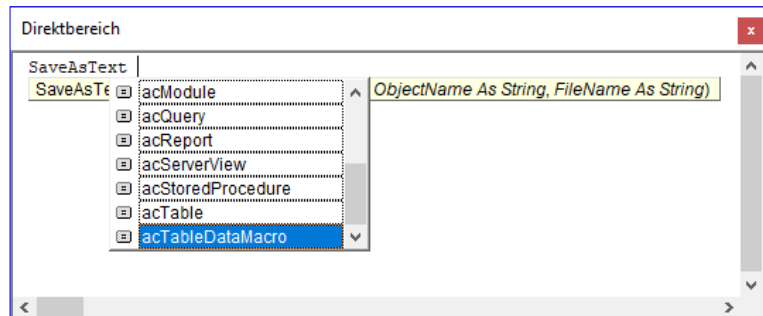


Bild 5: Die neue Konstante **acTableDataMacro**

wird klar, dass wir für den zweiten Parameter den Tabellennamen angeben, dessen Datenmakros wir im Dateisystem speichern wollen.

Also legen wir für die Tabelle **tblArtikel** einmal je ein Makro für die verschiedenen Typen an (das Ribbon sieht dann für diese Makros wie in Bild 6 aus).

Zusätzlich fügen wir auch ein benanntes Makro zu der Tabelle hinzu, für das wir allerdings im Gegensatz zu den vorgegebenen Makros einen Namen angeben müssen. Das ist logisch, denn es kann mehr als ein benanntes Makro je Tabelle geben.

Danach rufen wir den folgenden Befehl auf, um den Code der Datenmakros auf der Festplatte zu speichern:

```
SaveAsText acTableDataMacro, "tblArtikel", CurrentProject.Path & "\tblArtike1_Makros.txt"
```

Das Ergebnis sieht dann in der neu angelegten Datei **tblArtikel_Makros.txt** wie in Listing 1 aus. Hier sehen wir, dass es sich um eine XML-Datei handelt. Diese

verwendet als Hauptobjekt das Element **DataMacros**. Darunter finden Sie einige Elemente namens **DataMacro**, die mit einem Attribut namens **Event** ausgestattet sind. Dieses gibt an, durch welches Ereignis das Datenmakro ausgelöst wird:

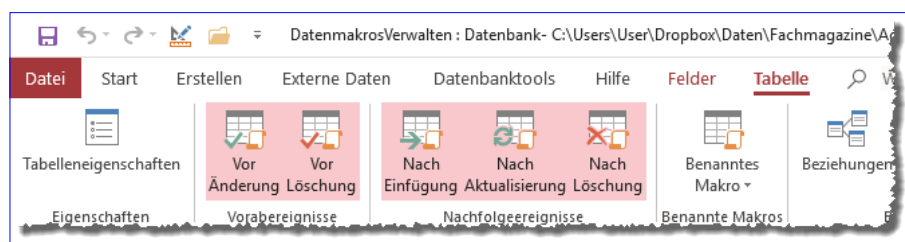


Bild 6: Die Einträge für die Datenmakros im Ribbon

- **AfterInsert:** Nach Einfügung
- **AfterUpdate:** Nach Aktualisierung
- **AfterDelete:** Nach Löschung
- **BeforeChange:** Vor Änderung
- **BeforeDelete:** Vor Löschung

Das ist aber nicht das einzige mögliche Attribut. Genau genommen wird dieses nur für die Datenmakros eingesetzt, die durch ein Ereignis ausgelöst werden. Für die anderen Makros von Tabellen, also die benannten Makros, gibt es das Attribut **Name**.

Unterhalb der **DataMacro**-Elemente finden wir ein **Statements**-Element, das dann die eigentlichen Befehle enthält. In unserem Beispiel haben wir für alle Makros einfach nur den Makrobefehl **Kommentar** eingefügt.

Hier finden dann auch die übrigen Befehle und Strukturen Platz, wobei bei den Strukturen noch untergeordnete Elemente integriert werden. Gleiches gilt für die Datenmakros.

Datenmakro analysieren

Wie aber nun können wir ermitteln, für welche Tabelle welche Datenmakros existieren und diese etwa im

```
<?xml version="1.0" encoding="UTF-16" standalone="no"?>
<DataMacros xmlns="http://schemas.microsoft.com/office/accessservices/2009/11/application">
  <DataMacro Event="AfterInsert">
    <Statements>
      <Comment>Kommentar Nach Einfügung</Comment>
    </Statements>
  </DataMacro>
  <DataMacro Event="AfterUpdate">
    <Statements>
      <Comment>Kommentar Nach Aktualisierung</Comment>
    </Statements>
  </DataMacro>
  <DataMacro Event="AfterDelete">
    <Statements>
      <Comment>Kommentar Nach Löschung</Comment>
    </Statements>
  </DataMacro>
  <DataMacro Event="BeforeChange">
    <Statements>
      <Comment>Kommentar Vor Änderung</Comment>
    </Statements>
  </DataMacro>
  <DataMacro Event="BeforeDelete">
    <Statements>
      <Comment>Kommentar Vor Löschung</Comment>
    </Statements>
  </DataMacro>
  <DataMacro Name="dmkBeispiel">
    <Statements>
      <Comment>Kommentar Benanntes Makro</Comment>
    </Statements>
  </DataMacro>
</DataMacros>
```

Listing 1: Die in eine Textdatei exportieren Datenmakros

Direktfenster ausgeben? Wir müssen tatsächlich über den Umweg des Dateixports auf die Festplatte gehen. Dort würden wir die so gespeicherte Datei dann mit der **Open**-Anweisung öffnen und mit den Methoden der Bibliothek **Microsoft XML, v3.0** analysieren.

Diese Bibliothek fügen Sie zunächst über den **Verweise**-Dialog hinzu (siehe Bild 7). Achtung: Mit der Version **v6.0** funktioniert der nachfolgend angegebene VBA-Code nicht.

Alle Tabellen durchlaufen

Danach bauen wir zunächst ein Grundgerüst, mit dem wir alle Tabellen einer Datenbank durchlaufen. Dieses sieht wie folgt aus:

```
Public Sub TabellenAnalysieren()
    Dim db As dao.Database
    Dim tbl As dao.TableDef
    Dim objXML As MSXML2.DOMDocument60
    Set db = CurrentDb
    For Each tbl In db.TableDefs
        If Not (Left(tbl.Name, 4) = "MSys") Then
            If EnthaelMakros(tbl.Name, objXML) = True Then
                Debug.Print "Makros vorhanden in '" & tbl.Name & "'."
            Else
                Debug.Print "Keine Makros in '" & tbl.Name & "'."
            End If
        End If
    Next tbl
End Sub
```

Damit durchlaufen wir alle Tabellen der Anwendung, deren Name nicht mit **MSys...** beginnt – also alle Tabellen außer den Systemtabellen. Innerhalb der **If...Then**-Bedingung können wir dann den Aufruf einer weiteren Prozedur unterbringen, welche die Makros in eine Datei exportiert und in ein XML-Dokument einliest – in diesem Fall eine Funktion namens **EnthaelMakros**.

Diese soll den Wert **True** liefern, wenn eine Tabelle Datenmakros enthält und **False**, wenn dies nicht der Fall ist.

Makros nach XML

Die Funktion **EnthaelMakros** erwartet den Namen der zu untersuchenden Tabelle als Parameter sowie eine Variable des Typs **MSXML2.DOMDocument**. Aus diesen bildet sie zunächst einen Dateinamen, der aus dem Pfad zur aktuellen Datenbank, einem Backslash, dem Namen der Tabelle und der Dateiergung **.xml** besteht:

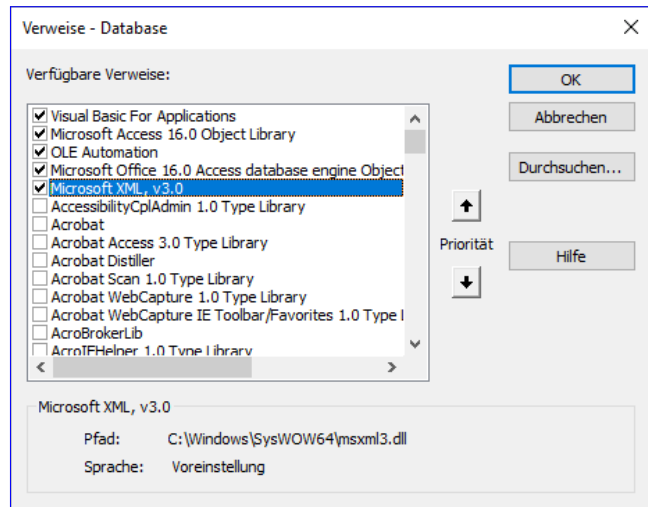


Bild 7: Hinzufügen eines Verweises auf die XML-Bibliothek

```
Public Function EnthaelMakros(strTabelle As String, objXML As MSXML2.DOMDocument60) As Boolean
    Dim strDateiname As String
    strDateiname = CurrentProject.Path & "\" & strTabelle & ".xml"
```

Dann versuchen wir, bei deaktivierter Fehlerbehandlung die **SaveAsText**-Anweisung mit dem Namen der Tabelle für die zu exportierenden Datenmakros und dem Dateinamen als Parameter aufzurufen. Wenn die Tabelle keine Datenmakros enthält, wird der Fehler mit der Nummer **2950** ausgelöst (**Reservierter Fehler**). In diesem Fall soll die Funktion enden, ohne den Funktionswert auf **True** einzustellen:

```
On Error Resume Next
SaveAsText acTableDataMacro, strTabelle, strDateiname
If Err.Number = 2950 Then
    Exit Function
End If
On Error GoTo 0
```

Konnten die Datenmakros hingegen erfolgreich exportiert werden, erstellt die Funktion ein neues Objekt des Typs **DOMDocument60**, das wir im nächsten Schritt mit dem XML-Dokument aus der soeben mit **SaveAsText** ange-

legten Datei füllen und danach den Inhalt des XML-Dokuments testweise im Direktbereich ausgeben:

```
Set objXML = New MSXML2.DOMDocument60
objXML.Load strDateiname
Debug.Print objXML.XML
EnthaelMakros = True
End Function
```

In diesem Fall gibt die Funktion den Wert **True** zurück – und auch der Parameter **objXML** liefert das gewünschte XML-Dokument.

XML-Dokument analysieren

In der aufrufenden Prozedur **TabellenAnalysieren** ersetzen wir nun die **Debug.Print**-Anweisung im **If**-Teil der **If...Then**-Bedingung durch den Aufruf der Prozedur **XMLAnalysieren**. Diesem stellen wir die Ausgabe des Tabellennamens voran. Dann übergeben wir das mit **objXML** referenzierte XML-Dokument als Parameter, sodass die Prozedur nun wie folgt aussieht:

```
Public Sub TabellenAnalysieren()
    Dim db As dao.Database
    Dim tbl As dao.TableDef
```

```
Public Sub XMLAnalysieren(objXML As MSXML2.DOMDocument)
    Dim objDataMacro As MSXML2.IXMLDOMElement
    Dim objAttribute As MSXML2.IXMLDOMAttribute
    For Each objDataMacro In objXML.selectNodes("DataMacros/DataMacro")
        Set objAttribute = objDataMacro.selectSingleNode("@Event")
        If Not objAttribute Is Nothing Then
            Debug.Print " Event: " & objAttribute.nodeTypedValue
        Else
            Set objAttribute = objDataMacro.selectSingleNode("@Name")
            If Not objAttribute Is Nothing Then
                Debug.Print " Name: " & objAttribute.nodeTypedValue
            End If
        End If
    Next objDataMacro
End Sub
```

Listing 2: Diese Prozedur analysiert den Code der enthaltenen Makrodefinitionen anhand der Attribute.

```
Dim objXML As MSXML2.DOMDocument
Set db = CurrentDb
For Each tbl In db.TableDefs
    If Not (Left(tbl.Name, 4) = "MSys") Then
        If EnthaelMakros(tbl.Name, objXML) = True Then
            Debug.Print "Tabelle: " & tbl.Name
            Debug.Print String(Len("Tabelle: " &
                & tbl.Name), "=")
            XMLAnalysieren objXML
            Debug.Print
        End If
    End If
Next tbl
End Sub
```

Die dadurch aufgerufene Prozedur **XMLAnalysieren** finden Sie in Listing 2.

Die Prozedur erwartet das XML-Dokument als Parameter. Sie deklariert zwei Variablen: Eine für ein Element des Typs **Element** des XML-Dokuments und eines für ein **Attribut**-Element. Die erste verwendet sie als Laufvariable für eine **For Each**-Schleife über alle mit einer per **select-Nodes** abgesetzten XPath-Abfrage. Die Abfrage lautet **DataMacros/DataMacro** und liefert alle **DataMacro**-

Elemente unterhalb des **DataMacros**-Elements. Das sind für unser obiges Beispiel sechs Elemente – die fünf Datenmakros, die durch die verschiedenen Ereignisse ausgelöst werden, und ein benanntes Makro.

Innerhalb der Schleife versucht die Prozedur, mit einer weiteren XPath-Abfrage, diesmal für das bereits in **objDataMacro** gespeicherte XML-Element

des Typs **DataMacro**, das Attribut **Event** zu referenzieren. Um Attribute zu referenzieren, stellt man unter XPath dem Attributnamen das @-Zeichen voran.

Wenn **objAttribute** danach nicht leer ist, wurde das Attribut gefunden und die Prozedur gibt den Wert des Attributs über die Eigenschaft **NodeTypeValue** aus. Wurde kein Attribut namens **Event** gefunden, versucht die Prozedur im **Else**-Teil, auf die gleiche Art ein Attribut namens **Name** zu finden.

Ist dieses vorhanden, gibt die Prozedur auch dessen Wert samt Attributnamen aus.

Auf diese Weise durchläuft die Prozedur alle Elemente des Typs **DataMacro**. Wenn wir noch für eine weitere Tabelle namens **tblBestelldetails** ein Datenmakro hinzufügen, sieht die Ausgabe im Direktfenster wie folgt aus:

```
Tabelle: tblArtikel
=====
Event: AfterInsert
Event: AfterUpdate
Event: AfterDelete
Event: BeforeChange
Event: BeforeDelete
Name: dmkBeispiel
```

```
Tabelle: tblBestelldetails
=====
Event: AfterInsert
```

Datenmakros in Tabelle speichern

Wir wollen nun nicht weiter ins Detail gehen, was die in den Datenmakros enthaltenen Makroaktionen angeht, sondern eine Möglichkeit schaffen, die Tabellen und die enthaltenen Makros übersichtlich in der Datenbank anzugeben.

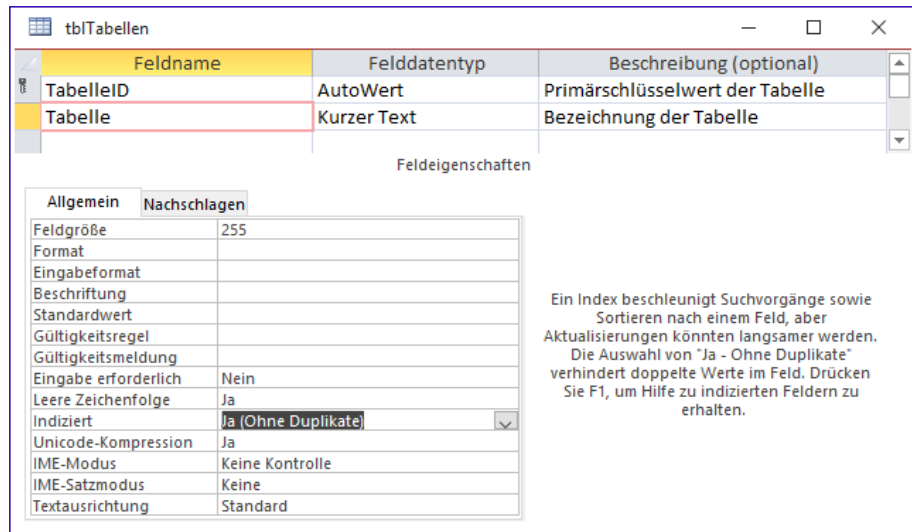


Bild 8: Tabelle zum Speichern der Tabellennamen

zeigen, damit der Benutzer diese auf die Schnelle öffnen und bearbeiten kann. Dazu benötigen wir zwei Tabellen zum Speichern der Tabellennamen und der enthaltenen Datenmakros. Auf diese greifen wir dann später von einer Kombination aus Formular und Bericht zu.

Dazu legen wir zwei Tabellen an. Die erste heißt **tblTabellen** und speichert die Namen der Tabellen, die Datenmakros enthalten. Sie enthält ein Primärschlüsselfeld namens **TabelleID** und ein Textfeld namens **Tabelle**, welches den Namen der Tabelle speichert. Für dieses Feld haben wir einen eindeutigen Index angelegt (siehe Bild 8).

Die zweite Tabelle heißt **tblDatenmakros**. Das Primärschlüsselfeld nennen wir logischerweise **Datenmakroid**. Aber wie speichern wir die Informationen aus den beiden Attributen **Event** und **Name**? Wir legen in einem Nachschlagefeld namens **DatenmakrotypID** den Typ des Datenmakros fest, der einen der folgenden sechs Werte annehmen können soll:

- **AfterInsert**
- **AfterUpdate**
- **AfterDelete**

- BeforeChange
- BeforeDelete
- NamedDataMacro

Diese Werte speichern wir in der Lookuptabelle **tblDatenmakrotypen**, die im Entwurf wie in Bild 9 aussieht.

Jetzt fehlt noch ein Feld, das den Namen des Datenmakros speichert. Dieses nennen wir Makroname. Es soll für Datenmakros, die durch Ereignisse ausgelöst werden, den gleichen Wert erhalten, der auch über das Nachschlagefeld ausgewählt wurde, also **Vor Änderung, Vor Löschung** und so weiter. Für die benannten Makros hinterlegen wir den angegebenen Namen. Schließlich geben wir in einem weiteren Nachschlagefeld namens **TabelleID** noch an, zu welcher Tabelle das Datenmakro gehört. Der Entwurf der Tabelle **tblDatenmakros** sieht dann wie in Bild 10 aus.

Die Beziehungen der Tabellen haben wir in Bild 11 abgebildet.

Tabellen mit den Datenmakros füllen

Nun passen wir die Prozeduren **TabellenAnalysieren** und **XMLAnalysieren** noch so an, dass die ermittelten Daten nicht im Direktbereich des VBA-Editors landen, sondern in den soeben erstellten Tabellen. Die Frage hierbei ist noch, ob wir nur die Tabellen in die Tabelle **tblTabellen** schreiben sollen, die bereits über Datenmakros verfügen, oder ob wir dort alle Tabellen eintragen sollen. Das

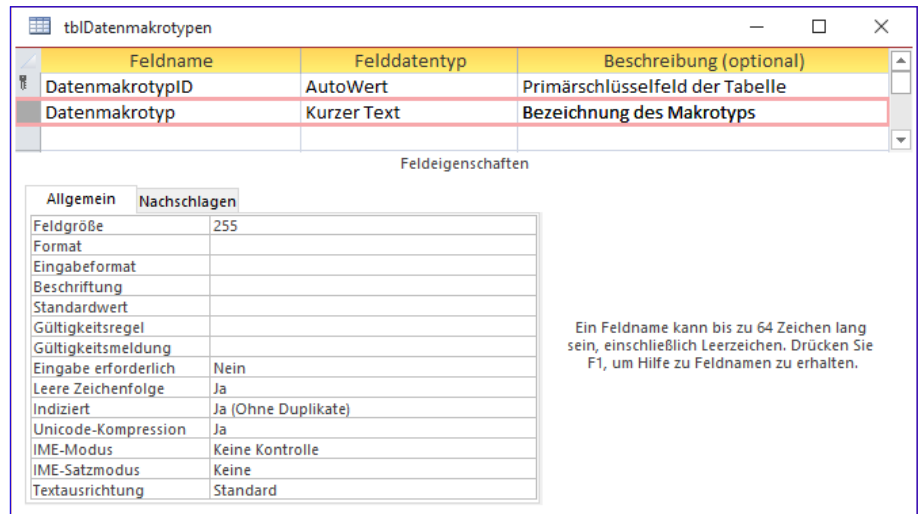


Bild 9: Tabelle zum Speichern der Datenmakrotypen

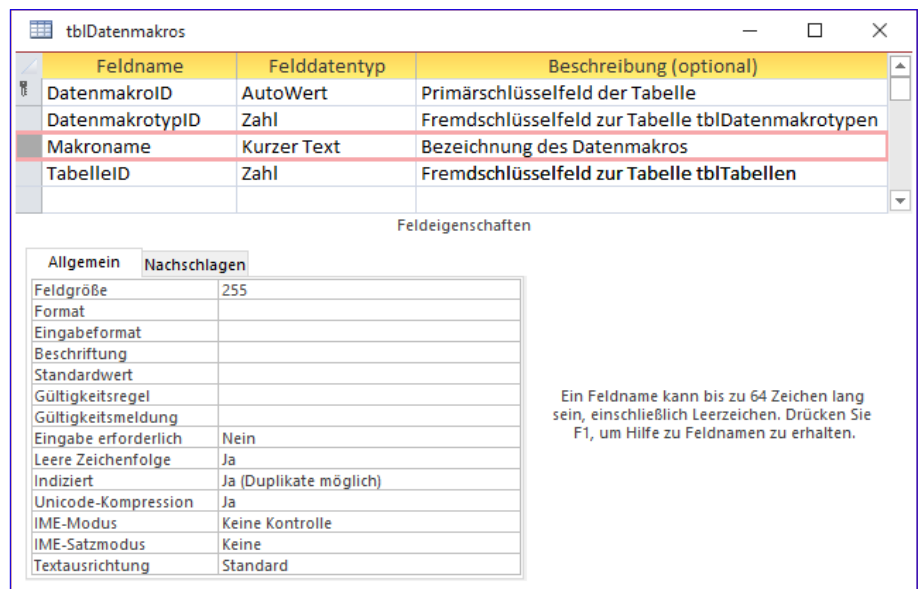


Bild 10: Tabelle zum Speichern der Datenmakro-Eigenschaften

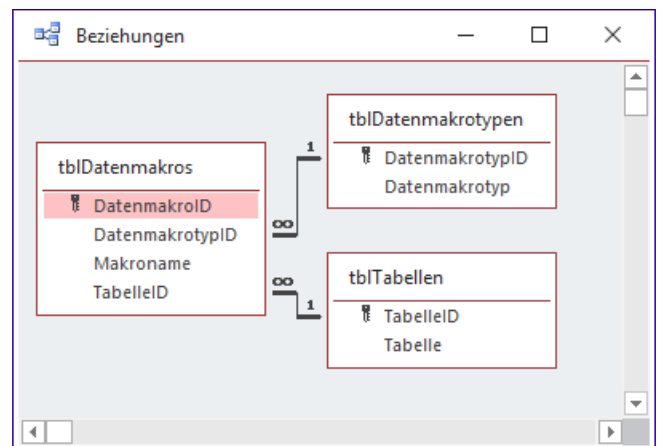


Bild 11: Beziehung der beteiligten Tabellen

wäre sinnvoll, wenn wir in unserer noch zu erstellenden Benutzeroberfläche nicht nur per Klick den Entwurf eines Makros öffnen, sondern auch noch neue Datenmakros anlegen wollen. Wir belassen es zunächst bei den Tabellen, die bereits Datenmakros enthalten.

Dazu haben wir die Prozedur **TabellenAnalysieren** wie in Listing 3 angepasst. Sie löscht zunächst alle Einträge der Tabelle **tblTabellen**.

Damit dies ausreicht, um auch die mit diesen Einträgen verknüpften Datensätze der Tabelle **tblDatenmakros** zu löschen, haben wir für die Beziehung zwischen diesen beiden Tabellen referenzielle Integrität mit Löschweitergabe definiert (siehe Bild 12).

Die Prozedur durchläuft nach wie vor alle benutzerdefinierten Tabellen der Datenbank und prüft mit der Funktion **EnthaeiltMakros**, ob die Tabelle Datenmakros enthält. In diesem Fall schreibt sie die Tabelle als neuen Datensatz in die Tabelle **tblTabellen** und ermittelt den Primärschlüsselwert des neuen Datensatzes. Damit ausgestattet, ruft sie die Prozedur **XMLAnalysieren** auf.

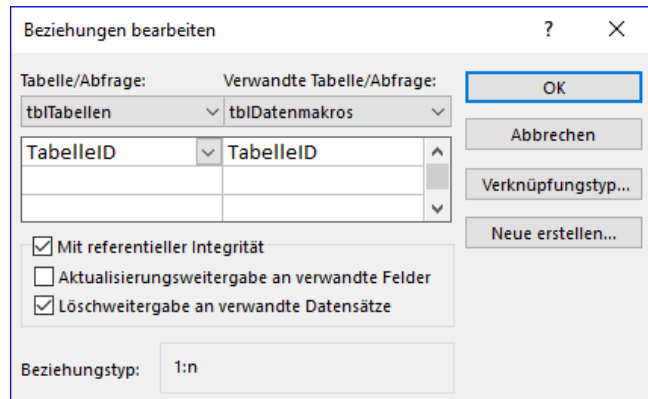


Bild 12: Referenzielle Integrität mit Löschweitergabe

Diese Prozedur erwartet neben dem Verweis auf das XML-Dokument nun noch zwei weitere Parameter: den Primärschlüsselwert des durch die Prozedur **TabellenAnalysieren** hinzugefügten Datensatzes und den Verweis auf das **Database**-Objekt der aktuellen Datenbank (siehe Listing 4).

Sie durchläuft wieder alle **DataMacro**-Elemente des XML-Dokuments und prüft, ob diese das Attribut **Event** enthalten. Ist das der Fall, ermittelt sie aus der Tabelle **tblDatenmakrotypen** den Primärschlüsselwert zu dem Eintrag,

```
Public Sub TabellenAnalysieren()
    Dim db As DAO.Database
    Dim tbl As DAO.TableDef
    Dim objXML As MSXML2.DOMDocument
    Dim lngTabelleID As Long
    Set db = CurrentDb
    db.Execute "DELETE FROM tblTabellen", dbFailOnError
    For Each tbl In db.TableDefs
        If Not (Left(tbl.Name, 4) = "MSys") Then
            If EnthaeiltMakros(tbl.Name, objXML) = True Then
                db.Execute "INSERT INTO tblTabellen(Tabelle) VALUES(' & tbl.Name & ')", dbFailOnError
                lngTabelleID = db.OpenRecordset("SELECT @@IDENTITY", dbOpenDynaset).Fields(0)
                XMLAnalysieren objXML, lngTabelleID, db
            End If
        End If
    Next tbl
End Sub
```

Listing 3: Die neue Version der Prozedur TabellenAnalysieren schreibt direkt die Tabellen in **tblTabellen**.

UNION-Abfragen

Auswahlabfragen liefern normalerweise immer die Daten aus einer einzigen Tabelle oder aus mehreren miteinander verknüpften Tabellen. In letzterem Fall soll die Abfrage immer die Kombination der Daten von verknüpften Tabellen liefern. Gelegentlich sollen die Daten von zwei oder mehr Tabellen in einer Abfrage aber auf andere Art zusammengestellt werden – nämlich so, dass die Datensätze dieser Tabellen jeweils als einzelne Datensätze erscheinen. So lassen sich dann beispielsweise Daten wie die Namen und Geburtsdaten von Personal und Kunden in einer Abfrage zusammenstellen und können als Geburtstagsliste dienen. Wie Sie UNION-Abfragen formulieren und was dabei zu beachten ist, zeigt dieser Beitrag.

Zusammenführen von Daten

Es gibt verschiedene Gründe für den Einsatz von **UNION**-Abfragen. Sie können diese nutzen, um die Daten verschiedener Tabellen wie der oben genannten Personal- und Kundentabelle zusammenzuführen, um daraus eine Liste der Geburtstage von Mitarbeitern und Kunden zu erstellen.

Oder Sie nutzen **UNION**-Abfragen, um das Ergebnis einer solchen Abfrage mit **INSERT INTO** in eine neue Tabelle zu transferieren. Diese Möglichkeit können Sie beispielsweise nutzen, um erst Daten aus externen Dateien wie etwa CSV-Dateien in jeweils eigene Tabellen zu importieren und diese dann per **UNION**-Abfrage zusammenzuführen.

Eine weitere Möglichkeit der Nutzung ist das Hinzufügen von Daten, die normalerweise nicht in einer Datenquelle vorkommen, aber dennoch angezeigt werden sollen – etwa, um die Liste der Einträge eines Kombinationsfeldes um Einträge wie **<Auswählen>** oder **<Neuer Eintrag>** zu ergänzen. Wie letzteres gelingt, schauen wir uns im Detail im Beitrag **Kombinationsfeld mit Extraeinträgen** an (www.access-im-unternehmen.de/1206).

Beispieldatenbank

Zu Beispielpzwecken wollen wir davon ausgehen, dass wir die Artikellisten verschiedener Kategorien von verschiedenen Quellen beispielsweise im CSV-Format erhalten und

diese dann zunächst über den Import der Textdateien in jeweils eine eigene Tabelle für jede Kategorie überführen.

Das Ergebnis sieht dann so aus, dass wir im Navigationsbereich sieben Tabellen mit verschiedenen Bezeichnungen wie **tblArtikel_Fleischprodukte**, **tblArtikel_Getreideprodukte** und so weiter vorfinden, die dann tatsächlich nur die Artikel der jeweiligen Kategorien enthalten. Der Einfachheit halber gehen wir davon aus, dass wir die Tabellen beim Import bereits in eine passende Struktur überführt haben (siehe Bild 1).

Wir könnten die Tabellen nun auch mit mehreren **INSERT INTO**-Abfragen in eine Zieltabelle mit gleicher Struktur einfügen, aber in diesem Beitrag soll es sich ja um die Verwendung von **UNION**-Abfragen drehen.

Erstellen von UNION-Abfragen

Access ist deshalb so beliebt, weil es für fast alle Aufgaben eine grafische Benutzeroberfläche bietet. Aber auch nur für fast alle Aufgaben. **UNION**-Abfragen sind eine der wenigen Ausnahmen, denn diese können Sie nur in der sogenannten SQL-Ansicht definieren.

Das zieht nach sich, dass Sie zumindest über grundlegende SQL-Kenntnisse verfügen sollten. In unserem Fall reicht es aus, dass Sie den einfachsten Aufbau verstehen. Die notwendigen SQL-Teilabfragen, die wir später per UNION-

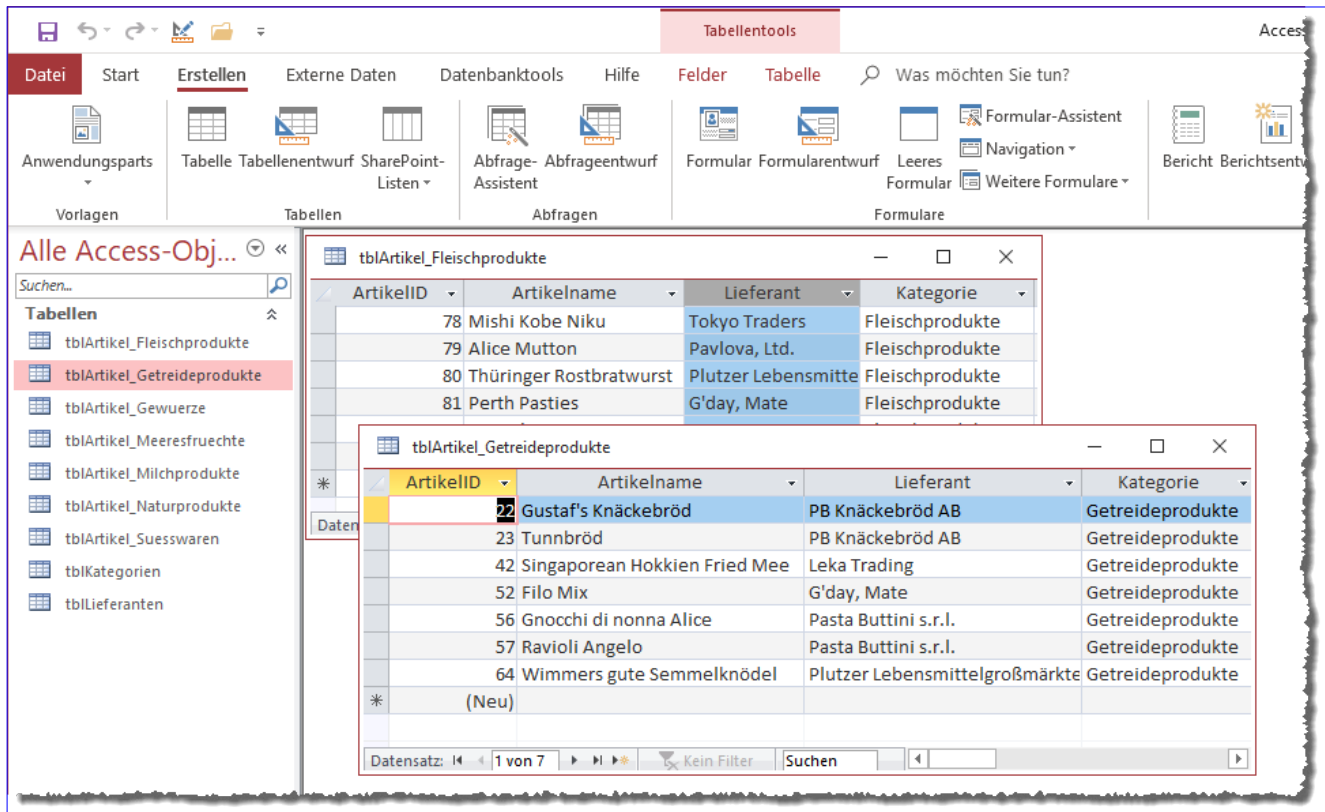


Bild 1: Beispielmaterial: sieben Tabellen mit Artikeln unterschiedlicher Kategorien

Schlüsselwort verknüpfen wollen, können Sie allerdings auch über die Entwurfsansicht für Abfragen erstellen – jedoch müssen Sie dann in die SQL-Ansicht wechseln, um an den passenden SQL-Code heranzukommen. Für den Start erstellen Sie also zunächst eine neue, leere Abfrage. Dazu verwenden Sie den Ribbon-Eintrag **Erstellen!Abfragen!Abfrageentwurf**. In die nun erscheinende Entwurfsansicht fügen wir per Doppelklick auf den Eintrag **tblArtikel_Fleischprodukte** die erste Tabelle ein (siehe Bild 2).

Dann schließen wir den Dialog **Tabelle anzeigen** und klicken doppelt auf das Sternchen (*) in der Feldliste der Tabelle **tblArti-**

kel_Fleischprodukte, um diesen Eintrag zum Entwurfsraster der Abfrage hinzuzufügen. Anschließend können Sie den Ribbon-Eintrag **Entwurf!Ergebnisse!Ansicht!SQL-**

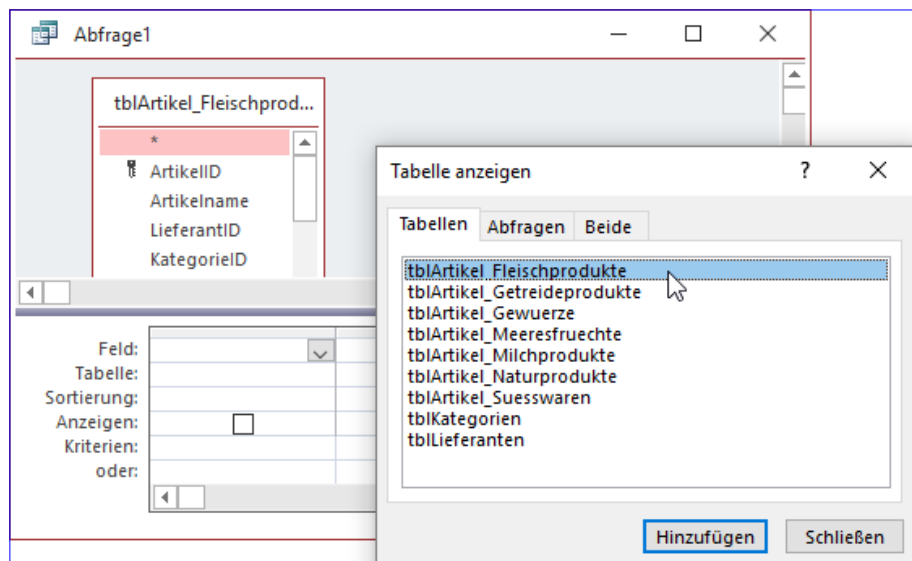


Bild 2: Hinzufügen einer Tabelle zum Abfrageentwurf

Ansicht betätigen, um in die SQL-Ansicht der neu erstellten Abfrage zu wechseln (siehe Bild 3).

Damit erhalten Sie nun die SQL-Ansicht der Abfrage (siehe Bild 4) und gleichzeitig das Werkzeug für die folgenden Beispiele dieses Beitrags.

Wenn Sie nun den Ribbon-Befehl **EntwurfErgebnisblattAnsicht** betätigen, erhalten Sie das Ergebnis der Abfrage mit allen Datensätzen der Tabelle **tblArtikel_Fleischprodukte**.

UNION einsetzen

Damit naht der Auftritt des **UNION**-Schlüsselworts: Wir wollen nun nicht mehr nur die Daten der Tabelle **tblArtikel_Fleischprodukte** ausgeben, sondern auch noch die Einträge der Tabelle **tblArtikel_Getreideprodukte** hinzufügen. Dazu erweitern wir die Abfrage nun wie folgt:

```
SELECT tblArtikel_Fleischprodukte.*
FROM tblArtikel_Fleischprodukte
UNION
SELECT tblArtikel_Getreideprodukte.*
FROM tblArtikel_Getreideprodukte;
```

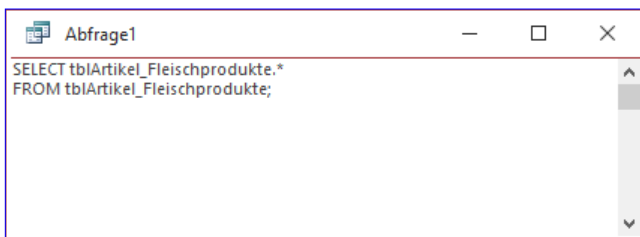


Bild 4: SQL-Ansicht einer Abfrage

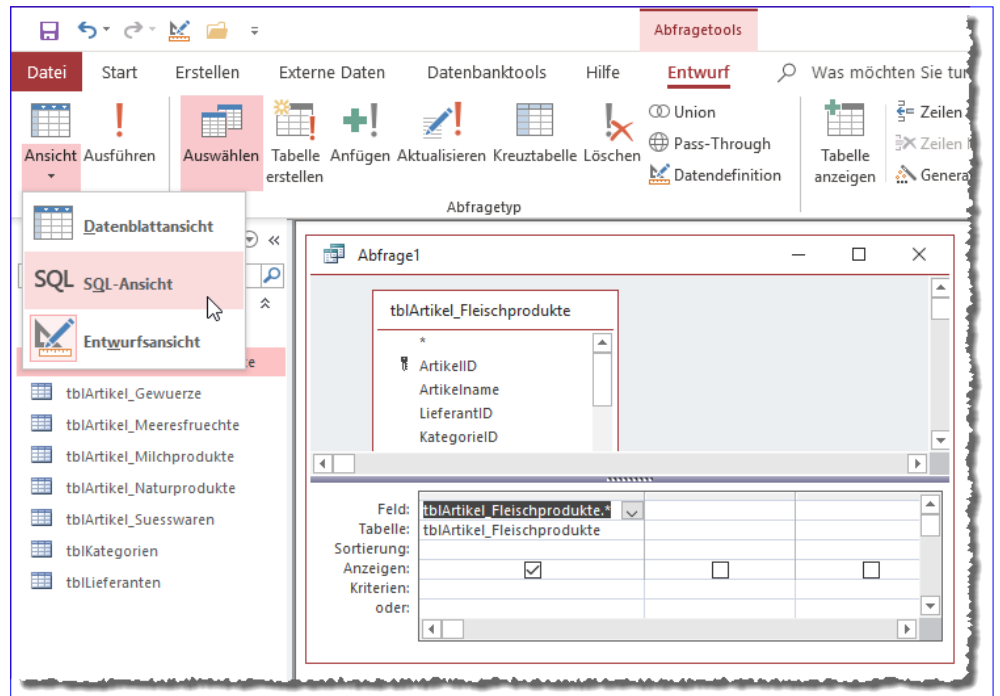


Bild 3: Wechsel zur SQL-Ansicht einer Abfrage

Wir fügen also eine weitere, genau gleich aufgebaute **SELECT**-Abfrage hinzu, die lediglich die Daten einer anderen Tabelle liefert, und verknüpfen die beiden Abfragen mit dem **UNION**-Schlüsselwort (die erste **SELECT**-Abfrage darf dabei nicht mehr mit dem Semikolon abschlossen werden).

Wenn Sie nun in die Datenblattansicht wechseln, erhalten Sie das Ergebnis aus Bild 5. Eine Besonderheit bei diesem

ArtikelID	Artikelname	LieferantID	KategorieID
22	Gustaf's Knäckebröd	9	5
23	Tunnbröd	9	5
42	Singaporean Hokkien Fried Mee	20	5
52	Filo Mix	24	5
56	Gnocchi di nonna Alice	26	5
57	Ravioli Angelo	26	5
64	Wimmers gute Semmelknödel	12	5
78	Mishi Kobe Niku	4	6
79	Alice Mutton	7	6
80	Thüringer Rostbratwurst	12	6
81	Perth Pasties	24	6
82	Tourtière	25	6
83	Pâté chinois	25	6

Bild 5: Ergebnis einer UNION-Abfrage

Abfrageergebnis ist, dass die in den ursprünglichen Tabellen als Nachschlagfelder ausgelegten Felder **LieferantID** und **KategorieID** nun nicht mehr als solche dargestellt werden, sondern nur noch die tatsächlichen Feldwerte anzeigen. Die Abfrage können Sie auch noch ein wenig vereinfacht darstellen:

```
SELECT * FROM tblArtikel_Fleischprodukte
UNION
SELECT * FROM tblArtikel_Getreideprodukte;
```

Speichern der UNION-Abfrage

Schließlich speichern wir die **UNION**-Abfrage, beispielsweise über die Tastenkombination **Strg + S**, unter dem Namen **qryArtikel_Fleisch_Getreide**.

Ein Blick in den Navigationsbereich zeigt, dass Access für **UNION**-Abfragen ein eigenes Symbol vorsieht, nämlich das Schnittmengensymbol (siehe Bild 6).

UNION als Unterabfrage

Wenn Sie die **UNION**-Abfrage nicht mit diesem Symbol anzeigen lassen wollen, können Sie den gesamten Ab-

frageausdruck als Unterabfrage einer weiteren **SELECT**-Abfrage festlegen:

```
SELECT * FROM (
    SELECT * FROM tblArtikel_Fleischprodukte
    UNION
    SELECT * FROM tblArtikel_Getreideprodukte);
```

Diese Abfrage liefert genau das gleiche Ergebnis, wird aber mit dem Symbol einer herkömmlichen Auswahlabfrage im Navigationsbereich dargestellt.

Diese Abfrage können wir nun auch in der Entwurfsansicht öffnen. Das Ergebnis ist sehr interessant – siehe Bild 7.

Die hier verwendete Bezeichnung der Unterabfrage, nämlich **\$\$\$@_Alias**, können Sie noch ersetzen, indem Sie mit dem **AS**-Schlüsselwort einen anderen Namen für die Unterabfrage definieren:

```
SELECT * FROM (
    SELECT * FROM tblArtikel_Fleischprodukte
    UNION
```

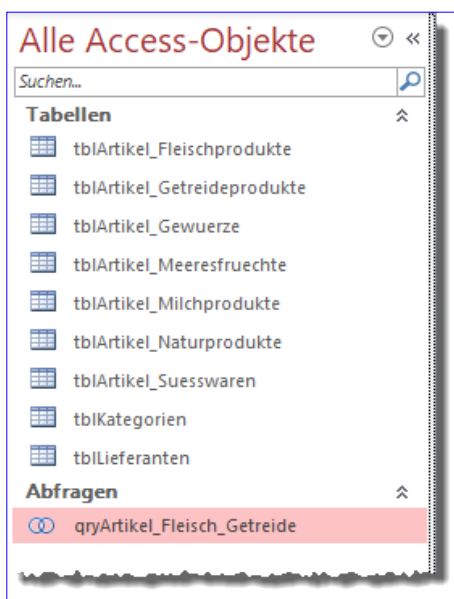


Bild 6: Eine **UNION**-Abfrage im Navigationsbereich

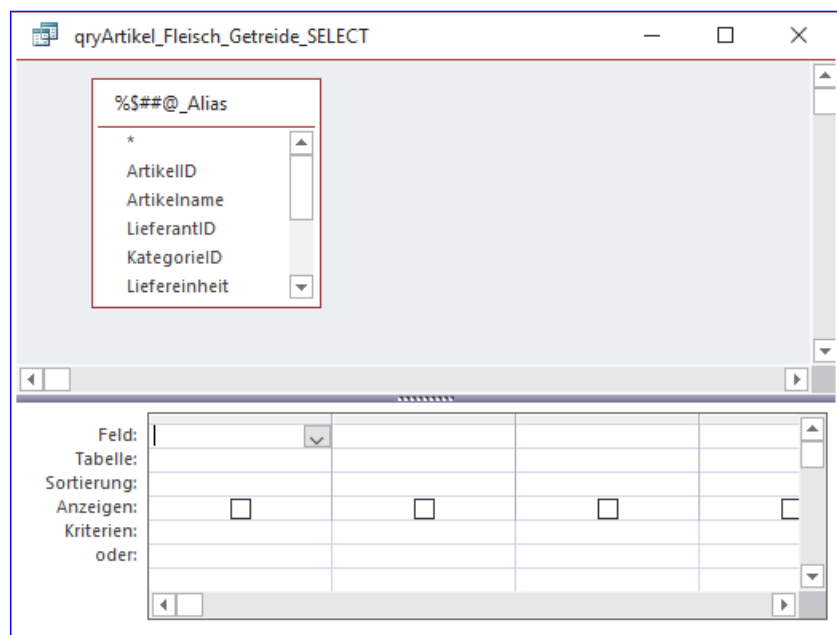


Bild 7: Eine in eine **SELECT**-Abfrage eingefasste **UNION**-Abfrage in der Entwurfsansicht

Rückgängig in Memofeldern

Seit Access 2007 bietet Access für Memofelder die Möglichkeit, die verschiedenen Versionen zu speichern. Wie aber können wir solche Änderungen wieder rückgängig machen, ohne dass der Benutzer die eingebauten Funktionen dafür nutzen muss oder gar ein extra Formular, das die bisher gespeicherten Versionen anzeigt? Sondern einfach etwa mit zwei Schaltflächen, mit denen er zwischen den vorhandenen History-Einträgen hin- und herblättern kann? Wie das gelingt, zeigen wir in diesem Beitrag.

Vorbereitungen

Wenn Sie ein Memofeld verwenden wollen, um verschiedene Versionsstände des enthaltenen Textes speichern zu können, brauchen Sie prinzipiell nur eine einzige Eigenschaft des Memofeldes zu ändern (in aktuelleren Versionen von Access heißt der Datentyp für längere Texte übrigens nicht mehr **Memofeld**, sondern **Langer Text**).

Diese Eigenschaft stellen Sie im Entwurf der Tabelle mit dem Memofeld ein, indem Sie dieses markieren und dann für die Eigenschaft **Nur Anfügen** den Wert **Ja** festlegen – wie für das Feld **Inhalt** in Bild 1.

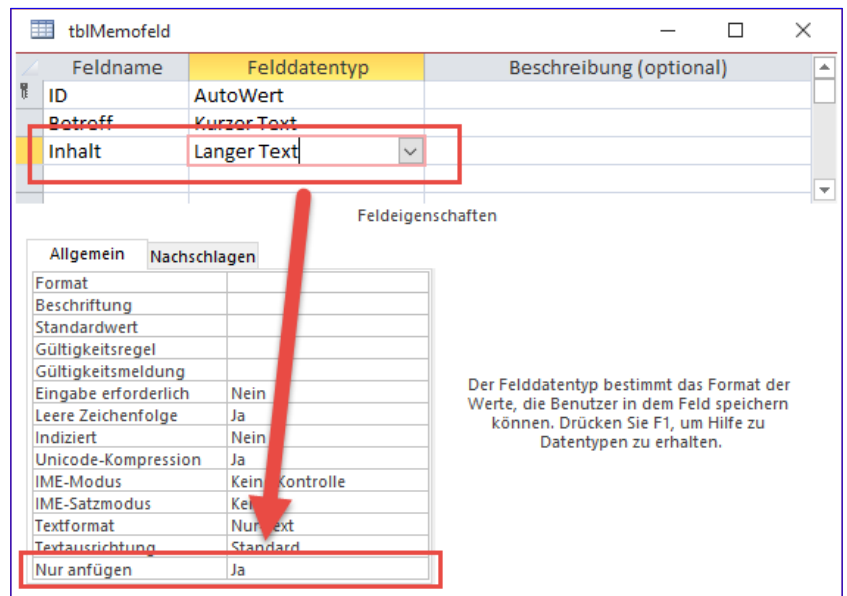


Bild 1: Memofeld mit History-Funktion

Wenn Sie nun in die Datenblattansicht wechseln, können Sie wie gewohnt Texte in das Memofeld eingeben – etwa so wie in Bild 2.

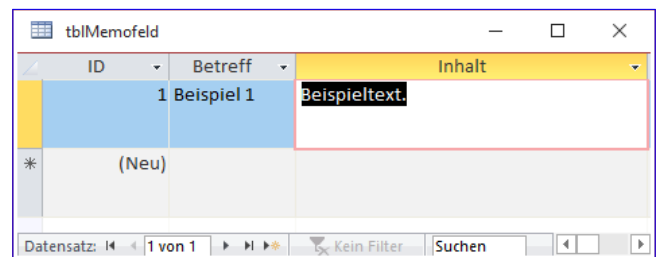


Bild 2: Eingeben eines Textes

Speichern Sie den Text nun, indem Sie das Feld verlassen, und fügen Sie dann weiteren Text hinzu – siehe Bild 3.

Die Datenblattansicht zeigt nun die neueste Version des Inhalts des Memofelds an.

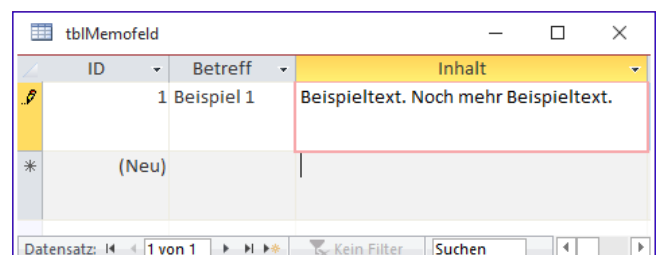


Bild 3: Eingeben eines weiteren Textes

Wie greifen wir nun auf die Historie zu? Diese erhalten wir, indem wir aus dem Kontextmenü des Feldes in der Datenblattansicht den Eintrag **Spaltenverlauf anzeigen...** aufrufen (siehe Bild 4).

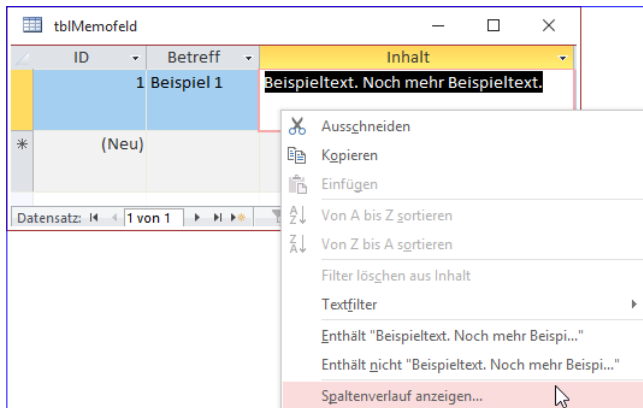


Bild 4: Aufruf der Historie

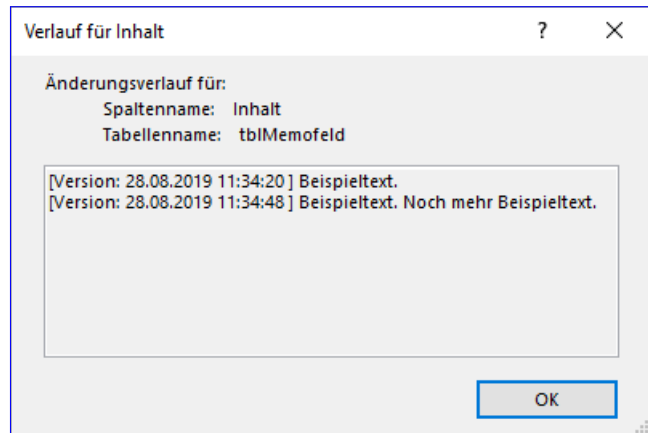


Bild 5: Anzeige der Historie

Danach erscheint der Dialog **Verlauf für Inhalt**, der in einem Textfeld alle Änderungsdaten samt des zu diesem Zeitpunkt vorliegenden Inhalts anzeigt (siehe Bild 5).

Damit erhalten wir allerdings noch nicht die Gelegenheit, auf einen bestimmten Versionsstand zuzugreifen und diesen wiederherzustellen.

Versionsstände per VBA abfragen

Wenn wir versuchen, die Versionsstände per **DLookup** abzufragen, erhalten wir jeweils nur die aktuelle Version des Memofeld-Inhalts:

```
Debug.Print Dlookup("Inhalt", "tblMemofeld")
Beispieltext. Noch mehr Beispieltext.
```

Es gibt jedoch eine neue Methode des **Application**-Objekts namens **ColumnHistory**, welche auch den im Textfeld des Dialogs **Verlauf für Inhalt** angezeigten Text liefert. Diese erwartet drei Parameter:

- **TableName:** Name der Tabelle, in der sich das Memofeld befindet.
- **ColumnName:** Name des Memofelds
- **queryString:** Bedingung für den zu untersuchenden Datensatz. Hierbei handelt es sich im Gegensatz zu

dem entsprechenden Parameter etwa bei der **DLookup**-Funktion um einen Pflichtparameter. Die Bedingung darf nur genau einen Datensatz zurückliefern!

Ein Aufruf könnte etwa wie folgt aussehen:

```
? Application.ColumnHistory("tblMemofeld", "Inhalt", "ID = 1")
[Version: 28.08.2019 11:34:20 ] Beispieltext.
[Version: 28.08.2019 11:34:48 ] Beispieltext. Noch mehr
Beispieltext.
```

Wann wird der Versionsstand gespeichert?

Das Hinzufügen eines neuen Versionsstandes erfolgt erst, wenn der Benutzer den Datensatz speichert. Das Verlassen des Feldes und somit der Fokusverlust reicht nicht aus.

Auf einzelne Einträge zugreifen

Damit können wir nun noch nicht auf die einzelnen Versionsstände des Textes des Memofelds zugreifen, was aber nötig ist, um komfortabel zwischen den verfügbaren Versionsständen hin- und herzuschalten.

Um dies zu realisieren, erstellen wir zunächst ein neues Formular, das über die Eigenschaft **Datensatzquelle** an die Tabelle **tblMemofeld** gebunden ist. Wir ziehen alle Felder aus der Feldliste in den Entwurf des Formulars.

Dann fügen wir drei Steuerelemente zum Formular hinzu, die wir wie in Bild 6 neben dem Textfeld für das Memofeld platzieren. Die beiden Schaltflächen heißen **cmdVorheriger** und **cmdNaechster**. Das Textfeld soll die Bezeichnung **txtVersion** erhalten. Außerdem stellen wir die Eigenschaft **Aktiviert** auf **Nein** und **Gesperrt** auf **Ja** ein – so wird der Inhalt weiterhin angezeigt, kann aber vom Benutzer nicht geändert werden.

Für die nachfolgend hinzuzufügenden Prozeduren benötigen wir zwei allgemein deklarierte Variablen, die Sie wie folgt zum Klassenmodul des Formulars **frmMemofelder** hinzufügen:

```
DDim colVersions As Collection
Dim lngVersion As Long
```

Die Variable **colVersions** soll eine Collection aufnehmen, die wir mit den Texten der verschiedenen Versionen des Memofeldes füllen.

Versionsstände ermitteln und in Collection eintragen

Dies erledigen wir zu verschiedenen Gelegenheiten. Wichtig ist zunächst, dass wir die Prozedur erstellen, mit der die verschiedenen Versionsstände des Inhalts des Memofeldes für den aktuellen Datensatz ermittelt und in das **Collection**-Objekt eingetragen werden können. Diese Prozedur finden Sie in Listing 1. Sie deklariert zunächst einige Variablen und initialisiert dann die Objektvariable **colVersions** des Typs **Collection**.

Die Prozedur prüft mit **Not IsNull(Me!ID)** zuerst, ob das Primärschlüsselfeld einen Wert ungleich **Null** enthält und somit, ob das Formular einen gültigen Datensatz anzeigt. Falls nicht, wird lediglich der Wert der Variablen **lngVersion** auf den Wert **0** eingestellt. Das ist beispielsweise der Fall, wenn das Formular einen neuen, leeren Datensatz anzeigt.

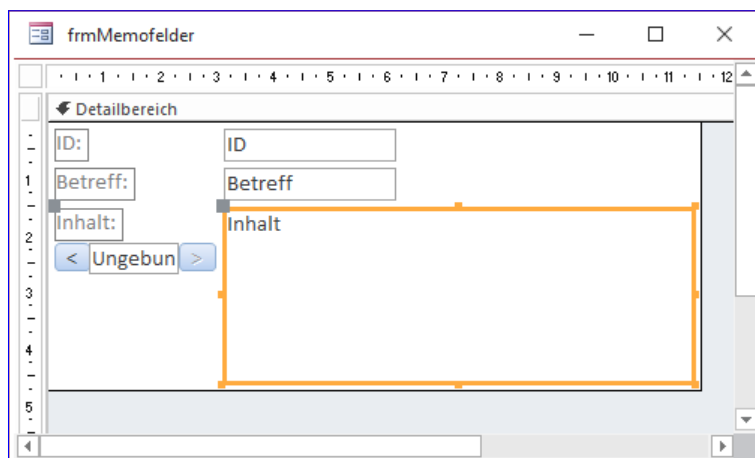


Bild 6: Steuerelemente zum Navigieren in den Versionen

Liegt jedoch bereits ein Datensatz vor, lesen wir den kompletten Inhalt des Memofeldes in die Variable **strVersions** ein. Dazu nutzen wir die Methode **ColumnHistory**, die wir bereits weiter oben vorgestellt haben.

Die folgende Anweisung ermittelt die Position des ersten Auftretens der Zeichenkette **[Version:** und speichert den Zahlenwert in der Variablen **lngPosStart**.

Hat **lngPosStart** danach einen Wert ungleich **0** und entspricht es nicht der Länge der gesamten in **strVersions** gespeicherten Zeichenkette plus drei (Erläuterung dazu weiter unten), folgen weitere Schritte. So ermittelt die Prozedur für die Variable **lngPosEnde** die Position der schließenden eckigen Klammer. Unmittelbar danach wird wieder die Position des nächsten Vorkommens von **[Version:** in die Variable **lngPosStart** eingetragen.

Hat **lngPosStart** an dieser Stelle den Wert **0**, was bedeutet, dass keine weiteren Vorkommen mehr gefunden werden konnten, stellen wir den Wert von **lngPosStart** auf den Wert der Länge von **strVersion** plus drei ein.

Nun wissen wir, an welcher Stelle sich der eigentliche Text der aktuellen Version des Memofeldes befindet. Diesen Text lesen wir dann mit der **Mid**-Funktion mit den entsprechenden Zahlenangaben in die Variable **strVer-**

Kombinationsfeld mit Extraeinträgen

Normalerweise enthalten Kombinationsfelder nur die Einträge einer Tabelle oder Abfrage als Datensatzherkunft. Manchmal wollen Sie dem Benutzer aber mit dem Kombinationsfeld noch mehr Funktionen bieten. Zum Beispiel wollen Sie vielleicht den Text <Auswählen> anzeigen, damit der Benutzer sieht, dass hier eine Auswahl vorzunehmen ist oder Sie möchten dem Benutzer mit einem Eintrag wie <Hinzufügen> die Möglichkeit geben, direkt über die Auswahl dieses Eintrags ein Detailformular zur Eingabe eines neuen Datensatzes zu öffnen. Dieser Beitrag zeigt, wie das funktioniert und welche Herausforderungen es gibt, wenn das Kombinationsfeld sonst noch keine Daten anzeigt.

Wenn Sie in einem Kombinationsfeld die Daten einer Abfrage oder Tabelle zur Auswahl anbieten wollen, gelingt das ganz einfach durch Einstellen weniger Eigenschaften. Die wichtigste ist dabei die Eigenschaft **Datensatzherkunft**, der Sie den Namen der zu verwendenden Tabelle oder der Abfrage zuweisen oder der für die Sie direkt einen SQL-Ausdruck hinterlegen.

Dann verwenden Sie beispielsweise einen Ausdruck wie den folgenden, den Sie leicht über den Abfrage-Generator der Eigenschaft **Datensatzherkunft** zusammenstellen können:

```
SELECT tb1Kunden.KundeID, [Nachname] & ", "
    & [Vorname] AS Kunde
FROM tb1Kunden
ORDER BY [Nachname] & ", " & [Vorname];
```

Wenn Sie nun noch die Eigenschaft **Spaltenanzahl** auf 2 und die Eigenschaft **Spaltenbreiten** auf 0cm einstellen, wird auch nur der Inhalt des zweiten Feldes im Kombinationsfeld angezeigt, nicht aber der Primärschlüsselwert aus dem ersten Feld. Dieser wird allein als Wert der gebundenen Spalte genutzt (siehe Bild 1).

Hier haben wir die beiden Felder **Nachname** und **Vorname**, getrennt durch ein Komma, zu einem neuen Feld namens **Kunde** zusammengefasst, damit beide Feldinhalte sowohl in der Auswahlliste als auch als ausgewählter Eintrag angezeigt werden (siehe Bild 2).

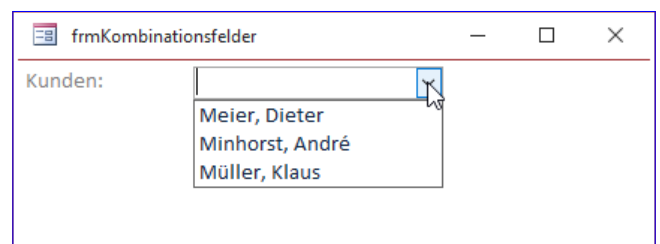


Bild 1: Einfache Auswahl

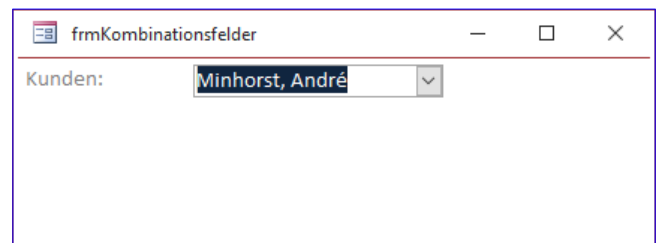


Bild 2: Anzeige des gewählten Eintrags

Eintrag <Auswählen> hinzufügen

Nun wollen wir dem Benutzer deutlich machen, dass er in diesem Feld einen Eintrag auswählen soll. Dazu soll das Feld einen Eintrag mit dem Text <Auswählen> hinzufügen, der immer direkt beim Öffnen des Formulars angezeigt wird.

Dazu wollen wir per **UNION**-Abfrage einen Eintrag voranstellen. Die **UNION**-Abfrage führt zwei einzelne **SELECT**-Abfragen zusammen und sieht wie folgt aus:

```
SELECT 0 AS KundeID, '<Auswählen>' AS Kunde
FROM tb1Kunden
UNION
```

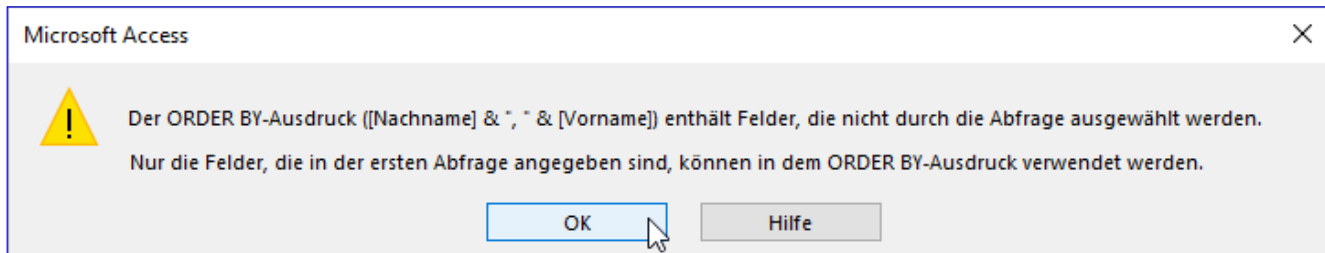


Bild 3: Fehlermeldung durch die **ORDER BY**-Klausel

```
SELECT tb1Kunden.KundeID, [Nachname] & ", " &
    & [Vorname] AS Kunde
FROM tb1Kunden
ORDER BY [Nachname] & ", " & [Vorname];
```

Damit erhalten wir beim Aufklappen des Kombinationsfeldes, das wir als neues Kombinationsfeld namens **cboAuswählen** angelegt haben, allerdings direkt die erste Fehlermeldung (siehe Bild 3). Wir müssten also noch die Felder **Nachname** und **Vorname** in den ersten **SELECT**-Ausdruck übernehmen, damit die Sortierung funktioniert.

Das lässt sich allerdings leicht beheben, indem wir einfach in der vom Abfragegenerator erzeugten **ORDER BY**-Klausel **[Nachname] & ", " & [Vorname]** durch das weiter vorn aus diesen Feldern zusammengesetzte Feld **Kunde** ersetzen:

```
SELECT 0 AS KundeID, '<Auswählen>' AS Kunde
FROM tb1Kunden
UNION
SELECT tb1Kunden.KundeID, [Nachname] & ", " &
    & [Vorname] AS Kunde FROM tb1Kunden
ORDER BY Kunde;
```

Damit erhalten wir dann auch wie in Bild 4 das angestrebte Zwischenergebnis – der Eintrag **<Auswählen>** wird als erster Eintrag der Auswahlliste angezeigt.

<Auswählen> als Standardwert

Wenn das Kombinationsfeld nun gleich beim Anzeigen den Eintrag **<Auswählen>** anzeigen soll, müssen Sie noch eine Ereignisprozedur anlegen, die dies beim Öffnen des

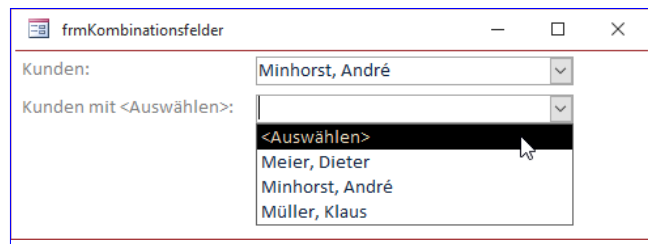


Bild 4: Anzeige des Eintrags **<Auswählen>**

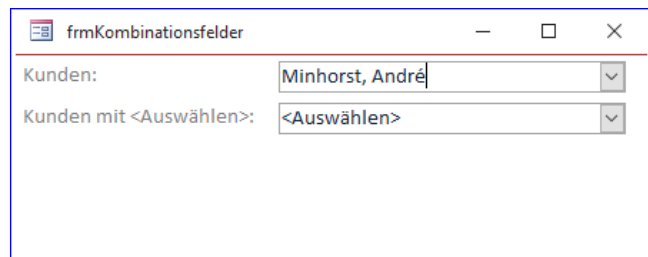


Bild 5: Anzeige des Eintrags **<Auswählen>** gleich beim Öffnen des Formulars

Formulars einrichtet. Das erledigen wir mit der Ereigniseigenschaft **Beim Laden**, für die wir den Wert **[Ereignisprozedur]** einstellen und dann mit einem Klick auf die Schaltfläche mit den drei Punkten eine Ereignisprozedur erstellen, die wir wie folgt füllen:

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

Dann erscheint der **<Auswählen>**-Eintrag auch direkt beim Öffnen des Formulars (siehe Bild 5).

Dadurch, dass wir als Wert des Feldes **KundeID** für den **<Auswählen>**-Eintrag die Zahl **0** angegeben haben, könnten wir die Anweisung auch wie folgt formulieren:

```
Private Sub Form_Load()
    Me!cboAuswaehlen = 0
End Sub
```

Und dass der Eintrag **<Auswählen>** direkt als erster Eintrag des Kombinationsfeldes erscheint, ist auch kein Zufall: Wir haben extra das Kleiner-Zeichen als erstes Zeichen gewählt, weil es in der Sortierung nach dem ASCII-Code vor den Buchstaben liegt.

Sortierung nach dem Primärschlüsselwert

Wenn Sie etwa das Feld **AnredeID** in einem Kombinationsfeld namens **cboAnredeID** auf die gleiche Weise abbilden wollen – also etwa wie in Bild 6 –, können Sie die Sortierung auch anders erzeugen.

In diesem Fall verwenden wir die folgende Datensatzherkunft:

```
SELECT 0 AS AnredeID, '<Auswählen>' AS Anrede
FROM tb1Anreden
UNION
SELECT tb1Anreden.AnredeID, tb1Anreden.Anrede
FROM tb1Anreden
ORDER BY AnredeID;
```

Der Unterschied ist, dass wir hier nach dem Feld **AnredeID** sortieren und nicht nach dem Feld **Anrede**. Der Grund ist, dass wir die Werte dieser Tabelle direkt in der gewünschten Reihenfolge eingegeben haben und daher eine Sortierung nach dem Primärschlüsselwert möglich ist. Auch hier können wir dann direkt den Wert des Kombinationsfeldes auf **0** einstellen:

```
Private Sub Form_Load()
    Me!cboAnreden = 0
End Sub
```

Datensatzherkunft leer?

Es kann sein, dass Sie diese Konstellation für eine leere Kundentabelle verwenden. Wenn Sie das Formular dann

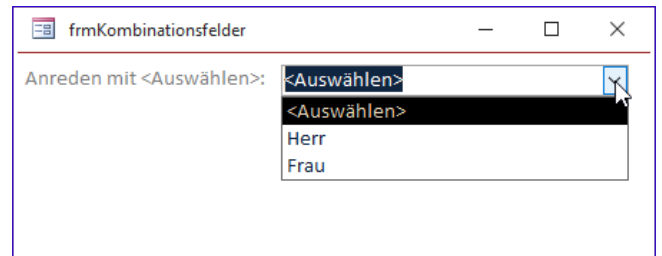


Bild 6: Auswählen von Anreden

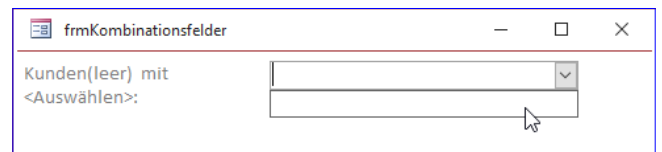


Bild 7: UNION mit einer leeren Tabelle zeigt keine Datensätze.

öffnen, zeigt das Kombinationsfeld überhaupt keinen Eintrag an – auch der Eintrag **<Auswählen>** ist verschwunden (siehe Bild 7).

Das ist übrigens ein Unterschied zu dem Fall, wo die zweite Abfrage lediglich keinen Wert zurückliefert, die referenzierte Tabelle aber bereits Datensätze enthält. Wenn **tblKunden** nicht leer ist, wir aber als Kriterium **1=2** definieren, wird nur der Eintrag **<Auswählen>** angezeigt:

```
SELECT 0 AS KundeID, '<Auswählen>' AS Kunde
FROM tblKunden
UNION
SELECT tblKunden.KundeID, [Nachname] & ", "
    & [Vorname] AS Kunde
FROM tblKunden
WHERE 1=2 ORDER BY Kunde;
```

Wie aber zeigen wir nun den Eintrag **<Auswählen>** an, auch wenn die Tabelle **tblKunden** keine Einträge enthält? Das Problem ist genau genommen, dass wir auch im ersten Teil der **UNION**-Abfrage die Tabelle **tblKunden** als Datenquelle angegeben haben.

Wir brauchen also nur einen anderen Tabellennamen anzugeben, zum Beispiel **MSysObjects** – diese Tabelle ist in einer Access-Anwendung immer vorhanden:

```
SELECT 0 AS KundeID, '<Auswählen>' AS Kunde
FROM MSysObjects
UNION
SELECT tblKundenLeer.KundeID, [Nachname] & ", "
    & [Vorname] AS Kunde
FROM tblKundenLeer
ORDER BY Kunde;
```

Damit gelingt dann auch die Anzeige des Eintrags **<Auswählen>**, obwohl die hier verwendete Tabelle **tblKundenLeer** keine Datensätze enthält (siehe Bild 8).

Wenn Sie auch für den Fall sichergehen wollen, dass Sie einmal die Daten aus den Tabellen etwa einer SQL Server-Datenbank beziehen, können Sie auch eine Hilfstabelle namens **tblKombihelfer** anlegen, die wie in Bild 9 aufgebaut ist.

In der Tabelle **tblKombihelfer** legen wir dann die verschiedenen benötigten Einträge an (siehe Bild 10). In der **UNION**-Abfrage fügen wir dann die entsprechenden Einträge zur Datensatzquelle des Kombinationsfeldes hinzu:

```
SELECT ID AS KundeID, Wert AS Kunde
FROM tblKombihelfer
WHERE ID IN (0)
UNION
SELECT tblKundenLeer.KundeID, [Nachname] & ", "
    & [Vorname] AS Kunde
FROM tblKundenLeer
ORDER BY Kunde
```

ID	Wert	Zum H
-1	<Neuer Eintrag>	
0	<Auswählen>	
0	0	

Bild 10: Datenblattansicht der Hilfstabelle **tblKombihelfer**

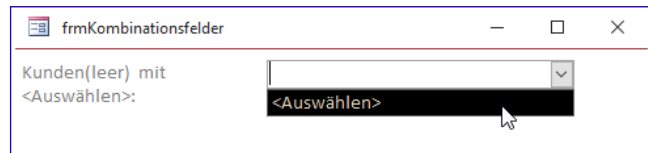


Bild 8: **UNION** mit einer leeren Tabelle zeigt den Eintrag **<Auswählen>**.

Feldname	Felddatentyp	Beschreibung (optional)
ID	Zahl	Primärschlüsselfeld
Wert	Kurzer Text	Text des Eintrags

Bild 9: Entwurf der Hilfstabelle **tblKombihelfer**

Wir müssen allerdings nach wie vor die Alias-Bezeichnungen **KundeID** und **Kunde** verwenden, da die Feldnamen der per **UNION** zusammengeführten Tabellen gleich lauten müssen. Nun finden wir den Eintrag ebenfalls im Kombinationsfeld vor – hier in **cboUNIONMitttblKombihelfer**.

<Neuer Eintrag> im Kombinationsfeld

Auf die gleiche Weise können wir einen Eintrag namens **<Neuer Eintrag>** hinzufügen. Dazu erweitern wir den Wert der Eigenschaft **Datensatzherkunft** einfach um den entsprechenden Wert für die **WHERE**-Klausel der ersten Tabelle vor dem **UNION**-Schlüsselwort:

```
SELECT ID AS KundeID, Wert AS Kunde
FROM tblKombihelfer
WHERE ID IN (0, -1)
UNION
SELECT tblKundenLeer.KundeID, [Nachname] & ", "
    & [Vorname] AS Kunde
FROM tblKundenLeer
ORDER BY Kunde;
```

Die Voreinstellung realisieren wir wieder wie folgt:

```
Private Sub Form_Load()
    Me!cboNeuerEintrag = 0
End Sub
```

Kopier- und Löschrerienfolge in MySQL

Im Beitrag »Kopier- und Löschrerienfolge für Tabellen« (www.access-im-unternehmen.de/926) haben wir ermittelt, wie wir die richtige Reihenfolge für das Löschen von Tabellen und das Kopieren von Tabelleninhalten von einer Datenbank in die nächste ermitteln. Wenn Sie die Reihenfolge nicht beachten, kann es nämlich sein, dass Datensätze wegen Fremdschlüsselverletzungen nicht gelöscht und auch nicht kopiert werden können. Im vorliegenden Beitrag zeigen wir, wie Sie die vorgestellte Lösung für das Löschen und Kopieren von Tabellen in MySQL-Datenbanken nutzen können.

In oben genanntem Beitrag haben wir bereits erläutert, in welcher Reihenfolge die Tabellen gelöscht und deren Daten kopiert werden müssen, damit dies ohne Fehlermeldungen durch Restriktionen möglich ist. Wir fassen das noch einmal zusammen: Wenn Sie etwa nur die beiden Tabellen **tblKunden** und **tblAnreden** verwenden, bei denen die Tabelle über das Fremdschlüsselfeld **AnredeID** mit dem Feld **AnredeID** der Tabelle **tblAnreden** verknüpft ist, gelten folgende Regeln:

- Beim Löschen der enthaltenen Datensätze müssen erst die Datensätze gelöscht werden, die in der Tabelle mit dem Fremdschlüsselfeld der Beziehung enthalten sind. Wir löschen also zunächst die Datensätze der Tabelle **tblKunden** und erst dann die Datensätze der Tabelle **tblAnreden**.
- Beim Kopieren der Daten von Tabellen einer Datenbank in die Tabellen der anderen Datenbank müssen erst die Datensätze der Tabelle mit dem Primärschlüsselfeld der Beziehung kopiert werden und dann die der anderen Tabelle. Wir kopieren also erst die Datensätze der Tabelle **tblAnreden** und erst dann die der Tabelle **tblKunden**.

Auf die gleiche Weise können Sie bei komplexeren Datenmodellen vorgehen. Um das zu automatisieren, sucht man sich zum Löschen der Daten zunächst die Tabellen heraus, auf die keine andere Tabelle per Fremdschlüsselfeld verweist. Im zweiten Schritt ermitteln wir alle Tabellen, die

nicht bereits ermittelt wurden und auf die keine andere der noch übrigen Tabellen per Fremdschlüsselfeld verweist und so weiter.

Beim Kopieren von Daten suchen wir im ersten Schritt zuerst die Tabellen heraus, die nicht per Fremdschlüsselfeld auf andere Tabellen verweisen. Dann suchen wir im zweiten Schritt die Tabellen heraus, die per Fremdschlüsselfeld nur auf die im ersten Schritt ermittelten Tabellen verweisen und so weiter.

Reihenfolge per VBA bestimmen

Im weiter oben erwähnten Beitrag haben wir bereits einen Algorithmus definiert, mit dem wir die Reihenfolge der Tabellen beim Löschen und beim Kopieren der Daten der Tabellen ermitteln. Dabei haben wir die Elemente der DAO-**TableDefs**-Auflistung durchlaufen und für die Informationen bezüglich der Beziehungen die Daten der Relations-Auflistung genutzt.

Unter MySQL stehen uns diese Elemente nicht zur Verfügung. Wenn Sie zwei miteinander in Beziehung stehende und per ODBC eingebundene MySQL-Tabellen in das Beziehungen-Fenster ziehen, sehen Sie bereits, dass die für diese Tabellen definierte Beziehung nicht in Access verfügbar ist (siehe Bild 1).

Alle Tabellen einbinden?

Hier stehen wir am Scheideweg, denn nun gibt es zwei Möglichkeiten:

- Sie fügen alle per ODBC eingebundenen Tabellen zum **Beziehungen**-Fenster hinzu und fügen die Beziehungen manuell ein. Das ist allerdings je nach Anzahl der Tabellen eine Fleißarbeit, die – wie bei solchen Arbeiten üblich – mit einer Fehleranfälligkeit ausgestattet ist. Danach können Sie die Reihenfolge der Tabelle zum Löschen und Kopieren mit den bereits vorgestellten Prozeduren ermitteln.
- Sie schreiben eine Prozedur, die alle Tabellen der Datenbank unter MySQL hinsichtlich der Beziehungen untersucht und diese automatisiert zu den per ODBC eingebundenen Tabellen hinzufügt. Danach können Sie die Reihenfolge der Tabelle zum Löschen und Kopieren ebenfalls mit den bereits vorgestellten Prozeduren ermitteln.
- Oder wir passen die vorhandenen Prozeduren so an, dass wir per ADO direkt auf die Informationen über die Beziehungen zwischen den Tabellen zugreifen. Hierzu müssten wir noch herausfinden, wie wir auf die gewünschten Informationen zugreifen. Da wir hierbei etwas Neues lernen, entscheiden wir uns an dieser Stelle für diese Option.

Tabellen durchlaufen

Das Durchlaufen der Tabellen erfolgt nicht über eine Auflistung wie **TableDefs**, sondern wir nutzen die Anweisung **SHOW TABLES**, mit der wir wie mit einer **SELECT**-Anweisung auf die Daten der MySQL-Datenbank zugreifen. **Show Tables** liefert uns ebenfalls ein Ergebnis in Form eines Recordsets zurück. In den folgenden Codezeilen nutzen wir eine Hilfsfunktion namens **GetRecordset**, deren ersten Parameter wir mit einer Konstanten füllen, welche die Verbindungszeichenfolge zu der gewünschten Datenbank enthält:

```
Dim rstTabellen As ADODB.Recordset
Set rstTabellen = GetRecordset(GetConnection(7
    cStrVerbindungszeichenfolgeAlt), "SHOW TABLES")
Do While Not rstTabellen.EOF
```

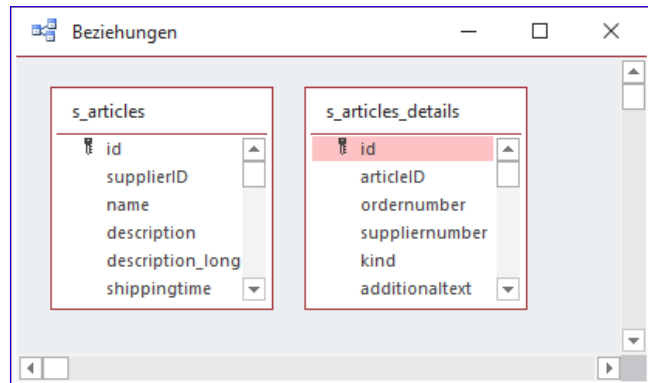


Bild 1: Keine Beziehungen von ODBC-verknüpften Tabellen sichtbar

```
Debug.Print rstTabellen.Fields(0)
rstTabellen.MoveNext
Loop
```

Hier füllen wir so ein Recordset namens **rstTabellen** mit allen Einträgen, die der Befehl **SHOW TABLES** zurückliefert und geben diese im Direktbereich des VBA-Fensters aus. Für die verwendeten Hilfsfunktionen benötigen wir noch einen Verweis auf die Bibliothek **Microsoft ActiveX Data Objects x.y Library**, den Sie über den **Verweise**-Dialog hinzufügen (VBA-Editor, Menüeintrag **Extras/Verweise**).

Beziehungen ermitteln

Nun wollen wir wissen, ob eine Tabelle über ein Fremdschlüsselfeld mit einer anderen Tabelle verknüpft ist oder ob eine Tabelle vom Fremdschlüsselfeld einer anderen Tabelle referenziert wird.

Dazu verwenden wir testweise die Prozedur **Relationinfo** aus Listing 1. In dieser weisen wir der Variablen **strSQL** die folgende Abfrage zu:

```
SELECT TABLE_NAME, COLUMN_NAME, REFERENCED_TABLE_NAME,
REFERENCED_COLUMN_NAME
FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
WHERE REFERENCED_TABLE_NAME IS NOT NULL;
```

Hier ermitteln wir die Felder mit den relevanten Informationen aus der Tabelle **INFORMATION_SCHEMA.KEY_COLUMN_USAGE** für die Bedingung **REFERENCED_TABLE_**


```
Public Sub Relationinfo()
    Dim rst As ADODB.Recordset
    Dim strSQL As String
    Dim fld As ADODB.Field
    strSQL = "SELECT TABLE_NAME, COLUMN_NAME, REFERENCED_TABLE_NAME, REFERENCED_COLUMN_NAME " & vbCrLf _
        & "FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE" & vbCrLf _
        & "WHERE REFERENCED_TABLE_NAME IS NOT NULL;"
    Set rst = GetRecordset(GetConnection(cStrVerbindungszeichenfolgeAlt), strSQL)
    For Each fld In rst.Fields
        Debug.Print fld.Name,
    Next fld
    Debug.Print
    Do While Not rst.EOF
        For Each fld In rst.Fields
            Debug.Print fld.Value,
        Next fld
        Debug.Print
        rst.MoveNext
    Loop
End Sub
```

Listing 1: Ermitteln aller Beziehungen mit beteiligten Tabellen und Feldern

NAME IS NOT NULL. Danach durchlaufen wir alle Felder der Tabelle und geben die Namen der Felder als Kopfzeile im Direktbereich des VBA-Editors aus und liefern direkt anschließend die Werte dieser Felder für alle gefundenen Datensätze.

Das sieht dann etwa wie folgt aus:

TABLE_NAME	COLUMN_NAME	REFERENCED_TABLE_NAME	REFERENCED_COLUMN_NAME
s_order_basket_attributes	basketID	s_order_basket	id
s_core_auth_attributes	authID	s_core_auth	id
s_articles_attributes	articleID	s_articles	id

Damit kommen wir nun zumindest an alle Informationen heran, die wir theoretisch benötigen, um die Reihenfolge der Tabellen für das Löschen und das Kopieren zu bestimmen. Wir müssen die Informationen nur noch geschickt kombinieren, was wir in den folgenden Schritten erledigen werden.

Reihenfolge für das Kopieren ermitteln

Die Reihenfolge der Tabellen für das Kopieren der Daten ermitteln wir in der Funktion **TabellenreihenfolgeKopieren** aus Listing 2. In dieser Funktion, welche die zu verwendende Verbindungszeichenfolge als Parameter erwartet und die ein **Collection**-Objekt als Ergebnis zurückliefert, erstellen wir zwei Collections. Die erste heißt **colOffen** und speichert alle Tabellen, die noch nicht in die Reihenfolge eingearbeitet wurden. Die zweite heißt **colReihenfolge** und enthält die Tabellen in der ermittelten Reihenfolge.

Um die Collection **colOffen** zu füllen, erstellen wir ein Recordset namens **rstTabellen** auf Basis der Abfrage **SHOW TABLES**. Dieses durchlaufen wir und tragen den Namen der Tabelle für jeden Datensatz in die Collection **colOffen** ein.

Dann verwenden wir die weiter oben vorgestellte SQL-Abfrage zum Ermitteln aller Beziehungen der Datenbank, um ein weiteres Recordset namens **rstBeziehungen** zu füllen.

```
Public Function TabellenreihenfolgeKopieren(strVerbindungszeichenfolge As String) As Collection
    Dim colOffen As Collection, colReihenfolge As Collection, bolIstDetailtabelle As Boolean
    Dim varOffen As Variant, varReihenfolge As Variant, intCountVorher As Integer
    Dim rstTabellen As ADODB.Recordset, rstBeziehungen As ADODB.Recordset, strSQL As String
    Set colOffen = New Collection
    Set colReihenfolge = New Collection
    Set rstTabellen = GetRecordset(GetConnection(strVerbindungszeichenfolge), "SHOW TABLES")
    Do While Not rstTabellen.EOF
        colOffen.Add rstTabellen.Fields(0).Value, rstTabellen.Fields(0).Value
        rstTabellen.MoveNext
    Loop
    strSQL = "SELECT TABLE_NAME, COLUMN_NAME, REFERENCED_TABLE_NAME, REFERENCED_COLUMN_NAME " & vbCrLf _
        & "FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE" & vbCrLf & "WHERE REFERENCED_TABLE_NAME IS NOT NULL;"
    Set rstBeziehungen = GetRecordset(GetConnection(cStrVerbindungszeichenfolgeAlt), strSQL)
    Do While colOffen.Count > 0
        If Not intCountVorher = colOffen.Count Then
            intCountVorher = colOffen.Count
            For Each varOffen In colOffen
                bolIstDetailtabelle = False
                rstBeziehungen.MoveFirst
                Do While Not rstBeziehungen.EOF
                    If rstBeziehungen!TABLE_NAME = varOffen Then
                        bolIstDetailtabelle = True
                        For Each varReihenfolge In colReihenfolge
                            If rstBeziehungen!REFERENCED_TABLE_NAME = varReihenfolge Then
                                bolIstDetailtabelle = False
                                Exit For
                            End If
                        Next varReihenfolge
                    End If
                    rstBeziehungen.MoveNext
                Loop
                If bolIstDetailtabelle = False Then
                    colReihenfolge.Add varOffen, varOffen
                    colOffen.Remove varOffen
                End If
                DoEvents
            Next varOffen
        Else
            Exit Do
        End If
    Loop
    MsgBox "Anzahl offener Tabellen (da Zirkelbezug): " & colOffen.Count & vbCrLf _
        & "Die Tabellen werden an die Collection angehängt."
    For Each varOffen In colOffen
        colReihenfolge.Add varOffen, varOffen
    Next varOffen
    Set TabellenreihenfolgeKopieren = colReihenfolge
End Function
```

Listing 2: Ermitteln der Reihenfolge für das Kopieren von Daten

Von Version zu Version

Neulich wollte ich meine Shopsoftware aktualisieren. Allerdings ging dies nicht mit dem vom Hersteller dafür bereitgestellten Plug-In – zumindest nicht mit allen Daten, die ich von der alten in die neue Version überführen wollte. Also musste ich manuell ermitteln, welche Daten der alten Version benötigt werden, damit die neue Version läuft. Damit erhielt ich recht schnell eine lauffähige Version des neuen Systems. Allerdings kam mir dann etwas dazwischen, wodurch zum alten, noch aktiven System wieder neue Kunden und Bestellungen hinzukamen – die Arbeit war also umsonst. Um das Übertragen der Daten beim nächsten Mal einfacher zu gestalten, wollte ich nun das alte und das neue Datenmodell nun automatisch analysieren und die SQL-Befehle erstellen lassen, um diese danach ebenfalls automatisch ausführen zu können. Wie das gelingt, zeigt der vorliegende Beitrag.

Bei der Shopsoftware handelt es sich um ein Produkt namens **Shopware**. Es verwendet eine MySQL-Datenbank.

Die hier vorgestellten Techniken lassen sich jedoch auch auf andere Systeme übertragen – und auch auf andere Anwendungen.

Auch MySQL als Datenbanksystem kann durch andere Systeme ersetzt werden – Sie müssen dann schlicht den Treiber in der Verbindungszeichenfolge austauschen und

gegebenenfalls ein paar Abfragen auf spezifische Eigenarten des jeweiligen Dialekts anpassen. Voraussetzung für die Nutzung der hier vorgestellten Technik ist der Zugriff auf die Datenbank. Ob dies möglich ist, müssen Sie mit dem jeweiligen Provider klären.

Zugriff per ADODB

Damit wir per ODBC auf die Daten der beiden zu vergleichenden MySQL-Datenbanken zugreifen können, fügen wir der Datenbank einen Verweis auf die Bibliothek **Microsoft ActiveX Data Objects 2.8 Library** zum VBA-Projekt einer neuen Access-Datenbank hinzu (siehe Bild 1).

Tabellen abgleichen

Als Erstes wollen wir die Tabellen der beiden Datenbanken abgleichen, also prüfen, ob es in der ersten Datenbank Tabellen gibt, die nicht in der zweiten Datenbank vorkommen und umgekehrt.

Dazu legen wir zwei Konstanten mit den Verbindungszeichenfolgen der beiden Datenbanken in einem neuen Modul namens **mdlADODB** an:

```
Const cStrVerbindungszeichenfolgeAlt As String =  
"DRIVER={MySQL ODBC 5.3 ANSI Driver};SERVER=xxx.xxx.xxx.xx
```

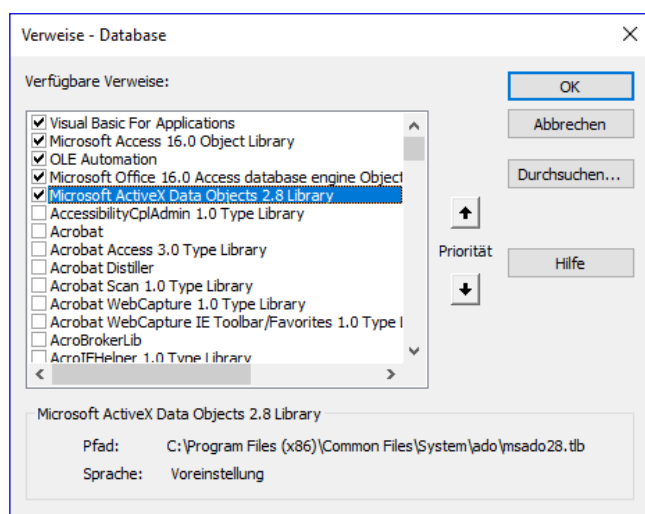


Bild 1: Verweis auf die ADODB-Bibliothek

```
x;DATABASE=<Datenbankname>;
UID=<Benutzername>;PWD=<Kennwort>"
Const cStrVerbindungszeichenfolgeNeu As String =
"DRIVER={MySQL ODBC 5.3 ANSI Driver};SERVER=xxx.xxx.xxx;DATABASE=<Datenbankname>;UID=<Benutzername>;PWD=<Kennwort>"
```

Die Funktion **GetConnection** erwartet die Angabe der zu verwendenden Verbindungszeichenfolge als Parameter. Sie erstellt ein neues Objekt des Typs **ADODB.Connection** und öffnet diese unter Angabe der Verbindungszeichenfolge (siehe Listing 1).

```
Public Function GetConnection(strConnection As String) As ADODB.Connection
    Dim objConnection As ADODB.Connection
    Set objConnection = New ADODB.Connection
    objConnection.Open strConnection
    Set GetConnection = objConnection
End Function
```

Listing 1: Die Funktion **GetConnection**

```
Public Function GetRecordset(objConnection As ADODB.Connection, strSQL As String) _
    As ADODB.Recordset
    Dim rst As ADODB.Recordset
    Set rst = New ADODB.Recordset
    With rst
        .ActiveConnection = objConnection
        .CursorLocation = adUseClient
        .CursorType = adOpenDynamic
        .LockType = adLockBatchOptimistic
        .Source = strSQL
        .Open , , , , adCmdText
    Set GetRecordset = rst
    End With
End Function
```

Listing 2: Die Funktion **GetRecordset**

Die Funktion **GetRecordset** erwartet ein **ADODB.Connection**-Objekt, das wir zuvor beispielsweise mit der Funktion **GetConnection** ermitteln, sowie eine SQL-Anweisung mit der zu verwendenden Abfrage (siehe Listing 2). Die Funktion erstellt ein neues **ADODB.Recordset**-Objekt und stellt seine Eigenschaften ein, darunter auch die SQL-Abfrage für die Eigenschaft **Source**. Das Ergebnis wird dann als **ADODB.Recordset**-Objekt zurückgegeben.

Danach nutzen wir diese beiden Funktionen in einer neuen Prozedur namens **TabellenAbgleichen** (siehe Listing 3). Diese Prozedur verwendet zwei **ADODB.Recordset**-Objekte namens **rstAlt** und **rstNeu**, um die Namen der Tabellen der alten und der neuen Version der Datenbank zu speichern. Diese füllen wir jeweils mit einem Aufruf der Funktion **GetRecordset**, wobei wir einmal die Verbindungszeichenfolge aus **cStrVerbindungszeichenfolgeAlt** und einmal die aus **cStrVerbindungszeichenfolgeNeu** übergeben. In beiden Fällen geben wir als SQL-Anweisung

keine einfache **SELECT**-Anweisung an, sondern die Anweisung **Show Tables**. Dies ist eine MySQL-spezifische Anweisung, die alle Tabellen einer Datenbank liefert. Danach sortieren wir die Inhalte der beiden Recordsets **rstAlt** und **rstNeu** aufsteigend nach dem Inhalt des einzigen Feldes.

Schließlich durchlaufen wir das Recordset **rstAlt** in einer **Do While**-Schleife, bis die Datensatzmarkierung des auf **EOF** steht. In dieser Schleife durchlaufen wir auf bestimmte Weise beide Recordsets, wobei gegebenenfalls noch ein oder mehrere Datensätze im Recordset **rstNeu** übrig bleiben.

Dabei prüfen wir zunächst, ob **rstNeu.EOF** wahr ist. Das ist zu Beginn in der Regel noch nicht der Fall – außer, die neue Datenbank enthält gar keine Tabellen. Anderenfalls prüfen wir, ob der aktuelle Datensatz von **rstAlt** und **rstNeu** den gleichen Wert aufweisen, was darauf hindeutet,

```
Public Sub TabellenAbgleichen()  
    Dim rstAlt As ADODB.Recordset  
    Dim rstNeu As ADODB.Recordset  
    Dim strKriterium As String  
    Dim fld As ADODB.Field  
    Set rstAlt = GetRecordset(GetConnection(cStrVerbindungszeichenfolgeAlt), "SHOW TABLES")  
    Set rstNeu = GetRecordset(GetConnection(cStrVerbindungszeichenfolgeNeu), "SHOW TABLES")  
    rstAlt.Sort = rstAlt.Fields(0).Name & " ASC"  
    rstNeu.Sort = rstNeu.Fields(0).Name & " ASC"  
    Do While Not rstAlt.EOF  
        If Not rstNeu.EOF Then  
            If rstAlt.Fields(0) = rstNeu.Fields(0) Then  
                Debug.Print rstAlt.Fields(0), rstNeu.Fields(0)  
                rstAlt.MoveNext  
                rstNeu.MoveNext  
            ElseIf rstAlt.Fields(0) > rstNeu.Fields(0) Then  
                Debug.Print "----", rstNeu.Fields(0)  
                rstNeu.MoveNext  
            ElseIf rstAlt.Fields(0) < rstNeu.Fields(0) Then  
                Debug.Print rstAlt.Fields(0), "----"  
                rstAlt.MoveNext  
            End If  
        Else  
            Debug.Print rstAlt.Fields(0), "----"  
            rstAlt.MoveNext  
        End If  
    Loop  
    Do While Not rstNeu.EOF  
        Debug.Print "----", rstNeu.Fields(0)  
        rstNeu.MoveNext  
    Loop  
End Sub
```

Listing 3: Die Prozedur **TabellenAbgleichen**

dass die Tabelle in beiden Datenbanken vorhanden ist. In diesem Fall geben wir im Direktbereich den Namen der Tabelle zwei Mal aus und verschieben die Datensatzzeiger beider Recordsets mit **MoveNext** zur nächsten Position. Ist **rstAlt.Fields(0) > rstNeu.Fields(0)**, dann stimmen die aktuellen Tabellennamen nicht überein und der Tabellename aus **rstNeu** liegt im Alphabet vor dem aus **rstAlt**. In diesem Fall geben wir eine Zeichenkette aus vier Minuszeichen (----) aus und den Namen der Tabelle aus **rstNeu**. Außerdem verschieben wir den Datensatzzeiger von **rstNeu** um eine Position weiter.

Das heißt, wir bleiben bei dem Tabellennamen von **rstAlt** und wechseln zum nächsten Eintrag von **rstNeu**. Die nächst **ElseIf**-Bedingung behandelt den umgekehrten Fall, nämlich dass der Eintrag aus **rstAlt** im Alphabet vor dem Eintrag aus **rstNeu** liegt. Dann geben wir den aus **rstAlt** aus und vier Minuszeichen statt des Eintrags aus **rstNeu** und verschieben den Datensatzzeiger in **rstAlt** um eine Position weiter nach hinten.

Wenn also etwa in beiden Datenbanken **tblA** und **tblB** vorhanden sind, werden diese so ausgegeben:

tb1A tb1A
tb1B tb1B

Wenn in **tb1A** nur in **rstNeu** vorhanden ist, erhalten wir dieses Ergebnis:

---- tb1A
tb1B tb1B

Ist **tb1A** in beiden Datenbanken vorhanden und **tb1B** nur in der alten, sieht das Ergebnis so aus:

tb1A tb1A
tb1B ----

Nun gibt es drei Fälle für das Ende des Durchlaufens der Datensätze mit den Tabellennamen in der ersten **Do While**-Schleife:

- Die letzte Tabelle ist in beiden Datenbanken enthalten. Dann wird die erste **Do While**-Schleife mit dem **If**-Teil der inneren **If...Then**-Bedingung verlassen, weil **NOT rstAlt.EOF** danach nicht mehr erfüllt ist. **rstNeu.EOF** ist dann auch **True**.
- Die letzte Tabelle beziehungsweise die letzten Tabellen sind nur in **rstAlt** enthalten. Dann ist die äußere **If...Then**-Schleife **False** und der **Else**-Teil wird ausgeführt, wo der Datensatzzeiger in **rstAlt** solange weitergeschoben wird, bis die **Do While**-Bedingung **rstAlt.EOF** erfüllt ist.
- Die letzte Tabelle beziehungsweise die letzten Tabellen sind nur in **rstNeu** enthalten. Dann wird die erste **Do While**-Schleife verlassen, nachdem der Datensatzzeiger für **rstAlt** auf **EOF** verschoben wird.

In allen Fällen wird die Bedingung der zweiten **Do While**-Schleife noch einmal geprüft, nämlich **Not rstNeu.EOF**. Sollten also nach dem Durchlaufen von **rstAlt** noch Datensätze in **rstNeu** vorhanden sein, durchläuft die Prozedur

diese in der zweiten **Do While**-Schleife. Dort gibt sie für die übrigen Tabellen der neuen Version der Datenbank die Tabellennamen aus, also etwa so:

...
---- tb1Y
---- tb1Z

Interpretation des Ergebnisses

Damit erhalten wir nun eine Gegenüberstellung der Tabellen der alten und der neuen Version der Datenbank, die etwa so aussehen könnte:

tb1A tb1A
tb1B tb1B
---- tb1C
tb1D ----
...
tb1Y tb1Y
tb1Z tb1Z

Die Tabelle **tb1C** kommt nicht in der alten Datenbank vor, aber in der neuen Version. **tb1D** hingegen kommt nur in der alten, aber nicht in der neuen Version vor. Alle anderen Tabellen sind in beiden Versionen der Datenbank vorhanden.

Die Interpretation könnte wie folgt lauten:

- Eine Tabelle kommt in der alten Version vor, aber nicht mehr in der neuen und umgekehrt: Die Tabelle wurde schlicht unbenannt, daher wird sie nicht mehr als gleiche Tabelle erkannt.
- Eine Tabelle kommt in der alten Version vor, aber nicht mehr in der neuen und eine Umbenennung ist ausgeschlossen: Die Funktion, für die die Daten der Tabelle benötigt wurden, ist gegebenenfalls weggefallen. Oder, in meinem Beispiel mit der neuen Version des Shops geschehen: Die alte Version des Shops enthielt Plugins, die der neuen Version noch nicht hinzugefügt wurden,

daher waren die entsprechenden Tabellen dort auch noch nicht enthalten. Sie können also so gegebenenfalls erkennen, dass Sie noch Plugins hinzufügen müssen, damit der neue Shop wie der alte funktioniert.

- Eine Tabelle kommt in der neuen Version der Datenbank vor, aber nicht in der alten, und eine Umbenennung ist ausgeschlossen: Dann handelt es sich wohl um eine Tabelle, die im Rahmen einer neuen Funktion der neuen Version der Software benötigt wird.

Felder vergleichen

Wenn wir nun die Daten aus der alten Version der Datenbank in die neue Version übertragen wollen, könnten wir theoretisch für jede Tabelle, die in der alten und in der neuen Version vorhanden ist, folgende Anweisung schreiben (vorausgesetzt, die Datenbanken befinden sich auf dem gleichen MySQL-Server):

```
INSERT INTO <Neue Datenbank>.<Tabellenname> SELECT * FROM  
<Alte Datenbank>.<Tabellenname>
```

Das klappt aber auch nur, wenn die Struktur der Tabelle in der alten und der neuen Version der Datenbank identisch ist, also wenn die Tabellen die gleichen Felder enthalten. Anderenfalls gibt es eine Fehlermeldung.

In diesem Fall müssen wir untersuchen, welche der Felder der Tabelle der alten Version der Datenbank in der Tabelle der neuen Version der Datenbank vorkommen und diese statt des Platzhalters Sternchen (*) in die **SELECT...INTO**-Anweisung einfügen.

Gemeinsame Felder ermitteln

Um diese gemeinsamen Felder zu ermitteln, fügen wir dem **If**-Zweig der inneren **If...Then**-Bedingung den Aufruf der Funktion **FelderAbgleichen** mit dem Namen der Tabelle als Parameter hinzu. Das Ergebnis der Aufrufe dieser Funktion fügen wir in der Variablen **strSQLInsert** zusammen. Auf die gleiche Weise fügen wir in der Variablen **strSQLDelete** jeweils eine **DELETE**-Anweisung für

die betroffenen Tabellen der neuen Version der Datenbank zusammen:

```
Public Sub TabellenAbgleichen()  
...  
Dim strSQLDelete As String  
Dim strSQLInsert As String  
...  
Do While Not rstAlt.EOF  
    If Not rstNeu.EOF Then  
        If rstAlt.Fields(0) = rstNeu.Fields(0) Then  
            Debug.Print rstAlt.Fields(0), "  
                rstNeu.Fields(0)  
            strSQLDelete = strSQLDelete _  
                & "DELETE FROM " & cStrNeueDatenbank _  
                & "." & rstAlt.Fields(0) & vbCrLf  
            strSQLInsert = strSQLInsert _  
                & FelderAbgleichen(rstAlt.Fields(0)) _  
                & vbCrLf  
            ...  
        End If  
    End If  
    rstAlt.MoveNext  
    rstNeu.MoveNext  
End Do
```

Die Funktion **FelderAbgleichen** finden Sie in Listing 4. Sie erwartet den Namen der zu untersuchenden Tabelle als Parameter. Sie ist grundsätzlich so aufgebaut wie die Prozedur **TabellenAbgleichen**, nur dass sie diesmal nicht die Tabellen der beiden Datenbanken, sondern die Felder der beiden Tabellen untersucht.

Daher verwenden wir wieder zwei Recordsets, welche die Verbindungszeichenfolgen aus den Konstanten nutzen. Als Datenquelle verwenden wir diesmal wieder keine klassische **SELECT**-Abfrage, sondern die Anweisung **DESCRIBE** mit den Namen der zu beschreibenden Tabelle. Dies liefert die Namen der Felder der angegebenen Tabelle zurück – neben einigen anderen Feldern. Uns interessiert aber nur der Feldname, den wir im Feld **Field** finden.

Deshalb sortieren wir die Datensätze beider Recordsets zuerst aufsteigend nach dem Inhalt des Feldes **Field**. Dann durchlaufen wir in einer **Do While**-Schleife die Datensätze des Recordsets **rstAlt**. Solange der Datensatz-

Ticketsystem, Teil V

In den vorherigen Teilen dieser Beitragsreihe haben wir den Aufbau einiger Funktionen eines Ticketsystems beschrieben. Es fehlt noch der letzte Feinschliff: Wir wollen die erneuten Antworten von Kunden auf unsere als Antwort versendeten E-Mails automatisch in die Ticketverwaltung aufnehmen. Bevor wir das im nächsten Teil erledigen können, haben wir in diesem Teil noch einige Feinheiten ergänzt und werfen außerdem noch einmal einen zusammenfassenden Blick über die bisher programmierten Funktionen und die Inbetriebnahme der Ticketverwaltung inklusive der Einrichtung in Outlook. Außerdem fügen wir noch einen Dialog zur Verwaltung der Optionen der Lösung hinzu.

Einrichtung des Ticketsystems

Um das Ticketsystem einzurichten, benötigen Sie zunächst die Access-Datenbank aus dem Download zu diesem Beitrag namens **Ticketsystem.accdb**. Außerdem sind kleine Anpassungen an Outlook erforderlich, genau genommen am VBA-Projekt von Outlook.

Einrichtung in Outlook

Es gibt einen kurzen und einen etwas umständlicheren Weg, die für das Ticketsystem nötigen Elemente in Outlook zu integrieren. Dabei brauchen wir einige Module. Diese können Sie einzeln zum aktuellen VBA-Projekt von Outlook hinzufügen. Das bietet sich an, wenn Sie bereits Änderungen am VBA-Projekt von Outlook vorgenommen haben. Wenn Sie jedoch eine jungfräuliche Instanz von Outlook nutzen, können Sie auch das komplette VBA-Projekt austauschen gegen das aus dem Download zu diesem Beitrag.

Outlook-VBA-Projekt austauschen

Wenn Sie das komplette VBA-Projekt für die Ticketverwaltung nutzen wollen, gehen Sie wie folgt vor:

- Navigieren Sie zur Datei **VbaProject.OTM**. Diese finden Sie beispielsweise im Ordner **C:\Users\User\AppData\Roaming\Microsoft\Outlook**.
- Benennen Sie die Datei **VbaProject.OTM** um, beispielsweise in **_VbaProject.OTM**.

- Fügen Sie dann die Datei **VbaProject.OTM** aus dem Download zu diesem Beitrag in das angegebene Verzeichnis ein.
- Anschließend starten Sie Outlook und fahren wie weiter unten unter **Outlook starten** beschrieben fort.

Module einzeln zu bestehendem VBA-Projekt hinzufügen

Wenn Sie Outlook starten und dann die Tastenkombination **Alt + F11** drücken, öffnet sich der VBA-Editor von Outlook. In diesem finden Sie, wenn der Projekt-Explorer geöffnet ist, ein Projekt mit einem Unterordner **Microsoft Outlook Objekte**. Dieser Ordner stellt das Klassenmodul **This- OutlookSession** bereit. In diesem legen Sie eine neue Prozedur an, die wie in Bild 1 aussieht. Das erledigen Sie am einfachsten, indem Sie aus dem linken Kombinationsfeld des Codefensters den Eintrag **Application** und aus dem rechten **Startup** auswählen.

Die einzige Anweisung dieser Prozedur soll eine weitere Prozedur namens **Application_Startup_Ticketverwaltung** aufrufen. Diese haben wir der Zip-Datei mit den Beispieldateien unter dem Namen **mdlTicketverwaltung.bas** untergebracht. Kopieren Sie dieses Modul aus der Zip-Datei mit den Beispieldateien in das VBA-Projekt von Outlook, damit die Prozedur **Application_Startup_Ticketverwaltung** beim Start von Outlook aufgerufen werden kann.

Sie benötigen noch weitere Module im VBA-Projekt von Outlook, und zwar **mdlTicketverwaltung.bas**, **mdlTicketverwaltung_Folders.bas**, **mdlTicketverwaltung_Global.bas** und **mdlTicketverwaltung_Outlook.bas**.

Schließlich benötigen Sie noch das Klassenmodul **clsFolderArchiv.cls**, das Sie in der Zip-Datei mit den Beispieldateien finden. Sie können dieses ebenso wie die anderen Module einfach aus dem Windows Explorer in den Projekt-Explorer des VBA-Editors von Outlook ziehen.

Verweise zum VBA-Projekt von Outlook hinzufügen

Im VBA-Editor stellen Sie nun noch zwei Verweise ein, und zwar für die Bibliotheken **Microsoft Access 16.0 Object Library** und **Microsoft Office 16.0 Access Database Engine Object Library**.

Danach können Sie Outlook einmal neu starten. Es erscheint dann direkt der Dateiauswahl-Dialog namens **Datenbankpfad auswählen**. Hier wählen Sie die Datenbank **Ticket-system.accdb** aus.

Outlook starten

Wenn Sie Outlook nun starten, wird die Ereignismethode **Application_Startup**

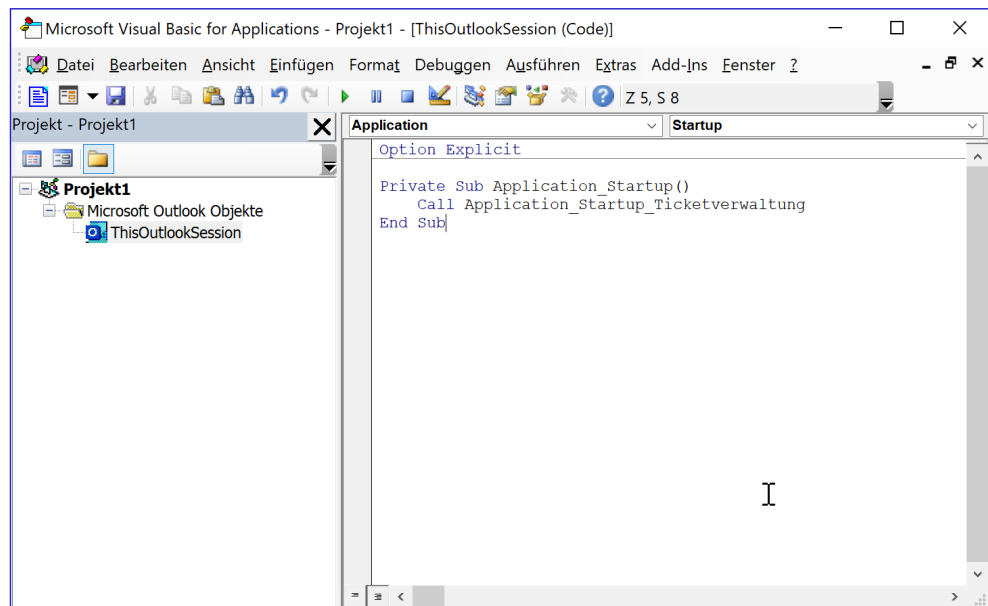


Bild 1: Anlegen der Prozedur, die beim Start von Outlook ausgelöst wird

ausgelöst. Diese ruft die Prozedur **Application_Startup_Ticketverwaltung** auf, die mit der Funktion **DatenbankpfadHolen** prüft, ob der Pfad zur Datenbank bereits in einer UserProperty innerhalb von Outlook gespeichert ist – und ob die angegebene Datei vorhanden ist. Falls nicht, erscheint ein Dateiauswahl-Dialog, mit dem der Benutzer die Datei **Ticketssystem.accdb** auswählen kann (siehe

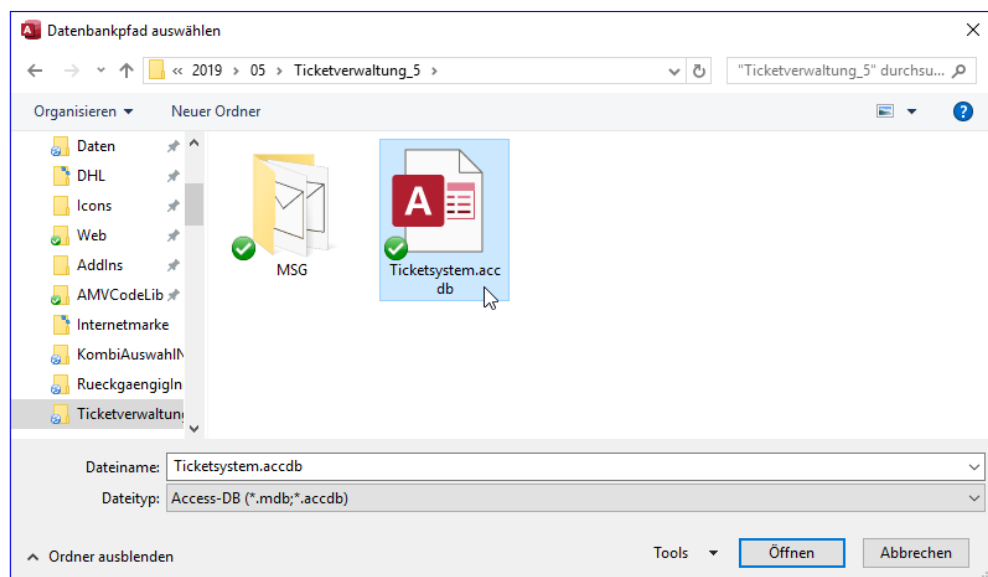


Bild 2: Auswählen der Datei **Ticketssystem.accdb**

Bild 2). In diesem Fall wird der Pfad zur Ticketmanager-Datenbank in der **UserProperty** gespeichert, wo er beim nächsten Start von Outlook wieder ausgelesen werden kann.

Die Prozedur **Application_Startup_Ticketverwaltung** öffnet nun die Datenbank und liest den Inhalt der Tabelle **tblOptionen** ein, die wir weiter unten noch beschreiben.

Die Tabelle enthält im Wesentlichen Informationen darüber, welche Verzeichnisse in Outlook zum Sammeln der Kunden-E-Mails zur Bearbeitung im Ticketsystem verwendet werden sollen.

Hier sind beim ersten Aufruf in der Regel noch keine Outlook-Verzeichnisse festgelegt und somit auch noch keine Datensätze in der Tabelle **tblOptionen**. Dementsprechend bietet die Prozedur dem Benutzer nun die Möglichkeit, einen Outlook-Ordner auszuwählen und diesen zur Tabelle **tblOptionen** hinzuzufügen. Damit der Benutzer weiß, was auf ihn zukommt, erscheint nun zunächst die Meldung aus Bild 5.

Anschließend erscheint der Dialog **Ordner auswählen** von Outlook. Dieser zeigt alle in Outlook verfügbaren Ordner an. Wir wählen hier den dafür vorbereiteten Ordner **Ticketsystem** aus (siehe Bild 3). Danach wird Outlook wie gewohnt gestartet und auch bei nachfolgenden Startvorgängen

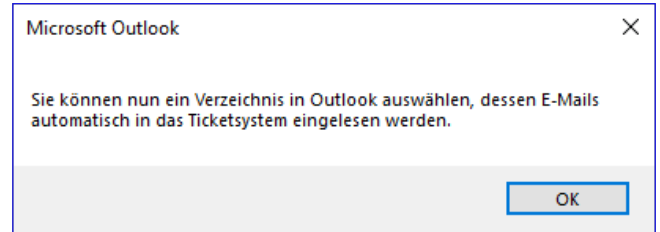


Bild 5: Hinweis auf das anstehende Auswählen eines Verzeichnisses in Outlook

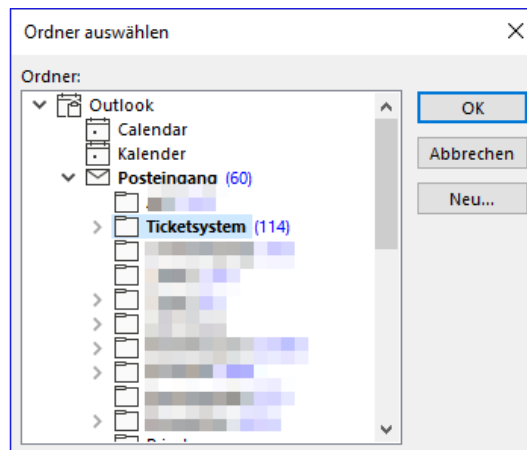


Bild 3: Auswahl des zu trackenden Outlook-Ordners

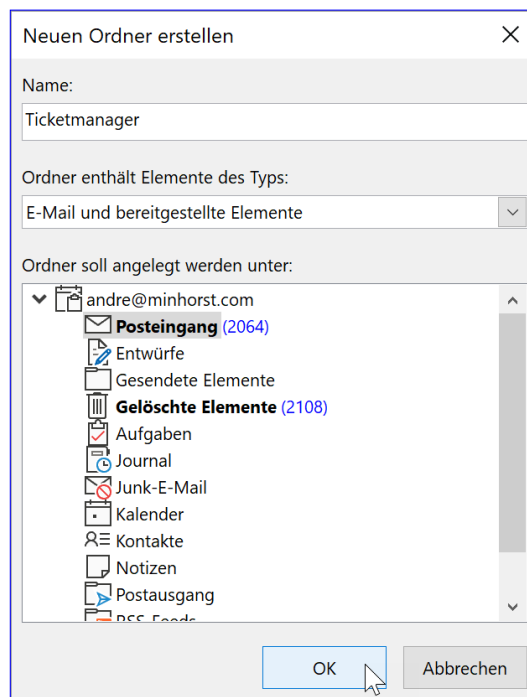


Bild 4: Anlegen eines neuen Ordners in Outlook

müssen die Informationen nicht erneut eingegeben werden.

Sollten Sie den Ordner für die E-Mails für die Ticketverwaltung noch nicht angelegt haben, können Sie das nachträglich erledigen – und zwar direkt im Dialog **Ordner auswählen** von Outlook. Dazu klicken Sie dort auf die Schaltfläche **Neu...** und legen dann mit dem Dialog aus Bild 4 einen neuen Outlook-Ordner für E-Mails an.

Ändern des Ordners in Outlook oder in der Datenbank

Es kann sein, dass Sie den Ordner in Outlook umbenennen oder dass der Wert des Feldes Verzeichnis in der Tabelle **tblOptionen** so geändert werden, dass beide nicht mehr übereinstimmen. In diesem Fall findet die Prozedur **Application_Startup_Ticketverwaltung** den Outlook-Ordner nicht und zeigt die Meldung aus Bild 6 an.

Diese Funktion war in der ersten Version der Prozedur **Application_Startup_Ticketver-**

waltung noch nicht enthalten. Die erweiterte Version finden Sie in Listing 1.

Zur Erinnerung: Die Prozedur ermittelt zunächst über die Funktion **DatenbankpfadHolen** den aktuell in einer **UserProperty** in Outlook gespeicherten Pfad der Datenbank, also in unserem Fall der Datei **Ticketverwaltung.accdb**. Wenn dies eine leere Zeichenkette liefert, wird eine Meldung angezeigt, dass die Verbindung zur Ticketverwaltung nicht hergestellt werden konnte und die Prozedur beendet.

Anderenfalls füllt die Prozedur eine **Database**-Variable mit einem Verweis auf die angegebene Datenbank, und zwar über die **OpenDatabase**-Methode. Dann öffnet sie ein Recordset auf Basis der Tabelle **tblOptionen** und referenziert dieses mit der Variablen **rst**.

Die Tabelle enthält keinen, einen oder mehrere Einträge mit Angabe von Outlook-Verzeichnissen, in die der Benutzer Kunden-E-Mails verschieben kann, damit auf ihrer Basis ein Ticket in der Ticketverwaltung angelegt wird. Hier prüft die Prozedur zunächst, ob überhaupt ein Datensatz vorliegt (**rst.EOF**).

Ist das der Fall, erscheint die Meldung, dass nun ein Outlook-Verzeichnis ausgewählt werden kann, das als Auffangbehälter für die Kunden-E-Mails dienen soll. Dieses Verzeichnis wird dann mit dem über die **PickFolder**-Methode geöffneten Dialog **Ordner auswählen** selektiert. Hat der Benutzer einen Ordner ausgewählt, wird ein neuer Datensatz in der Tabelle **tblOptionen** angelegt und mit dem Verzeichnisnamen aus Outlook gefüllt. Außerdem wird das **Folder**-Objekt der Collection aus **colFolders** hinzugefügt.

Sollte **rst.EOF** den Wert **False** zurückliefern, enthält die Tabelle **tblOptionen** mindestens einen Datensatz. In diesem Fall durchläuft die Prozedur in einer **Do While**-Schleife alle Einträge der Tabelle und versucht jeweils

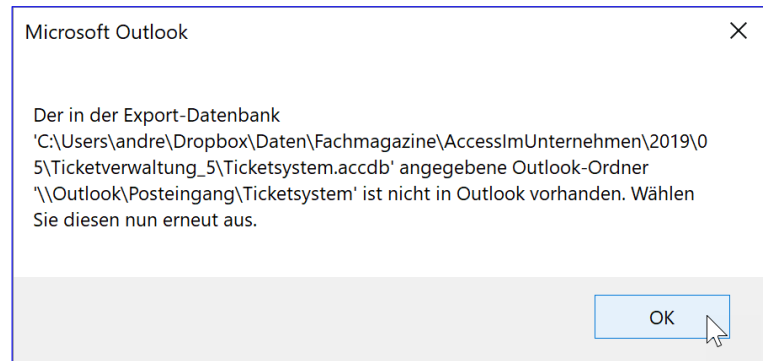


Bild 6: Es fehlt ein Ordner, der vom Ticketsystem getrackt werden soll.

über die Funktion **GetFolderByPath** ein **Folder**-Objekt auf Basis der Pfadangabe zu erhalten – also etwa über **\\Outlook\Posteingang\Ticketssystem**. Eingebettet sind diese Anweisungen in das Erstellen eines neuen Objekts auf Basis der Klasse **clsFolderArchiv**, dem das **Folder**-Objekt und die Datenbank als Eigenschaften zugewiesen werden. Diese Klasse enthält die Automatismen, die dafür sorgen, dass eine in ein damit referenziertes **Folder**-Objekt eingefügten Mails automatisch in die Ticketverwaltungs-Datenbank eingetragen und auf spezielle Art markiert werden – nämlich durch Voranstellen der Zeichenfolge **[Ticket x]**, wobei **x** der Nummer des Datensatzes entspricht.

Kann kein **Folder**-Objekt zu diesem Verzeichnis gefunden werden, erscheint die Meldung, dass der Outlook-Ordner nicht vorhanden ist und dass der Benutzer einen neuen Ordner auswählen soll. Dies geschieht dann wiederum über den Dialog **Ordner auswählen**.

Der so ausgewählte Ordner wird dann in das Feld **Verzeichnis** des aktuellen Datensatzes eingetragen. Außerdem werden auch hier das **Folder**-Objekt und die Referenz auf die Datenbank in ein neu erstelltes Objekt des Typs **clsFolderArchiv** eingetragen. Dieser Vorgang kann auch für mehr als einen Outlook-Ordner erfolgen, sofern dieser in der Tabelle **tblOptionen** eingetragen ist. Hier werden dann auch noch weitere Eigenschaften zum neu erstellten Objekt **objFolderArchiv** hinzugefügt, die in der nachfolgend beschriebenen Tabelle **tblOptionen** gespeichert werden.

```

Public Sub Application_Startup_Ticketverwaltung()
    Dim db As DAO.Database, rst As DAO.Recordset
    Dim objFolder As Outlook.Folder, objFolderArchiv As clsFolderArchiv, strTicketssystemDatenbank As String
    strTicketssystemDatenbank = DatenbankpfadHolen("Ticketssystem", "Datenbankpfad")
    If Len(strTicketssystemDatenbank) = 0 Then
        MsgBox "Verbindung zur Ticketdatenbank konnte nicht hergestellt werden."
        Exit Sub
    End If
    Set db = DBEngine.OpenDatabase(strTicketssystemDatenbank, , True)
    Set rst = db.OpenRecordset("SELECT * FROM tblOptionen", dbOpenDynaset)
    Set colFolders = New Collection
    If rst.EOF Then
        MsgBox "Sie können nun ein Verzeichnis in Outlook auswählen, dessen E-Mails automatisch in das " _
            & "Ticketssystem eingelesen werden."
        Set objFolderArchiv = New clsFolderArchiv
        With objFolderArchiv
            Set objFolder = Outlook.GetNamespace("MAPI").PickFolder
            If Not objFolder Is Nothing Then
                rst.AddNew
                rst!Verzeichnis = objFolder.FolderPath
                rst.Update
                Set .Folder = objFolder
                Set .Database = db
                colFolders.Add objFolderArchiv
            End If
        End With
    Else
        Do While Not rst.EOF
            Set objFolderArchiv = New clsFolderArchiv
            With objFolderArchiv
                Set objFolder = GetFolderByPath(rst!Verzeichnis)
                If objFolder Is Nothing Then
                    MsgBox "Der in der Export-Datenbank '" & strTicketssystemDatenbank & "' angegebene Outlook-Ordner '" _
                        & rst!Verzeichnis & "' ist nicht in Outlook vorhanden. Wählen Sie diesen nun erneut aus."
                    Set objFolder = Outlook.GetNamespace("MAPI").PickFolder
                    rst.Edit
                    Verzeichnis = objFolder.FolderPath
                    rst.Update
                End If
                Set .Folder = objFolder
                .AnlagenSpeichern = rst!AnlagenSpeichern
                Set .Database = db
                .NeuEinlesen = rst!NeuEinlesen
                .Groesse = Nz(rst!Groesse)
            End With
            colFolders.Add objFolderArchiv
            If rst!Rekursiv Then
                UnterordnerInstanzieren objFolder, db, Nz(rst!Groesse), rst!NeuEinlesen, rst!AnlagenSpeichern, colFolders
            End If
            rst.MoveNext
        Loop
    End If
End Sub

```

Listing 1: Die Prozedur **Application_Startup_Ticketverwaltung**

AutoComplete schnell zu Textfeld hinzufügen

Im Beitrag »AutoComplete in Textfeldern« in der vorherigen Ausgabe von Access im Unternehmen haben wir eine Lösung vorgestellt, mit der Sie Textfelder um eine AutoComplete-Funktion erweitern können. Dort haben wir uns die grundlegenden Techniken angesehen, im vorliegenden Beitrag kümmern wir uns darum, dieses Tool in eine Form zu bringen, die wir einfach und schnell einem Textfeld hinzufügen können. Und das, ohne den Code für jedes Textfeld erneut in die Klassenmodule der Formulare kopieren zu müssen.

Autocomplete-Klasse

Wenn wir eine nützliche Funktion herauskristallisiert haben, nutzen wir in der Regel ein Klassenmodul, um den entstandenen Code zusammenzufassen und diesen wiederverwendbar zu machen, ohne dass er jedes Mal in die jeweiligen Module hineinkopiert werden muss. Das Klassenmodul muss dann ein paar private Variablen bieten, die über öffentliche Eigenschaften gefüllt oder gelesen werden können und welche die Informationen für den Ablauf der Funktion bereitstellen.

Also erstellen wir zunächst ein neues Klassenmodul namens **clsAutoComplete**.

Für die Umsetzung der AutoComplete-Funktion in der Klasse benötigen wir drei Informationen:

- Einen Verweis auf das Textfeld, welches wir mit der AutoComplete-Funktion ausstatten wollen,
- den Namen der Tabelle, aus der die Daten für **AutoComplete** stammen und
- den Namen des Feldes, welches diese Daten enthält.

Dafür fügen wir der Klasse zunächst die folgenden privat deklarierten Variablen hinzu:

```
Private WithEvents m_Textbox As TextBox
Private m_Table As String
```

```
Private m_Field As String
```

Die Objektvariable **m_Textbox** versehen wir mit dem Schlüsselwort **WithEvents**. Damit legen wir die Grundlage dafür, dass wir innerhalb des Klassenmoduls auch Ereignisprozeduren für das Textfeld implementieren können.

Öffentliche Eigenschaften

Damit wir etwa bei Öffnen des Formulars mit dem auszustattenden Textfeld die notwendigen Informationen an das neu erstellte Objekt auf Basis dieser Klasse übermitteln können, benötigen wir drei entsprechende öffentliche Eigenschaften. Diese bilden wir mit **Property Set**- beziehungsweise **Property Let**-Prozeduren ab. Erstere sind für Objektvariablen geeignet, zweitere für einfache Variablen wie Text- oder Zahlenfelder. Die Tabelle und das Feld erfassen wir mit solchen **Property Let**-Prozeduren:

```
Public Property Let Table(str As String)
    m_Table = str
End Property
```

```
Public Property Let Field(str As String)
    m_Field = str
End Property
```

Die **Property Set**-Prozedur für die Eigenschaft **TextBox** hat noch eine zusätzliche Anweisung:

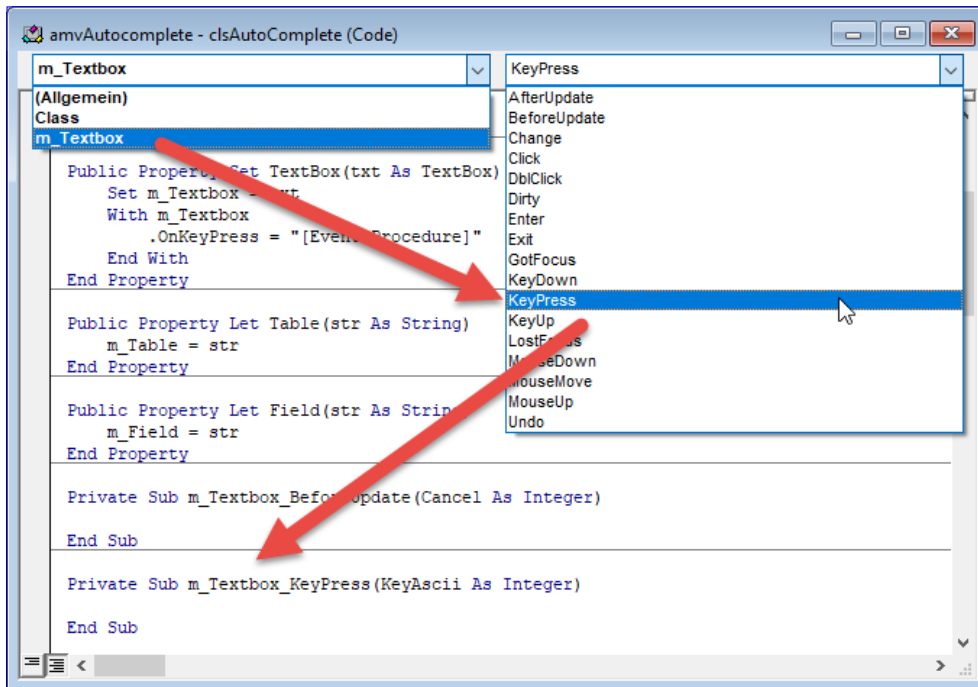


Bild 1: Hinzufügen der Ereignisprozedur für das Ereignis **KeyPress**

```
Public Property Set TextBox(txt As TextBox)
    Set m_Textbox = txt
    With m_Textbox
        .OnKeyPress = "[Event Procedure]"
    End With
End Property
```

Die Zuweisung des Wertes **[Event Procedure]** zur Eigenschaft **OnKeyPress** teilt dem Klassenmodul mit, dass es nach einer Implementierung des Ereignisses **KeyPress** für das Textfeld aus **m_Textbox** suchen soll, wenn der Benutzer dieses Ereignis auslöst.

Durch diese Vorbereitung finden Sie im linken Kombinationsfeld des Codefensters für das Klassenmodul **clsAutoComplete** den Eintrag **m_TextBox**. Haben Sie diesen ausgewählt, erscheint automatisch die Ereignisprozedur **m_Textbox_BeforeUpdate** – das ist die Standardereignisprozedur für diesen Steuerelementtyp. Wir benötigen aber das Ereignis **KeyPress**, das wir danach im rechten Kombinationsfeld auswählen (siehe Bild 1). Unten sehen Sie die danach erscheinende Ereignisprozedur.

Diese brauchen wir dann nur noch mit dem Code zu füllen, den wir im oben genannten Beitrag im **KeyPress**-Ereignis für die jeweilige Schaltfläche programmiert haben – nebst einigen Anpassungen. Natürlich müssen wir alle Verweise auf das Textfeld und den Namen der zu verwendenden Tabelle samt Feld durch unsere drei Variablen **m_Textbox**, **m_Table** und **m_Field** ersetzen. Das Ergebnis sieht dann wie in Listing 1 aus.

Anpassungen

Hier haben wir noch kleinere Anpassungen durchgeführt. Unsere Lösung hatte zum Beispiel den folgenden Fehler: Wenn Sie ein Leerzeichen eingegeben haben und die Prozedur keinen passenden Eintrag gefunden hat, wurde das Leerzeichen nicht im Textfeld ausgegeben. Diesen Fall haben wir mit einer zusätzlichen **If...Then**-Bedingung im **If**-Teil der ersten **If...Then**-Bedingung abgefangen:

```
...
If Not KeyAscii = 32 Then
    KeyAscii = 0
End If
...
```

AutoSelect in der Praxis

Nun wollen Sie auch ein Formular mit mehreren Textfeldern mit der **AutoSelect**-Funktion ausstatten. Dazu legen wir ein einfaches neues Formular an, dessen Datensatzquelle entsprechende Felder enthält. Ein gutes Beispiel dafür ist die Tabelle **tblKunden**, die passende Felder liefert.