HAUFE.
Ausgabe 06/2015

ACCESS

IM UNTERNEHMEN

KOMFORTABEL FILTERN IN FORMULAREN

Das Zusammenstellen von Filterkriterien ist mit Fallstricken verbunden. Wir zeigen, wie Sie diese kinderleicht zusammenstellen (S. 44) und die gewonnenen Erkenntnisse in Ihren Formularen umsetzen (S. 18).

In diesem Heft:

DATENSÄTZE NEBENEINANDER

Stellen Sie Datensätze nicht nur untereinander, sondern auch nebeneinander dar – mit Unterformularen und kleinen Tricks!

SEITE 8

PROJEKTZEITEN MIT OUTLOOK UND ACCESS

Nutzen Sie Outlook als Access-Frontend und geben Sie Tätigkeiten per Drag and Drop in den Terminkalender ein.

SEITE 56

KOMFORTABLE ANLAGEFELDER

Anlagefelder lassen sich noch annehmlicher gestalten – etwa mit einer Drag-and-Drop-Funktion oder einer Anlagenübersicht.

SEITE 24

Access 2016 ist da!

Schon seit einer Weile ist die neue Version der Datenbanksoftware von Microsoft auf dem Markt, um die sich ja die vorliegende Publikation hauptsächlich dreht. Und es gab noch nicht einmal einen Beitrag, der die Neuigkeiten der neuen Version auflistet! Genau genommen gab es das auch zur Version 2013 nicht. Warum? Weil es im Desktop-Bereich nichts Neues zu berichten gibt. Die gute Nachricht dafür ist: Damit sind nun auch keine essenziellen Funktionen weggefallen, wie vor einigen Jahren das Sicherheitssystem oder die Replikation.

Außerdem bedeutet es für uns Access-Entwickler, dass wir uns nicht in Neuigkeiten einarbeiten müssen. Das ist gar nicht ironisch gemeint: Wenn man weiß, dass die mit Access 2010 oder Access 2013 in vielen Stunden zusammengestellte Anwendung auch unter Access 2016 noch problemlos funktioniert, erspart das eine Menge Ärger.

Dass es aber auch mit der gleichen, unspektakulären Entwicklungsumgebung immer wieder Neues zu entwickeln gibt, zeigen die Beiträge der aktuellen Ausgabe. Hier haben wir uns zum Beispiel einmal um das Thema Filtern gekümmert, speziell um das Zusammenstellen von Filterausdrücken, die ja an vielen Stellen einer Anwendung vorkommen können. Der Beitrag **Filterbedingungen einfach zusammenstellen** zeigt ab S. 44, wie Sie Fehlern, die beim Zusammenstellen seitenlanger Filterkriterien per VBA und Zeichenkettenfunktionen auftreten, vorbeugen. Dazu lernen Sie eine Klasse kennen, mit der Sie Filterbedingungen stabil zusammenstellen können.

Wie Sie diese Klasse einsetzen, zeigt der Beitrag **Komfortabel filtern in Formularen** ab S. 18. Hier erfahren Sie, wie Sie die Techniken zum Filtern in echten Formularen einsetzen können.

Die mit Access 2007 eingeführten Anlagefelder (eine der letzten echten Neuerungen unter Access) sind praktisch, wenn Sie Dateien in einer Access-Datenbank speichern wollen. Unpraktisch wird es, wenn Sie auf die Schnelle ein paar Dateien hinzufügen wollen – beispielsweise per Drag and Drop aus dem Windows Explorer heraus.

Diese Funktion rüsten wir nach, und zwar mit einem kleinen Trick: Ein **ListView**-Steuerelement hält dabei als Zielsteuerelement her, die Dateien landen dann aber VBA-gesteuert im Anlagefeld. Dies alles finden Sie im Beitrag **Drag and Drop in Anlagefelder** ab S. 24.

Ein weiterer Beitrag zu diesem Thema heißt **Inhalte von Anlagefeldern verwalten** und zeigt ab S. 29, wie Sie die in einem Anlagefeld enthaltenen Dateien in einem **ListView**-Steuerelement übersichtlich anzeigen und diese öffnen, speichern oder löschen können.

Uns erreichen gelegentlich Leseranfragen, die sich eine Lösung für den Export von Berichten im PDF-Format wünschen. Dem kommen wir mit dem Beitrag **Verteiler für Berichte** gerne nach (ab S. 39).

Hier können Sie verschiedenen Verteiler festlegen, die einen oder mehrere Berichte der aktuellen Datenbank enthalten. Per Knopfdruck exportieren Sie die angegebenen Berichte dann im PDF-Format in das zuvor festgelegte Verzeichnis.

Wer Primärschlüsselfelder mit Autowert-Datentyp verwendet, kann theoretisch durch gelöschte Daten auftretende Lücken in der Nummerierung wieder auffüllen. Wie das geht und welche Probleme dies hervorrufen kann, zeigt der Beitrag **Probleme mit Primärschlüsseln und Autowert** ab S. 2.

Mancher Anwendungsfall lässt das Speichern einer individuellen Nummerierung von Datensätzen sinnvoll erscheinen. Allerdings kann es dann auch vorkommen, dass durch gelöschte Datensätze oder durch Umsortierungen eine Aktualisierung der Nummerierung erforderlich ist. Der Beitrag **Individuelle Nummerierungen** zeigt ab S. 4, wie Sie die gewünschte Nummerierung herstellen können.

Und schließlich liefert der Beitrag **Projektzeiterfassung mit Outlook und Access** eine Lösung, mit der Sie Ihre Arbeitszeiten ganz einfach per Drag and Drop in den Terminkalender von Outlook erfassen können – die Daten werden im Hintergrund in einer Access-Datenbank gespeichert (ab S. 56).

Und nun: Viel Spaß beim Lesen!

Ihr Michael Forster

Probleme mit Primärschlüsseln und Autowert

Primärschlüsselfelder mit Autowert-Funktion bieten eine komfortable Möglichkeit, für neue Datensätze automatisch einen neuen, eindeutigen Primärschlüsselwert zu vergeben. Wenn man zwischendurch keine Datensätze löscht, ist diese Nummerierung sogar lückenlos. Allerdings kann es sein, dass jemand versucht, gelöschte Datensätze mit den fehlenden Primärschlüsselwerten zu ersetzen. Zu welchen Problemen dies führt und wie Sie diese lösen, zeigt der vorliegende Beitrag.

Autowert-Primärschlüsselfelder

Fast alle Tabellen verwenden ein Primärschlüsselfeld, das über die Autowert-Funktion mit einem neuen, noch nicht vergebenen Wert gefüllt wird (s. Bild 1).

Mit dem Felddatentyp **Autowert** legt man automatisch die Eigenschaft **Neue Werte** auf Inkrement fest, was bedeutet, dass als Primärschlüsselwert beim Neuanlegen von Datensätzen standardmäßig der Primärschlüsselwert des zuletzt angelegten Datensatzes herangezogen und um eins erhöht wird.

Damit ist eine kleine Stolperfalle verbunden, denn dies funktioniert nur, wenn Sie den neuen Datensatz über die Tabelle oder ein an die Tabelle gebundenes Formular eingeben oder dies per VBA erledigen, wobei hier der Primärschlüsselwert explizit nicht festgelegt werden darf, sondern vom System erzeugt werden muss.

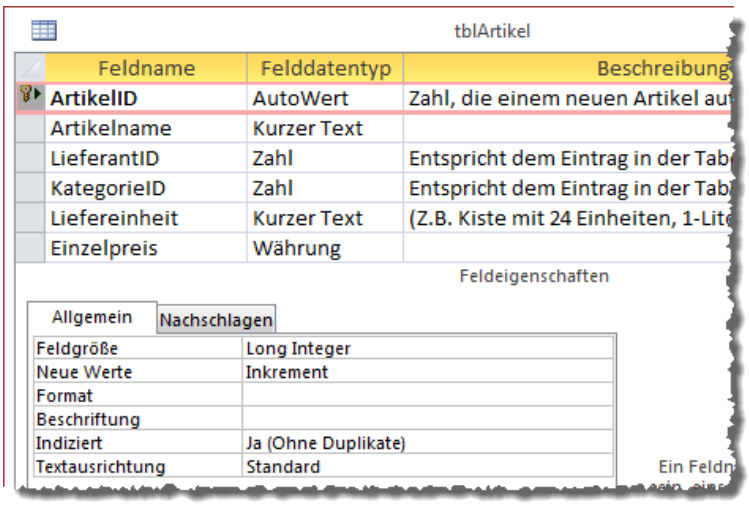

Bild 1: Entwurf einer Tabelle mit Autowert-Primärschlüsselfeld

Dies erreichen Sie beispielsweise durch folgende kleine Prozedur, die Sie in einem Standardmodul anlegen und per **F5** ausführen:

```
Public Sub NeuerArtikelMitAutowert()
    Dim db As DAO.Database
    Set db = CurrentDb
    db.Execute "INSERT INTO tblArtikel(Artikelname) 
        VALUES('Beispielartikel')", dbFailOnError
End Sub
```

Der Autowert wird hier automatisch auf den zuletzt vergebenen Wert plus eins festgelegt.

Die folgende Variante (siehe Prozedur **NeuerArtikel_MitVorgegebenemPK** im Modul **mdlBeispiele**) verwendet die folgende **INSERT INTO**-Anweisung:

```
db.Execute "INSERT INTO tblArtikel(ArtikelID, 
Artikelname) VALUES(79, 'Beispielartikel')", 
dbFailOnError
```

Diese legt ebenfalls einen neuen Datensatz an, weist dem Primärschlüsselfeld aber den Wert **79** zu. Sofern dieser bereits vergeben ist, löst dies den Fehler **3022** aus.

Wenn Sie einen Datensatz per DAO anlegen, sieht dies so aus:

```
Public Sub NeuerArtikel_DAO_MitAutowert()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
```

Individuelle Nummerierungen

Manche Anwendungsfälle erfordern jedoch eine individuelle Nummerierung in einem Feld. Dafür kann es verschiedene Wege geben – zum Beispiel, wenn eine Nummerierung nach einer bestimmten Sortierung eines anderen Feldes erfolgen soll oder nach dem Entfernen oder Hinzufügen von Datensätzen eine neue Nummerierung erforderlich ist. Daher stellen wir in diesem Beitrag Wege vor, um die Nummerierung ganz nach Wunsch zu gestalten.

Gründe für durchnummerierte Felder

In den meisten Fällen ist die Darstellung der Daten Auslöser für den Wunsch nach einer durchgehenden Nummerierung. Dies lässt sich natürlich auch über eine entsprechende Abfrage regeln.

Je nach Datenmenge kostet dies jedoch Performance, und gerade wenn die neu zu nummerierenden Daten selten oder gar nicht mehr geändert werden, ist das Hinzufügen eines Nummerierungsfeldes oder auch die Anpassung des Primärschlüsselfeldes sinnvoll. In allen anderen Fällen sollte die Nummerierung dynamisch per Abfrage erfolgen.

Unabhängig davon, ob Sie nun ein Primärschlüsselfeld oder ein separates Zahlenfeld für die Nummerierung verwenden, sind meist die folgenden zwei Gründe verantwortlich für den Wunsch nach einer neuen Nummerierung:

- Die Sortierkriterien als Grundlage für die Nummerierung haben sich geändert.
- Es wurden Datensätze entfernt, sodass die Nummerierung wieder durchgehend gestaltet werden soll.

Ein dritter Grund ist, dass möglicherweise eine Menge aktuell gefilterter Daten nummeriert werden soll. Da dies jedoch meist mehrmals geschieht, sollte man hier grundsätzlich eine dynamische Nummerierung per Abfrage vornehmen.

Primärschlüsselfeld oder zusätzliches Feld nummerieren?

In manchen Fällen soll das Primärschlüsselfeld selbst nummeriert werden. Dies führt erstens dazu, dass man die Aktualisierung der Fremdschlüsselfelder verknüpfter Datensätze für die Beziehung aktivieren muss.

Zweitens muss man sich hier einen kleinen Algorithmus einfallen lassen, da man ja nicht einfach neu nummerieren kann – eine Neunummerierung nach einer anderen Reihenfolge zum Beispiel würde dazu führen, dass ein Datensatz die Nummer **1** erhalten soll, während ein anderer Datensatz, der nach unten sortiert wurde, diesen Wert ebenfalls enthält.

Und in Primärschlüsselfeldern darf nun einmal jeder Wert nur einmal vorkommen. Man müsste in diesem Fall also zunächst den größten Wert für das Primärschlüsselfeld ermitteln und alle Datensätze mit einem Wert im Primärschlüsselfeld füllen, der größer ist als dieser Wert. Erst dann könnte man die Datensätze in der gewünschten Sortierung mit **1** beginnend durchnummerieren.

Deutlich einfacher ist es, wenn Sie für die Nummerierung ein eigenes Feld vorsehen. Dies kostet zwar etwas Spei-

tblArtikel

Feldname	Felddatentyp	Beschreibung (optional)
ArtikelID	AutoWert	Zahl, die einem neuen Artikel automatisch zugewiesen wird.
Artikelnummer	Zahl	Durchlaufende Artikelnummer
Artikelname	Kurzer Text	
LieferantID	Zahl	Entspricht dem Eintrag in der Tabelle "Lieferanten".
KategorieID	Zahl	Entspricht dem Eintrag in der Tabelle "Kategorien".
Liefereinheit	Kurzer Text	(Z.B. Kiste mit 24 Einheiten, 1-Liter-Flasche).
Einzelpreis	Währung	

Bild 1: Zusätzliches Feld für eine Nummerierung

cherplatz, aber das sollte in der heutigen Zeit keine übergeordnete Rolle mehr spielen.

Nummerieren zusätzlicher Felder

Beginnen wir mit den Techniken zum Nummerieren zusätzlicher Felder, die nicht mit einem eindeutigen Index versehen sind (in diesem Fall würde man diese auch erst mit höheren Zahlen nummerieren, bevor man diese beginnend mit dem Wert **1** nummerieren könnte).

Zu Beispielzwecken haben wir der Tabelle **tblArtikel** ein Feld namens **Artikelnummer** hinzugefügt, welches wir durchnummerieren wollen (s. Bild 1).

Die Füllung des Feldes **Artikelnummer** übernimmt die Prozedur aus Listing 1. Sie öffnet ein Recordset auf Basis der Tabelle **tblArtikel**, sortiert nach dem Feld **ArtikelID**.

```
Public Sub Sortierung()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim i As Integer
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM tblArtikel ORDER BY ArtikelID", dbOpenDynaset)
    Do While Not rst.EOF
        rst.Edit
        i = i + 1
        rst!Artikelnummer = i
        rst.Update
        rst.MoveNext
    Loop
End Sub
```

Listing 1: Einfache Sortierung im Feld **Artikelnummer** der Tabelle **tblArtikel**

Dann durchläuft sie alle Datensätze und erhöht dabei jeweils den Wert der Variablen **i**, der dann im Feld **Artikelnummer** landet.

Diese Prozedur lässt sich noch etwas vereinfachen, denn eigentlich benötigen wir die Variable **i** in diesem Fall gar nicht: Wenn mit dem Wert **1** beginnend durchgehend nummeriert werden soll, können wir dazu auch den Wert der Eigenschaft **AbsolutePosition** nutzen. Dies liefert die Position des Datensatzzeigers. Da dieser Wert für den ersten Datensatz den Wert **0** enthält, müssen wir nur noch den Wert **1** hinzuaddieren (s. Listing 2).

```
Public Sub SortierungMitAbsolutePosition()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM tblArtikel ORDER BY ArtikelID", dbOpenDynaset)
    Do While Not rst.EOF
        rst.Edit
        rst!Artikelnummer = rst.AbsolutePosition + 1
        rst.Update
        rst.MoveNext
    Loop
End Sub
```

Listing 2: Nummer für die Nummerierung aus **AbsolutePosition** entnehmen

Nummerierung mit anderem Startwert oder Intervall

Wenn die Nummerierung gar nicht mit 1 beginnen soll oder wenn ein anderes Intervall als 1 gefragt ist, können Sie die nachfolgende beschriebene Prozedur verwenden.

Ein anderes Intervall mit einer Nummerierung wie 1, 4, 7, ... könnte beispielsweise interessant sein, wenn man nicht häufig eine neue Nummerierung benötigt, aber dennoch neue Datensätze zwischen den vorhandenen Datensätzen einfügen möchte.

In diesem Fall nutzen Sie die Prozedur **SortierungStartwertIntervall** aus Listing 3. Diese erwartet als Parameter den Startwert für die Nummerierung sowie die Angabe des Intervalls. Die Werte **10** und **2** würden dann also zu einer Nummerierung wie **10, 12, 14 ...** führen.

```
Public Sub SortierungStartwertIntervall(lngStartwert As Long, lngIntervall As Long)
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM tblArtikel ORDER BY ArtikelID", dbOpenDynaset)
    Do While Not rst.EOF
        rst.Edit
        rst!Artikelnummer = rst.AbsolutePosition * lngIntervall + lngStartwert
        rst.Update
        rst.MoveNext
    Loop
End Sub
```

Listing 3: Sortierung mit Startwert und Intervall

Nummerierung mit Sortierung oder Filter

Nun möchten Sie vielleicht flexibel angeben, welche Tabelle (oder Abfrage) mit einer neuen Nummerierung versehen werden soll.

Dazu fügen wir der Prozedur noch zwei Parameter hinzu:

- **strDatenquelle**: Nimmt den Namen der Tabelle oder Abfrage oder einen SQL-Ausdruck entgegen.
- **strNummerierungsfeld**: Erwartet den Namen des Feldes, in das die Nummerierung eingetragen werden soll.

```
Public Sub SortierungFlexibel(strDatenquelle As String, strNummerierungsfeld As String, _
        lngStartwert As Long, lngIntervall As Long)
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset(strDatenquelle, dbOpenDynaset)
    Do While Not rst.EOF
        rst.Edit
        rst(strNummerierungsfeld) = rst.AbsolutePosition * lngIntervall + lngStartwert
        rst.Update
        rst.MoveNext
    Loop
End Sub
```

Listing 4: Sortierung mit Datenquelle, Sortierfeld, Startwert und Intervall

Die beiden Parameter werden in der Prozedur **SortierungFlexibel** verarbeitet (s. Listing 4) – **strDatenquelle** in der **OpenRecordset**-Methode und **strNummerierungsfeld** in der Anweisung zum Zuweisen der Nummer.

Ein Beispielaufruf sieht etwa wie folgt aus, das Ergebnis finden Sie in Bild 2:

```
SortierungFlexibel "SELECT * FROM tblArtikel ORDER BY
Artikelname DESC", "Artikelnummer", 1, 1
```

Nummerieren nach Sortierung

In den Parameter **strDatenquelle** lassen Sie natürlich auch direkt die Sortierung einfließen, falls gewünscht – entweder, indem Sie eine Abfrage mit der Sortierung anlegen, speichern und den Namen der Abfrage als Parameter angeben oder indem Sie den SQL-Ausdruck für die Datenherkunft samt Sortierung als Parameter übergeben – beispielsweise sortiert nach dem Artikelnamen:

```
SELECT * FROM tblArtikel ORDER BY Artikelname
```

Nummerierung nach Gruppen

Auf diese Weise können prinzipiell alle Nummerierungen vorgenommen werden – auch solche, die nur Teilmengen betreffen wie etwa Artikel einer bestimmten Kategorien.

Wenn Sie die Nummerierung für jede Kategorie separat vornehmen wollen, müssen Sie die Prozedur nur für jede Kategorie einmal aufrufen. Das gelingt, da ja auch die Kategorien in einer Tabelle gespeichert werden, am besten innerhalb einer **Do While**-Schleife, die alle Kategorien durchläuft. Dies sieht etwa wie in Listing 5 aus.

ArtikelID	Artikelnummer	Artikelname	
47	1	Zaanse koeken	Zaanse Snoe
64	2	Wimmers gute Semmelknödel	Plutzer Lebe
63	3	Vegie-spread	Pavlova, Ltd.
50	4	Valkoinen suklaa	Karkki Oy
7	5	Uncle Bob's Organic Dried Pears	Grandma Kel
23	6	Tunnbröd	PB Knäckebrö
54	7	Tourtière	Ma Maison
14	8	Tofu	Mayumi's
29	9	Thüringer Rostbratwurst	Plutzer Lebe
19	10	Teatime Chocolate Biscuits	Specialty Bis
62	11	Tarte au sucre	Forêts d'érab
35	12	Steeleye Stout	Bigfoot Brew
46	13	Spegesild	Lyngbysild
61	14	Sirop d'érable	Forêts d'érab

Bild 2: Nummerierung nach dem Artikelnamen in absteigender Reihenfolge

Die Prozedur erstellt ein Recordset auf Basis der Tabelle **tblKategorien**, also nach der Tabelle, nach deren Datensätzen die Sortierung aufgebaut werden soll. Sie durchläuft alle Datensätze dieses Recordsets in einer **Do While**-Schleife und ruft mit jedem Durchlauf einmal die bereits zuvor beschriebene Prozedur **SortierungFlexibel** auf. Dieser übergibt sie als ersten Parameter eine **SELECT**-Anweisung, die alle Datensätze der Tabelle **tblArtikel** enthält, deren Feld **KategorieID** der Kategorie des aktuellen Datensatzes des Recordsets entspricht. Der zweite Parameter liefert den Namen des Feldes, das die Nummerierung erhalten soll, der dritte und vierte die Einstellungen für den Startwert und das Intervall.

```
Public Sub Beispiel_NummerierungNachKategorie()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM tblKategorien", dbOpenDynaset)
    Do While Not rst.EOF
        SortierungFlexibel "SELECT * FROM tblArtikel WHERE KategorieID = " _
            & rst!KategorieID, "Artikelnummer", 1, 1
        rst.MoveNext
    Loop
End Sub
```

Listing 5: Aufruf der Nummerierungsroutine nach Kategorien

ArtikelID	Artikelnummer	Artikelname	Kategorie
1	1	Chai	Getränke
2	2	Chang	Getränke
3	1	Aniseed Syrup	Gewürze
4	2	Chef Anton's Cajun Seasoning	Gewürze
5	3	Chef Anton's Gumbo Mix	Gewürze
6	4	Grandma's Boysenberry Spread	Gewürze
7	1	Uncle Bob's Organic Dried Pears	Naturprodukte
8	5	Northwoods Cranberry Sauce	Gewürze

Bild 3: Nummerierung je Kategorie

Das Ergebnis finden Sie in Bild 3.

Zusammenfassung

Wenn eine Tabelle ein zusätzliches Nummerierungsfeld aufweist, können Sie die Nummerierung ganz einfach nach den gewünschten Kriterien durchführen. Dabei helfen Ihnen die in diesem Beitrag vorgestellten Beispielprozeduren.

Mehrere Datensätze pro Spalte in Formularen

Üblicherweise landen in einem Access-Formular entweder die Details eines Datensatzes oder mehrere Datensätze. Erstere können über das Formular verteilt werden, Letztere erscheinen untereinander in der Datenblattansicht oder der Endlosansicht. Mit einigen Unterformular-Steuerelementen lassen sich jedoch auch mehrere Datensätze nebeneinander anzeigen. Dieser Beitrag zeigt die Grundlagen zur Anzeige mehrerer Datensätze in einer Matrix von Unterformularen.

Grundlage für die Anzeige mehrerer Datensätze nebeneinander ist, dass Sie für jeden Datensatz ein Unterformular anlegen und diese dann nebeneinander anordnen. Die Datenherkünfte der Unterformulare müssen Sie dann noch so gestalten, dass diese jeweils den gewünschten Datensatz anzeigen. Wenn Sie also beispielsweise drei Kundendatensätze nebeneinander anzeigen wollen, müssten Sie dem ersten Unterformular den ersten Datensatz der Datenherkunft , dem zweiten den zweiten Datensatz und so weiter zuweisen.

Das Blättern in solchen Daten ist natürlich nicht mehr über die Bildlaufleiste möglich – dazu müssten Sie sich mit eigenen Steuerelementen wie etwa einer **Nach oben**- oder **Nach unten**-Schaltfläche behelfen. Natürlich könnte man auch die herkömmliche Bildlaufleiste nutzen, aber dann müsste das Formular so viele Unterformulare enthalten, dass alle Datensätze der Datenherkunft angezeigt werden können.

Mit steigender Datensatzanzahl sinkt hier die Performance, und auch die Anzahl der Unterformularsteuerelemente ist natürlich begrenzt. Also wollen wir es lieber bei einer vordefinierten Anzahl belassen und eigene Schaltflächen zum Blättern zwischen den Datensätzen verwenden. Das Ergebnis soll in der Rohfassung wie in Bild 1 aussehen. Das Hauptformular legen Sie als einfaches Formular mit der Bezeichnung **frmKundenNebeneinander** an und speichern dieses.

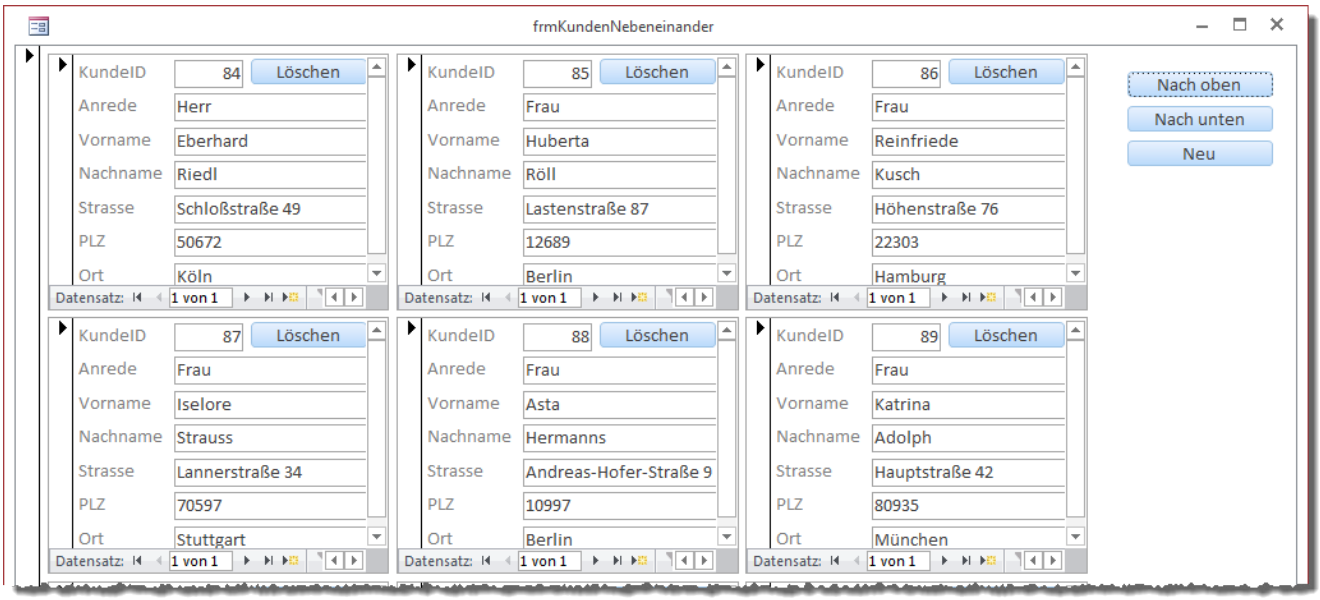

Bild 1: Unterformulare in mehreren Spalten zur Anzeige von Daten

Unterformulare hinzufügen

Im folgenden Beispiel wollen wir zwölf Unterformulare in drei Spalten und vier Zeilen anordnen. Die Unterformulare sollen die Bezeichnungen **sfm01**, **smf02** und so weiter erhalten. Damit die Unterformulare korrekt positioniert und mit den richtigen Abmessungen ausgestattet werden, wollen wir diese mit einem kleinen Skript zum Hauptformular hinzufügen.

Dieses sieht wie in Listing 1 aus und ist in der Beispieldatenbank im Modul **mdlTools** zu finden. Die Prozedur erwartet einige Parameter, welche den Namen des mit Unterformularen auszustattenden Formulars sowie die Abmessungen für die Unterformularsteuerelemente enthalten. Dabei liefern **intX** und **intY** zunächst die Anzahl der anzulegenden Unterformulare je Zeile und Spalte. **lngBreite** und **lngHoehe** enthalten die Abmessungen, **lngAbstandX** und **lngAbstandY** den Abstand der Unterformulare untereinander und zum linken und oberen Rand des Formulars. Schließlich legen Sie mit **strSubform**, soweit gewünscht, noch den Wert der Eigenschaft **Herkunftsobjekt** (VBA: **SourceObject**) fest.

```
Public Sub UnterformulareAnlegen(strForm As String, intX As Integer, intY As Integer, lngBreite As Long, _
        lngHoehe As Long, lngAbstandX As Long, lngAbstandY As Long, Optional strSubform As String)
    Dim frm As Form
    Dim ctl As Control
    Dim i As Integer
    Dim x As Integer
    Dim y As Integer
    On Error Resume Next
    DoCmd.Close acForm, strForm
    On Error GoTo 0
    DoCmd.OpenForm strForm, acDesign
    Set frm = Forms(strForm)
    For i = frm.Controls.Count - 1 To 0 Step -1
        Set ctl = frm.Controls(i)
        Select Case ctl.ControlType
            Case acLabel, acSubform
                DeleteControl frm.Name, ctl.Name
        End Select
    Next i
    i = 0
    For y = 1 To intY
        For x = 1 To intX
            i = i + 1
            Set ctl = CreateControl(frm.Name, acSubform, acDetail, , , lngAbstandX + (lngAbstandX + lngBreite) _
                * (x - 1), lngAbstandY + (lngAbstandX + lngHoehe) * (y - 1), lngBreite, lngHoehe)
            ctl.Name = "sfm" & Format(i, "00")
            If Len(strSubform) > 0 Then
                ctl.SourceObject = strSubform
            End If
        Next x
    Next y
End Sub
```

Listing 1: Hinzufügen der Unterformulare zum Hauptformular

Für die Konstellation in der Beispieldatenbank haben wir den folgenden Aufruf der Prozedur zusammengestellt:

```
UnterformulareAnlegen "frmKundenNebeneinander", 3, 4,
4000, 3000, 100, 100, "sfmKundenNebeneinander"
```

Damit legt diese 3 x 4 Unterformularsteuerelemente an und füllt diese gleich mit dem Unterformular **sfmKundenNebeneinander**. Die Prozedur schließt zunächst das mit **strForm** übergebene Formular, sofern dieses noch geöffnet ist, und öffnet es erneut, diesmal in der Entwurfsansicht. Dann speichert es einen Verweis auf das Formular in der Variablen **frm**.

In einer **For...Next**-Schleife über alle im Formular enthaltenen Steuerelemente löscht sie dann alle bereits vorhandenen Unterformulare. Sie sollten die Routine also nicht für Formulare nutzen, die bereits für andere Zwecke vorgesehene Unterformulare enthalten, oder aber eine Prüfung einbauen, welche nur die in vorherigen Aufrufen angelegte Unterformularsteuerelemente löscht. Zum Löschen der Steuerelemente verwendet die Routine die **DeleteControl**-Methode, die den Namen des Formulars und des zu löschenden Steuerelements enthält.

Danach durchläuft die Prozedur zwei verschachtelte **For... Next**-Schleifen über die Werte von **1** bis **intY** beziehungsweise **1** bis **intX**. In der Schleife erhöht sie den Wert der Variablen **i**, die zuvor auf **0** eingestellt wurde, um **1**. **i** soll die Nummer für den Steuerelementnamen (**sfm01**, **sfm02** und so weiter) liefern. Nun folgt die Methode **CreateControl**, welche das eigentliche Steuerelement erstellt. Sie erwartet den Namen des Zielformulars, den Typ des zu erstellenden Steuerelements, den Zielbereich (hier den Detailbereich) sowie die Koordinaten und Abmessungen.

Die Abmessungen entnimmt die Routine den Parametern **intHoehe** und **intBreite**, die Koordinaten werden für jedes Steuerelement neu berechnet. Der Abstand vom linken Formularrand stammt dabei aus der Formel **lngAbstandX**

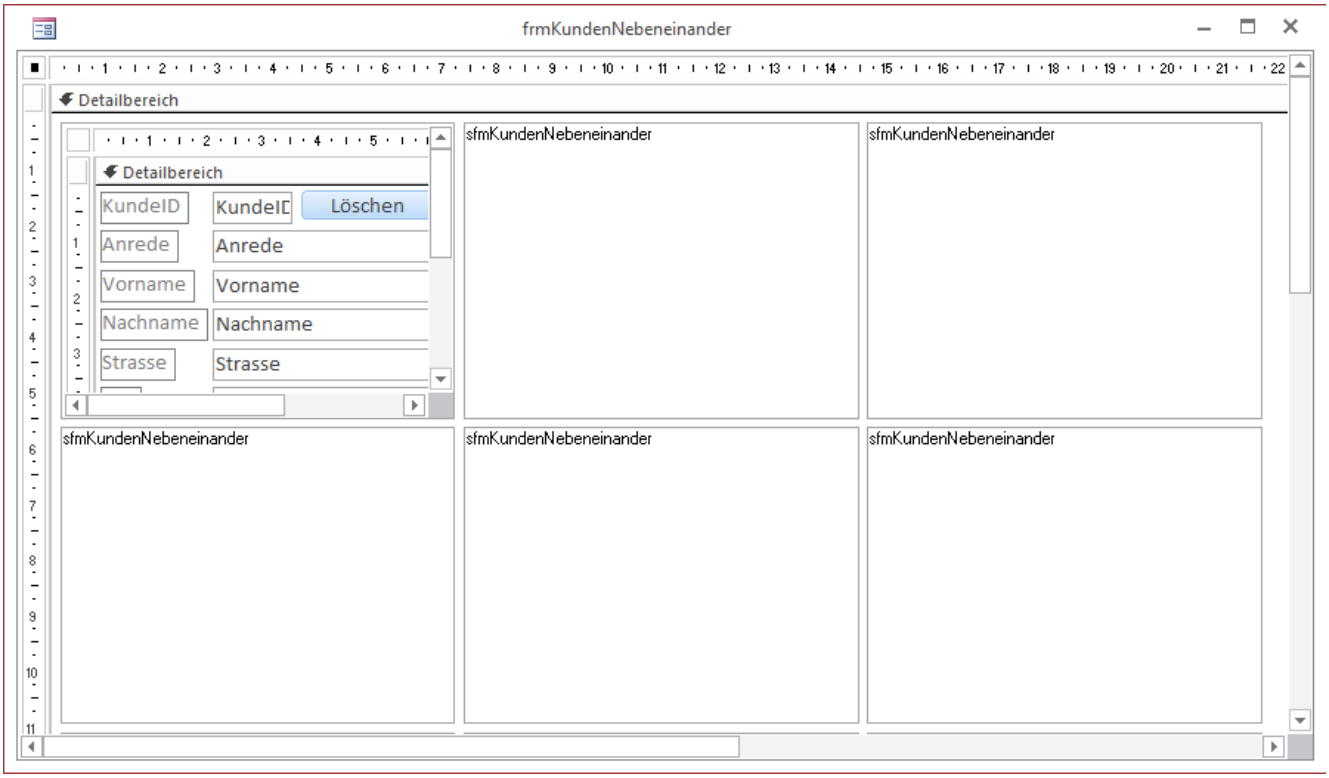

Bild 2: Frisch angelegte Unterformulare in der Entwurfsansicht

+ (lngAbstandX + lngBreite) * (x - 1). Im ersten Durchlauf ist **x = 0**, also ist der Abstand der mit **lngAbstandX** übergebene Wert. Für die folgenden Elemente, die natürlich rechts neben dem jeweils zuvor platzierten Element landen sollen, ermittelt die Routine die Position aus dem Abstand des ersten Elements vom linken Rand plus dem Produkt der Breite eines Elements plus dem einfachen Abstand und dem Index des Steuerelements. Auf die gleiche Art berechnet die Routine den Abstand vom oberen Formularrand.

Die Variable **ctl** speichert den Verweis auf das soeben erstellte Steuerelement. Damit weist die Routine noch den Namen für das Steuerelement zu. Wenn **strSubform** den Namen des Unterformulars enthält, der im Unterformularsteuerelement angezeigt werden soll, weist die Prozedur diesen der Eigenschaft **SourceObject** des Unterformularsteuerelements zu. Diesen Vorgang wiederholt die Prozedur, bis alle Zeilen und Spalten in den **For... Next**-Schleifen durchlaufen wurden. Das Ergebnis sieht zu diesem Zeitpunkt etwa wie in Bild 2 aus.

Unterformular anlegen

Schließlich benötigen wir auch noch das Unterformular, das in den Unterformularsteuerelementen des Hauptformulars erscheinen und die verschiedenen Datensätze anzeigen soll. Die Herkunftstabelle heißt **tblKunden** und liefert einige Felder, die wir wie in Bild 3 im Unterformular **sfmKundenNebeneinander** anorden. Außerdem legen wir dort schon einmal eine Schaltfläche namens **cmdLoeschen** an, die später das Löschen des im Unterformular angezeigten Datensatzes erlauben soll.

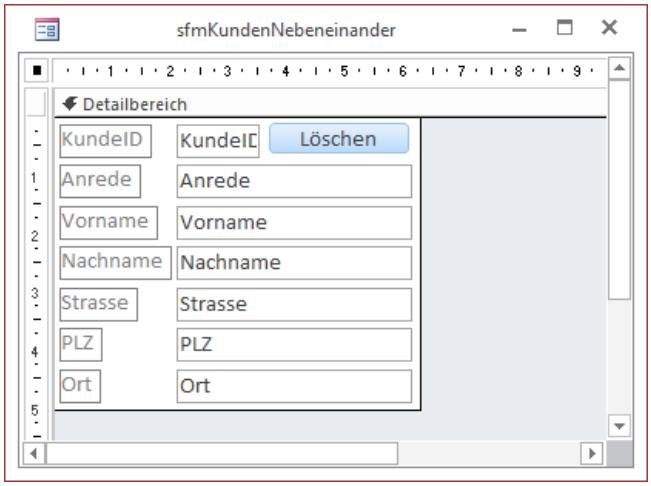

Bild 3: Das Unterformular der Beispielanwendung

Wenn Sie nun das Hauptformular in der Formularansicht öffnen, zeigt dieses für jedes Unterformular den gleichen Datensatz an (s. Bild 4). Kein Wunder: Das Unterformular ist ja in allen Fällen genau gleich aufgebaut und zeigt dementsprechend auch die gleichen Daten an – zumindest, bis wir gleich mit einigen Zeilen Code eingreifen.

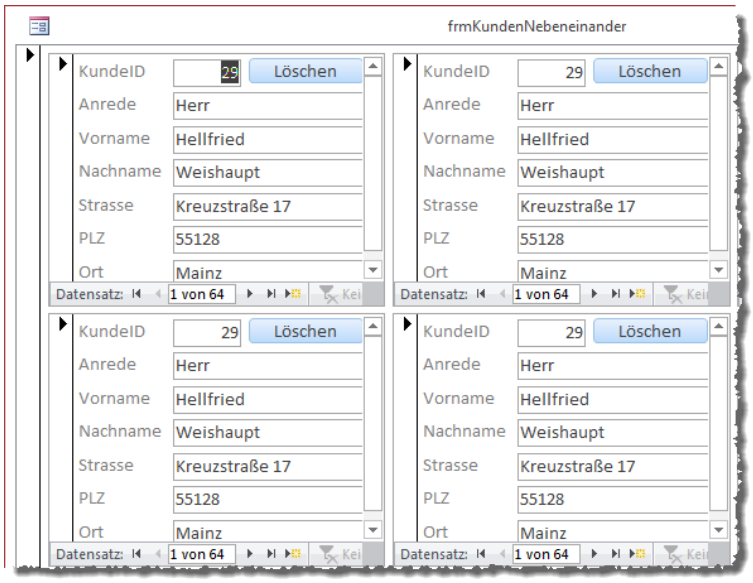

Bild 4: Gleicher Datensatz in allen Unterformularen

Unterformulare mit verschiedenen Datensätzen füllen

Die Unterformulare füllen wir gleich beim Öffnen des Hauptformulars, und zwar in der Ereignisprozedur, die durch das Ereignis **Beim Laden** ausgelöst wird (s. Listing 2). Diese Prozedur verwendet bereits einige Variablen, die wir modulweit im Kopf des Klassenmoduls **Form_frmKundenNebeneinander** deklarieren:

```
Dim db As DAO.Database
Dim rst As DAO.Recordset
```

```
Private Sub Form_Load()
    Dim intElemente As Integer
    Dim j As Integer
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM tblKunden", dbOpenDynaset)
    intStartdatensatz = 1
    UnterformulareFuellen intElemente
    Me!cmdNachUnten.Enabled = Not (intStartdatensatz + intElemente - 1 >= rst.RecordCount)
    Me!cmdNachOben.Enabled = Not intStartdatensatz = 1
End Sub
```

Listing 2: Füllen der Unterformulare mit den gewünschten Datensätzen

```
Dim intStartdatensatz As Integer
```

Die Prozedur füllt die Variable **db** mit einem Verweis auf das aktuelle **Database**-Objekt. Die Recordset-Variable **rst** füllt die Prozedur mit den Daten der Tabelle **tblKunden**.

Mit der Variablen **intElemente** als Parameter ruft die Prozedur damit die Routine **UnterformulareFuellen** auf. Diese ist letztlich dafür verantwortlich, in den Unterformularen den richtigen Datensatz anzuzeigen. Da wir später mit dem Formular durch die Kundendatensätze blättern wollen, benötigen wir noch die aktuelle Position. Dazu tragen wir den Wert **1** in die Variable **intStartdatensatz** ein, was bedeutet, dass das erste Unterformular den ersten Datensatz der Datenherkunft anzeigt. Schließlich soll das Formular zwei Schaltflächen namens **cmdNachOben** und **cmdNachUnten** enthalten, mit denen der Benutzer zu den vorherigen beziehungsweise nächsten Datensätzen blättern kann, sowie eine Schaltfläche zum Anlegen eines neuen Datensatzes (**cmdNeu**) – s. Bild 5.

Wenn das Formular gerade die ersten Datensätze anzeigt, soll die Schaltfläche **cmdNachOben** natürlich deaktiviert sein, das Gleiche gilt für die Schaltfläche **cmdNachUnten** auf der letzten Seite. Dies erledigen die letzten beiden Anweisungen der Prozedur. Die Bedingung für das Deaktivieren der Schaltfläche **cmdNachUnten** lautet **Not (intStartdatensatz + intElemente - 1 >= rst.RecordCount)**. Der Startdatensatz (in diesem Fall **1**) plus der Anzahl der angezeigten Datensätze (von der Prozedur **UnterformulareFuellen** mit dem Parameter **intElemente** zurückgeliefert) minus **1** soll nicht größer oder gleich der Anzahl der Datensätze der Datenherkunft sein. Die **Nach oben**-Schaltfläche wird immer aktiviert, wenn der Startdatensatz nicht der erste ist (**intStartdatensatz = 1**).

Unterformulare füllen

Die Ereignisprozedur **Form_Load** ruft die Routine **UnterformulareFuellen** aus Listing 3 auf, um die entsprechenden Datensätze in den einzelnen Unterformularen anzuzeigen. Diese verwendet einen Parameter namens **intElemente**, der die Anzahl der auf dieser Seite gefüllten Unterformulare zurückgeben soll. Die Prozedur durchläuft eine **Do While**-Schleife, die erst abgebrochen wird, wenn das Ende des Recordsets erreicht ist, also keine Datensätze mehr in Unterformularen abzubilden sind, oder **intElemente** kleiner als **intUnterformulare** wird, also alle Unterformulare gefüllt sind. **intElemente** wird gleich in der ersten Anweisung um **1** erhöht. Dann ermittelt die Prozedur den Wert des Primärschlüsselfeldes des aktuellen Datensatzes und speichert diesen in der Variablen **lngKundeID**.

Damit beginnen die Arbeiten am Unterformular: Dieses erhält zunächst eine Abfrage als Datensatzquelle, die den Kunden mit dem in **lngKundeID** gespeicherten Primärschlüsselwert liefert. Die **Visible**-Eigenschaft erhält den Wert **True**, damit das Unterformularsteuerelement eingeblendet wird. Dies wiederholt die Schleife so lange, bis die Abbruchbedingung erfüllt ist.

Wenn alle Unterformulare vor dem Aufruf von **UnterformulareFuellen** gefüllt sind und dann die letzte Seite

Komfortabel filtern in Formularen

Im Beitrag »Filterbedingungen einfach zusammenstellen« erfahren Sie weiter hinten in dieser Ausgabe, wie Sie den fehleranfälligen Code für die Zusammenstellung von Filterausdrücken in eine Klasse auslagen und übersichtlicher gestalten. Der vorliegende Beitrag zeigt nun anhand eines Praxisbeispiels, wie Sie die gewonnenen Erkenntnisse und die verwendete Klasse in Access-Formularen zum Filtern der enthaltenen Daten einsetzen.

Praxisbeispiel

In der **Suedsturm**-Datenbank gibt es die Tabelle **tblBestellungen**. Die Daten aus dieser Tabelle sollen in einem Bestellübersichtsformular angezeigt und gefiltert werden. Filtern wollen wir nach Kundenname, Lieferadresse, Bestelldatum, Lieferdatum, Versanddatum, offene Lieferung und bestellte Artikel.

Für die Darstellung der Suchmaske und der Ergebnisliste wählen wir ein ungebundenes Hauptformular (**frmBestellUebersicht**) mit ungebundenen Steuerelementen für die Filtereingabe. In das Hauptformular fügen wir ein Formular in Datenblattansicht (**frmBestellUebersicht_SF_Bestellungen**) als Unterformular ein.

Als Datenquelle für das Unterformular verwenden wir die Abfrage **qryBestellListe**. Für die Eingabe der Filterwerte fügen wir im Formularkopf des Hauptformulars entsprechende Steuerelemente ein. Eine Schaltfläche dient zum Anwenden der Filterwerte.

Als Alternative zur Schaltfläche fügen wir noch eine Checkbox ein, mit der wir eine automatische Filterung nach jeder Filterwert-Änderung aktivieren können. Zum

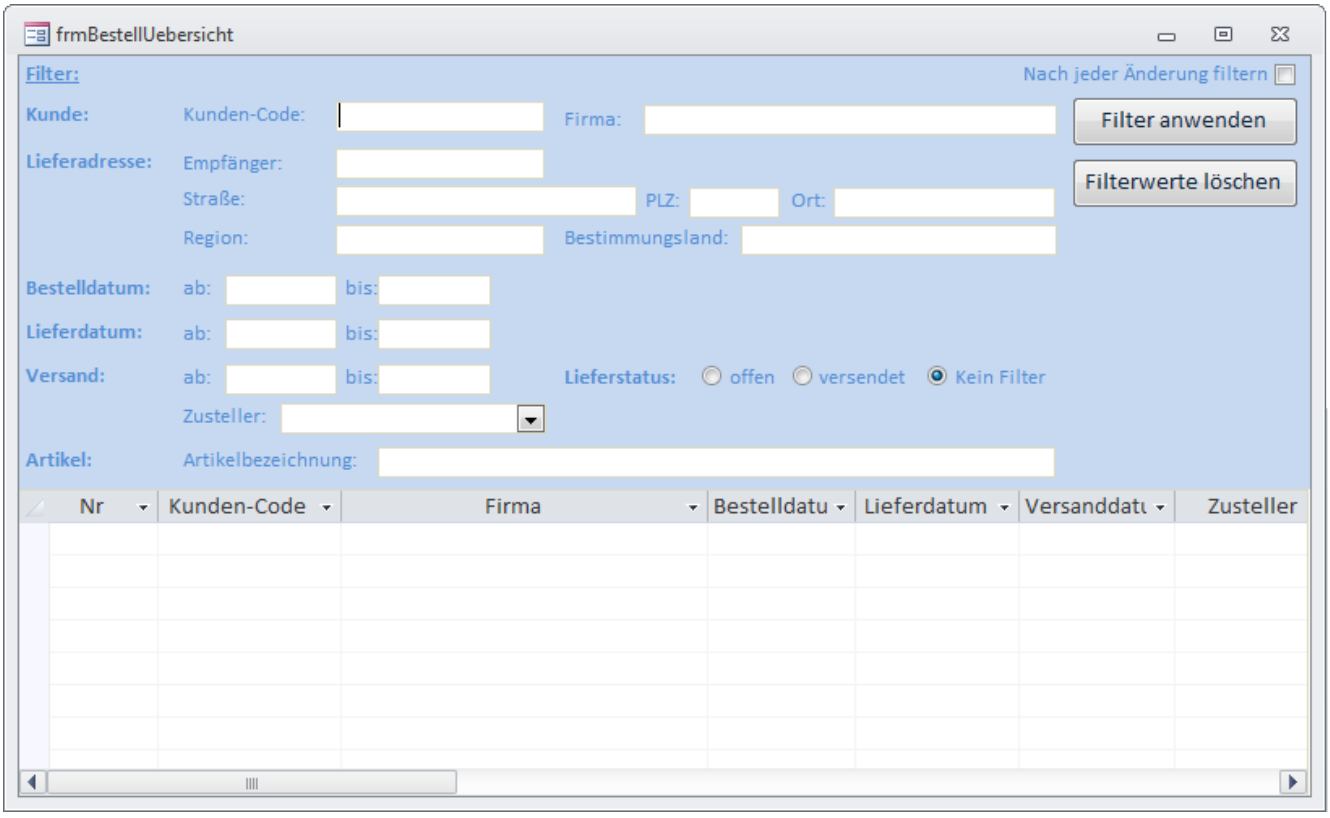

Bild 1: Filterformular

Entfernen aller eingegeben Filterwerte dient eine weitere Schaltfläche. In Bild 1 sehen Sie das Filterformular direkt nach dem Öffnen. Damit im Unterformular direkt nach dem Öffnen keine Datensätze angezeigt werden, ist in der Formular-Eigenschaft **Filter** der Ausdruck **1=0** eingetragen.

Die Namen der Filtersteuerelemente beginnen alle mit dem Präfix **fctl**. Das erleichtert uns später im Code das Ansprechen der Steuerelemente in **For Each**-Schleifen.

Variante I: Filtern ohne Hilfsklasse

Erstellen wir als erste Variante den Code zum Filtern ohne Verwendung von Hilfsklassen. Wenn auf die Schaltfläche **Filter anwenden** geklickt wird, soll das Unterformular gefiltert werden.

Damit der Code im Formular übersichtlich bleibt, erstellen wir die Prozedur **UseFilter** zum Starten des Filtervorgangs und rufen diese in der **Beim Klicken**-Ereignisbehandlung der Schaltfläche auf.

In der **UseFilter**-Prozedur übergeben wir den über eine weitere Prozedur erzeugten Filterausdruck an das Unterformular.

Den vollständigen Inhalt der Prozedur **CreateFilterString** zur **Filtertext**-Erstellung werden wir später ergänzen. Zuerst geben wir zum Testen nur den Filterausdruck für den Filterwert in **Firma** zurück. Mit diesem Code können wir bereits einen ersten Test durchführen.

Dabei ruft die Ereignisprozedur, die durch das Ereignis **Beim Klicken** der Schaltfläche **cmdUseFilter** ausgelöst wird, eine weitere Routine namens **UseFilter** auf:

```
Private Sub cmdUseFilter_Click()
   UseFilter
End Sub
```

Diese startet eine weitere Prozedur namens **SetSubFormFilter** und übergibt als Parameter einen Wert, der von der Funktion **CreateFilterString** beigesteuert wird und wie folgt aussieht:

```
Private Sub UseFilter()
' Diese Prodedur startet die Filterung mit den Werten
' in den Filter-Steuerelementen
   Call SetSubFormFilter(CreateFilterString)
End Sub
```

Diese sieht wie in Listing 1 aus.

Die Funktion prüft, ob das Filter-Textfeld **fctlKundeFirma** einen Wert enthält (**Len(.Value) > 0**). Falls ja, stellt sie einen Filterausdruck zusammen und gibt diesen als Funktionswert zurück.

Die Prozedur **SetSubFormFilter** nimmt diesen Filterstring entgegen und prüft, ob dieser bereits angewendet wurde. Dazu vergleicht sie den Wert der Eigenschaft **Filter** mit **FilterString**.

Nur wenn es sich um einen neuen Filter-Ausdruck handelt, wird **FilterString** der Eigenschaft **Filter** zugewiesen und

```
Private Function CreateFilterString() As String
    'Diese Prozedur erzeugt den Filterausdruck
    'aus den Filter-Steuerelementen

    With Me.fctlKundeFirma
        If Len(.Value) > 0 Then
            CreateFilterString = "Kunde_Firma LIKE '" _
                & Replace(.Value, "'", "''") & "'"
        Else
            CreateFilterString = vbNullString
        End If
    End With

End Function
```

Listing 1: Zusammensetzen des Filterausdrucks

der Filter mit der Eigenschaft **FilterOn** aktiviert oder deaktiviert – je nachdem, ob **FilterString** einen Ausdruck enthält oder nicht (s. Listing 2).

```
Private Sub SetSubFormFilter(ByVal FilterString As String)
    'Diese Prozedur kapselt die Datenfilterung im
    'Unterformular
    'Vorteil: wenn man auf eine andere Variante filtern
    'will, muss man nur diese Prozedur ändern

    With Me.sfrBestellListe.Form
        If .Filter <> FilterString Then
            'eine Änderung ist nur notwendig,
            'wenn der Filtertext nicht gleich ist
            .Filter = FilterString
            .FilterOn = (Len(FilterString) > 0)
        End If
    End With
End Sub
```

Listing 2: Die Prozedur zum Setzen des Filters

Wenn Sie im Textfeld für den **Firma**-Filterwert einen Text eintragen, wird nach diesem Text gefiltert, sobald Sie auf die Schaltfläche **Filter anwenden** klicken. Sie können das Textfeld auch leer lassen und filtern. Dann werden alle Datensätze aus der Tabelle angezeigt.

Als Nächstes ergänzen wir den Code zum Leeren der Filtersteuerelemente. Da wir die Namen der Filtersteuerelemente mit dem Präfix **fctl** versehen haben, können wir die Formularsteuerelemente mittels **For Each**-Schleife durchlaufen und alle Filterfelder leeren.

Außerdem wollen wir nach dem Löschen der Filterwerte nicht alle Datensätze anzeigen, sondern im Unterformular keine Daten anzeigen, damit erkennbar ist, dass noch nicht gefiltert wurde.

Dazu setzen wir den Filter im Unterformular wieder auf **1=0**. Mit dem zusätzlichen Code aus Listing 3 können wir das Löschen der Filter testen.

Filtern direkt nach der Eingabe

Als weiteres Feature wollen wir sofort nach Eingabe oder Änderung eines Filterwertes die Datensätze im Unterformular filtern. Wir müssen dazu auf das Ereignis **AfterUp-**

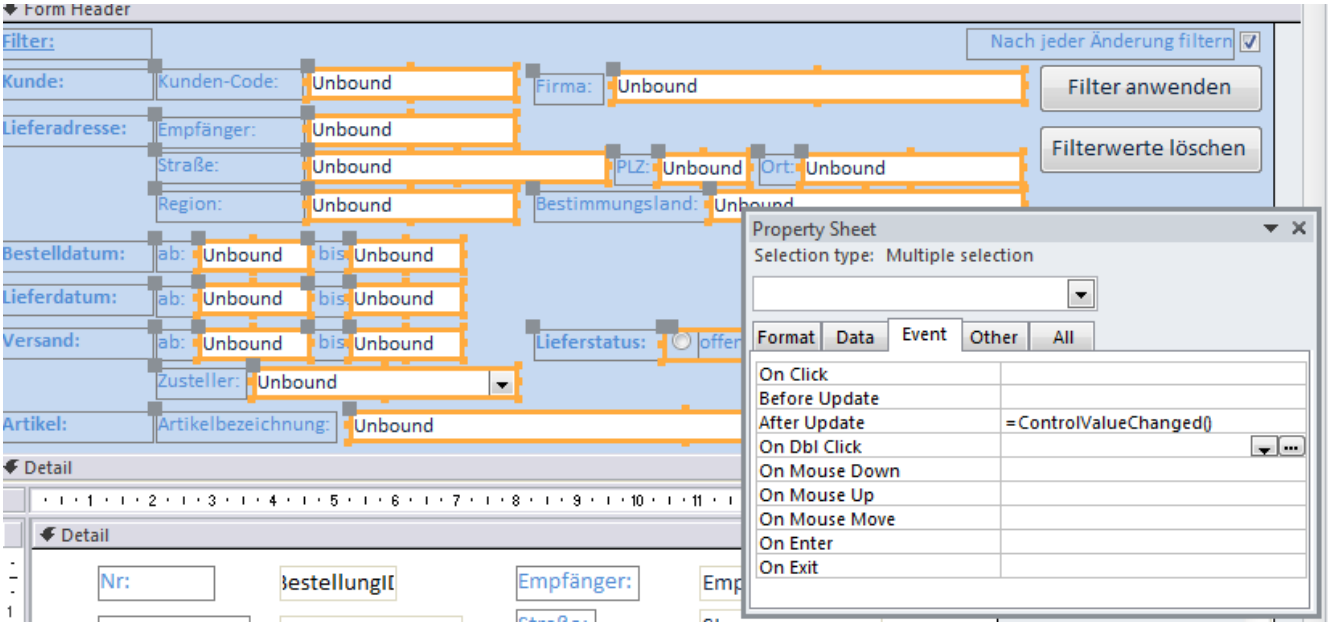

Bild 2: Ereignisbehandlung für AfterUpdate einstellen

Drag and Drop in Anlagefelder

Mit den Anlagefeldern hat Microsoft unter Access 2007 eine Möglichkeit zum Speichern von Dateien in Tabellenfeldern eingeführt, die etwas einfacher zugänglich ist als das bis dahin für solche Zwecke verwendete OLE-Feld. Leider fehlt noch eine Möglichkeit, einfach per Drag and Drop eine Datei aus dem Windows Explorer in ein Anlagefeld einer Tabelle zu ziehen. Dieser Beitrag zeigt, wie Sie ein solches Feature nachträglich hinzufügen können.

Anlagefelder sind eigentlich eine prima Sache: Sie können ganz einfach über die Benutzeroberfläche Dateien hinzufügen und darauf ins Dateisystem übertragen. Auch ein Öffnen der Dateien direkt aus dem Anlagefeld heraus ist möglich. Allerdings ist der Zugriff auf die enthaltenen Dateien mitunter etwas mühselig.

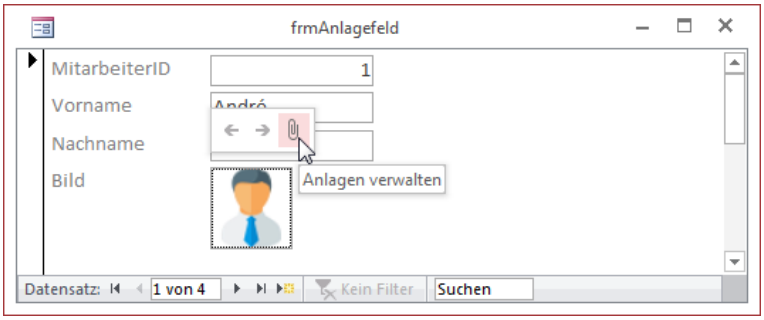

Bild 1: Öffnen des Dialogs zum Verwalten von Anlagen

Immerhin müssen Sie zunächst auf das Anlage-Steuerelement klicken und dann auf das erscheinende Menü, um überhaupt erst die Liste der enthaltenen Dateien anzuzeigen (s. Bild 1).

Dort erwartet Sie dann ein Dialog, der die im Anlagefeld enthaltenen Anlagen anzeigt und verschiedene Möglichkeiten bietet, zum Beispiel zum Hinzufügen, Entfernen oder Öffnen einer der im Anlagefeld gespeicherten Dateien (s. Bild 2).

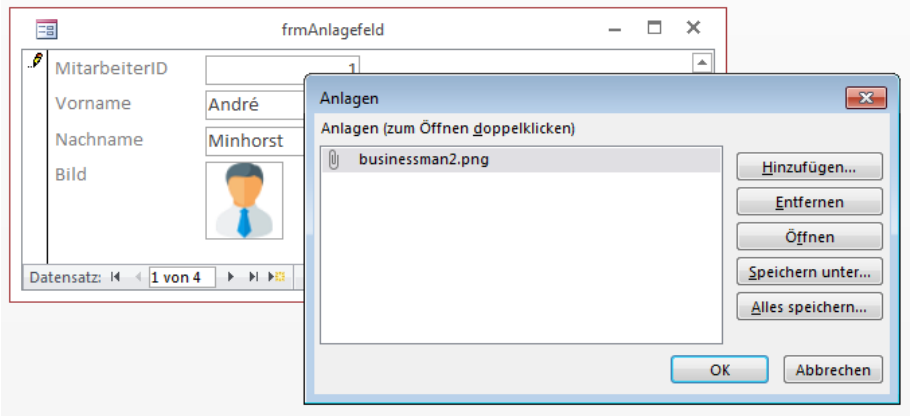

Bild 2: Dialog zum Verwalten von Anlagen

Was hier für einige Anwendungen fehlen dürfte, ist eine Möglichkeit, mal eben eine Datei aus dem Windows Explorer auf ein Steuerelement zu ziehen und so die Datei im Anlagefeld zu speichern. Das stellen wir uns etwa wie in Bild 3 vor.

Hier wählen wir komfortabel die Zieltabelle und das darin enthaltene Anlagefeld aus. Das Listenfeld darunter zeigt alle Datensätze der Tabelle an. Hier markieren Sie den Datensatz, dem Sie eine Datei hinzufügen möchten. Dann ziehen Sie die gewünschte Datei aus dem Windows Explorer einfach auf das darunter befindliche **ListView**-Steuerelement. Das erfolgreiche Speichern wird noch durch eine kurze Meldung bestätigt.

Formular für Drag und Drop

Das Formular der Beispieldatenbank heißt **frmDragAndDrop** und soll zeigen, wie Sie Dateien flexibel in beliebige Anlagefelder hineinziehen können. Dies ist erstens sinnvoll, weil Sie so die relevanten Codezeilen leicht anpassen können, zweitens können Sie das Formular gleich nutzen,

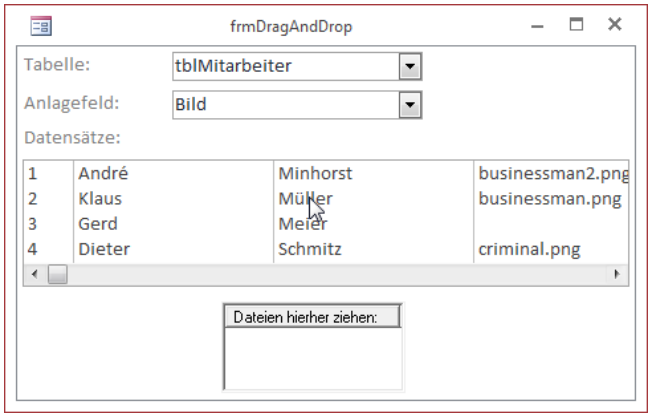

Bild 3: Formular mit Drag-and-Drop-Ziel für das Anlagefeld

wenn Sie einem Anlagefeld einer beliebigen Tabelle Dateien hinzufügen möchten.

In der Entwurfsansicht sieht das Formular wie in Bild 4 aus. Im oberen Bereich finden Sie die beiden Kombinationsfelder, mit denen Sie die Zieltabelle und das darin enthaltene Anlagefeld auswählen können. Darunter befindet sich ein Listenfeld, das die Inhalte der ersten vier Felder der gewählten Tabelle anzeigt.

Hier können Sie festlegen, zu welchem Datensatz die Anlage hinzugefügt werden soll. Schließlich folgt das wichtigste Steuerelement – das **ListView**-Steuerelement, das als Drag-and-Drop-Ziel dient.

Warum ausgerechnet ein **ListView**-Steuerelement? Nun: Dieses bietet im Gegensatz zu den Standardsteuerelementen von Access die Möglichkeit, auf Drag-and-Drop-Ereignisse zu reagieren und diese zu implementieren.

Tabelle und Feld auswählen

Das Kombinationsfeld **cboTabellen** erhält als Datenherkunft die folgende Abfrage:

```
SELECT Name FROM MSysObjects WHERE Type = 1
AND NOT Name LIKE 'MSys*' AND NOT Name LIKE '~*'
AND NOT Name LIKE 'f_*' ORDER BY Name;
```

Diese liefert alle Einträge der Systemtabelle **MSysObjects**, deren Feld **Type** den Wert **1** enthält (für lokale Access-Tabellen) und deren Feld **Name** nicht mit **MSys...**, **F_...** oder **~...** beginnt. Erstere sind die Systemtabellen, Letztere für den internen Gebrauch verwendete Tabellen.

Nach der Auswahl soll das zweite Kombinationsfeld die Felder der gewählten Tabelle anzeigen.

Dazu legen wir für **cboTabellen** eine Ereignisprozedur an, die durch das Ereignis **Nach Aktualisierung** ausgelöst wird:

```
Private Sub cboTabellen_AfterUpdate()
    Me!cboFelder.RowSource = Me!cboTabellen
    Me!lstDatensaetze.RowSource = Me!cboTabellen
End Sub
```

Die erste Anweisung stellt die Datensatzherkunft des Kombinationsfeldes zur Anzeige der Felder auf den Wert des ersten Kombinationsfeldes ein, also den Namen der Tabelle.

Die zweite legt die gleiche Tabelle als Datensatzherkunft des Listenfeldes **lstDatensaetze** zur Anzeige der enthaltenen Daten fest. Damit das zweite Kombinationsfeld nach

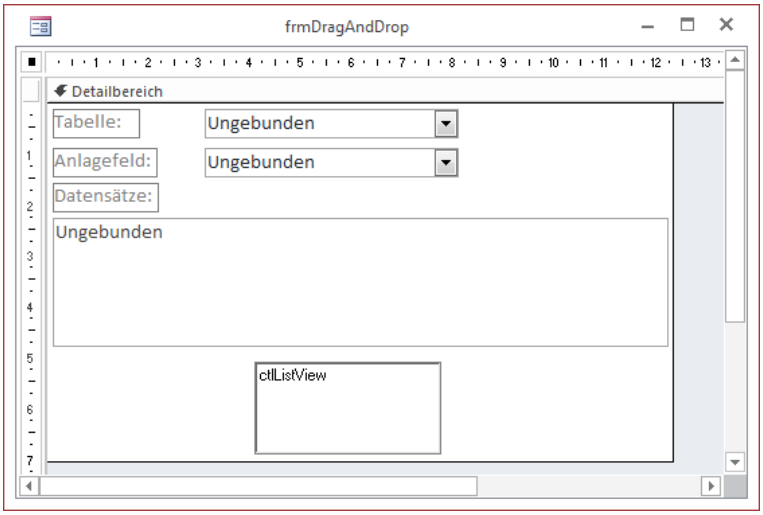

Bild 4: Das Formular mit dem Drag-and-Drop-Ziel in der Entwurfsansicht

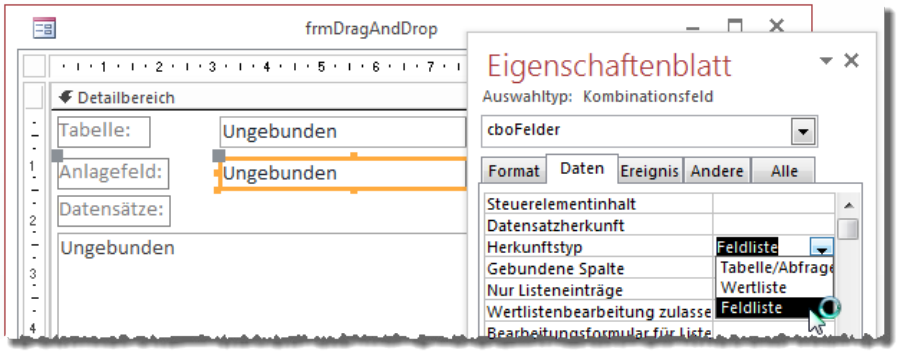

Bild 5: Einstellen der Herkunftsart für das Kombinationsfeld **cboFelder**

der Angabe eines Tabellennamens als Datensatzherkunft auch die enthaltenen Felder anzeigt, stellen Sie zuvor noch die Eigenschaft **Herkunftsart** auf den Wert **Feldliste** ein (s. Bild 5).

Fehlen noch einige Einstellungen für das **ListView**-Steuerelement. Diese nehmen wir gleich beim Öffnen des Formulars in der Ereignisprozedur **Beim Laden** vor.

Diese Prozedur aus Listing 1 deklariert zunächst eine Variable des Datentyps **ColumnHeader**. Dieses benötigen wir gleich, um eine Referenz auf ein neu angelegtes Objekt dieses Typs zu speichern. Das **ListView**-Element enthält nämlich nicht automatisch Spalten und somit auch keine Spaltenköpfe.

Nachdem die Prozedur eventuell noch vorhandene Spaltenköpfe mit der **Clear**-Anweisung gelöscht hat, legt sie mit der **Add**-Methode der **ColumnHeaders**-Auflistung des **ListView**-Steuerelements **ctlListView** einen neuen Spaltenkopf an und legt über den dritten Parameter dieser Methode die Spaltenbeschriftung fest. Im Grunde benötigen wir ja gar keine Spalten, aber wir wollen im **ListView**-Steuerelement die Beschriftung Dateien hierher ziehen: unterbringen, damit der Benutzer weiß, was er mit dem **ListView**-Steuerelement anfangen kann.

In der folgenden Anweisung stellen wir die Spaltenbreite für die neu erstellte und in der Variablen **objColumnheader** gespeicherten Spalte auf einen Wert ein, der etwas kleiner ist als die Breite des **ListView**-Steuerelements selbst. Schließlich legen wir noch die Ansicht auf **lvwReport** fest, damit der Spaltenkopf überhaupt erscheint, und stellen **OLEDropMode** auf **ccOLEDropManual** ein, damit wir das Ereignis für das Droppen von Elementen selbst programmieren können.

Die **ListItems.Clear**-Methode leert die Liste, falls diese aus irgendwelchen Gründen einen Eintrag enthalten sollte. Damit finden Sie das Formular in der Entwurfsansicht auch wie in der eingangs dargestellten Abbildung vor.

Drop-Ereignis

Wenn der Benutzer eine Datei über dem **ListView**-Steuerelement fallen lässt, löst dies das Ereignis **OLEDragDrop** aus. Dieses Ereignis können Sie nicht, wie bei den eingebauten Access-Steuerelementen üblich, über das Eigenschaftsfenster des Steuerelements anlegen. Stattdessen öffnen Sie direkt das Klassenmodul des Formulars. Wählen

```
Private Sub Form_Load()
    Dim objColumnheader As MSComctlLib.ColumnHeader
    With Me!ctlListview
        .ColumnHeaders.Clear
        Set objColumnheader = .ColumnHeaders.Add(, , "Dateien hierher ziehen:")
        objColumnheader.Width = Me!ctlListview.Width - 100
        .View = lvwReport
        .OLEDropMode = ccOLEDropManual
        .ListItems.Clear
    End With
End Sub
```

Listing 1: Einstellen des **ListView**-Steuerelements beim Öffnen des Formulars

Inhalte von Anlagefeldern verwalten

Anlagefelder sind eine prima Möglichkeit, um Dateien in Access-Tabellen zu speichern. Im Gegensatz zu OLE-Feldern stehen dazu sogar geeignete Elemente in der Benutzeroberfläche zur Verfügung. Diese sind allerdings nicht besonders ausgereift – so kann man nicht direkt im Formular die in einem Anlagefeld gespeicherten Elemente anzeigen oder neue hinzufügen. Dieser Beitrag zeigt, wie Sie ein Formular um diese Möglichkeit erweitern. Dabei fügen wir auch gleich die im Beitrag Drag and Drop im Anlagefeld vorgestellte Technik hinzu.

Beispieltabelle

Als Beispiel nutzen wir eine Tabelle namens **tblProjekte**, die neben dem Primärschlüsselfeld und der Projektbezeichnung nur ein Anlagefeld namens **Dateien** zum Speichern projektrelevanter Dateien enthält. Dies ist natürlich stark vereinfacht, aber es soll ja auch nur ein Beispiel sein. Bild 1 zeigt die Entwurfsansicht der Tabelle **tblProjekte**.

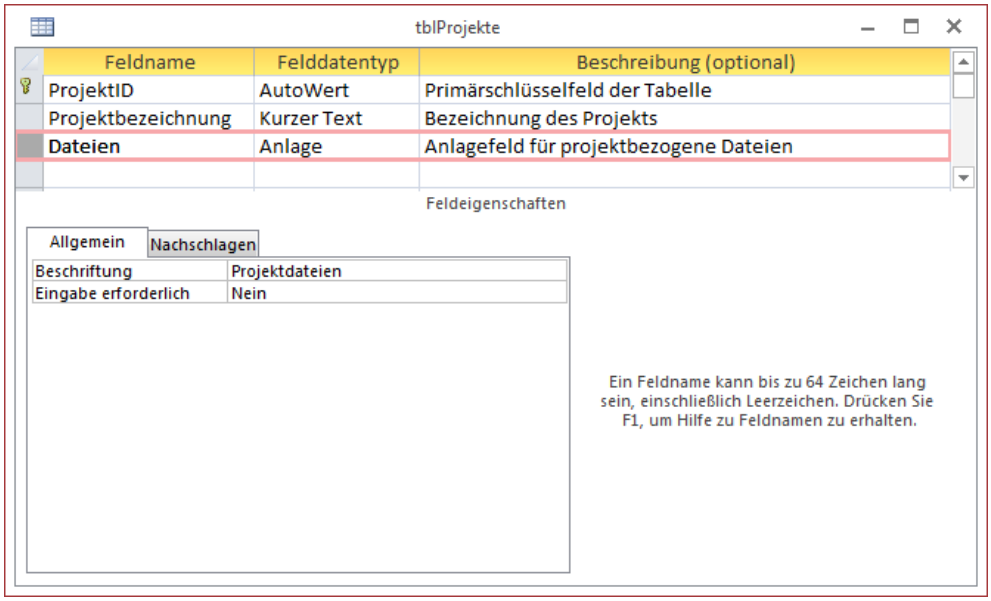

Bild 1: Tabelle mit Anlagefeld

Formular zum Verwalten der Anlagen

Das Formular zum Verwalten der Anlagen im Anlagefeld **Dateien** der Tabelle **tblProjekte** verwendet natürlich die Tabelle **tblProjekte** als **Datenherkunft** (s. Bild 2).

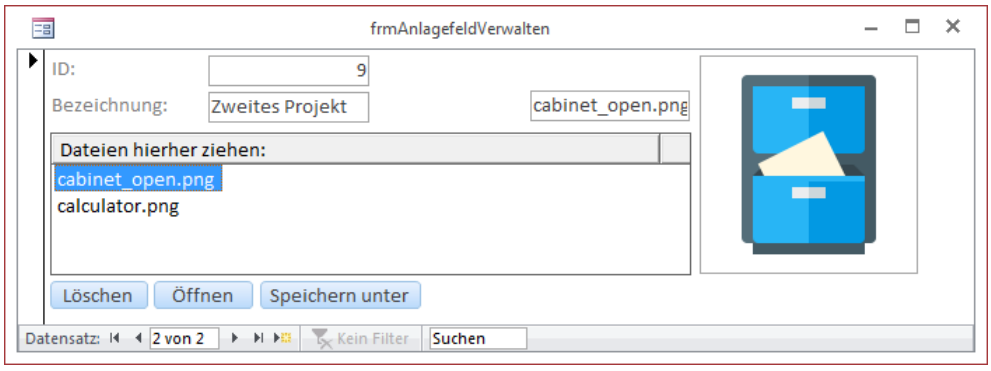

Bild 2: Formular zum Verwalten von Anlagefeld-Inhalten

Dabei soll sie den Inhalten der beiden Felder **ProjektID** und **Projektbezeichung** in entsprechenden Textfeldern anzeigen. Die im Anlagefeld gespeicherten Anlagen sollen in einem **ListView**-Steuerelement erscheinen. Drei Schaltflächen stellen die Funktionen zum Löschen, Öffnen und Speichern der aktuell markierten Anlage im **ListView**-Steuerelement bereit. Als Gimmick soll ein Anlage-Steuerelement im rechten Bereich auch noch die aktuell im Listenfeld markierte Anlage anzeigen. Im Falle einiger Dateien erscheint dann die Vorschau, wie in der Abbildung etwa für eine Bilddatei. Dateien können per

Drag and Drop zum **ListView**-Steuerelement hinzugefügt werden. Eine neu hinzugefügte Datei wird gleich markiert und im Anlage-Steuerelement angezeigt.

Zu den Einträgen des **ListView**-Steuerelements haben wir noch jeweils drei Kontextmenü-Befehle hinzugefügt, und zwar **Löschen**, **Öffnen** und **Speichern unter** (s. Bild 3).

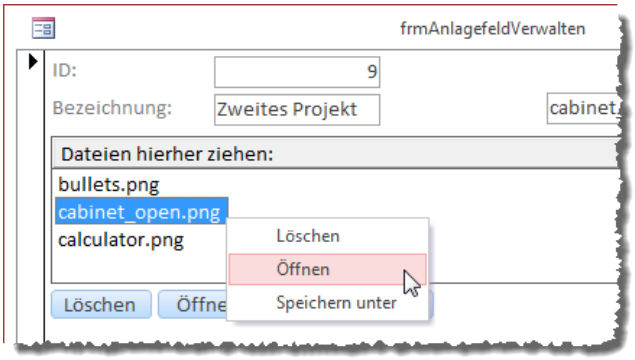

Bild 3: Kontextmenü mit den Anlagen-spezifischen Befehlen

Erstellen des Formulars

Das Formular sieht im Entwurf wie in Bild 4 aus. Hier können Sie erkennen, dass sich das Anlage-Steuerelement nicht direkt im Hauptformular befindet, sondern in ein Unterformular eingebettet ist. Dies haben wir so gemacht, weil wir ja das Anlagen-Feld der Datenherkunft nicht so an ein Anlagen-Steuerelement binden können, dass jeweils der Inhalt einer speziellen Anlage angezeigt wird. Wie dies technisch gelöst ist, schauen wir uns jedoch weiter unten an.

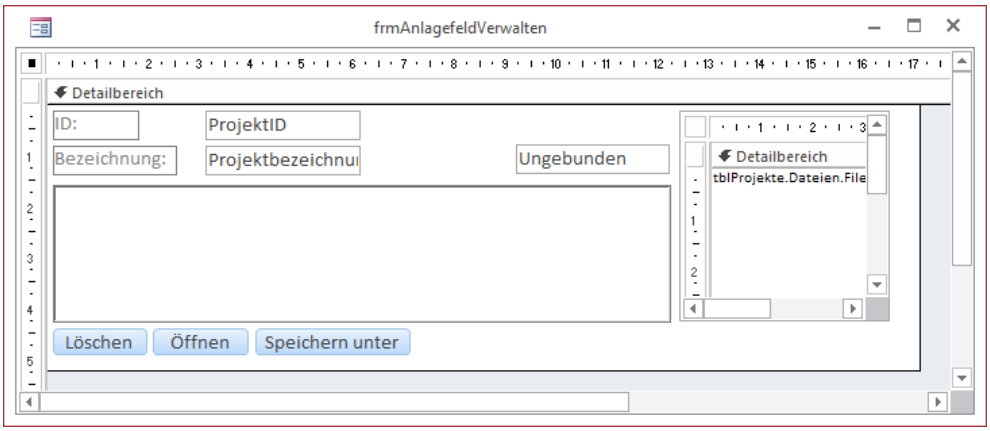

Bild 4: Das Formular **frmAnlagefeldVerwalten** in der Entwurfsansicht

```
Private Sub Form_Load()
    Dim objColumnheader As MSComctlLib.ColumnHeader
    With Me!ctlListview
        .ColumnHeaders.Clear
        Set objColumnheader = .ColumnHeaders.Add(, , "Dateien hierher ziehen:")
        objColumnheader.Width = Me!ctlListview.Width - 350
        .View = lvwReport
        .Font.Name = "Calibri"
        .Font.Size = 11
        .Appearance = ccFlat
        .OLEDropMode = ccOLEDropManual
        .ListItems.Clear
    End With
End Sub
```

Listing 1: Vorbereitungen, die beim Laden des Formulars durchgeführt werden

Laden des Formulars

Beim Laden des Formulars sind zunächst einige Schritte zur Konfiguration des **ListView**-Steuerelements nötig. Dies erledigen wir in der Ereignisprozedur **Beim Laden** (s. Listing 1).

Die Prozedur leert eventuell vorhandene Spaltenköpfe im **ListView**-Steuerelement und fügt dann einen neuen Spaltenkopf mit dem Text **Dateien hierher ziehen** hinzu. Die Breite dieses Elements stellt sie auf die Breite des Steuerelements minus 350 ein. Dies sorgt dafür, dass die einzige Spalte auch beim Einblenden der vertikalen Bildlaufleiste noch vollständig sichtbar ist und keine horizontale Bildlaufleiste eingeblendet werden muss.

Dann stellt sie die Ansicht auf **lvwReport** ein und legt die Schriftart und Schriftgröße so fest, dass diese zur Schrift der übrigen Steuerelemente passt. Der Wert **ccFlat** für die Eigenschaft **Appearance** sorgt dafür, dass das Steuerelement nicht vertieft angezeigt wird. **OLEDropMode = ccOLEDropManual** sorgt dafür, dass einige Ereignisse beim Drag and Drop auf das **ListView**-Steuerelement ausgelöst und implementiert werden können. Schließlich leert die Prozedur das **ListView**-Steuerelement, falls noch Einträge von vorherigen Anwendungen vorhanden sind.

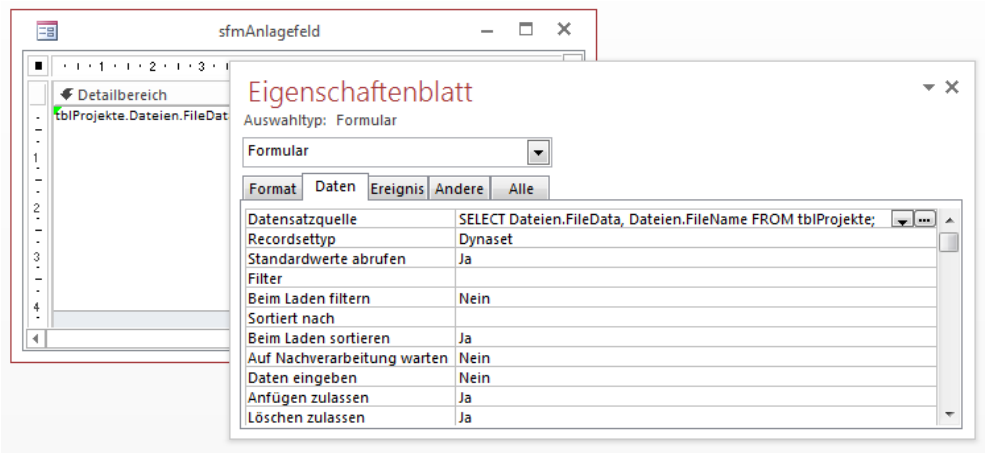

Bild 5: Das Unterformular **sfmAnlagefeld** in der Entwurfsansicht

Unterformular zur Ansicht der aktuell markierten Anlage

Das Unterformular **sfmAnlagefeld** enthält als einziges Steuerelement das Anlage-Steuerelement **ctlAttachment**. Das Formular selbst ist über die Eigenschaft **Datenherkunft** an eine Abfrage gebunden, welche die beiden Felder **Dateien.FileData** und **Dateien.FileName** liefert (s. Bild 5). Die »doppelten« Feldnamen mit dem trennenden Punkt erklären sich durch die Besonderheit des Anlagefeldes: Dieses hat in der Tabelle **tblProjekte** den Namen **Dateien**. Ein Anlagefeld enthält aber intern immer eine Tabelle, da es ja auch mehrere Anlagen samt Metadaten aufnehmen kann. Und zwei der Felder dieser Tabelle heißen **FileData** und **FileName**. Um direkt auf diese Felder zuzugreifen, verwendet man dann die Syntax **<Anlagefeldname>.<Feld der Anlagetabelle>**, also zum Beispiel **Dateien.FileData**.

Das Feld **Dateien.FileData** weisen wir direkt dem Anlage-Steuerelement **ctlAttachment** als **Steuerelementinhalt** zu, damit dieses den Inhalt der Datei oder ein entsprechendes Symbol anzeigt. Wofür aber benötigen wir das Feld **Dateien.FileName**? Dieses verwenden wir, um den aktuellen Datensatz des Unterformulars mit dem im **ListView**-Steuerelement des Hauptformulars angezeigten Anlage-Datensatz zu synchronisieren.

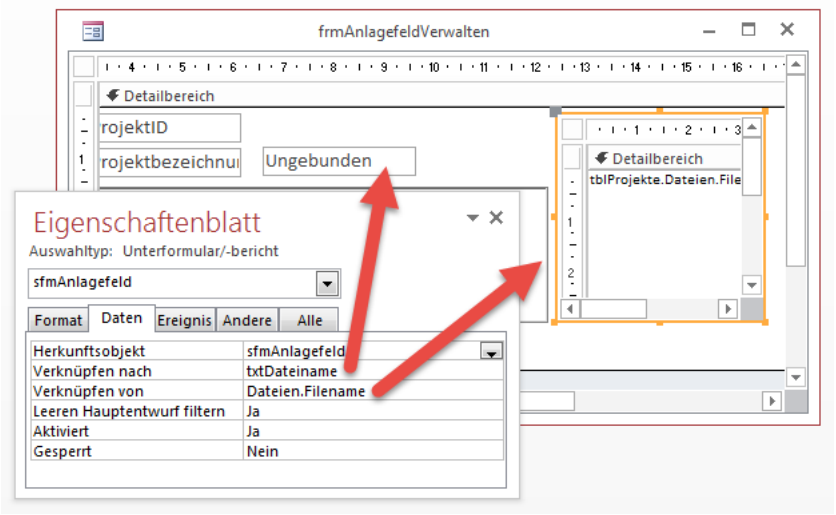

Bild 6: Einstellungen für das Unterformular-Steuerelement

Dazu verwenden wir die beiden Eigenschaften **Verknüpfen von** und **Verknüpfen nach** des Unterformular-Steuerelements, die sonst für die Verknüpfung von Daten in einer 1:n-Beziehung in Haupt- und Unterformular dienen. In diesem Fall stellen wir die Eigenschaft **Verknüpfen von** auf das Feld **Dateien.FileName** der Datenherkunft des Unterformulars und die Eigenschaft **Verknüpfen nach** auf

```
Private Sub Form_Current()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim rstDateien As DAO.Recordset
    Set db = CurrentDb
    If Not IsNull(Me!ProjektID) Then
        Set rst = db.OpenRecordset("SELECT Dateien.Filename FROM tblProjekte WHERE ProjektID = " & Me!ProjektID _
            & " ORDER BY Dateien.Filename", dbOpenDynaset)
        Me!ctlListview.ListItems.Clear
        Me!txtDateiname = rst("Dateien.Filename")
        Do While Not rst.EOF
            If Not IsNull(rst.Fields("Dateien.Filename")) Then
                Me!ctlListview.ListItems.Add , , rst.Fields("Dateien.FileName")
            End If
            rst.MoveNext
        Loop
    Else
        Me!txtDateiname = ""
        Me!ctlListview.ListItems.Clear
    End If
End Sub
```

Listing 2: Diese Ereignisprozedur wird beim Anzeigen eines jeden Datensatzes ausgelöst.

das Textfeld **txtDateiname** ein. Dieses finden Sie samt den Eigenschaften des Unterformular-Steuerelements in Bild 6.

Nun müssen wir nur noch klären, wie das Textfeld **txtDateiname** an den Namen des aktuell im **ListView**-Steuerelement ausgewählten Eintrags kommt.

Beim Anzeigen eines Datensatzes

Jeder Datensatzwechsel, und damit auch das erste Anzeigen eines Datensatzes im Formular **frmAnlagefeldVerwalten**, löst das Ereignis **Beim Anzeigen** und somit die Prozedur aus Listing 2 aus. Diese prüft zunächst, ob das Formular aktuell überhaupt einen Datensatz anzeigt. Dies ist der Fall, wenn **Me!ProjektID** den Wert **Null** aufweist und geschieht etwa dann, wenn der Benutzer zu einem neuen, leeren Datensatz wechselt. In diesem Fall soll dann einfach das **ListView**-Steuerelement geleert werden – siehe **Else**-Teil der **If...Then**-Bedingung.

Liegt jedoch ein Datensatz vor, erstellt die Prozedur ein neues Recordset, welches das Feld **Dateien.FileName** der Tabelle **tblProjekte** für alle Datensätze enthält, deren Feld **ProjektID** mit dem aktuellen Primärschlüsselwert des Datensatzes im Hauptformular übereinstimmt. Die Prozedur leert dann die Einträge des Listenfeldes.

Anschließend folgt eine Anweisung, die für die Synchronisierung mit dem Unterformular zur Anzeige der aktuell markierten Anlage verantwortlich ist: Sie stellt den Wert des Feldes **txtDateiname** auf den Dateinamen aus **rst("Dateien.FileName")** ein. Das Anlage-Steuerelement im Unterformular zeigt nun Inhalt des Anlagefeldes des ersten Datensatzes der Tabelle **tblProjekte** für das aktuellen Projekt an.

Nun durchläuft die Prozedur alle Datensätze der Abfrage und fügt dem **ListView**-Steuerelement jeweils einen neuen Eintrag hinzu, wobei dieser als Text jeweils den Inhalt des Feldes **Dateien.FileName** erhält.

Hinzufügen neuer Anlagen per Drag and Drop

Neue Dateien fügen Sie zum aktuellen Projekt hinzu, indem Sie diese aus dem Windows Explorer auf das

Verteiler für Berichte

Wer sich die Berichte seiner Access-Datenbank nicht in der Datenbank selbst ansehen, sondern anderen Personen präsentieren will, kommt nicht um einen Export herum. Dieser findet heutzutage in der Regel im PDF-Format statt. Einen einzelnen Bericht mal eben zu exportieren ist auch schnell erledigt. Anders sieht es aus, wenn immer mal wieder bestimmte Zusammenstellungen von Berichten exportiert werden sollen – einmal für das Controlling, einmal für den Chef et cetera. Dann kommt Ihnen die kleine Lösung aus diesem Beitrag zuhilfe, mit der Sie Verteiler für Berichte zusammenstellen und die Berichte per Mausklick in das gewünschte Verzeichnis exportieren.

Einfacher Export

Der übliche Weg zum Exportieren ist beispielsweise, im Navigationsfenster den zu exportierenden Bericht zu markieren und dann auf die rechte Maustaste zu klicken, um das Kontextmenü anzuzeigen. Daraus wählen Sie dann den Eintrag **ExportierenlPDF oder XPS** aus (s. Bild 1). Im folgenden **Datei auswählen**-Dialog legen Sie dann die Zieldatei für den Export im PDF- oder XPS-Format fest. Einen Mausklick später öffnet Access bereits die so angelegte Datei.

Export per VBA

Per VBA exportieren Sie einen Bericht am einfachsten mit der **OutputTo**-Methode des **DoCmd**-Objekts. Dieses erwartet die folgenden Parameter:

- **ObjectType**: Typ des Objekts, in diesem Fall **acOutputReport** für einen Bericht
- **ObjectName**: Name des Objekts, wie er im Datenbankfenster angegeben ist
- **OutputFormat**: Format des Exports, für unsere Anwendung **acFormatPDF**
- **OutputFile**: Pfad der zu erstellenden Datei
- **Autostart**: Gibt an, ob die Datei direkt in der entsprechenden Anwendung (hier meist Acrobat Reader) geöffnet werden soll. **True** öffnet das Dokument, bei **False** geschieht nichts (Standardwert).

Ein Beispiel für einen einfachen Aufruf sieht wie folgt aus:

```
DoCmd.OutputTo acOutputReport, "rptArtikel", acFormatPDF,
"c:\Export\Artikel.pdf", True
```

Dies erstellt das Dokument **Artikel.pdf** auf Basis des Berichts **rptArtikel** und öffnet dieses direkt (s. Bild 2).

Funktion der Lösung

Unsere kleine Lösung soll ein Formular anbieten, mit dem Sie verschiedene Verteilerlisten verwalten können.

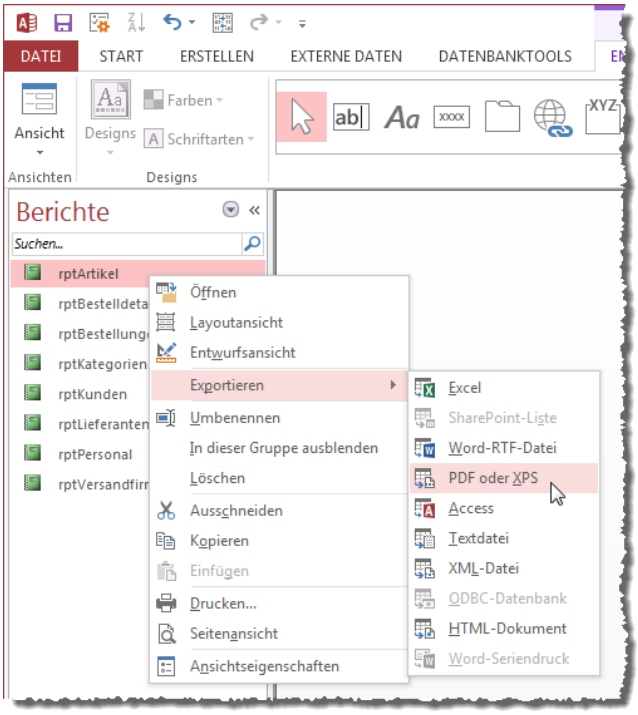

Bild 1: Export per Kontextmenü

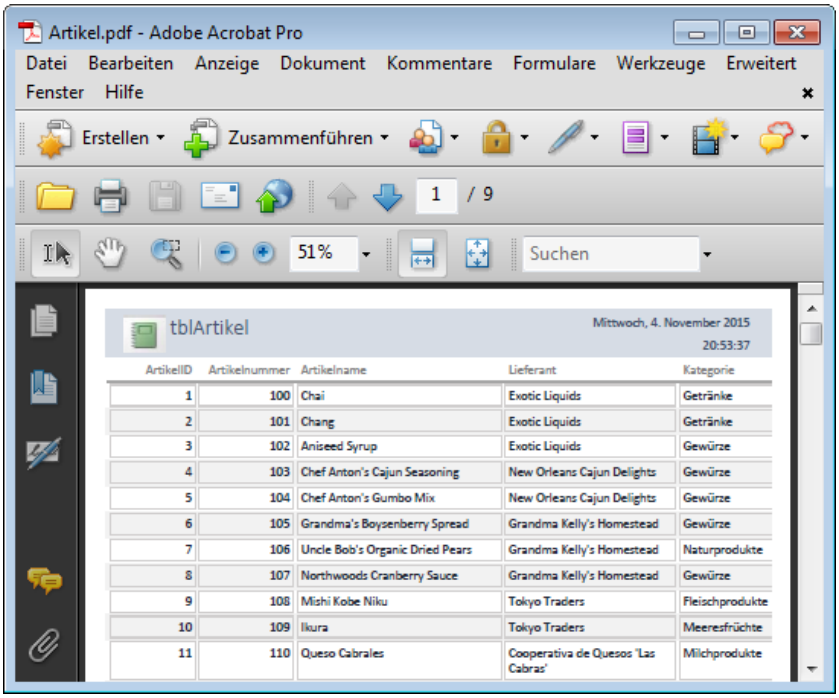

Bild 2: Frisch exportierter Bericht

Jeder Verteilerliste können Sie dabei einen oder mehrere der Berichte, die in der aktuellen Datenbank vorhanden sind, zuweisen. Ein Klick auf eine Schaltfläche soll dann alle Berichte für den aktuell angezeigten Verteiler in das PDF-Format exportieren. Das Formular sieht wie in Bild 3 aus. Es zeigt im oberen Bereich die Daten des aktuellen Verteilers an, der sich vor allem durch eine Bezeichnung und einen Export-Pfad auszeichnet. Den Pfad können Sie mit einem **Verzeichnis auswählen**-Dialog ermitteln, den Sie mit einem Klick auf die Schaltfläche rechts vom entsprechenden Textfeld öffnen.

Das Unterformular zeigt alle Berichte an, die aktuell zum Verteiler gehören. Hier werden der Berichtsname sowie der Dateiname angezeigt, unter dem die PDF-Datei erzeugt werden soll. Die Schaltfläche **Exportieren** schließlich sorgt für den Export der Berichte dieses Verteilers. Im Falle des hier abgebildeten Verteilers ergibt sich ein Ergebnis wie in Bild 4.

Datenmodell

Das Formular zeigt die Daten aus insgesamt drei Tabellen an, die Sie samt ihrer Beziehungen in Bild 5 sehen. Die Tabelle **tblVerteiler** enthält neben dem Primärschlüsselfeld die beiden Felder **Bezeichnung** und **Zielverzeichnis**.

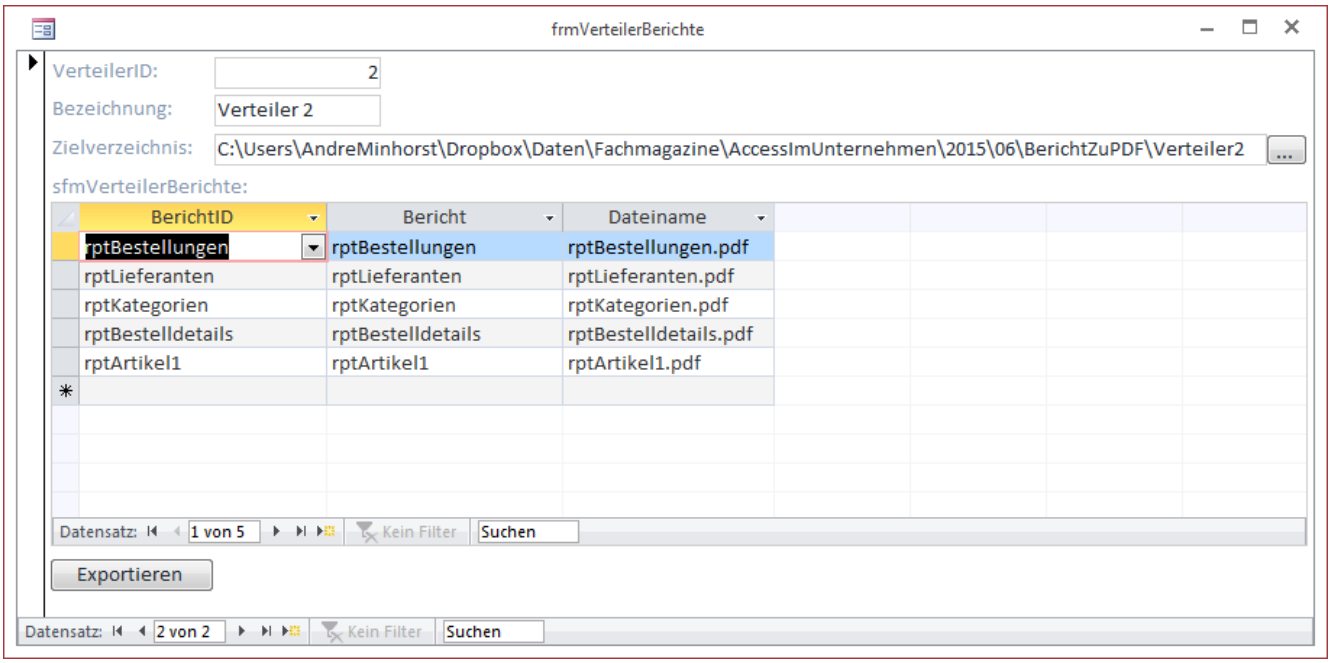

Bild 3: Das Formular zum Auswählen der Berichte, die in einem Verteiler landen und von dort exportiert werden sollen

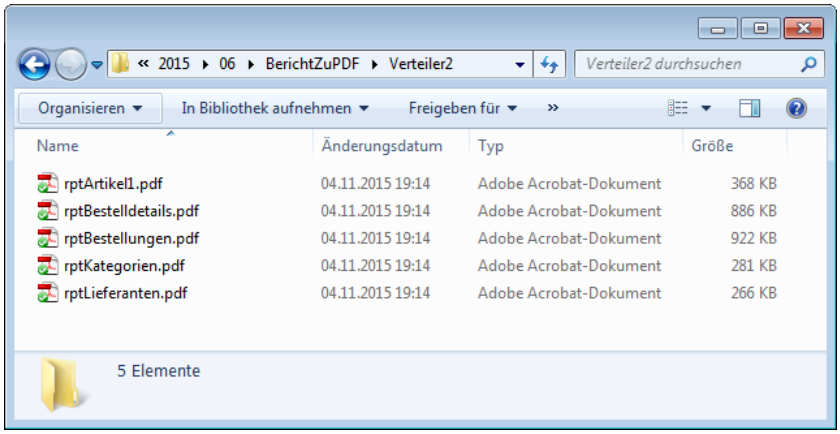

Bild 4: Die exportierten Berichte zu einem Verteiler

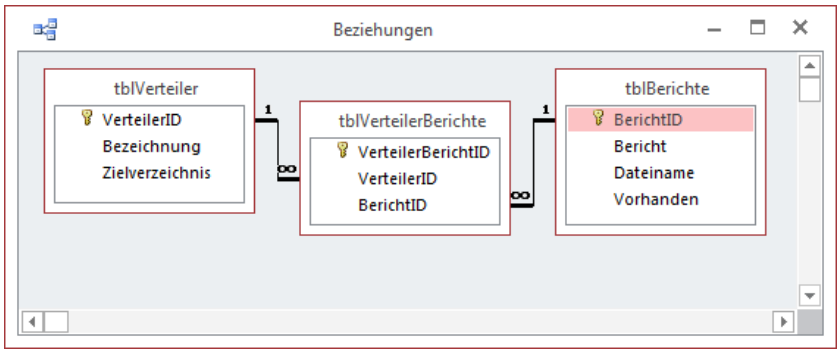

Bild 5: Datenmodell der Lösung

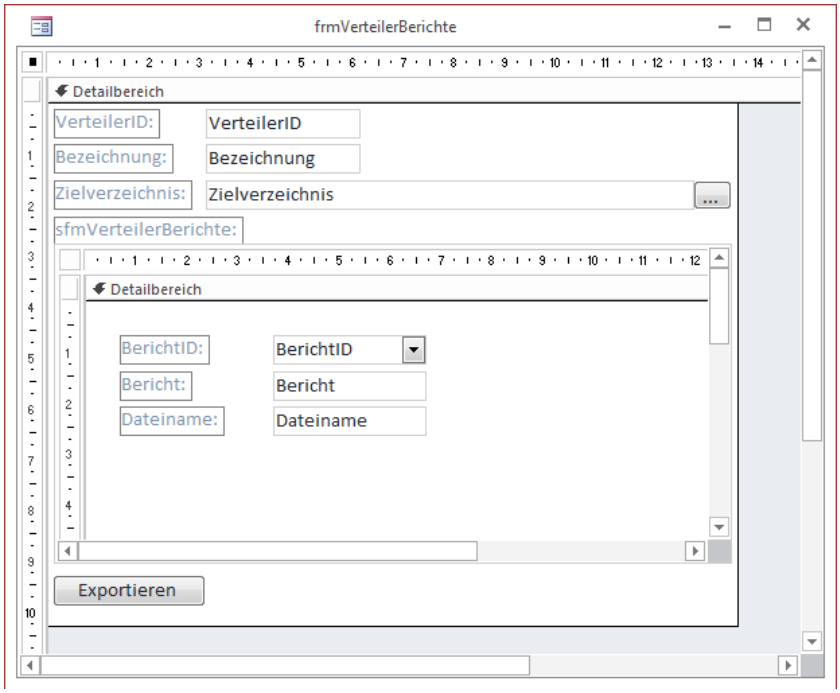

Bild 6: Das Formular **frmVerteilerBerichte** samt Unterformular in der Entwurfsansicht

Die Berichte speichern wir in der Tabelle **tblBerichte**, die im Feld **Bericht** den Namen des Berichts und in **Dateiname** den Namen der zu erstellenden Datei aufnimmt. Das Feld **Vorhanden** dient als Unterstützung beim Aktualisieren der Berichte, falls seit der letzten Verwendung des Formulars neue Berichte hinzugefügt oder alte entfernt wurden. Für das Feld **Bericht** legen wir einen eindeutigen Index fest, damit jeder Bericht nur einmal in der Tabelle auftauchen kann.

Die Tabelle **tblVerteilerBerichte** verknüpft die beiden Tabellen per m:n-Beziehung. Dazu enthält sie neben dem Primärschlüsselfeld je ein Fremdschlüsselfeld zu den beiden übrigen Tabellen. Hier ist wichtig, dass für die beiden Felder **VerteilerID** und **BerichtID** ein eindeutiger, zusammengesetzter Index festgelegt wird – so verhindern wir, dass ein Bericht mehr als einmal zu einem Verteiler hinzugefügt wird. Die beiden Verknüpfungsfelder sind als Nachschlagefelder definiert.

Erstellen des Formulars

Das Formular besteht aus dem Hauptformular **frmVerteilerBerichte** und dem Unterformular **sfmVerteilerBerichte**, die wie in Bild 6 angeordnet sind. Das Hauptformular verwendet die Tabelle **tblVerteiler** als Datenherkunft, das Unterformular eine Abfrage, die auf den beiden Tabellen **tblVerteilerBerichte** und **tblBerichte** basiert.

Die Schaltfläche **cmdOrdnerAuswaehlen** rechts neben dem Textfeld zur Eingabe des Zielverzeichnisses löst die folgende Prozedur aus:

Filterbedingungen einfach zusammenstellen

Erstellen Sie in Ihren Anwendungen mit VBA Filterausdrücke, um Formulare zu filtern oder Recordsets mit einer gefilterten Datenherkunft zu öffnen? Sind Ihnen dabei schon Fehler passiert, weil Sie die Werte nicht richtig konvertiert haben? Oder finden Sie den Code zum Filtern zu aufwendig und suchen eine Möglichkeit, die Erstellung des Filtertextes zu vereinfachen? Im vorliegenden Beitrag werden Sie verfolgen, wie eine Code-Struktur entsteht, die alle notwendigen Konvertierungsregeln berücksichtigt und trotzdem in der Anwendung übersichtlich ist und Programmierfehler vermeidet.

Problemstellung

In Access-Anwendungen wird relativ oft mittels VBA ein Filterausdruck zusammengestellt. Typische Beispiele sind:

- SQL-Anweisung zum Öffnen eines Recordset (**CurrentDb.OpenRecordset** und Ähnliche)
- **WHERE**-Parameter von **DoCmd.OpenForm** oder **DoCmd.OpenReport**
- Setzen der **Filter**-Eigenschaft in Formularen

Beim Zusammensetzen der SQL-Texte werden – wie Sie in vielen Beiträgen in Foren lesen können – regelmäßig Fehler gemacht. Es wird der Datentyp nicht SQL-konform in Text konvertiert, mögliche Null-Werte werden übersehen, statt dem Variableninhalt wird der Variablenname in den SQL-Ausdruck übernommen und so weiter. Damit Ihnen diese Fehler nicht passieren, beschreiben wir zuerst die Grundlagen für das Zusammenstellen von SQL-Anweisungen in VBA. Anschließend erstellen wir einen Code (in Form von Klassen), der die richtige Syntax für Sie erzeugt. Sie müssen den Klassen in Ihren Anwendungen nur noch Ihr Vorhaben mittels VBA-Anweisung mitteilen, um den gewünschten Filterausdruck zu erstellen.

Syntax in Abhängigkeit vom Datentyp

Bei der Konvertierung von Filterwerten in einen SQL-Text müssen Sie jeden Datentyp passend für das verwendete Datenbanksystem konvertieren. Die folgenden Absätze beschreiben dies für die relevanten Datentypen.

Text

Ein Text muss mit Hochkommata (**'**) umschlossen werden. Je nach Datenbanksystem sind auch Anführungszeichen (") oder Ähnliches möglich. Im weiteren Text werden wir zur Vereinfachung von Access/Jet-SQL für DAO ausgehen.

Beispiele:

```
... WHERE Textfeld = 'Filtertext'
... WHERE Textfeld LIKE 'Filtertext*'
```

Wenn im zu filternden Text ein Hochkomma (**'**) enthalten ist (Beispiel: **O'Neill**), muss das Zeichen verdoppelt werden, damit der SQL-Interpreter erkennen kann, dass der Text noch nicht zu Ende ist. Aus dem Ausdruck

```
… WHERE Textfeld = 'O'Neill'
```

wird dann der folgende Ausdruck (mit **O''Neill**):

```
… WHERE Textfeld = 'O''Neill'
```

Zahlen

Zahlen sind in englischer Schreibweise mit Punkt als Dezimaltrennzeichen zu verwenden.

```
... WHERE Zahlfeld = 1.23
```

Datum

Datumswerte sind im SQL-Text für Jet-SQL mit dem Raute-Zeichen (**#**) zu umschließen. Die Reihenfolge für Tag, Monat

und Jahr muss in amerikanischer Schreibweise (**mm/dd/yyyy**) oder im ISO-Format (**yyyy-mm-dd**) erfolgen. Die erforderlichen Datumsformate sehen je nach Datenbanksystem unterschiedlich aus. Beispiele für Jet-SQL:

```
… WHERE Datumsfeld = #12/24/2015#
… WHERE Datumsfeld = #2015-12-24#
… WHERE Datumsfeld = #2015-12-24 16:30:00#
```

T-SQL (SQL-Server):

```
… WHERE Datumsfeld = '20151224'
… WHERE Datumsfeld = '20151224 16:30:00'
```

Ja/Nein (Boolean)

In Access-Datenbanken wird **True** als **-1** in Zahlenschreibweise verwendet. Der SQL-Server (T-SQL) betrachtet **1** als **True** bei Bit-Datenfeldern. Im SQL-Text einer Access-Abfrage beziehungsweise DAO-Recordset und ähnlichen ist die Angabe von **True** statt **-1** zu empfehlen.

Dann gibt es auch bei Abfragen auf über ODBC verknüpfte SQL-Server-Tabellen keine Probleme.

```
… WHERE JaNeinFeld = True
```

Stolperfallen

Beim Zusammensetzen eines SQL-Textes entstehen die meisten Fehler beim Einbinden von Werten aus Variablen oder Steuerelementen in den SQL-Text.

- Die Konvertierung der Variablenwerte in einen SQL-Ausdruck wird nicht beziehungsweise falsch durchgeführt.
- Verdoppeln des Begrenzungszeichens vom Text (meist ') wird übersehen, falls das Zeichen im Text enthalten ist.
- Wenn die Zusammenstellung des SQL-Textes über mehrere VBA-Zeilen erfolgt, werden notwendige Leerzeichen übersehen.

Diese Stolperfallen können Sie problemlos vermeiden, wenn Sie die Vorgehensweise für das Zusammenstellen eines SQL-Textes verinnerlichen und die für Ihr Datenbanksystem gültige SQL-Syntax kennen.

Vom statischen Text zum dynamisch aus Werten in Steuerelementen erstellten Text

Wenn Sie nicht sicher sind, wie ein SQL-Text mit Filterwerten aus Steuerelementen oder Variablen zusammengesetzt wird, versuchen Sie die Erstellung in umgekehrter Reihenfolge anzugehen.

Nehmen Sie zum Ausprobieren der SQL-Anweisungen einen fixen Filterwert (welcher später etwa durch eine Benutzereingabe übergeben wird) an und erstellen im Access-Abfrageeditor eine SQL-Anweisung. Die Funktionsfähigkeit dieser SQL-Anweisung testen Sie nun. Wenn die Abfrage das erwartete Ergebnis liefert, wissen Sie, dass diese SQL-Anweisung passt.

Beispiel:

```
SELECT * FROM tabTest WHERE Textfeld LIKE 'abc*'
```

Stufe 1: Vollständiger SQL-Text in Variable

Nun kopieren Sie diese SQL-Anweisung und speichern sie in einer **String**-Variable in einer Test-Prozedur, wie in Listing 1 gezeigt, ab.

Zum Öffnen eines Recordsets mit dem in VBA gespeicherten SQL-Text und zur Ausgabe der enthaltenen Datensätze in den Direktbereich verwenden wir die Hilfsprozedur aus Listing 2.

Stufe 2: Filterausdruck als extra String verketten

Im nächsten Ausbauschritt (s. Listing 3) trennen wir den Filterausdruck von der restlichen SQL-Anweisung (**SqlText_VBA_Stufe2a**). Desweiteren speichern wir den Filterausdruck in einer zusätzlichen Variablen ab und verketten diese Variable mit dem restlichen SQL-Ausdruck (**SqlText_VBA_Stufe2b**). Zusätzlich zur Variable für den gesamten Filterausdruck können wir den Filterwert (im

```
Private Sub SqlText_VBA_Stufe1()
   Dim SqlText As String
   SqlText = "SELECT * FROM tabTest WHERE Textfeld LIKE 'abc*'"
   Call SqlAnweisungAuswerten(SqlText)
End Sub
```

Listing 1: VBA-Ausbaustufe 1

```
Private Sub SqlAnweisungAuswerten(ByVal SqlText As String)
   Debug.Print "SQL: "; SqlText
   With CurrentDb.OpenRecordset(SlText)
      Do While Not .EOF
         Debug.Print "ID: "; .Fields("idTest"), _
                     "Textfeld: "; .Fields("Textfeld")
         .MoveNext
      Loop
      .Close
   End With
End Sub
```

Listing 2: Hilfsprozedur zum Auswerten einer SQL-Anweisung

SQL-Format) in einer Stringvariable speichern (**SqlText_VBA_Stufe2c**).

Stufe 3: Filterwerte in passenden SQL-Text konvertieren

In den obigen Beispielen wurde der SQL-Text für den Filterwert im passenden SQL-Format in der Variable gespeichert.

Üblicherweise muss der Filterwert in SQL-Text konvertiert werden, weil die Werte zum Beispiel aus Steuerelementen in Formularen oder aus VBA-Variablen kommen, die nur den Wert und nicht dessen SQL-Ausdruck enthalten.

Ablauf des Codes in Listing 4: Aus dem String **abc*** soll der SQL-Ausdruck **'abc*'** entstehen. Dazu werden die Hochkommata (**'**) mit dem Filterwert verkettet (**"'" & "abc*" & "'"**). Von den anfangs genannten Regeln wissen wir, dass die im Filterwert enthaltenen Hochkommata verdoppelt werden müssen. Das erledigt die **Replace**-Funktion.

Bei anderen Datentypen können Sie die gleiche Vorgehensweise anwenden.

Sie müssen nur die Zeile **WertimSqlFormat = …** entsprechend anpassen, damit der Filterwert in das passende SQL-Format konvertiert wird.

Möglicher Umwandlungscode für Text:

```
WertImSqlFormat = "'" & Replace(FilterWert, "'", "''") & "'"
```

Für Zahlen:

```
WertImSqlFormat = Str(FilterWert)
```

```
Private Sub SqlText_VBA_Stufe2a()
    Dim SqlText As String
    SqlText = "SELECT * FROM tabTest WHERE " & "Textfeld LIKE 'abc*'"
    Call SqlAnweisungAuswerten(SqlText)
End Sub

Private Sub SqlText_VBA_Stufe2b()
   Dim SqlText As String
   Dim SqlFilter As String
   SqlFilter = "Textfeld LIKE 'abc*'"
   SqlText = "SELECT * FROM tabTest WHERE " & SqlFilter
   Call SqlAnweisungAuswerten(SqlText)
End Sub

Private Sub SqlText_VBA_Stufe2c()
   Dim SqlText As String
   Dim SqlFilter As String
   Dim WertImSqlFormat As String
   WertImSqlFormat = "'abc*'"
   SqlFilter = "Textfeld LIKE " & FilterWertImSqlFormat
   SqlText = "SELECT * FROM tabTest WHERE " & SqlFilter
   Call SqlAnweisungAuswerten(SqlText)
End Sub
```

Listing 3: VBA-Ausbaustufe 2

```
Private Sub SqlText_VBA_Stufe3()
   Dim SqlText As String
   Dim SqlFilter As String
   Dim WertImSqlFormat As String
   Dim FilterWert As String
   FilterWert = "abc*"
   WertImSqlFormat = "'" & Replace(FilterWert, "'", "''") & "'"
   SqlFilter = "Textfeld LIKE " & WertImSqlFormat
   SqlText = "SELECT * FROM tabTest WHERE " & SqlFilter
   Call SqlAnweisungAuswerten(SqlText)
End Sub
```

Listing 4: VBA-Ausbaustufe 3

```
Private Sub SqlText_VBA_Stufe4()
    Dim SqlText As String
    Dim SqlFilter As String
    Dim SqlFilterAusdruck As String
    Dim WertImSqlFormat As String
    Dim TextWert As Variant
    Dim DoubleWert As Double
    Dim DatumsWert As Date
    Dim BooleanWert As Boolean
    'Text
    TextWert = "abc*"
    WertImSqlFormat = "'" & Replace(TextWert, "'", "''") & "'"
    SqlFilterAusdruck = "Textfeld LIKE " & WertImSqlFormat
    SqlFilter = SqlFilterAusdruck
    'Zahlen
    DoubleWert = 1.23
    WertImSqlFormat = Trim(Str(DoubleWert))
    SqlFilterAusdruck = "DoubleZahlenfeld > " & WertImSqlFormat
    SqlFilter = SqlFilter & " And " & SqlFilterAusdruck
    'Datum
    DatumsWert = Date
    WertImSqlFormat = Format(DatumsWert, "\#yyyy-mm-dd\#")
    SqlFilterAusdruck = "Datumsfeld <= " & WertImSqlFormat
    SqlFilter = SqlFilter & " And " & SqlFilterAusdruck
    'Ja/Nein (Boolean)
    BooleanWert = True
    WertImSqlFormat = IIf(BooleanWert = True, "True", "False")
    SqlFilterAusdruck = "JaNeinFeld = " & WertImSqlFormat
    SqlFilter = SqlFilter & " And " & SqlFilterAusdruck
    'gesamten SQL-Text zusammenstellen
    SqlText = "SELECT * FROM tabTest WHERE " & SqlFilter
    Call SqlAnweisungAuswerten(SqlText)
End Sub
```

Listing 5: VBA-Ausbaustufe 4

Die **Str**-Funktion nicht mit der **CStr**-Funktion verwechseln!

Für Datumsangaben:

```
WertImSqlFormat = Format(FilterWert,
"\#yyyy-mm-dd\#")
```

Für Ja/Nein-Felder (**Boolean**):

```
WertImSqlFormat = IIf(FilterWert = True,
"True", "False")
```

Stufe 4: Filter kombinieren

Im nächsten Code-Beispiel (s. Listing 5) erstellen wir SQL-Filtertexte für verschiedene Datentypen und fügen sie zu einem kombinierten Filterausdruck zusammen.

Stufe 5: Filterwerte aus Formular-Steuerelementen

Häufig werden Formulare zum Filtern von Datenmengen verwendet. In diesen Formularen befinden sich Steuerelemente, in die der Anwender Werte eingeben kann. Wenn in einem Steuerelement ein Wert enthalten ist, sollen die Datensätze entsprechend gefiltert werden. Ist im Steuerelement kein Wert enthalten, soll dieser Filter ignoriert werden.

Typisches Code-Gerüst im Formular-Modul mit den Filtersteuerelementen:

```
With Me.Steuerelement
    If Not IsNull(.Value) then
        ' Filterausdruck erstellen
    End If
End With
```

Im Code in Listing 6 werden die Werte aus den Formular-Steuerelementen auf

vorhandene Filterwerte geprüft, aus den vorhandenen Einträgen Filterausdrücke erstellt und zu einem kombinierten Filterstring zusammengesetzt.

Nach diesem Prinzip können Sie Ihre Filterbedingungen zusammenstellen. Einen Nachteil hat diese Vorgehensweise: Der Code wird bei mehreren Filter-Feldern schnell unübersichtlich.

Ein Refactoring dieses Codes führt uns allerdings zu unserem Vorhaben, eine universell einsetzbare Code-Struktur zu gestalten, die statt dem Programmieren des Filtertextes ein Definieren der Filterbedingungen erlaubt und uns den fertigen SQL-Filtertext liefert.

Zuvor sehen wir uns noch ein bereits in Access eingebautes Hilfsmittel zum Erzeugen eines Filterausdrucks an.

```
Private Sub SqlText_VBA_Stufe5()
   Dim SqlText As String
   Dim SqlFilter As String
   Dim SqlFilterAusdruck As String
   Dim WertImSqlFormat As String
   'Text
   With Me.txtTextFilter
      If Len(.Value) > 0 Then ' Null und Leerstring ignorieren
         WertImSqlFormat = "'" & Replace(.Value, "'", "''") & "'"
         SqlFilterAusdruck = "Textfeld LIKE " & WertImSqlFormat
         SqlFilter = SqlFilter & " And " & SqlFilterAusdruck
      End If
   End With
   'Zahlen
   With Me.txtZahlenFilter
      If Not IsNull(.Value) Then
         WertImSqlFormat = Trim(Str(.Value))
         SqlFilterAusdruck = "DoubleZahlenfeld > " & WertImSqlFormat
         SqlFilter = SqlFilter & " And " & SqlFilterAusdruck
      End If
   End With
   'Datum
   With Me.txtDatumsFilter
      If Not IsNull(.Value) Then
         WertImSqlFormat = Format(.Value, "\#yyyy-mm-dd\#")
         SqlFilterAusdruck = "Datumsfeld <= " & WertImSqlFormat
         SqlFilter = SqlFilter & " And " & SqlFilterAusdruck
      End If
   End With
   'Filterausdruck aufbereiten
   If Len(SqlFilter) > 0 Then
      ' ersten " And "-Ausdruck wegschneiden
      SqlFilter = Mid(SqlFilter, Len(" And ") + 1)
   End If
   'gesamten SQL-Text zusammenstellen
   SqlText = "SELECT * FROM tabTest"
   If Len(SqlFilter) > 0 Then
      SqlText = SqlText & " WHERE " & SqlFilter
   End If
   Call SqlAnweisungAuswerten(SqlText)
End Sub
```

Listing 6: VBA-Ausbaustufe 5

Hilfsfunktion Access.BuildCriteria

Die Methode **BuildCriteria** vom Access-Objekt erlaubt eine Übergabe der Filterwerte als String. Dabei müssen keine SQL-relevanten Zeichen zum Eingrenzen eines Textes oder Datums ergänzt werden. **BuildCriteria** erzeugt die Zeichen in Abhängigkeit vom **FieldType**-Parameter selbst.

Syntax: **BuildCriteria(Field, FieldType, Expression)**

Beispiele finden Sie in Listing 7.

Ein Nachteil dieser Hilfsfunktion ist, dass die Funktion nur einen Filterausdruck für Jet/DAO-SQL erzeugt. Falls Sie einen SQL-Text für den SQL-Server in T-SQL für eine Pass-Through-Abfrage benötigen, können Sie diese

Projektzeiterfassung mit Outlook und Access

Die Erfassung von Projektzeiten ist eine Aufgabe, die man gern etwas stiefmütterlich handhabt. Am Ende des Tages, der Woche oder auch des Monats muss man dann die für verschiedene Aufgaben verbrauchte Zeit aus dem Gedächtnis nachtragen. Dem wollen wir mit einer Lösung, die Outlook und Access nutzt, entgegenwirken. Damit legen Sie eine Tätigkeit einer bestimmten Dauer – in diesem Fall 25 Minuten – ganz einfach per Drag and Drop in den Outlook-Kalender an. Der Rest geschieht im Hintergrund: Die Tätigkeit wird samt Projekt und Aufgabe in einer Access-Datenbank gespeichert und kann anschließend bequem ausgewertet werden.

Der Clou bei dieser Lösung ist, dass es wirklich nur weniger Vorbereitungen sowie einer Drag-and-Drop-Aktion bedarf, um schnell eine Tätigkeit zu dokumentieren. Die grundlegende Konfiguration erfordert Outlook und Access ab der Version 2007.

Wenn Sie Outlook öffnen, stellen Sie zunächst die Ansicht der linken Leiste auf **Ordner** um (s. Bild 1).

Danach erscheint links eine Übersicht der einzelnen Outlook-Elemente als Ordner – **Posteingang**, **Entwürfe**, **Gesendete Elemente** und so weiter. Für uns ist der Ordner **Aufgaben** interessant.

Dort legt die Lösung aus diesem Beitrag beim Starten von Outlook automatisch einen Unterordner namens **Projekte** an. Mit dem Eintrag **Neuer Ordner...** des Kontextmenüs dieses Eintrags legen Sie einen Unterordner an, der ein neues Projekt repräsentiert (s. Bild 2).

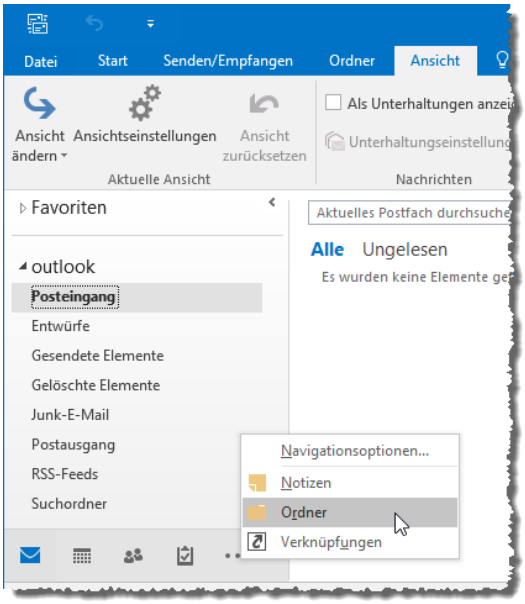

Bild 1: Wechsel auf die **Ordner**-Ansicht

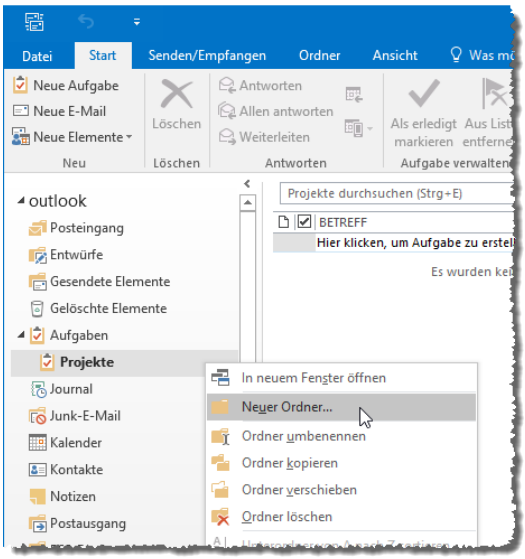

Bild 2: Anlegen eines neuen Projekt-Ordners

Zu Beispielzwecken wollen wir diesen Ordner **Projekt 1** nennen.

Ein Projekt soll eine oder mehrere Aufgaben aufnehmen. Damit Sie diese anlegen können, klicken Sie den frisch angelegten Ordner **Projekt 1** an. Dies blendet die aktuell für diesen Aufgabenordner vorhandenen Aufgaben im Hauptbereich von Outlook ein. Hier fügen Sie nun wie in Bild 3 die gewünschten Aufgaben hinzu – im **Beispiel Aufgabe 1** und **Aufgabe 2**.

Nun können wir schon zur Kalenderansicht wechseln. Dabei müssen Sie allerdings etwas beachten: Wenn Sie per Mausklick auf das Kalender-Icon unten links zum Kalender wechseln, verschwindet auch die Ordneransicht wieder – und die sollte angezeigt bleiben, damit wir auch während der Arbeit mit dem Kalender Zugriff auf die Projektordner haben. Also nutzen Sie für den Wechsel zur Kalenderansicht den Eintrag **Kalender** aus der Ordnerliste.

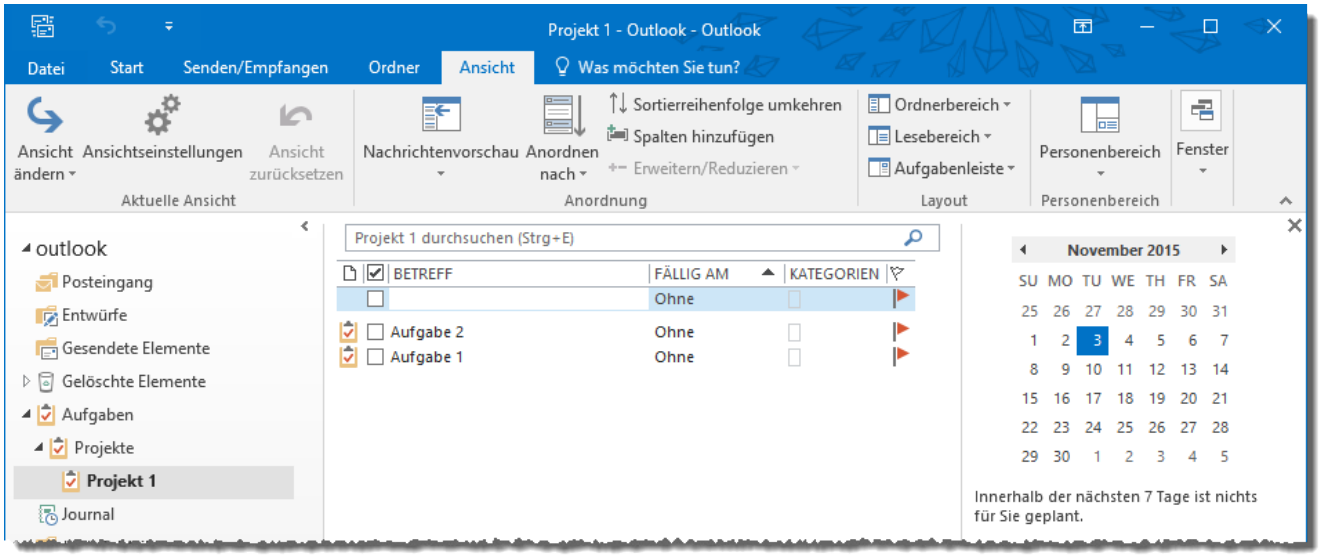

Bild 3: Hinzufügen von Aufgaben zu einem Projekt

Hier prüfen wir noch, ob der rechte Bereich die Aufgabenliste anzeigt. Ist dies nicht der Fall, können Sie dies etwa unter Outlook 2016 mit dem Ribbon-Eintrag **Ansicht|Layout|Aufgabenliste|Aufgaben** einblenden (s. Bild 4). Hier sehen Sie auch, wie Sie dann eine der Aufgaben aus der Aufgabenliste in den Kalenderbereich ziehen. Die Aufgabe wird dann als neue Tätigkeit genau an der Stelle angelegt, die dem aktuellen Datum und der Uhrzeit entspricht. Fertig – für die nächsten 25 Minuten steht die anstehende Aufgabe nun fest und Sie können diese bearbeiten. Danach geht es auf die gleiche Weise weiter – legen Sie eine neue Aufgabe an und bearbeiten

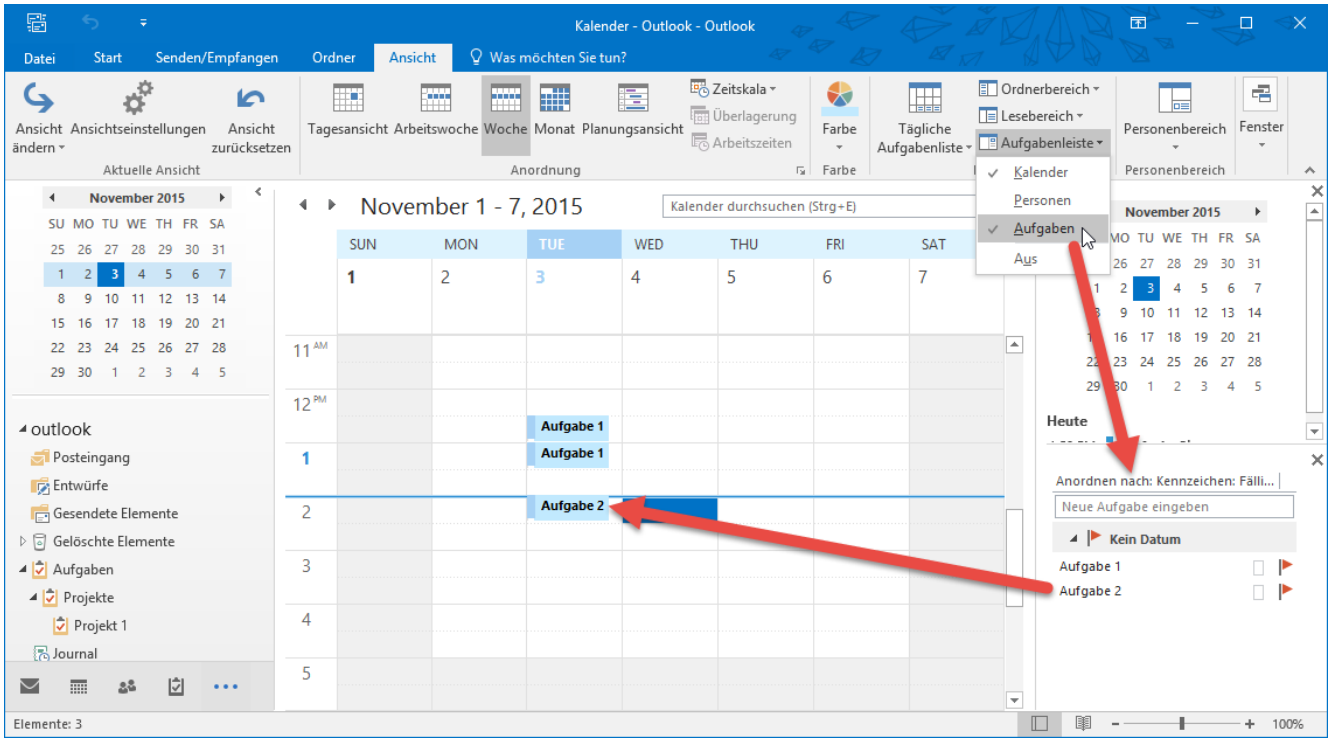

Bild 4: Einblenden der Aufgabenleiste und Hinzufügen von Aufgaben als Tätigkeiten zum Kalender

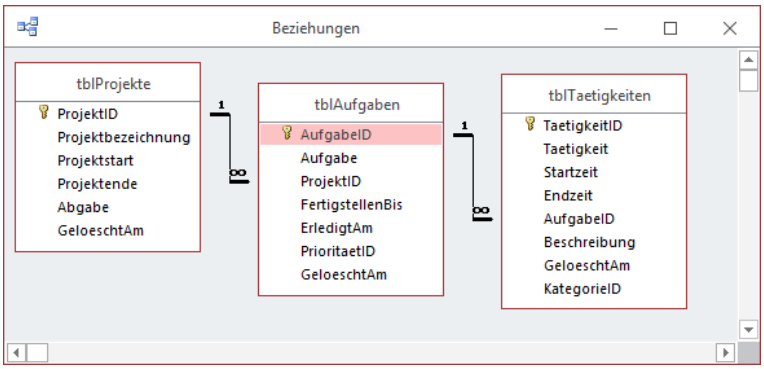

Bild 5: Datenmodell der Datenbank zur Projektzeiterfassung

Sie diese oder bearbeiten Sie einfach eine der vorhandenen Aufgaben weiter.

Datenbank zum Speichern der Projektzeitdaten

Dies war der Teil der Lösung, der sich auf die Benutzeroberfläche bezieht. Im Hintergrund geschieht noch einiges mehr: Zum Beispiel gibt es eine Access-Datenbank namens **Projektzeiterfassung.accdb**, welche die erfassten Daten in drei Tabelle speichert und die um Funktionen zum Auswerten der Projektzeiten erweitert werden kann.

Datenmodell der Datenbank

Das Datenmodell dieser Datenbank sieht wie in Bild 5 aus. Die Tabelle **tblProjekte** nimmt für jeden Ordner, den Sie unterhalb des Ordners Projekte anlegen, einen Datensatz auf und speichert dort die eingegebene Projektbezeichnung. Die übrigen Felder können nach Belieben genutzt werden, interessant ist lediglich noch das Feld **GeloeschtAm**. Dieses wird gefüllt, wenn der Benutzer den Projektordner in Outlook löscht.

Die Tabelle **tblAufgaben** speichert die einzelnen Aufgaben, die Sie im mittleren Bereich von Outlook hinzufügen, wenn Sie einen der Projektordner markiert haben. Hier finden Sie das Fremdschlüsselfeld **ProjektID**, das die Aufgabe einem Datensatz der Tabelle **tblProjekte** zuordnet. Das Feld **Aufgabe** ist für die Bezeichnung der Aufgabe wie in Outlook angegeben gedacht. Auch hier finden Sie wieder ein Feld namens **GeloeschtAm**, das beim Löschen des entsprechenden Elements aus Outlook mit dem aktuellen Datum und der Zeit gefüllt wird. Das Feld **ErledigtAm** schließlich wird gefüllt, wenn der Benutzer die Aufgabe in Outlook als erledigt markiert. Dies können Sie beispielsweise in der Übersichtsliste der Aufgaben erledigen (s. Bild 6). Eine andere Möglichkeit bietet der Eintrag **Als erledigt markieren** im Kontextmenü der Aufgabenliste in der Kalenderansicht.

Die Tabelle **tblTaetigkeiten** speichert schließlich die Aufgaben, die der Benutzer als Termine in den Kalender gezogen hat. Die Tabelle enthält einige Felder mehr als die zuvor vorgestellten Tabellen. Beim Anlegen einer Tätigkeit trägt die Lösung die Bezeichnung der Tätigkeit, die Startzeit, die Endzeit sowie den Wert des Primärschlüsselfeldes der Aufgabe, zu deren Erledigung die Tätigkeit dient, in die Tabelle ein. Außerdem gibt es auch hier wieder ein Feld namens **GeloeschtAm**, welches das Löschdatum von entfernten Tätigkeiten aufnimmt. Das Feld **KategorieID**, das Sie in der Abbildung des Datenmodells sehen, benötigen wir für eine Erweiterung der hier vorgestellten Lösung aus dem Beitrag **Kanban mit Outlook und Access** (**www.access-im-unternehmen.de/1017**, siehe folgende Ausgabe).

Bild 7 zeigt die drei Tabellen mit den Beispieldatensätzen, die wir weiter oben angelegt haben, in der Datenblatt-

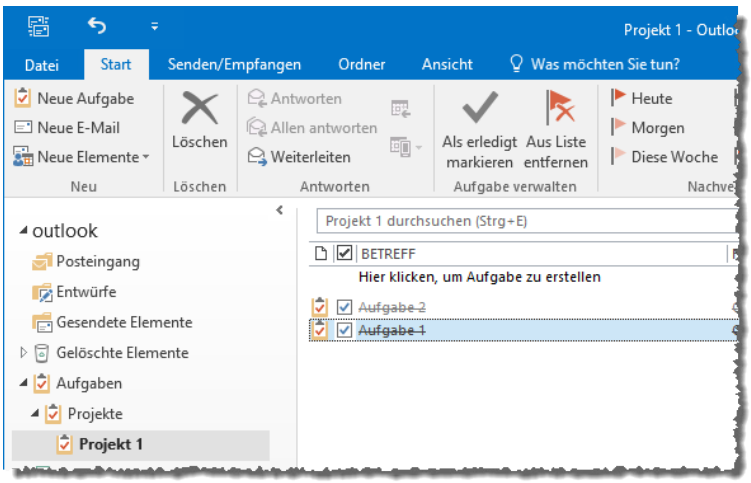

Bild 6: Markieren einer Aufgabe als »Erledigt«

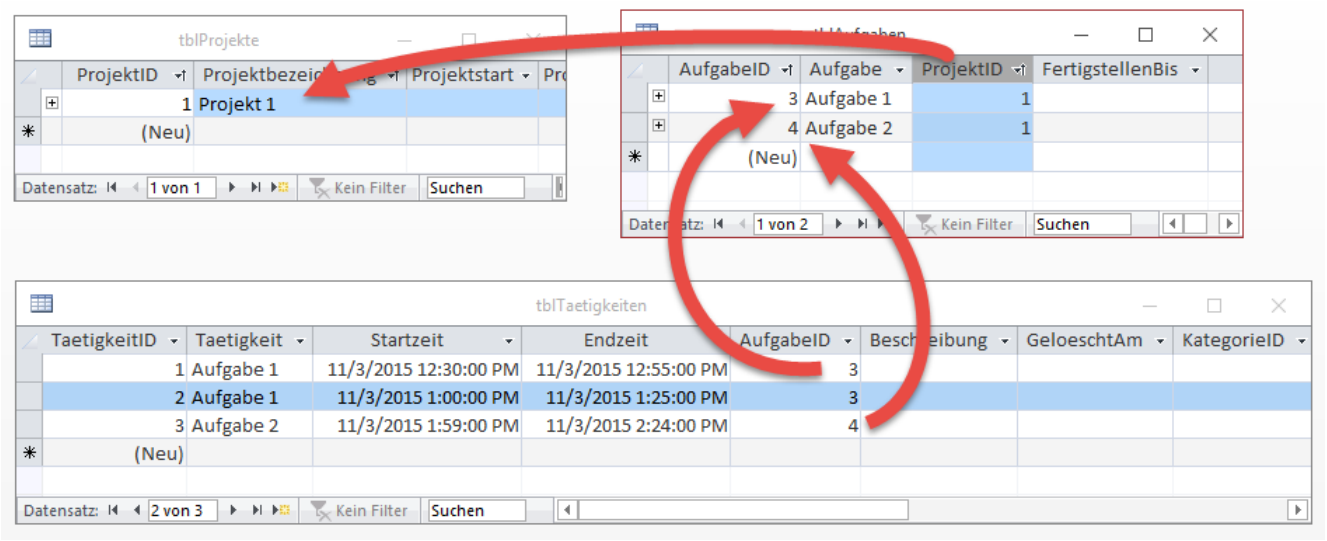

Bild 7: Tabellen der Lösung mit einigen Beispieldaten

ansicht und deutet die Verknüpfungen der vorhandenen Datensätze durch rote Pfeile an.

Programmierung der Lösung

Der Teil der Lösung, der in diesem Beitrag beschrieben wird, spielt sich VBA-technisch komplett in Outlook ab – die Access-Datenbank stellt nur die Tabellen zum Speichern der Projektzeiten bereit. In einem weiteren Beitrag namens **Projektzeiten auswerten** (**www.access-im-unternehmen.de/1018**) zeigen wir in der folgenden Ausgabe, wie Sie die gespeicherten Daten auswerten können.

Hier schauen wir uns nun an, wie die in Outlook erstellten Projekte, Aufgaben und Tätigkeiten ihren Weg in die Tabellen der Access-Datenbank finden.

VBA-Editor unter Outlook öffnen

Ich weiß nicht, wie es Ihnen geht, aber ich öffne den VBA-Editor unter Access immer mit der Tastenkombination **Strg + G** und lande damit sofort im Direktbereich. Unter Outlook funktioniert diese nicht.

Dort gelangen Sie am schnellsten über die Tastenkombination **Alt + F11** in den VBA-Editor (dies funktioniert unter Access und den übrigen Office-Anwendungen übrigens auch).

Sicherheit

Je nachdem, wie Ihre Sicherheitseinstellungen unter Outlook aussehen, können Sie keine VBA-Prozeduren im VBA-Projekt von Outlook ausführen. Aber keine Sorge – dies ändern Sie ganz schnell. Öffnen Sie den Optionen-Dialog von Outlook, wechseln Sie dort zum Bereich **Trust Center** und klicken Sie auf **Einstellungen für das Trust Center**. Hier übernehmen Sie die Option für den Bereich Makroeinstellungen wie in Bild 8.

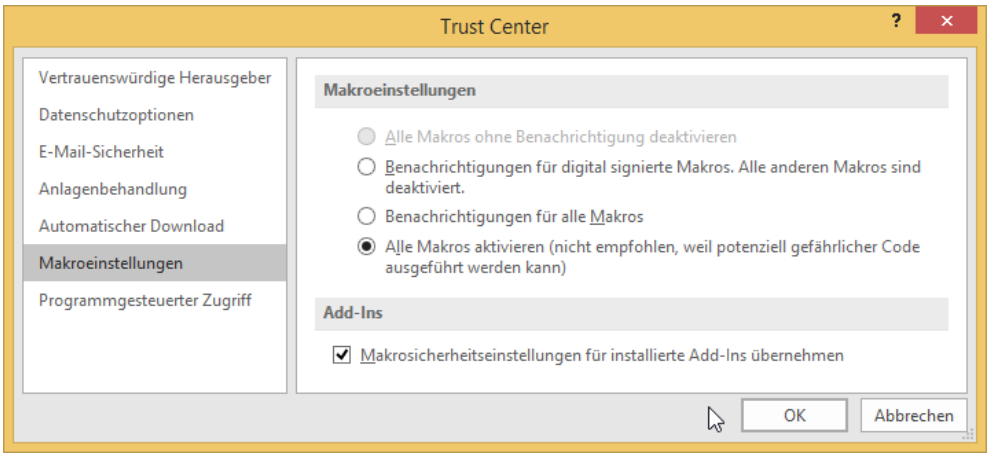

Bild 8: Einstellungen zum Ausführen von VBA-Code in Outlook

```
Public Sub Application_Startup()
    On Error GoTo Application_Startup_Err
    Dim objProject As Outlook.Folder
    Dim objTasks As clsTasks
    Dim lngError As Long
    Set objAppointments = GetMAPINamespace.GetDefaultFolder(olFolderCalendar).Items
    On Error Resume Next
    Set objProject = GetTaskfolder.Folders("Projekte")
    If Not Err.Number = 0 Then
        GetTaskfolder.Folders.Add "Projekte", olFolderTasks
    End If
    Set objProjects = GetTaskfolder.Folders("Projekte").Folders
    On Error GoTo Application_Startup_Err
    Set objAppointmentFolder = GetMAPINamespace.GetDefaultFolder(olFolderCalendar)
    Set colProjects = New Collection
    For Each objProject In objProjects
        Set objTasks = New clsTasks
        Set objTasks.Tasks = objProject.Items
        Set objTasks.Project = objProject
        colProjects.Add objTasks
    Next
Application_Startup_Exit:
    On Error Resume Next
    Exit Sub
Application_Startup_Err:
    Call Fehlerbehandlung("Projekt1/ThisOutlookSession", "Application_Startup", _
        Erl, "Bemerkungen: ./.")
    GoTo Application_Startup_Exit
End Sub
```

Listing 1: Diese Prozedur wird bei Start von Outlook automatisch ausgelöst.

Achtung: Die Änderungen wirken sich erst beim nächsten Start von Outlook aus.

Wichtigstes Element: Ereignisse

Eine sehr wichtige Rolle spielen dabei die Ereignisse, die beim Anlegen von Projektordnern, Aufgaben und Terminen/Tätigkeiten ausgelöst werden. Weitere Ereignisse, die wir tracken müssen, sind das Löschen dieser drei Objektarten sowie das Verschieben, also das Ändern der Eigenschaften von Objekte – in diesem Fall betrifft das nur die Termine/Tätigkeiten. Die Tätigkeiten sollen ja, wenn der Benutzer diese auf den Kalender zieht, immer genau für das aktuelle Datum und die entsprechende Zeit angelegt werden. Manchmal vergisst man jedoch, eine Tätigkeit vor Beginn in Outlook anzulegen und fügt diese nachträglich hinzu. Diese muss dann natürlich nach dem Anlegen noch an die der gewünschten Zeit entsprechende Stelle verschoben werden, üblicherweise per Drag and Drop oder durch manuelles Einstellen der Zeit nach dem Öffnen des Termins per Doppelklick.

Eine solche Änderung muss ebenfalls ein Ereignis auslösen, damit die neuen Zeiten die beim Anlegen in der Tabelle gespeicherten Zeiten überschreiben können. Aber auch die Projektordner und die Aufgaben sind Änderungen unterworfen. Diese beziehen sich jedoch nur auf die jeweilige Bezeichnung.

Damit wir die Ereignisse der drei Objektarten durch entsprechende Ereignisprozeduren implementieren können, benötigen wir jeweils entsprechende Objektvariablen, die wir mit dem Schlüsselwort **WithEvents** deklarieren. Da nach dem Öffnen von Outlook nicht immer nur neue Projekte, Aufgaben und Tätigkeiten angelegt, bearbeitet und gelöscht werden, sondern auch die bestehenden Elemente potenziell Änderungen unterworfen sind, müssen wir für alle infrage kommenden Elemente Objektvariablen anlegen und diese mit den entsprechenden Ereignisprozeduren ausstatten.

Wir benötigen also beim Start von Outlook zunächst eine Prozedur, die alle vorhandenen Elemente in Form von Objektvariablen referenziert. Diese Prozedur heißt **Application_Startup** und landet im standardmäßig im VBA-

Projekt von Outlook vorhandenen Klassenmodul **ThisOutlookSession** (s. Listing 1).

VBA-Projekt von Outlook

Bezüglich des VBA-Projekts von Outlook müssen Sie ein wenig von der Access-Denkweise abweichen: Unter Access hat ja jede Datenbankdatei ein eigenes VBA-Projekt – genau wie die Dokumente in Word oder Excel. Unter Outlook gibt es nur ein VBA-Projekt, das der Anwendung Outlook selbst zugeteilt ist.

Fehlerbehandlung

Die Prozedur enthält eine Fehlerbehandlung, die alle Fehler in Form einer entsprechenden Meldung behandelt. Dies dient primär dazu, die Lösung stabil zu gestalten. Üblicherweise werden beispielsweise Objektvariablen bei nicht behandelten Fehlern geleert, was im Falle dieser Lösung natürlich katastrophal wäre: Wenn der Benutzer nach einem nicht behandelten Fehler fleißig weiter Projekte, Aufgaben und Tätigkeiten bearbeitet, wird nämlich keines der mit den Objektvariablen verbundenen Ereignisse mehr ausgelöst und die Daten werden nicht in der Datenbank gespeichert.

Die Prozedur **Application_Startup** stellt nun zunächst die Variable **objAppointments** des Typs **Items** auf die im Kalender-Ordner enthaltenen Elemente ein. Die entsprechende Variable deklarieren wir wie folgt im Kopf des Klassenmoduls:

```
Dim WithEvents objAppointments As Outlook.Items
```

Um einen leichteren Zugriff auf die Elemente von Outlook zu schaffen, haben wir im Modul **mdlOutlook** ein paar Hilfsfunktionen beigefügt. Hier kommen gleich mehrere zum Einsatz. Die erste heißt **GetMAPINamespace** und liefert einen Verweis auf das MAPI-Namespace-Objekt von Outlook, über das wir auf alle Elemente von Outlook zugreifen können (siehe unten). Das von **GetMAPINamespace** gelieferte Objekt bietet die Methode **GetDefaultFolder**, die mit dem Parameter **olFolderCalendar** einen Verweis auf den Standard-Kalenderordner von Outlook holt und speichert ihn in **objAppointments**.

Nun folgt ein Schritt, der dem Benutzer ein wenig Arbeit abnehmen soll: Wenn unterhalb des Aufgaben-Ordners in der Ordnerliste von Outlook noch kein Eintrag namens **Projekte** vorhanden ist, soll die

```
Dim mOutlook As Outlook.Application
Dim mNamespace As Outlook.NameSpace
Dim mTaskFolder As Outlook.Folder

Public Property Get GetOutlook() As Outlook.Application
    If mOutlook Is Nothing Then
        Set mOutlook = Application
    End If
    Set GetOutlook = mOutlook
End Property

Public Property Get GetMAPINamespace() As Outlook.NameSpace
    If mNamespace Is Nothing Then
        Set mNamespace = GetOutlook.GetNamespace("MAPI")
    End If
    Set GetMAPINamespace = mNamespace
End Property

Public Property Get GetTaskfolder() As Outlook.Folder
    If mTaskFolder Is Nothing Then
        Set mTaskFolder = GetMAPINamespace.GetDefaultFolder(olFolderTasks)
    End If
    Set GetTaskfolder = mTaskFolder
End Property
```

Listing 2: Hilfsfunktionen für den einfachen Zugriff auf Outlook-Objekte

Prozedur diesen automatisch anlegen. Dazu referenziert die Prozedur bei deaktivierter Fehlerbehandlung diesen Ordner über das Element **Projekte** der **Folders**-Auflistung für den Aufgaben-Ordner.

Den Aufgaben-Ordner liefert ebenfalls eine unserer Hilfsfunktionen aus dem Modul **mdlOutlook**, nämlich **GetTaskfolder** (siehe unten). Führt dieser Zugriff auf den Ordner zu einem Fehler, ist offensichtlich, dass der Ordner noch nicht vorhanden ist. Die Prozedur verwendet dann die **Add**-Methode der **Folders**-Auflistung des mit **GetTaskfolder** ermittelten Aufgaben-Ordners, um den Ordner namens **Projekte** hinzuzufügen. Danach speichert sie einen Verweis auf die **Folders**-Auflistung dieses Ordners in der Variablen **objProjects**, die ebenfalls mit dem Schlüsselwort **WithEvents** im Kopf des Klassenmoduls deklariert wird:

```
Dim WithEvents objProjects As Outlook.Folders
```

Die nächste auf diese Variable namens **objAppointmentFolder** füllt die Prozedur mit einem Verweis auf den Kalender-Ordner:

```
Dim WithEvents objAppointmentFolder As Outlook.Folder
```

Auch diesen erhalten wir mit der **GetDefaultFolder**-Methode des mit **GetMAPINamespace** ermittelten MAPI-Namespace-Objekts.

Für die Implementierung weiterer Ereignisse benötigen wir auch noch Objekte auf Basis der Ordner, die der Benutzer unterhalb des **Projekte**-Ordners anlegt und die jeweils ein Projekt repräsentieren. Dies ist etwas aufwendiger, da wir jeden einzelnen Ordner referenzieren müssen. Dies erledigen wir mit einer Hilfsklasse, die jeweils einen Ordner aufnimmt und die dann in einer Collection namens **colProjects** landet. Diese deklarieren wir wie folgt im Kopf des Klassenmoduls:

```
Dim colProjects As Collection
```

Die Prozedur erzeugt diese Collection neu. Dann durchläuft sie alle Ordner im Folder **Projekte** in einer **For Each**-Schleife über die Elemente der Auflistung **objProjects** und legt für jeden Ordner ein neues Objekt der Klasse **clsTasks** an. Diese weist eine Eigenschaft namens **Tasks** auf, die mit der **Items**-Auflistung des aktuellen Projekte-Ordners gefüllt wird. Der Verweis auf den aktuellen Projekt-Ordner selbst landet in der Eigenschaft **Project**. Schließlich wird das so präparierte Objekt der Collection **colProjects** hinzugefügt.

Das waren bereits die Vorbereitungen. Nun müssen wir noch die zahlreichen Ereignisprozeduren implementieren, damit die in Outlook eingegebenen Projekte, Aufgaben und Tätigkeiten auch in den Tabellen der Datenbank landen.

Die Hilfsfunktionen GetOutlook, GetMAPINamespace und GetTaskFolder

Die drei Hilfsfunktionen aus Listing 2 finden Sie im Standardmodul **mdlOutlook** im VBA-Projekt von Outlook – dort natürlich mit Fehlerbehandlung, die wir hier aus Platzgründen gekürzt haben. Die erste Funktion liefert einen Verweis auf das **Outlook.Application**-Objekt. Dabei prüft sie, ob die lokale Variable **mOutlook** bereits mit einem

```
Private WithEvents mTasks As Outlook.Items
Private WithEvents mProject As Outlook.Folder

Public Property Set Tasks(objTasks As Outlook.Items)
    Set mTasks = objTasks
End Property

Public Property Set Project(objProject As Outlook.Folder)
    Set mProject = objProject
End Property
```

Listing 3: Elemente der Klasse **clsTasks**

Verweis auf das **Application**-Objekt gefüllt ist. Falls nicht, was eigentlich nur beim ersten Aufruf geschieht, füllt es diese Variable damit. Die Funktion gibt dann einen Verweis auf den Inhalt von **mOutlook** zurück.

Die Funktion **GetMAPINamespace** erledigt dies mit der Variablen **mNamespace** und nutzt dazu die **GetNamespace**-Methode des **Application**-Objekts, das es mit der Funktion **GetOutlook** holt.

Fehlt noch die Funktion **GetTaskfolder**, die einen Verweis auf den Aufgaben-Ordner von Outlook ermittelt und zurückgibt. Dabei nutzt diese die beiden zuvor beschriebenen Funktionen sowie die lokale Variable **mTaskfolder**.

Die Klasse clsTasks

Diese Klasse haben Sie soeben bereits kennen gelernt, als wir für jedes **Folder**-Objekt eine Instanz dieser Klasse erstellt und über die Eigenschaften **Tasks** und **Project** jeweils einen Verweis auf die Aufgaben-Auflistung und den Projekt-Ordner übergeben haben.

Diese beiden Eigenschaften stellt die Klasse über die beiden **Property Set**-Prozeduren **Tasks** und **Project** bereit, die wie in Listing 3 aussehen.

Damit die übergebenen Verweise in der Klasse gespeichert werden, tragen die **Property Set**-Prozeduren diese in die beiden Variablen **mTasks** und **mProject** ein. Beide deklarieren wir wieder mit dem Schlüsselwort **WithEvents**, denn auch für diese wollen wir wieder Ereignisprozeduren anlegen. Diese stellen wir in den folgenden Abschnitten vor.

Auf Outlook-Aktionen reagieren und Daten in Access speichern

Damit hätten wir das Grundgerüst erstellt. Nun folgt eine Reihe von Ereignisprozeduren, mit denen wir die Benutzereingaben in Outlook abfangen und die dabei anfallenden Daten in der Access-Datenbank speichern. Diese schauen wir uns in der Reihenfolge an, wie sie beim Arbeiten mit der Lösung auftreten.

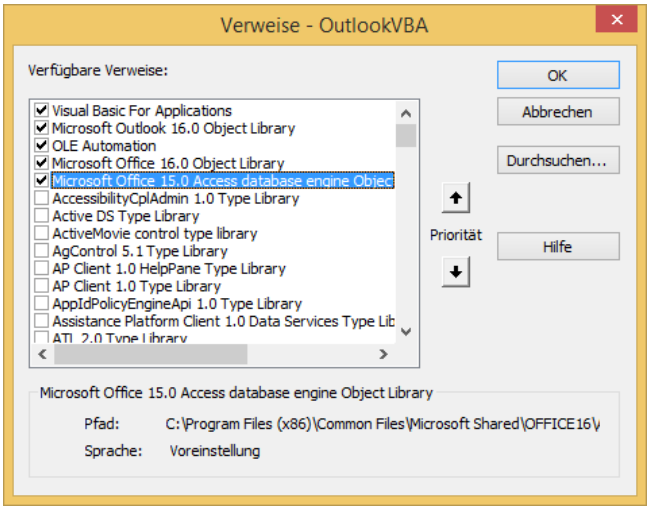

Bild 9: Verweis auf die DAO-Bibliothek

Verweis auf die DAO-Bibliothek

Spätestens jetzt, wo es an das Speichern von Daten in die Tabellen der Access-Datenbank geht, benötigen wir für das VBA-Projekt von Outlook einen Verweis auf die DAO-Bibliothek beziehungsweise auf die modernere Variante, die **Microsoft Office x.0 Access database engine Object Library**. Diesen fügen Sie wie in Bild 9 hinzu (VBA-Editor, Menüeintrag **Extras|Verweise**).

Projekt hinzufügen

Die erste Ereignisprozedur wird ausgelöst, wenn der Benutzer einen neuen Unterordner zum Aufgaben-Ordner hinzufügt – üblicherweise über den entsprechenden Kontextmenü-Eintrag. Dies löst das Ereignis **FolderAdd** des Objekts **objProjects** auf, das im Klassenmodul **ThisOutlookSession** deklariert ist. Dort legen Sie auch die Ereignisprozedur aus Listing 4 an.

Eine solche Ereignisprozedur erstellen Sie, indem Sie aus dem linken Kombinationsfeld des Codefensters den Namen des Objekts auswählen (hier **objProjects**) und aus dem rechten den Eintrag für die gewünschte Ereignisprozedur (in diesem Fall **FolderAdd**). Dies legt dann die nackte Prozedur an. Dem Prozedurkopf können wir dann schon entnehmen, dass die Prozedur als Parameter einen Verweis auf den neu angelegten Ordner liefert (**Folder**). Die Prozedur prüft zunächst den Typ des neu angelegten Ordners. Dazu vergleicht sie in einer **Select Case**-Bedin-