HAUFE.

Ausgabe 04/2015

ACCESS IM UNTERNEHMEN

BESTELLVERWALTUNG MIT VERSAND

Was hilft die schönste Bestellverwaltung, wenn Sie die Versandetiketten von Hand ausfüllen müssen? Erfahren Sie, wie Sie die Erstellung von DHL-Etiketten professionell direkt aus Ihrer Datenbank heraus steuern!

In diesem Heft:

ONLINEBANKING PER WEBSERVICE III

Das Ende der Trilogie: Diesmal rufen Sie Kontoumsätze ab und lernen, Überweisungen zu beauftragen.

SEITE 61

T-SQL PER FORMULAR

Fomulieren Sie Ihre Auswahl- oder Aktionsabfragen mit einem weiterem, hilfreichen Tool für den Zugriff auf den SQL Server!

SEITE 15

ABFRAGEN IM GRIFF

Erstellen Sie so viele Abfragen, wie Sie möchten, ohne damit den Navigationsbereich zu überfrachten.

SEITE 2

Geschickt verschicken

Wer die Bestellungen seiner eigenen Produkte mit einer Access-Datenbank verwaltet oder eine solche Datenbank für seine Kunden programmiert, schließt eine Bestellung in der Regel mit dem Versand des bestellten Produkts ab. Für den Versand gibt es nicht nur diverse Dienstleister, sondern auch verschiedene Möglichkeiten, die für den Versand nötigen Daten letztlich in ein Versandetikett umzuwandeln.

In dieser Ausgabe von **Access im Unternehmen** legen wir zunächst den Grundstein zur weitgehenden Automatisierung des Versands. Weitgehend deshalb, weil Sie ja immer noch die Ware verpacken, mit dem Versandetikett versehen und die Ware zur Post bringen oder abholen lassen müssen.

Um ein paar Handgriffe werden Sie dabei nicht herumkommen (außer, Sie lagern diese aus), aber zumindest die Erstellung der Versandetiketten wollen wir soweit es geht vereinfachen.

Deshalb schauen wir uns in dieser Ausgabe an, welche Möglichkeiten DHL in diesem Bereich für Geschäftskunden bietet. Wir könnten uns der Aufgabe zwar auch auf der Privatkunden-Schiene nähern, aber die besseren Möglichkeiten der technischen Anbindung erhält man als Geschäftskunde.

Im Beitrag **Versandetiketten mit DHL IntraShip** schauen wir uns ab S. 35 zunächst einmal an, wie Sie sich am Geschäftskundenkonto anmelden und dort eine CSV-Datei mit den Daten für die Erstellung der Versandetiketten einreichen.

Weiter geht es mit dem Beitrag **Klasse für DHL-Intraship-CSV-Dateien**, der ab S. 40 erklärt, wie Sie die Daten für die Erstellung der Versandetiketten strukturiert erstellen. Dabei tauchen wir einmal mehr in die objektorientierte Programmierung ein, denn wir verwenden dort eine kleine Struktur aus verschachtelten Klassen.

Das Ergebnis nutzen wir dann in der Lösung aus dem Beitrag **Bestellverwaltung mit Versand** (ab S. 51). Wir legen den Grundstein für die Zusammenstellung der Versanddaten, indem wir einen Satz von Tabellen und Formularen vorstellen, mit denen Sie schnell Kunden und Bestellungen eintragen können. Diese wandeln wir mithilfe der zuvor vorgestellten Klasse in eine CSV-Datei um, die wir dann über die Internetseite von DHL zur Erstellung der gewünschten Versandetiketten übergeben. Als Ergebnis erhalten wir dort unter anderem eine Liste der Sendungsnummern, die wir zum Zwecke der Sendungsnachverfolgung zu den Bestelldaten hinzufügen.

In den weiteren Beiträgen dieser Ausgabe schauen wir uns zum Beispiel an, wie Sie mal eben schnell eine Abfrage erstellen und speichern, ohne damit den Navigationsbereich der aktuellen Access-Datenbank zu überfrachten (siehe **Abfrageverwalter** ab S. 2). Das dazu benötigte Formular bietet auch die Möglichkeit, mehrere Abfrageergebnisse parallel auf mehreren Seiten eines Registersteuerelements anzuzeigen.

Außerdem beenden wir unsere Beitragsreihe zum Thema **Onlinebanking per Webservice**. Im dritten Teil erfahren Sie ab S. 61, wie Sie die Umsätze Ihrer Konten direkt von Access aus einlesen können und eine Überweisung von einem Access-Formular aus durchführen können.

Der Beitrag **T-SQL per Formular** liefert ein weiteres Tool für unsere Sammlung der RDBMS-Add-Ins. Hier können Sie eine SQL Server-Abfrage auswählen und Abfragen auf Basis der Tabellen der gewählten Datenbank ausführen – egal, ob es sich um Auswahl- oder Aktionsabfragen handelt (ab S. 15).

Schließlich finden Sie in dieser Ausgabe den zweiten Teil der Beitragsreihe **Outlook-Mails in Access archivieren** (ab S. 26). Dort liefern wir die Techniken, um die im ersten Teil der Beitragsreihe eingelesenen E-Mails zu durchsuchen, anzuzeigen und diese sogar in Outlook wiederherzustellen.

Viel Spaß beim Lesen!

Ihr Michael Forster

Abfrageverwalter

Geschieht Ihnen das auch regelmäßig? Sie wollen mal eben per Abfrage ein paar Daten filtern, sortieren oder zusammenführen, aber die Abfrage nach dieser Anwendung gleich wieder löschen. Dummerweise vergessen Sie Letzteres und irgendwann ist das Datenbankfenster voller gespeicherter Abfragen. Oder Sie benötigen gleichzeitig die Ergebnisse verschiedener Abfragen, finden die vielen geöffneten Fenster im Access-Hauptfenster aber unübersichtlich. Für beides liefert unser Abfrageverwalter die Lösung: Sie können damit mal eben eine Abfrage erstellen, ohne dass diese dauerhaft im Navigationsbereich verbleibt. Oder Sie zeigen die Ergebnisse mehrerer Abfragen übersichtlich im Registersteuerelement eines einzigen Formulars an.

Damit Sie gleich eine Vorstellung davon bekommen, wie diese Lösung aussehen soll, schauen wir uns das Formular aus Bild 1 an. Hier finden Sie eine ganze Reihe Elemente, zuoberst ein Listenfeld. Dieses zeigt alle mit dem Formular erstellten und gespeicherten Abfragen an. Darunter folgen die Steuerelemente für die Verwaltung der Abfragen:

- **Neu**: Öffnet den Entwurf einer neuen, leeren Abfrage.
- **Entwurf**: Zeigt den Entwurf der im Listenfeld markierten Abfrage an.
- **Anzeigen**: Öffnet die im Listenfeld ausgewählte Abfrage in der Datenblattansicht.
- **Löschen**: Entfernt die Abfrage aus dem Listenfeld.
- **Zur Übersicht hinzufügen**: Zeigt das Ergebnis der ausgewählten Abfrage auf einer Registerseite im unteren Bereich des Formulars an.
- **Aus Übersicht entfernen**: Entfernt eine Abfrage aus dem Registersteuerelement und blendet die entsprechende Registerseite aus.

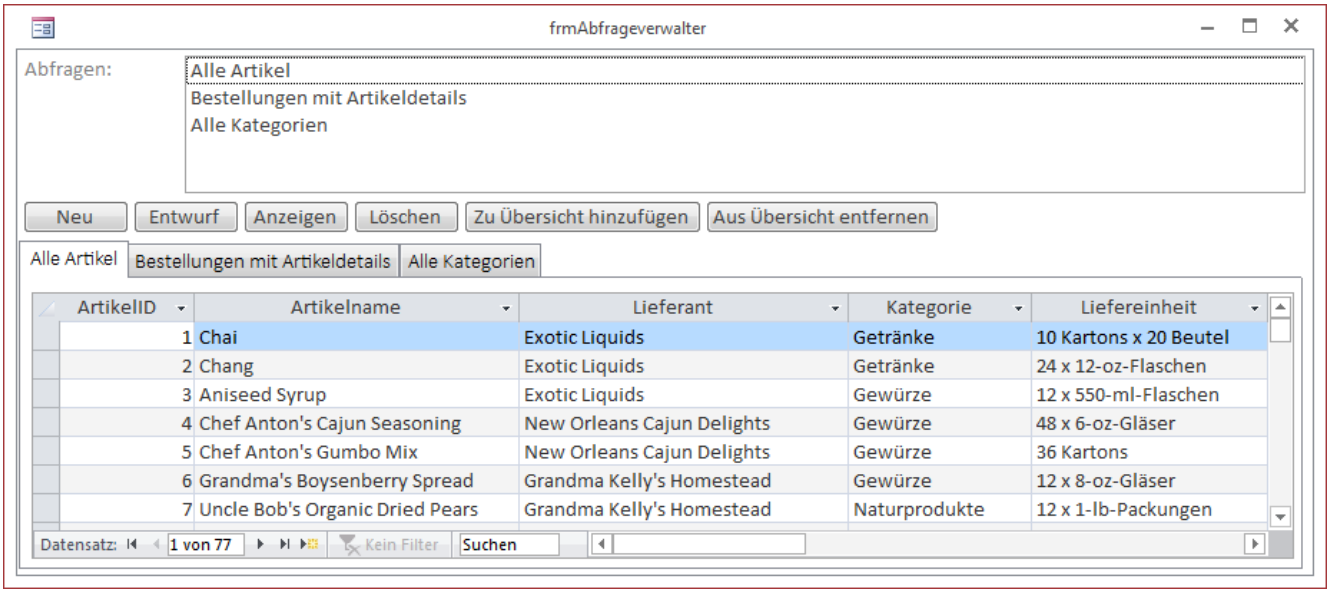

Bild 1: Benutzeroberfläche des Abfrageverwalters

Speicherort der Abfragen

Nun werden Sie sich zu Recht fragen: Wenn die Abfragen schon nicht im Navigationsbereich der Datenbank auftauchen, wo werden diese dann gespeichert? Ganz einfach: Wie legen für jede hier hinzugefügte Abfrage einen neuen Datensatz in einer speziell für diesen Zweck vorbereiteten Tabelle an.

Diese Tabelle heißt **tblAbfragen** und sieht in der Entwurfsansicht wie in Bild 2 aus. Das Feld **Abfragebezeichnung** soll die Bezeichnung aufnehmen und ist als eindeutiger Index ausgelegt. Auf diese Weise verhindern wir, dass die Tabelle mehrere Abfragen mit der gleichen Bezeichnung aufnimmt. Das Feld **AbfrageSQL** enthält später den SQL-Ausdruck, den Sie mithilfe der Entwurfsansicht für Abfragen zusammengestelt haben. Das Feld **Abfrageeigenschaften** nimmt die wichtigsten Eigenschaften der Abfrage auf, die Sie üblicherweise über das Eigenschaftsfenster der Abfrage eingeben – wie etwa in Bild 3 die Beschreibung der Abfrage.

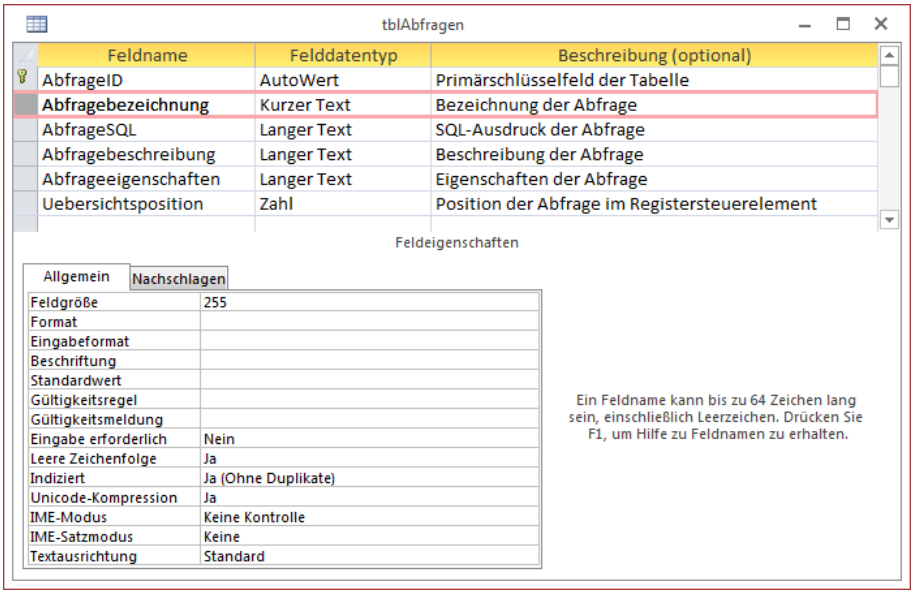

Bild 2: Entwurf der Tabelle zum Speichern der Abfragen

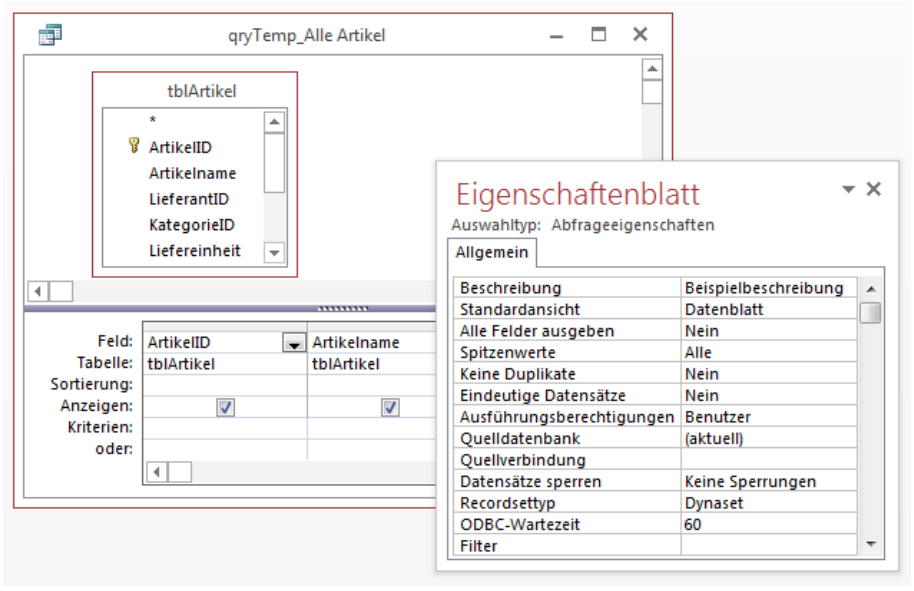

Bild 3: Einstellen von Eigenschaften für die Abfrage

Die meisten der hier angegebenen Eigenschaften landen indes ohnehin in Form entsprechender Schlüsselwörter im SQL-Text der Abfrage – zum Beispiel die Eigenschaft **Spitzenwerte** als **TOP**.

Das Feld **Uebersichtsposition** enthält einen Wert, der Folgendes angibt: Wenn das Feld leer ist, soll diese Abfrage nicht in der Übersicht der Abfrageergebnisse im Registersteuerelement erscheinen. Nur wenn dieses Feld einen Zahlenwert enthält, soll dies geschehen. In diesem Fall werden die Abfragen in aufsteigender Reihenfolge entsprechend dem Wert des Feldes dort angezeigt.

Damit können wir uns nun an das Formular zur Verwaltung der Abfragen begeben.

Das Formular frmAbfragen

Das Formular sieht in der Entwurfsansicht wie in Bild 4 aus. Es enthält ein Listenfeld zur Anzeige der in der Tabelle **tblAbfragen** gespeicherten Abfragen, einige Schaltflächen sowie ein Registersteuerelement mit zehn Registerseiten.

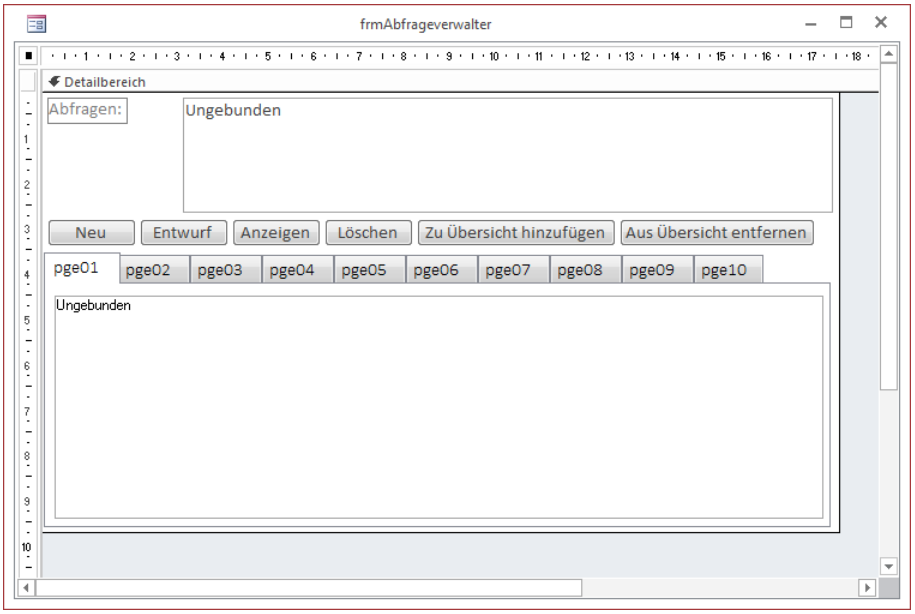

Bild 4: Entwurf des Formulars **frmAbfragen**

Listenfeld zur Anzeige der Abfragen

Das Listenfeld heißt **lstAbfragen** und verwendet die Abfrage aus Bild 5 als Datensatzherkunft. Diese liefert das Primärschlüsselfeld **AbfrageID** und die Bezeichnung der Abfrage aus dem Feld **Abfragebezeichnung**, nach der die Ergebnisse dieser Abfrage auch sortiert werden. Die Eigenschaft **Spaltenanzahl** des Listenfeldes erhält den Wert **2**, die Eigenschaft **Spaltenbreiten** den Wert **0cm**.

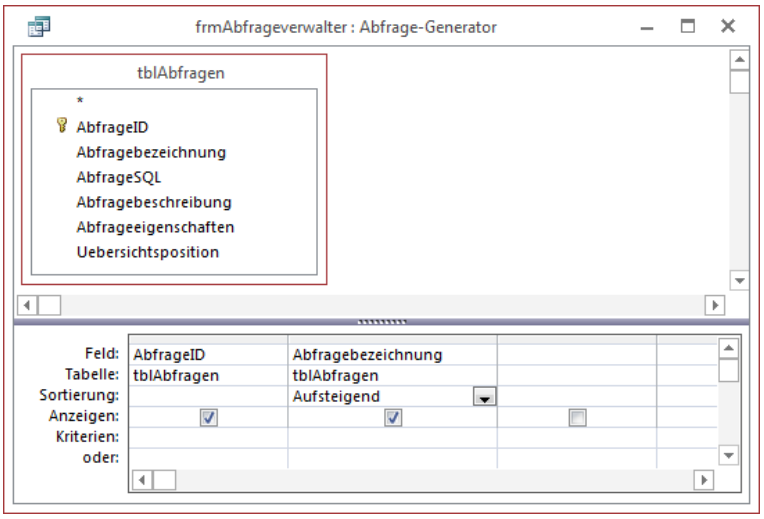

Bild 5: Abfrage, die als Datensatzherkunft des Listenfeldes **lstAbfragen** dient

Neue Abfrage anlegen

Ein Klick auf die Schaltfläche **cmdNeu** soll eine neue Abfrage in der Entwurfsansicht öffnen und dem Benutzer die Möglichkeit geben, die gewünschten Tabellen und Felder in den Entwurf zu ziehen. Dieser Teil ist einfach und wird durch die folgende Prozedur realisiert:

```
Private Sub cmdNeu_Click()
    Dim db As DAO.Database
    Dim qdf As DAO.QueryDef
    Set db = CurrentDb
    strAktuelleAbfrage = "qryTemp_New"
    On Error Resume Next
    db.QueryDefs.Delete strAktuelleAbfrage
    On Error GoTo 0
    Set qdf = db.CreateQueryDef(strAktuelleAbfrage)
    DoCmd.OpenQuery strAktuelleAbfrage, acViewDesign
End Sub
```

Die Prozedur definiert in **strAktuelleAbfrage** den temporären Namen der zu erstellenden Abfrage. Die Variable **strAktuelleAbfrage** ist modulweit deklariert:

```
Private strAktuelleAbfrage As String
```

Die Prozedur löscht dann eine eventuell noch vorhandene temporäre Abfrage mit dem Namen **qryTemp_New** und legt diese mit der Methode **CreateQueryDef** neu an. Danach öffnet die **OpenQuery**-Methode des **DoCmd**-Objekts

diese Abfrage in der Entwurfsansicht.

Die auf diese Weise geöffnete Abfrage zeigt gar nicht erst das **Tabelle anzeigen**-Fenster an, sodass Sie die als Datenquelle zu verwendenden Tabellen oder Abfragen direkt aus dem Navigationsbereich in den Abfrageentwurf ziehen können (s. Bild 6).

Mit dem entsprechenden Kontextmenü-Eintrag können Sie dieses Fenster aber auch nachträglich noch anzeigen.

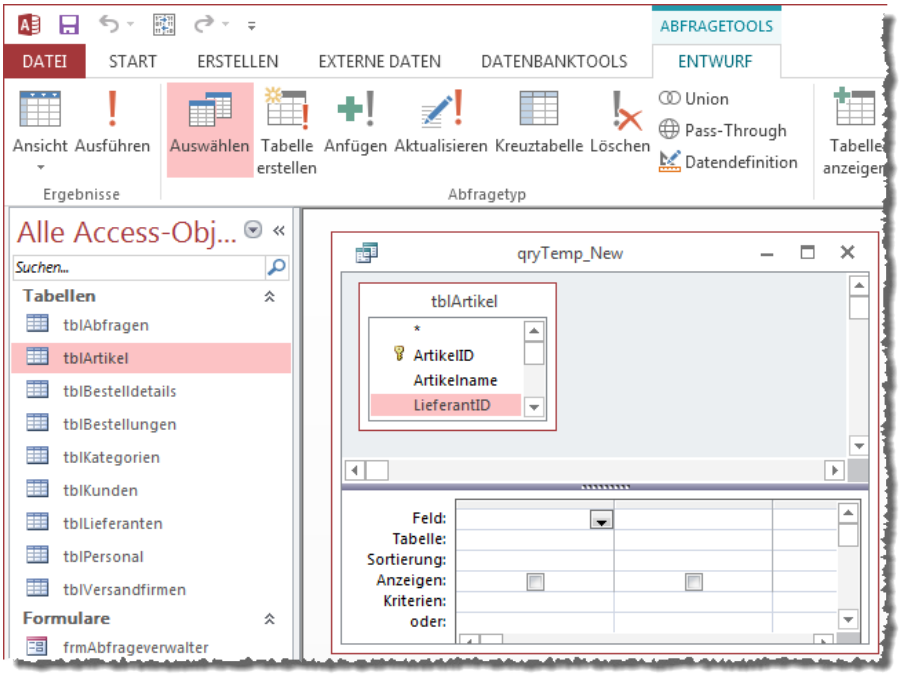

Bild 6: Anlegen einer neuen Abfrage in der Entwurfsansicht

Was aber geschieht, wenn die Abfrage fertiggestellt ist – wie bekommt das Formular mit, dass es soweit ist und wie wird dann die Abfrage in der Tabelle **tblAbfragen** gespeichert? Dazu nutzen wir das Ereignis **Bei Aktivierung** des Formulars **frmAbfragen**. Dieses wird beispielsweise ausgelöst, wenn der Benutzer über die Schaltfläche **cmdNeu** eine neue Abfrage erstellt hat und diese dann schließt. Das Formular wird dann wieder aktiviert und führt die genannte Ereignisprozedur aus.

Deren Code finden Sie in Listing 1. Die Prozedur prüft zunächst, ob die Variable **strAktuelleAbfrage** einen Wert enthält. Dies ist beispielsweise der Fall, wenn Sie kurz zuvor die Schaltfläche **cmdNeu** betätigt haben. Ist dies der Fall, prüft die Prozedur auch noch, ob die Abfrage aktuell noch geöffnet ist. Dazu nutzt sie die Hilfsfunktion **IstAbfrageGeoeffnet**:

```
Private Function IstAbfrageGeoeffnet(strAbfrage As _
        String) As Boolean
    IstAbfrageGeoeffnet = _
        SysCmd(acSysCmdGetObjectState, acQuery, strAbfrage)
End Function
```

Ist die Abfrage noch geöffnet, erscheint eine Meldung, die den Benutzer darüber informiert, dass die Abfrage gespeichert und geschlossen werden muss. Die Prozedur wird dann beendet.

Ist die Abfrage nicht mehr geöffnet, referenziert die Prozedur das **QueryDef**-Objekt zu dieser Abfrage mit der Variablen **qdf** und vergleicht das Datum der Erstellung mit dem Änderungsdatum. Sind die Daten gleich, hat der Benutzer seit der Erstellung keine Änderungen vorgenommen und die Abfrage braucht nicht gespeichert zu werden.

Anderenfalls liest die Prozedur den SQL-Code der Abfrage in die Variable **strAbfrageSQL** ein und die Eigenschaften in die Variable **strAbfrageEigenschaften** – dazu später mehr. Sie liest dann den Primärschlüsselwert einer eventuell bereits in der Tabelle **tblAbfragen** gespeicherten Abfrage in die Variable **lngAbfrageID** ein.

Nur wenn noch keine Abfrage gleichen Namens vorhanden ist, fragt die Prozedur den Benutzer, ob dieser die erstellte Abfrage sichern möchte. Dann fragt die folgende

```
Private Sub Form_Activate()
    Dim db As DAO.Database, qdf As DAO.QueryDef
    Dim strAbfrageSQL As String, strAbfragebezeichnung As String, strAbfrageeigenschaften As String
    Dim lngAbfrageID As Long, strSQL As String
    If Len(strAktuelleAbfrage) > 0 Then
        If IstAbfrageGeoeffnet(strAktuelleAbfrage) Then
            MsgBox "Bitte speichern und schließen Sie die temporäre Abfrage '" & strAktuelleAbfrage & "'"
            Exit Sub
        End If
        Set db = CurrentDb
        Set qdf = db.QueryDefs(strAktuelleAbfrage)
        If CDec(qdf.DateCreated) = CDec(qdf.LastUpdated) Then
            Exit Sub
        Else
            strAbfrageSQL = qdf.SQL
            strAbfrageeigenschaften = Abfrageeigenschaften(qdf)
            lngAbfrageID = Nz(DLookup("AbfrageID", "tblAbfragen", "'qryTemp_' & Abfragebezeichnung = '" & qdf.Name & "'"), 0)
            If lngAbfrageID = 0 Then
                If MsgBox("Soeben erstellte Abfrage sichern?", vbYesNo) = vbYes Then
                    strAbfragebezeichnung = Mid(qdf.Name, 9)
                    strAbfragebezeichnung = InputBox("Bezeichnung der Abfrage: ", "Abfrage speichern", strAbfragebezeichnung)
                    Do While Not IsNull(DLookup("AbfrageID", "tblAbfragen", "Abfragebezeichnung = '" _
                            & strAbfragebezeichnung & "'"))
                        strAbfragebezeichnung = InputBox("Unter diesem Namen ist bereits eine Abfrage vorhanden. " _
                            & "Geben Sie eine neue Bezeichnung ein: ", "Abfrage speichern", strAbfragebezeichnung)
                    Loop
                    If Len(strAbfragebezeichnung) = 0 Then
                        Exit Sub
                    End If
                    If Len(strAbfragebezeichnung) > 0 Then
                        strSQL = "INSERT INTO tblAbfragen(Abfragebezeichnung, AbfrageSQL, Abfrageeigenschaften) " _
                            & "VALUES('" & strAbfragebezeichnung & "', '" & strAbfrageSQL & "', '" _
                            & Replace(Replace(strAbfrageeigenschaften, """", "'"), "'", "''") & "')"
                        db.Execute strSQL, dbFailOnError
                    End If
                End If
                Me!lstAbfragen.Requery
            Else
                If MsgBox("Soeben geänderte Abfrage speichern?", vbYesNo) = vbYes Then
                    strSQL = "UPDATE tblAbfragen SET AbfrageSQL = '" & strAbfrageSQL & "', Abfrageeigenschaften = '" _
                        & strAbfrageeigenschaften & "' WHERE AbfrageID = " & lngAbfrageID
                    db.Execute strSQL, dbFailOnError
                End If
            End If
        End If
        strAktuelleAbfrage = ""
    End If
End Sub
```

Listing 1: Diese Prozedur prüft, ob der Benutzer neue Abfragen angelegt hat, und speichert gegebenenfalls die Daten.

T-SQL per Formular

Wer Access-Frontends mit einem SQL Server-Backend entwickelt, wird früher oder später nicht immer zum SQL Server Management Studio wechseln wollen, um mal eben eine Abfrage auf die Daten der SQL Server-Datenbank abzusetzen. Viel schöner wäre es doch, wenn man dies direkt vom Access-Fenster aus erledigen könnte! Kein Problem: Da wir in den vorherigen Ausgaben ohnehin schon Add-Ins für den Einsatz mit dem SQL Server vorgestellt haben, machen wir hier gleich weiter.

In den Beiträgen **RDBMS-Tools: Verbindungen verwalten** (**www.access-im-unternehmen.de/976**), **RDBMS-Tools: Tabellen verknüpfen** (**www.access-im-unternehmen.de/977**) und **RDBMS-Tools als Add-In nutzen** (**www.access-im-unternehmen.de/978**) haben wir bereits zwei kleine Tools vorgestellt und als Add-In aufbereitet.

Diese Add-In-Datei namens **RDBMSTools.mda** bauen wir jetzt weiter aus, indem wir ein Formular hinzufügen, mit dem Sie direkt Abfragen an die Datenbank eines SQL Servers absetzen können.

Dieses Formular sieht wie in Bild 1 aus und soll ebenfalls über einen Befehl des Add-In-Menüs verfügbar sein.

Das Formular soll zunächst über das obere Kombinationsfeld die Auswahl einer Verbindung ermöglichen. Diese haben wir natürlich zuvor mit einem der anderen beiden Add-Ins hergestellt, nämlich mit dem aus dem Beitrag **RDBMS-Tools: Verbindungen verwalten**.

Das Kombinationsfeld zeigt also, soweit bereits eingegeben, die vorhandenen Verbindungen an. Alternativ klicken Sie auf die Schaltfläche rechts vom Kombinationsfeld und öffnen so den Dialog aus Bild 2.

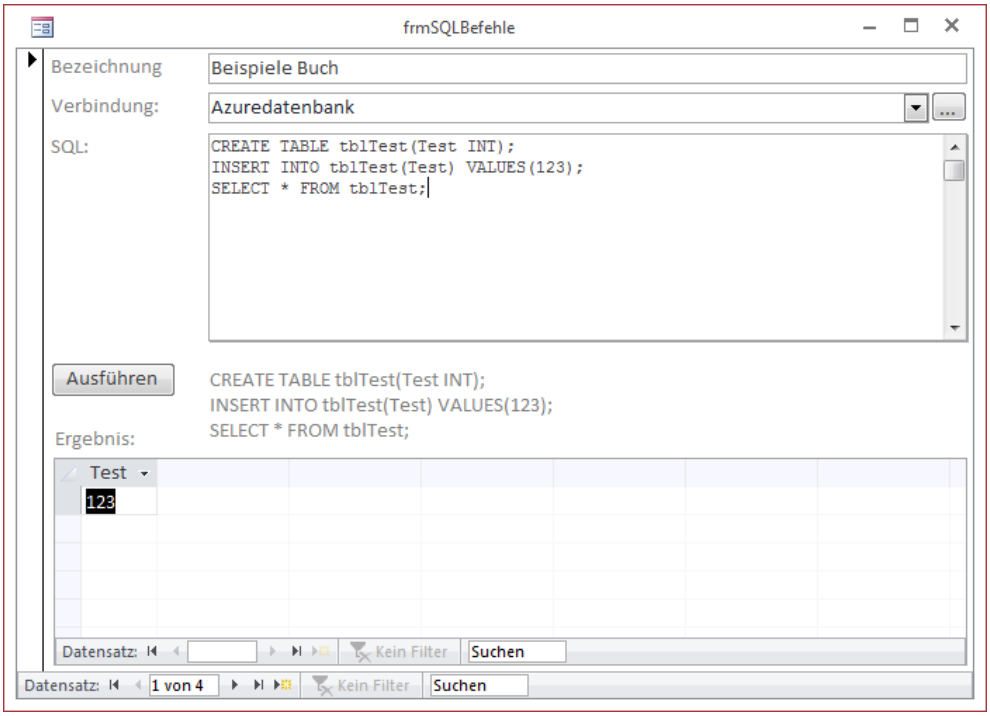

Bild 1: Formular zur direkten Eingabe von SQL-Anweisungen gegen ein SQL Server-Backend

Nach dem Auswählen der Verbindungszeichenfolge können Sie dann gleich mit der Eingabe von SQL-Anweisungen loslegen. Diese tragen Sie einfach in das große Textfeld direkt unter der Verbindungsauswahl ein. Dabei können Sie dort eine oder mehrere SQL-Anweisungen unterbringen.

Der Clou ist: Wenn Sie mehrere Anweisungen unterbringen, werden diese auch nacheinander abgearbeitet. Manchmal möchten

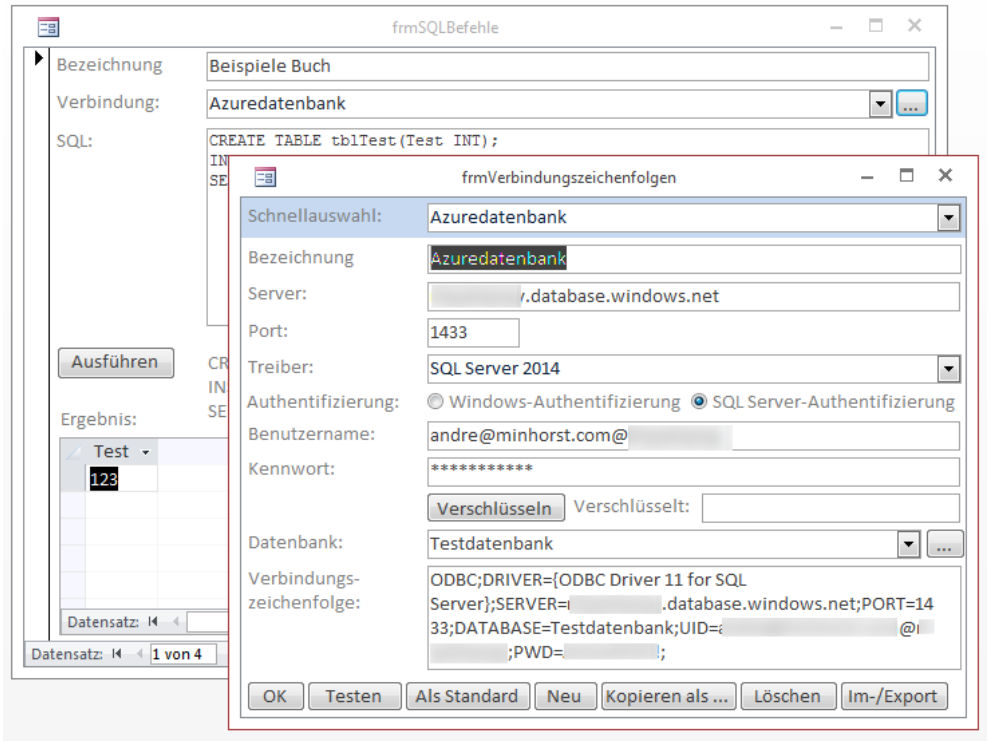

Bild 2: Verwalten der zu verwendenden Verbindungszeichenfolge

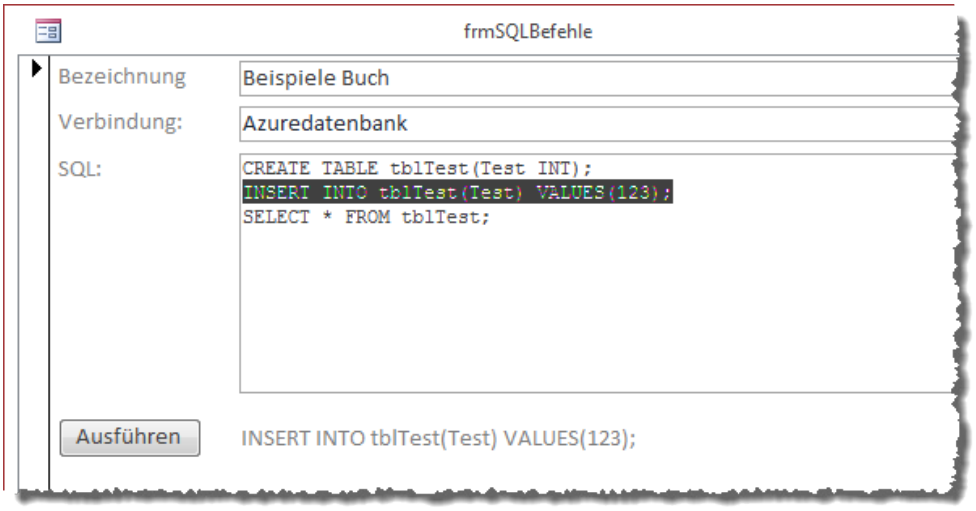

Bild 3: Markieren der auszuführenden SQL-Anweisung

Sie aber vielleicht ein paar Anweisungen eingeben und diese nacheinander ausführen.

Dann können Sie die jeweils auszuführende Anweisung einfach markieren und diese dann durch Betätigen der Schaltfläche **Ausführen** oder der Taste **F5** starten.

Damit Sie wissen, welche Anweisungen aktuell für die Ausführung vorgesehen sind, zeigt das Formular diese Zeilen unter dem Eingabefeld für die SQL-Anweisungen an. Wenn Sie also nur eine einzige Zeile wie in Bild 3 markieren, dann wird auch nur diese angezeigt – und durch **F5** oder **Ausführen** auch ausgeführt.

Fehlt noch das Unterformularsteuerelement im unteren Bereich des Formulars: Dieses zeigt die Ergebnisse der ausgeführten Anweisungen an.

Bei Datensatzänderungen wie dem Löschen, Ändern oder Hinzufügen wird hier beispielsweise die Anzahl der betroffenen Datensätze ausgegeben. Wenn Sie eine **SELECT**-Abfrage ausführen, liefert das Unterformular bis zu zehn Felder des Abfrageergebnisses.

Sie können also die folgenden Abfragetypen mit dem hier vorgestellten Formular ausführen:

- Auswahlabfragen (**SELECT**)
- Aktualisierungsabfragen (**INSERT INTO**, **SELECT INTO**, **UPDATE**, **DELETE**)

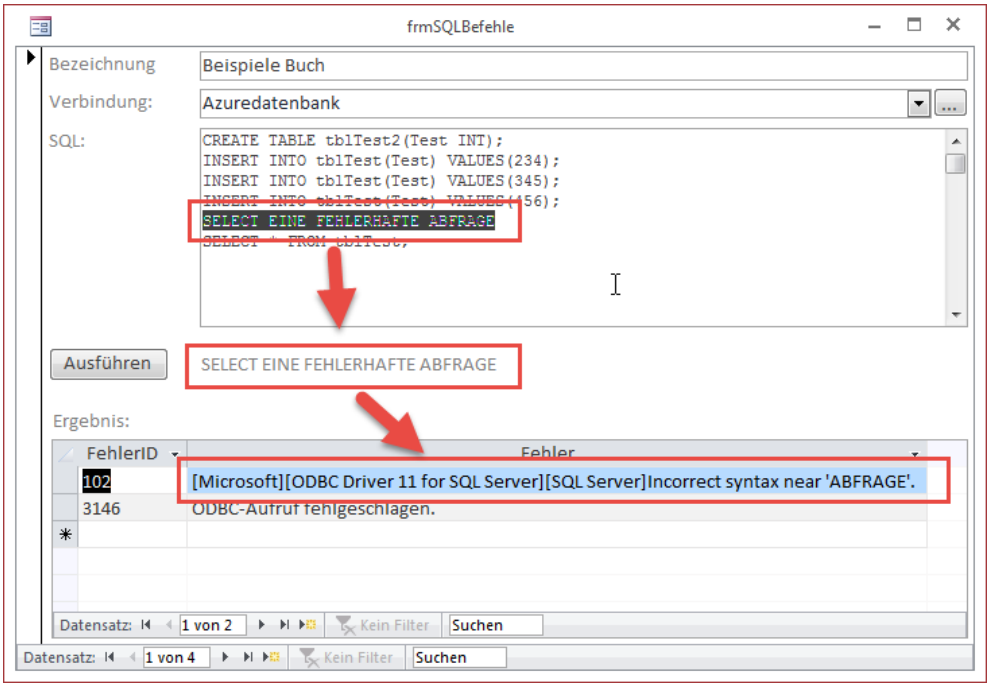

Bild 4: Ausgabe von Fehlermeldungen

- Datendefinitionsabfragen (**CREATE TABLE** et cetera)

Das Unterformular liefert sogar eine entsprechende Fehlermeldung, wenn Sie eine ungültige SQL-Abfrage absetzen (s. Bild 4).

Und noch ein wichtiges Feature: Möglicherweise benötigen Sie die eine oder andere SQL-Anweisung im Laufe der Entwicklung der Anwendung mehrmals. Die eingegebenen Daten samt Bezeichnung, Verbindungszeichenfolge und SQL-Befehlen werden in einer Tabelle gespeichert, sodass Sie durch Blättern durch die Datensätze die verschiedenen SQL-Befehle ansteuern können.

Formular erstellen

Das Formular sieht in der Entwurfsansicht wie in Bild 5 aus. Da die eingegebenen SQL-Befehle gespeichert und bei Bedarf wieder aufrufbar sein sollen, binden wir das Formular an die Tabelle **tblSQLBefehle** als Datenherkunft. Das Feld **SQLBefehl** soll dabei als Memofeld ausgelegt sein, da eine SQL-Anweisung leicht einmal mehr als 255 Zeichen umfasst. **VerbindungID** ist ein Fremdschlüsselfeld zur Tabelle **tblVerbindungszeichenfolgen** und **Bezeichnung** ein einfaches Textfeld.

Diese Tabelle finden Sie in der Entwurfsansicht in Bild 6. Die Felder der Tabelle werden – mit Ausnahme des Primärschlüsselfeldes – im oberen Teil des

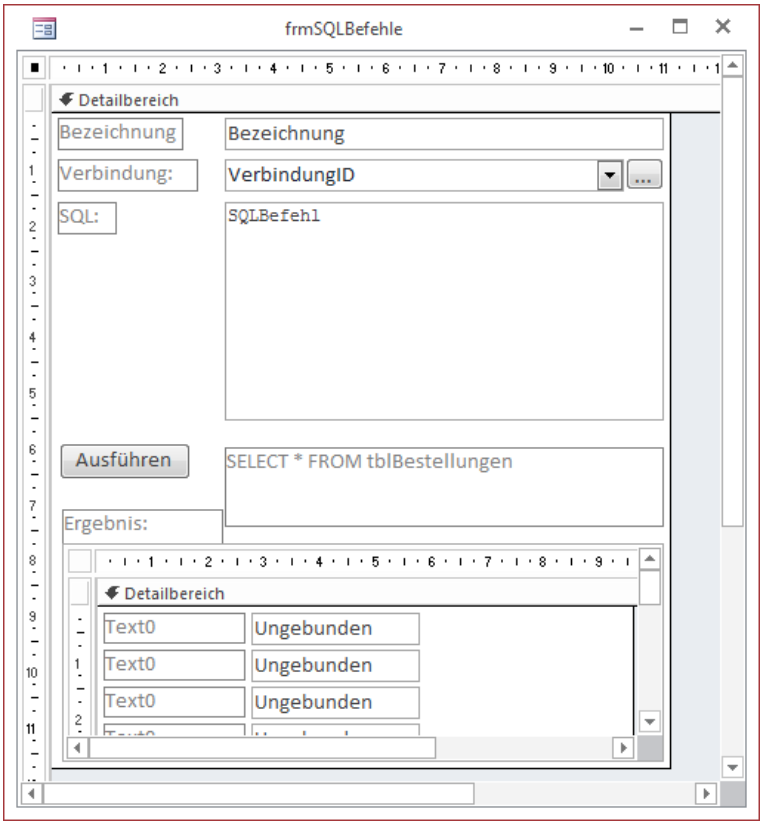

Bild 5: Entwurf des Formulars **frmSQLBefehle**

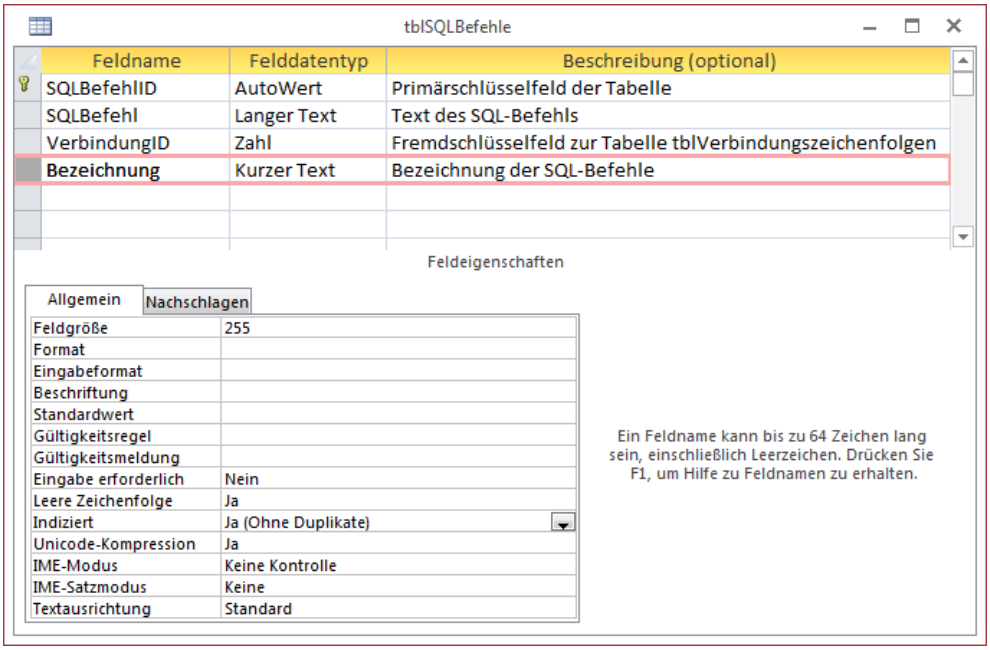

Bild 6: Entwurf der Tabelle zum Speichern der SQL-Befehle

Damit die Schaltfläche **cmdVerbindungenBearbeiten** auf Knopfdruck das Formular **frmVerbindungszeichenfolgen** öffnet, hinterlegen Sie die Ereignisprozedur aus Listing 1 für das Ereignis **Beim Klicken**. Dieses öffnet das Formular **frmVerbindungszeichenfolgen** und übergibt den Primärschlüsselwert der aktuell im Kombinationsfeld **cboVerbindung** ausgewählten Verbindungszeichenfolge per Öffnungsargument.

Formulars **frmSQLBefehle** angezeigt. Das Feld **VerbindungID** führen wir dabei als Kombinationsfeld aus, damit Sie damit leicht die Einträge der Tabelle **tblVerbindungszeichenfolgen** auswählen können. Rechts neben dem Kombinationsfeld finden Sie eine Schaltfläche namens **cmdVerbindungenBearbeiten**, mit der Sie das bereits im Beitrag **RDBMS-Tools: Verbindungen verwalten** beschriebene Formular öffnen können.

Dieses zeigt dann direkt die aktuell im Kombinationsfeld ausgewählte Verbindung an. Das Kombinationsfeld **cboVerbindung** erhält also die folgende Abfrage als Datensatzherkunft:

```
SELECT VerbindungszeichenfolgeID, Bezeichnung
FROM tblVerbindungszeichenfolgen;
```

Dann speichert die Prozedur einen Verweis auf das Formular in einer Objektvariablen namens **frmVerbindungszeichenfolgen**, die im Kopf des Klassenmoduls wie folgt deklariert wird:

```
Dim WithEvents frmVerbindungszeichenfolgen As Form
```

Der Hintergrund ist, dass wir im Klassenmodul des aufrufenden Formulars **frmSQLBefehle** eine Ereignisprozedur definieren wollen, die beim Schließen des Formulars **frmVerbindungszeichenfolgen** ausgelöst wird. Dazu müssen wir die Objektvariable mit dem Schlüsselwort **WithEvents** auszeichnen. Nun können wir per VBA eine Ereignisprozedur für die Ereigniseigenschaft **Beim Entladen** angeben (**OnUnload = [Event Procedure]**).

```
Private Sub cmdVerbindungenBearbeiten_Click()
    DoCmd.OpenForm "frmVerbindungszeichenfolgen", OpenArgs:=Nz(Me!cboVerbindung, 0)
    Set frmVerbindungszeichenfolgen = Forms!frmVerbindungszeichenfolgen
    With frmVerbindungszeichenfolgen
        .Modal = True
        .OnUnload = "[Event Procedure]"
    End With
End Sub
```

Listing 1: Formular zum Auswählen der Verbindungszeichenfolge öffnen

Damit brauchen wir nun nur noch die folgende Ereignisprozedur im Klassenmodul des Formulars **frmSQLBefehle** zu hinterlegen:

```
Private Sub frmVerbindungszeichenfolgen_
Unload(Cancel _
        As Integer)
    Me!cboVerbindung.Requery
    Me!cboVerbindung = _
        frmVerbindungszeichenfolgen!Verbindungszeic
henfolgeID
End Sub
```

Damit rufen wir nun das Formular **frmVerbindungszeichenfolgen** auf und können im Formular **frmSQLBefehle** auf das **Beim Schließen**-Ereignis von **frmVerbindungszeichenfolgen** reagieren.

Dort aktualisieren wir den Inhalt von **cboVerbindung** (falls im Formular **frmVerbindungszeichenfolgen** neue Verbindungszeichenfolgen hinzugefügt oder welche gelöscht wurden) und stellen das Kombinationsfeld **cboVerbindung** auf die zuletzt im Formular **frmVerbindungszeichenfolgen** gewählte Verbindungszeichenfolge ein.

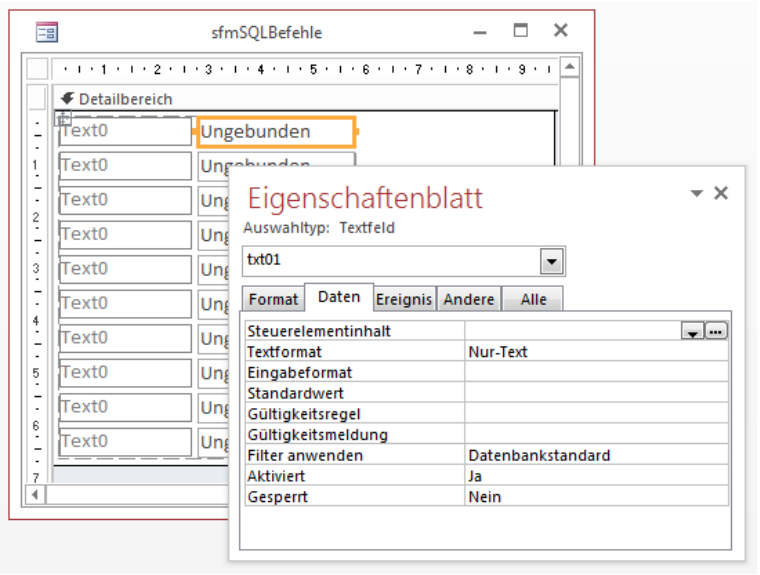

Bild 7: Entwurf des Unterformulars **sfmSQLBefehle**

Das Textfeld **txtSQL** ist an das Feld **SQLBefehl** der Tabelle **tblSQLBefehle** gebunden. Damit Sie darin Zeilenumbrüche einfach mit der Eingabetaste herbeiführen können, ohne dass der Fokus auf das nächste Steuerelement verschoben wird, stellen wir die Eigenschaft **Eingabetastenverhalten** auf **Neue Zeile im Feld** ein.

Das Bezeichnungsfeld **lblSQL** soll jeweils den SQL-Ausdruck aus dem Textfeld **txtSQL** anzeigen, der beim Betätigen von **F5** beziehungsweise beim Anklicken der Schaltfläche **cmdAusfuehren** ausgeführt würde.Das ist normalerweise der komplette SQL-Text, außer der Benutzer markiert nur einen Teil des enthaltenen Textes. Dann wird nur dieser Teil in **lblSQL** abgebildet und auch ausgeführt.

Das Formular enthält außerdem eine Schaltfläche namens **cmdAusfuehren**, auf deren Funktion wir weiter unten eingehen.

Außerdem enthält das Formular ein Unterformular-Steuerelement, dessen Eigenschaft **Herkunftsobjekt** wir auf das Formular **sfmSQLBefehle** einstellen.

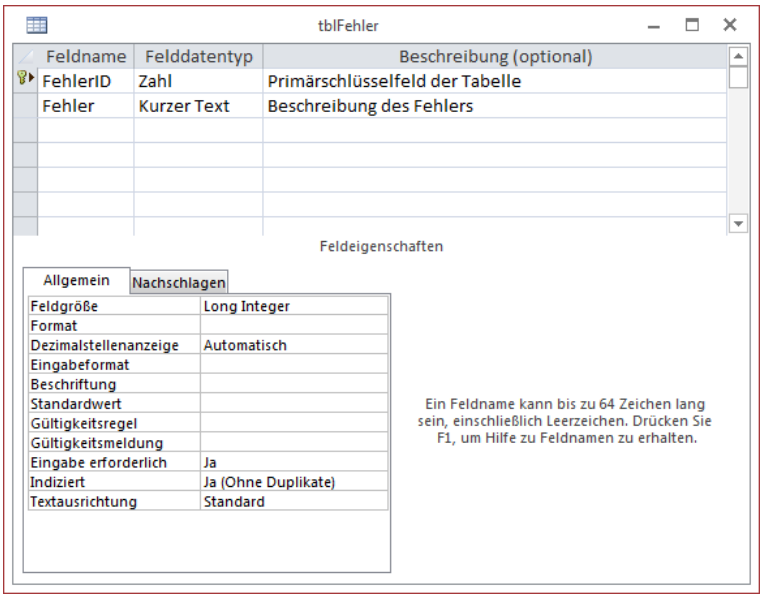

Bild 8: Entwurf der Tabelle **tblFehler**

Outlook-Mails in Access archivieren II

Im ersten Teil dieser Beitragsreihe haben wir alle Outlook-Mails eines Ordners und gegebenenfalls auch die in den Unterordnern in Access archiviert. Im vorliegenden zweiten Teil der Reihe schauen wir uns an, welche Daten wir dort nun produziert haben und was wir mit diesen alles anfangen können. Dazu gehört unter anderem, dass wir die Daten, da wir diese nun in Tabellenform vorliegen haben, mit den gewohnten Mitteln durchsuchen können. Damit wollen wir nun nicht mehr benötigte Mails endgültig löschen oder Mails wiederherstellen, die wir in Outlook doch nochmal brauchen.

Mit den im ersten Teil vorgestellten Prozeduren im Formular **frmMailimport** haben wir die Tabellen der Datenbank mit den Daten der **Outlook.pst**-Datei gefüllt – und auch noch zwei Verzeichnisse, die sich auf der gleichen Ebene wie die Datenbankanwendung befinden.

Die Verzeichnisstruktur sieht nun wie in Bild 1 aus. Das Verzeichnis **MSG** enthält die im Dateisystem abgelegten **.msg**-Dateien. Die Lösung bietet die Möglichkeit, diese Dateien entweder mit dem Datensatz in einem Anlagefeld der Tabelle **tblMailItems** zu speichern oder aber diese im Dateisystem abzulegen. Letzteres legen Sie durch den Wert für die maximale Größe der in der Datenbank zu speichernden **.msg**-Dateien in den Optionen der Lösung fest. Alle Dateien, die größer sind, landen im Verzeichnis **MSG**.

Das Verzeichnis **Anlagen** enthält, soweit Anlagen nochmals in separate Dateien extrahiert werden sollen, alle Anlagen der archivierten E-Mails. Die Anlagen befinden

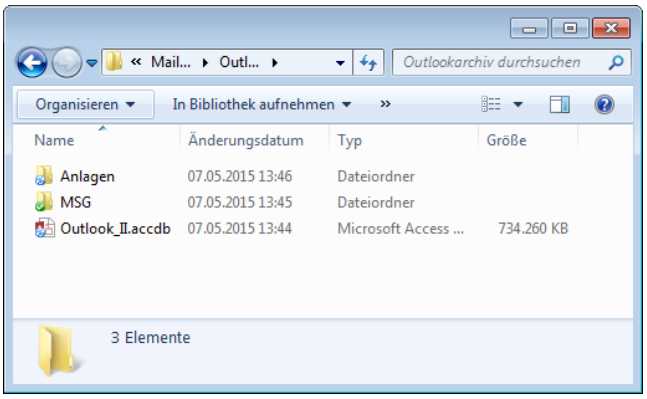

Bild 1: Verzeichnisstruktur der Lösung

sich auch in den jeweiligen **.pst**-Dateien, sodass ein separates Speichern grundsätzlich nicht erforderlich ist. Die Tabelle **tblMailItems** enthält nun einige Datensätze mit den Metadaten einer E-Mail sowie dem Inhalt. Außerdem enthält die Tabelle ein Anlagefeld, das die **.msg**-Datei aufnehmen kann, sofern diese die maximal zulässige Größe nicht überschreitet (s. Bild 2).

Die Tabelle **tblAnlagen** speichert zu jeder Anlage einer Mail einen Datensatz, der über das Feld **MailItemID** mit der Tabelle **tblMailItems** verknüpft ist (s. Bild 3). Die dortigen Felder geben an, in welchem Verzeichnis die Anlagen gespeichert sind und wie die entsprechenden Dateien heißen. Damit haben wir eigentlich alle Informationen, die wir benötigen, um unsere Such-Funktion zu implementieren!

tblMailItems

MailitemID	EntryID	Betreff	Body
43335	00000000809CC08917748749/	AW: Kontaktformular Shopware	Hallo Herr Minnhors
43336	00000000809CC08917748749/	Re: AW: Intellisense für SQL/Onlir	Hallo André,
43337	00000000809CC08917748749/	Kündigung Abo Access Basic/Acce	
43338	00000000809CC08917748749/	Kontaktformular Shopware	Kontaktformular Sh
43339	00000000809CC08917748749/	WG: Anfrage Json Datei	
43340	00000000809CC08917748749/	Stammtisch-Thema	Hi Andre,
43341	00000000809CC08917748749/	Stammtisch	
43342	00000000809CC08917748749/	Frage zu VBA	Guten Abend!
43343	00000000809CC08917748749/	Re: Anfrage-Formular Shopware	... super

Datensatz: 1 von 48560 Kein Filter Suchen

Bild 2: Die Tabelle **tblMailItems** mit einigen Datensätzen

Suchformular

Das Suchformular soll wie in Bild 4 aussehen. Dort finden wir im oberen Bereich zunächst die Felder zur Eingabe der Suchkriterien. Darunter befinden sich drei Schaltflächen:

- **Suchen**: Startet die Suche auf Basis der angegebenen Suchkriterien
- **Alle anzeigen**: Hebt den durch die Suche angewendeten Filter wieder auf.
- **In OL wiederherstellen**: Legt eine Kopie der E-Mail in Outlook in dem Ordner an, aus dem sie ursprünglich kopiert wurde.

Der Entwurf des Formulars sieht wie in Bild 5 aus. Das Unterformular **sfmMailItems** verwendet die Tabelle **tblMailItems** als **Datenherkunft**.

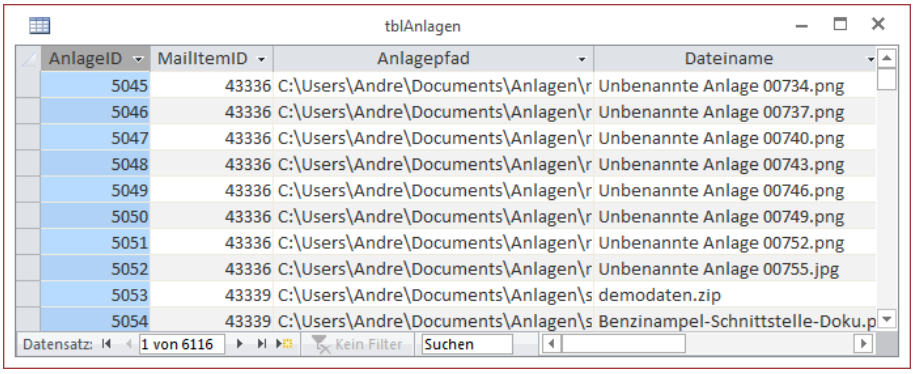

Bild 3: Tabelle zum Speichern der Pfade zu den Anlagen

Mails über das Unterformular anzeigen

Bevor wir auf die Suchfunktion eingehen, schauen wir uns an, wie Sie über das Unterformular eine einzelne Mail im Mailfenster von Outlook öffnen können. Dazu soll der Benutzer einfach doppelt auf den entsprechenden Datensatz im Unterformular klicken können. Da es dazu kein allgemeines Ereignis namens **Beim Doppelklicken** für den kompletten Formularbereich gibt, müssen wir dieses für die einzelnen Textfelder im Unterformular anlegen. Die folgenden Ereignisprozeduren legen wir also im Klassenmodul des Unterformulars an:

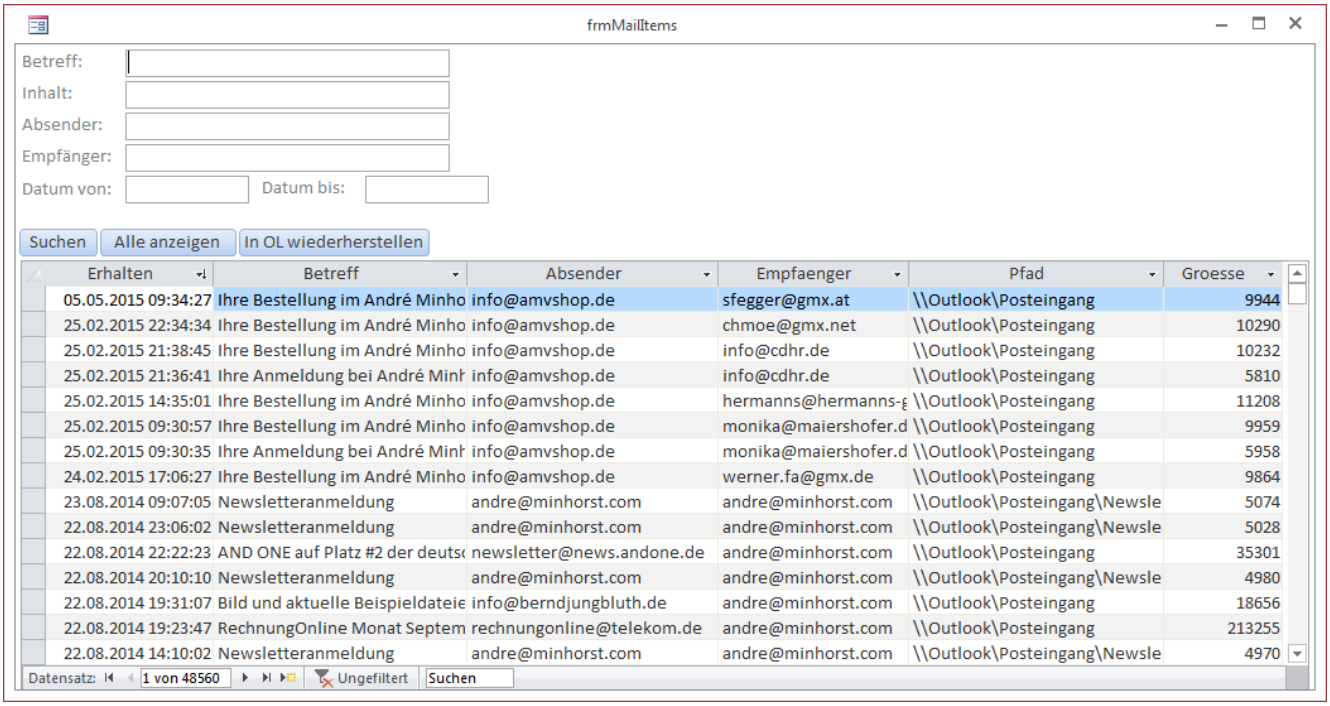

Bild 4: Formular zum Durchsuchen der E-Mails

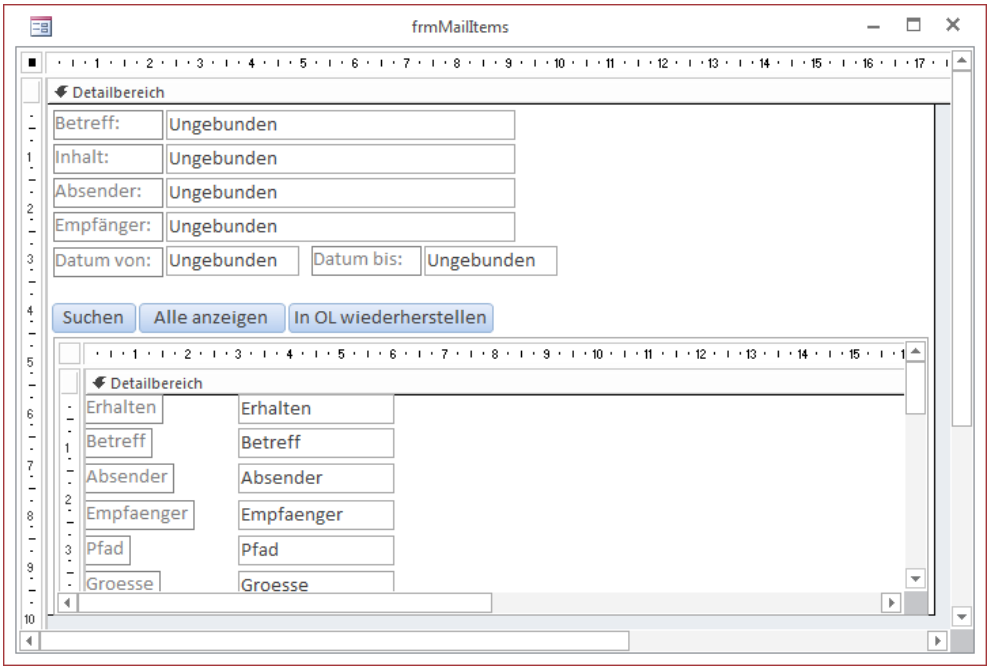

Bild 5: Entwurf des Formulars **frmMailItems**

```
Private Sub Absender_DblClick(Cancel As Integer)
    MailOeffnen Me!MailitemID
End Sub

Private Sub Betreff_DblClick(Cancel As Integer)
    MailOeffnen Me!MailitemID
End Sub

Private Sub Empfaenger_DblClick(Cancel As Integer)
    MailOeffnen Me!MailitemID
End Sub

Private Sub Erhalten_DblClick(Cancel As Integer)
    MailOeffnen Me!MailitemID
End Sub

Private Sub Pfad_DblClick(Cancel As Integer)
    MailOeffnen Me!MailitemID
End Sub
```

Damit ist sichergestellt, dass die von diesen Ereignisprozeduren aufgerufene Prozedur **MailOeffnen** immer ausgelöst wird – egal, auf welches der Textfelder in der Datenblattansicht des Unterformulars der Benutzer klickt.

Die Prozedur **MailOeffnen** finden Sie in Listing 1. Die Prozedur nimmt als Parameter den Primärschlüsselwert des im Unterformular angeklickten Datensatzes der Tabelle **tblMailItems** entgegen. Sie öffnet dann ein Recordset auf Basis der Tabelle **tblMailItems**, wobei diese nach dem übergebenen Primärschlüsselwert gefiltert wird.

Nun ist es interessant, ob die Originalmail im **.msg**-Format im Anlagefeld **MailItem** der Tabelle **tblMailItems** gespeichert wurde oder ob sich diese im Unterverzeichnis **MSG** befindet.

Um dies herauszufinden, referenziert die Prozedur zunächst das im Anlagefeld **MailItem** enthaltene und über die **Value**-Eigenschaft verfügbare Recordset mit der eigentlichen Anlage (hier bitte Anlage eines Anlagefeldes nicht mit Anlage einer E-Mail verwechseln) und speichert es in **rstAttachment**. Enthält **rstAttachment** nun genau einen Recordset (**RecordCount = 1**), dann liegt die Originalmail im **.msg**-Format im Anlagefeld der Tabelle vor.

Das Feld **FileData** des Recordsets **rstAttachment** füllt die Prozedur dann in die Variable **fldAttachment**. Das **Field2**-Objekt **fldAttachment** bietet dann mit der Methode **SaveToFile** die Möglichkeit, den Inhalt des Anlagefeldes als Datei zu speichern. Da es durchaus vorkommen kann, dass wir einmal mehrere Mail gleichzeitig anzeigen, müssen wir eine Konvention für die Dateinamen der **.msg**-Dateien festlegen, die ausschließt,

```
Private Sub MailOeffnen(lngMailItemID As Long)
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim rstAttachment As DAO.Recordset2
    Dim fldAttachment As DAO.Field2
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM tblMailItems WHERE MailItemID = " & lngMailItemID, dbOpenDynaset)
    Set rstAttachment = rst.Fields("MailItem").Value
    If rstAttachment.RecordCount = 1 Then
        Set fldAttachment = rstAttachment.Fields("FileData")
        On Error Resume Next
        Kill CurrentProject.Path & "\MailItemTemp_" & lngMailItemID & ".msg"
        On Error GoTo 0
        fldAttachment.SaveToFile CurrentProject.Path & "\MailItemTemp_" & lngMailItemID & ".msg"
    Else
        FileCopy CurrentProject.Path & "\MSG\" & lngMailItemID & ".msg", CurrentProject.Path & "\MailItemTemp_" _
            & lngMailItemID & ".msg"
    End If
    Call ShellExecute(Me.hWnd, "open", CurrentProject.Path & "\MailItemTemp_" & lngMailItemID & ".msg", "", "", _
        SW_NORMAL)
End Sub
```

Listing 1: Die Prozedur zum Anzeigen einer E-Mail im Mail-Inspector

dass Dateien doppelt vorhanden sind. Das ist aber kein Problem, denn wir haben ja zu jeder Mail einen Primärschlüsselwert, den wir in den Dateinamen integrieren können. Also fügen wir den Namen etwa nach folgendem Schema zusammen:

```
MailItemTemp_<ID>.msg
```

Eventuell vorhandene Dateien gleichen Namens löscht die Prozedur vor dem Anlegen mit der **Kill**-Anweisung. Dann speichert sie die Datei mit **SaveToFile** in das gleiche Verzeichnis wie die Datenbank.

Sollte das Anlagefeld keine Datei enthalten, haben wir beim Import eine Kopie der **.msg**-Datei im Verzeichnis **MSG** im Verzeichnis der Datenbank gespeichert. Diese kopieren wir nun mit der **FileCopy**-Anweisung ebenfalls in das Verzeichnis der Datenbank. Schließlich öffnen wir die **.msg**-Datei mit der **ShellExecute**-Methode, welche sich die passende Anwendung für die Anzeige der E-Mail, in diesem Fall Outlook, aussucht und die E-Mail schließlich öffnet. Das Ergebnis sieht etwa wie in Bild 6 aus.

Suchfunktion

Die Suchfunktion wertet die Eingaben in die Textfelder **txtAbsender**, **txtEmpfaenger**, **txtBetreff**, **txtInhalt**, **txtVon** und **txtBis** aus und stellt einen entsprechenden Filter zusammen.

Sie wird durch einen Klick auf die Schaltfläche **cmdSuchen** ausgelöst und ist in Listing 2 zu finden. Die Prozedur prüft in einer If...Then-Bedingung den Inhalt der einzelnen Textfelder, und zwar über die Länge der enthaltenen Zeichenketten. Beträgt die mit der **Len**-Funktion Länge mehr als **0** Zeichen, ist ein Vergleichswert vorhanden und die Prozedur fügt einen entsprechenden, mit dem **AND**-Schlüsselwort beginnenden Ausdruck zu der in der Variablen **strFilter** gespeicherten Zeichenkette hinzu.

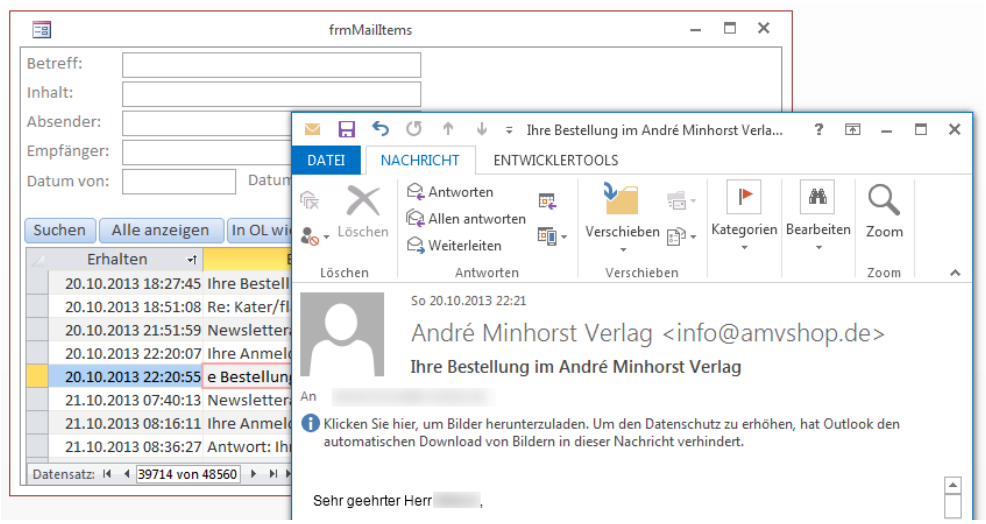

Bild 6: Eine per Doppelklick geöffnete E-Mail

Wenn der Benutzer beispielsweise als Absender den Wert **andre@minhorst.com** eingibt, lautet der Ausdruck in **strFilter** für diesen Schritt wie folgt:

```
AND Absender LIKE '*andre@minhorst.com*'
```

Sie erkennen hier bereits, dass wir mit dem **LIKE**-Operator und dem Sternchen als führenden und abschließenden Platzhalter arbeiten. Dadurch liefert das Suchergebnis auch solche Einträge, die den angegebenen Text nur enthalten, nicht aber komplett mit diesem identisch sein müssen. Auf diese Weise fügt die Prozedur schrittweise die in den Textfeldern angegebenen Vergleichswerte zu einem Filterausdruck zusammen.

Ein Unterschied ergibt sich bei den beiden Textfeldern **txtVon** und **txtBis**, mit denen ja der Zeitraum für die E-Mails zusammengesetzt werden soll. Die Prozedur prüft dann in einer ersten **If...Then**-Bedingung, ob der Benutzer im Feld **txtVon** ein Startdatum angegeben hat. Falls ja, prüft sie auch noch, ob **txtBis** gefüllt ist. In diesem Fall kann sie einen Ausdruck wie den folgenden zusammenstellen:

```
Erhalten >= #2015/01/01 00:00:00# AND Erhalten <=
#2015/01/31 00:00:00#
```

Hat das Feld **txtBis** keinen Wert, enthält der Ausdruck nur das Startdatum für den Datumsbereich:

```
Erhalten >= #2015/01/01
00:00:00#
```

Fehlt noch die Variante, dass nur **txtBis** gefüllt ist:

```
Erhalten <= #2015/01/31
00:00:00#
```

Nun hat die Prozedur kein, ein oder mehrere Kriterien zusammengestellt, die jeweils mit dem **AND**-Schlüsselwort beginnen. Das erste Auftreten dieses Schlüsselworts müssen wir natürlich abschneiden, was die Prozedur mit der **Mid**-Funktion erledigt, sofern **strFilter** nicht leer ist. Schließlich stellt sie die Eigenschaft **Filter** des Unterformulars auf den Wert aus **strFilter** ein und aktiviert den Filter durch Einstellen der Eigenschaft **FilterOn** auf den Wert **True**.

Alle Mails wieder einblenden

Wenn Sie nicht gefunden haben, was Sie suchen, können Sie die Filterkriterien zurücksetzen und alle in der Tabelle **tblMailItems** enthaltenen Einträge wieder einblenden.

Dazu klicken Sie einfach auf die Schaltfläche **cmdAlleAnzeigen**, was die folgende Prozedur auslöst:

```
Private Sub cmdAlleAnzeigen_Click()
    Me!txtAbsender = Null
    Me!txtEmpfaenger = Null
    Me!txtBetreff = Null
    Me!txtInhalt = Null
    Me!txtVon = Null
    Me!txtBis = Null
    Me!sfmMailItems.Form.Filter = ""
End Sub
```

Versandetiketten mit DHL-IntraShip

Wer regelmäßig Pakete mit DHL verschickt, ist möglicherweise Geschäftskunde bei DHL. Das ist Voraussetzung, um das Onlineportal zur Eingabe der Empfängeradressen und zum Erstellen der Versandetiketten zu nutzen. Als Datenbankentwickler wollen wir aber keine Adressen von Hand in ein Formular eingeben, sondern diese direkt aus der Datenbank heraus übergeben – mit möglichst wenig Aufwand. Dieser Beitrag zeigt, wie Sie die Daten übergeben, wenn die notwendige Datei einmal erzeugt ist.

Voraussetzungen

Den Zugang als Geschäftskunde erhalten Sie, wenn Sie sich unter der folgenden Adresse registrieren:

```
https://www.dhl-geschaeftskundenportal.de/
```

Auf die Einzelheiten der Registrierung wollen wir an dieser Stelle nicht eingehen.

Vorteile IntraShip

Neben IntraShip für Geschäftskunden gibt es ja auch noch die Möglichkeit, als Privatkunde online die Etiketten für den Versand von DHL-Paketen zu erhalten. Dort müssen Sie allerdings etwa Ihre Portokasse aufladen und bezahlen das Porto dann aus dieser Kasse.

Eine genaue Auflistung, welche Positionen damit bezahlt wurden, erhält man nicht. Bei Intraship brauchen Sie sich beim Einreichen der CSV-Dateien für die Erstellung der Versandetiketten nicht um die Bezahlung zu kümmern – es erfolgt alle zehn Tage eine Abrechnung. Außerdem erhalten Sie bei IntraShip kostenlos Etiketten für den Ausdruck der Versandmarken.

Ablauf der Versandabwicklung

Sind Sie einmal registriert, loggen Sie sich auf dieser Seite mit Ihren Zugangsdaten ein. Anschließend sehen Sie bereits die Menüstruktur mit den unterschiedlichen Funktionen (s. Bild 1). Hier interessiert uns aktuell der Bereich **Versenden**.

Einen Klick später landen Sie auf der IntraShip-Seite von DHL (s. Bild 2). Hier können Sie nun manuell die Daten für ein Paket eingeben (**Versandabwicklung Paket|Neuer Auftrag**). Unter **Absender** legen Sie Ihre eigenen Daten fest, unter **Empfänger** die Adresse des Empfängers. Das ist aber nicht unser Ziel – wir wollen ja DHL-Sendungen auf Basis der Daten einer Tabelle mit den Empfängerdaten erzeugen.

Um einen ersten Einblick zu gewinnen, was wir dazu benötigen, wählen wir den Menüpunkt **Versandabwick-**

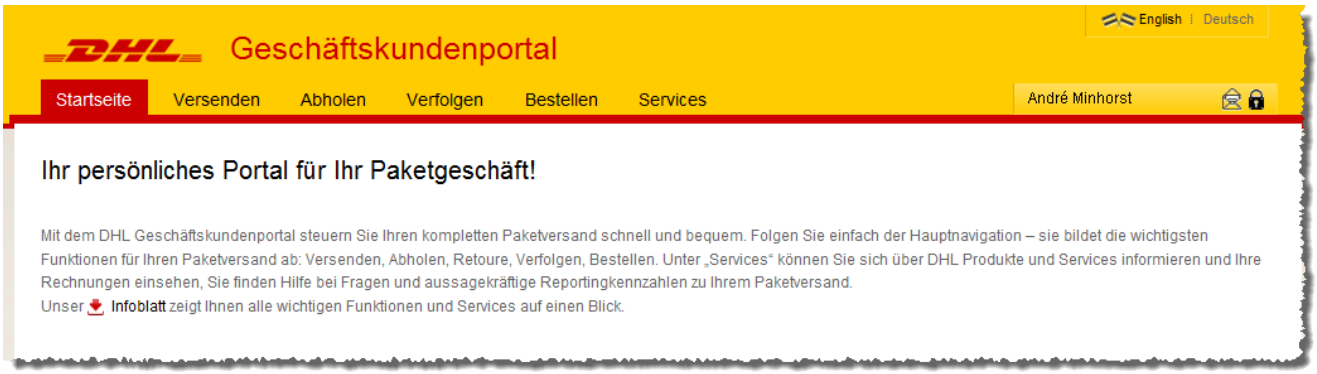

Bild 1: Funktionen im Geschäftskundenportal

Bild 2: Die Startseite von Intraship

Bild 3: Seite zum Importieren der CSV-Datei

Kombinationsfeld namens **Typ des Sendungsimports** auf einen der Werte **Standard Sendungsimport**, **Personalisierter Sendungsimport** oder **Ad-hoc-Sendungsimport** einstellen. Schließlich löst ein Klick auf die Schaltfläche **Datei importieren** den Import aus.

Informationen zu allen möglichen Belangen des Umgangs mit IntraShip erhalten Sie, wenn Sie auf den Menüpunkt **Info** klicken.

Dort finden Sie dann beispielsweise Links zu einem Benutzerhandbuch, zu den verschiedenen Import-Spezifikationen oder auch die Telefonnummern der unterschiedlichen Kundendienste.

Wir schauen also zunächst in das IntraShip-Benutzerhandbuch. Dieses liefert genau zu dem von uns benötigten Menüpunkt keine Informationen, also schauen wir uns die übrigen Dokumente an.

lung Paketlimport von Aufträgen aus. Damit landen wir in dem Dialog aus Bild 3. Hier erfahren wir, dass wir ein **Mandanten Standard Sendungsimport Format: 6.1_DE.csv** benötigen. Davon abgesehen können wir noch ein

Hier ist kein Dokument namens **Mandanten Standard Sendungsimport Format: 6.1_DE.csv** zu finden. Jede weitere Suche an dieser Stelle ist vergeblich, denn Sie erhalten nach der Freischaltung von IntraShip eine E-Mail

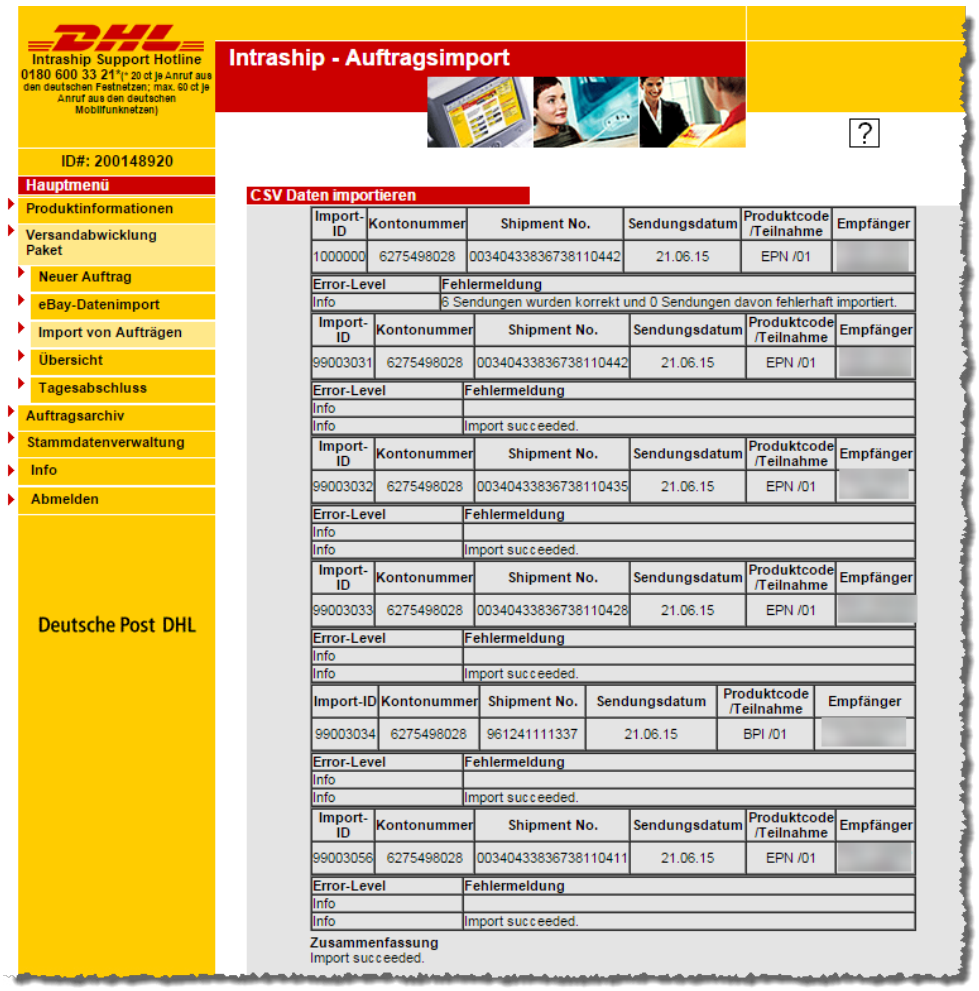

Bild 4: Ergebnis des Imports

mit der entsprechenden Dokumentation sowie Beispieldateien.

Ablauf beim Auftragsimport

Bevor wir uns im Detail ansehen, wie wir die benötigte CSV-Datei mit den Daten aus einer Datenbank füllen, sehen wir uns den grundlegenden Ablauf bei der Verwendung von IntraShip an.

Nach dem Klick auf **Versandabwicklung Paketlimport von Aufträgen** klicken Sie auf die Schaltfläche **Datei auswählen**. Im nun erscheinenden Dateiauswahl-Dialog geben Sie die zu importierende Datei an und klicken dann unter Beibehaltung der übrigen Einstellungen auf **Datei importieren**.

Das Ergebnis sieht optimalerweise wie in Bild 4 aus. Alle bereitgestellten Daten der CSV-Datei wurden ohne Fehler eingelesen. Dies erkennen Sie gleich im ersten Datensatz mit der Import-ID **1.000.000** (eigentlich sollte dieser Datensatz ganz unten stehen, was aber wegen der von mir verwendeten hohen Zahlen für die Import-ID fehlschlägt).

Darunter folgen die eigentlichen Positionen für die einzelnen Lieferungen, wobei diese leider nur recht wenige Informationen enthalten.

Wenn Sie mehr Details sehen möchten, wechseln Sie mit dem Menübefehl **Versandabwicklung PaketlÜbersicht** zu einer Listenansicht der aktuell beauftragten Sendungen (s. Bild 5). Hier können Sie nun mit einem Klick auf die Einträge der Spalte **Absender Kundennr.** eine Detailansicht des Auftrags ansehen – hier sind unter anderem eine Bearbeitung sowie das Löschen des Auftrags möglich.

Hier können Sie gerade nach den ersten Importversuchen prüfen, ob alle Informationen wie gewünscht übergeben wurden.

Die interessanteren Funktionen finden sich jedoch gleich in der Übersicht. Hier können Sie ein oder mehrere Aufträge auswählen und mit einem Klick auf die Schaltfläche **Label drucken** die Versandmarken im PDF-Format erstellen lassen. Ein Klick auf die Schaltfläche **Alle auswählen**

Klasse für DHL-Intraship-CSV-Dateien

Die CSV-Dateien, die für die Einreichung bei DHL für die Erstellung von Versandetiketten gefordert werden, sind erstens sehr umfangreich und zweitens auch noch komplex. Diese einfach per Code zusammenzusetzen mag auf den ersten Blick schnell gehen, aber was geschieht, wenn später einmal Änderungen nötig sind? Um die Programmierung des Exports einer CSV-Datei für DHL-Intraship zu vereinfachen, erstellen wir in diesem Beitrag einige Klassen, mit denen das Zusammenstellen der CSV-Dateien zum Kinderspiel wird.

Im Beitrag **Versandetiketten mit DHL-IntraShip** (**www.access-im-unternehmen.de/991**) zeigen wir, wie Sie eine CSV-Datei mit den Versandinformationen an die Webseite von DHL/IntraShip übergeben und damit entsprechende Versandetiketten erhalten. Dabei setzen Sie dort einen Satz von Klassen ein, welcher die einfache Erstellung der CSV-Datei ermöglicht. Den Aufbau und die Erstellung dieser Klassen möchten wir im vorliegenden Beitrag erläutern.

Ausgangsmaterial: PDF-Dokument mit CSV-Beschreibung

Grundlage für die Erstellung der Klasse ist ein PDF-Dokument, das die verschiedenen Sätze beschreibt, die Sie benötigen, um die Informationen zu einer Versendung unterzubringen. Dieses Dokument namens **Auftragsimport DD-Shipments 6.1 DE.PDF**, das Sie als Geschäftskunde anfordern können (wir können es leider nicht zum Download anbieten), ist immerhin 27 Seiten lang. Wir haben davon die für eine Sendung von Deutschland aus an ein Ziel innerhalb Deutschlands oder in ein anderes EU-Land nötigen Elemente in Form entsprechender Klassen umgesetzt. Es gibt noch einige weitere Elemente, die beispielsweise zur Erstellung einer ProForma-Rechnung oder der Zollpapiere für den Versand etwa in die Schweiz benötigt werden. Deren Umsetzung beschreibt der vorliegende Beitrag nicht.

Das PDF-Dokument enthält die Beschreibung verschiedener Satzarten, von denen einige verpflichtend sind, ein paar andere können optional hinzugefügt werden. Für einen Auftrag benötigen Sie auf jeden Fall eine Zeile der Satzart **Sendung**, die etwa so aussieht – jede Information ist durch das Pipe-Zeichen von der nächsten Information getrennt:

```
1234|DPEE-SHIPMENT|EPN|20150629|1.3||||AEMA|||||||||||
||||||100|EUR|||||||01||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
```

Sie enthält die grundlegenden Informationen zu der Sendung wie die Sendungsart, das Datum, das Gewicht, den Warenwert oder eine Sendungsreferenz. Außerdem enthält sie als Erstes eine Zahl, die für alle weiteren Sätze für diese Sendung gleich sein muss. Dementsprechend beginnt die zweite Satzart namens **Absender** auch mit dieser Zahl, enthält aber die Absenderinformationen:

```
1234|DPEE-SENDER|6275498028|André Minhorst
Verlag||André Minhorst|Borkhofer Str.|17||47137|Duis
burg|DE||andre@minhorst.com|0123/4567890|||||||||||||
|||||||
```

Das Element der Satzart **Empfänger** enthält logischerweise die Adresse und weitere Informationen zum Empfänger der Sendung:

```
1234|DPEE-RECEIVER|André Minhorst Verlag||||Herr
Klaus Müller|Borkhofer Str.|17||47137|Duisburg|DE||kla
us@mueller.de|||||||||||||||||
```

Danach folgt eine Zeile, welche die Daten für die Satzart **Packstück** aufnimmt (diese Zeile kann, wenn es sich um

mehrere Packstücke handelt, auch mehrfach vorkommen). Sie sieht so aus:

1234|DPEE-ITEM|1.3|||||PK||

Die letzte von uns in diesem Beitrag behandelte Zeile enthält die E-Mail-Adresse des Empfängers der Versandbenachrichtigung (Satzart **Benachrichtigungen**).

Auch hier können Sie mehrere Sätze der gleichen Art angeben, beispielsweise um sich selbst und dem Empfänger eine Benachrichtigung zu schicken, die etwa die Nummer für die Sendungsverfolgung enthält. Sie ist beispielsweise wie folgt aufgebaut:

1234|DPEE-NOTIFICATION||andre@minhorst.com|||

Warum Klassen?

Man könnte beispielsweise das erste Element mit den grundlegenden Informationen zur Sendung wie folgt in die CSV-Datei schreiben:

```
Print #1, lngKundeID & "|DPEE-SHIPMENT|" & strPaketart &
"|" & Format(Date, "yyyymmdd") & "|" & Replace(curGewicht,
",", ".") & "||||" & strKuerzel & "||||||||||||||||||" &
Replace(curWert, ",", ".") & "|EUR|||||||01|||||||||||||||
|||||||||||||||||||||||||||||||||||||||||||||"
```

Sobald Sie aber mal einen Parameter in der CSV-Datei benötigen, den Sie hier nicht berücksichtigt haben und der irgendwo zwischen dem 80. und 81. Pipe-Zeichen eingefügt werden muss, dürfen Sie erstmal die Dokumentation suchen, die Zeile anpassen et cetera.

Der Ansatz in der beschriebenen Codezeile ist Teil meines ersten Versuchs, die Erstellung der CSV-Datei zu automatisieren, und funktioniert natürlich gut – zumindest, bis man dann etwas an der Programmierung ändern oder, noch schlimmer, den Code anderen Entwicklern zugänglich machen möchte – also beispielsweise den Lesern dieses Beitrags.

Gerade in letzterem Fall werden doch die unterschiedlichsten Anforderungen zusammentreffen, die sich unmöglich mit einer einzigen Zeile Code je Element der CSV-Datei erledigen lassen. So ist dann nach einigem Hin und Her die Idee entstanden, ein System von Klassenmodulen aufzusetzen, von denen eines die Hauptschnittstelle zum Erstellen der CSV-Datei bildet und weitere Klassen jeweils eine Zeile der CSV-Datei repräsentieren.

CSV erstellen mit Klassen

Schauen wir uns zunächst an, wie Sie mit den nachfolgend erläuterten Klassen die CSV-Datei zusammenstellen können. Dies sieht in einem Beispiel für eine einzelne Sendung wie in Listing 1 aus. Keine Sorge: Der Code wird nicht wesentlich länger, wenn Sie beispielsweise die CSV-Datei für ein paar hundert Sendungen erstellen wollen – Sie müssen dann nur hier und da ein paar Schleifen hinzufügen und ein paar feste Zuweisungen durch Verweise auf die jeweiligen Feldinhalt ersetzen.

Die Prozedur deklariert ein Objekt auf Basis der Klasse **clsDPEEMain**. Dieses erstellt sie gleich in der darauffolgenden Zeile mit der **New**-Anweisung. Da wir nachfolgend einige Eigenschaften dieser Klasse einstellen möchten, verwenden wir nun die **With**-Anweisung, damit wir den Objektbezeichner **objDPEEMain** nicht in jeder Zeile wiederholen müssen.

Als Erstes legen wir die Ordnungsnummer fest. Dies ist die Nummer, die wir in jedem Satz als erstes Element angeben müssen, damit wir die einzelnen Sätze als zu einer Versendung gehörend markieren können. Gleichzeitig ist das auch die einzige Information, die wir direkt an diese Klasse übergeben.

Über diese Klasse greifen wir jedoch über Eigenschaften wie **DPEEShipment** (Satzart **Sendung**), **DPEESender** (Satzart **Absender**), **DPEEReceiver** (Satzart **Empfaenger**), **DPEEItem** (Satzart **Packstück**) und **DPEENotification** (Satzart **Benachrichtigungen**) auf die Objekte für die einzelnen Sätze beziehungsweise Zeilen in der CSV-Datei zu.

```
Public Sub BeispielDHL()
    Dim objDPEEMain As clsDPEEMain
    Set objDPEEMain = New clsDPEEMain
    With objDPEEMain
        .Ordnungsnummer = 1234
        With .DPEEShipment
            .Produktcode = eDHLPaket
            .Sendungsdatum = Date
            .Gewicht = 1.3
            .Sendungsreferenz = "AEMA"
            .Warenwert = 100
            .WarenwertWaehrung = "EUR"
            .Teilnahme = "01"
        End With
        With .DPEESender
            .Kundennummer = "6275498028"
            .Firmenname1 = "André Minhorst Verlag"
            .Kontaktperson = "André Minhorst"
            .Strasse = "Borkhofer Str."
            .Hausnummer = "17"
            .PLZ = "47137"
            .Stadt = "Duisburg"
            .Laendercode = LaendercodeAusLand("Deutschland")
            .Emailadresse = "andre@minhorst.com"
            .Telefonnummer = "0123/4567890"
        End With
        With .DPEEReceiver
            .Firmenname = "André Minhorst Verlag"
            .Firmenname2 = ""
            .Kontaktperson = "Herr Klaus Müller"
            .Strasse = "Borkhofer Str."
            .Hausnummer = "17"
            .PLZ = "47137"
            .Stadt = "Duisburg"
            .Land = LaendercodeAusLand("Deutschland")
            .Emailadresse = "klaus@mueller.de"
        End With
        With .AddDPEEItem
            .GewichtDesPackstueckesInKg = 1.3
            .PackartKollitraeger = ePaket
        End With
        With .AddDPEENotification
            .Emailadresse = "andre@minhorst.com"
        End With
        With .AddDPEENotification
            .Emailadresse = "test@test.de"
        End With
        Open CurrentProject.Path & "\dhltest.csv" For Output As #1
        Print #1, .Satz
        Close #1
    End With
End Sub
```

Listing 1: Erstellen einer CSV-Datei für den Versand

Über **DPEEShipment** stellen Sie so etwa den Produktcode, das Sendungsdatum oder das Gesamtgewicht ein.

DPEESender stellt Eigenschaften bereit, mit denen Sie die Kundennummer und die Adresse des Absenders festlegen.

Dementsprechend dient das Objekt **DPEEReceiver** dazu, die Adresse des Empfängers zu übergeben.

Auf die Objekte **DPEEItem** und **DPEE-Notification** greifen Sie nicht wie auf die anderen Elemente zu. Der Grund ist, dass Sie beide mehrfach anlegen können. Daher legen Sie diese jeweils mit der entsprechenden **Add...**-Methode an, also etwa **AddDPEEItem** oder **AddDPE-ENotification**. Da die beiden Methoden aber auch jeweils ein Element des Typs **clsDPEEItem** beziehungsweise **clsDPE-ENotification** zurückliefern, können Sie auf die **Add...**-Methode jeweils mit der **With**-Anweisung verweisen und darüber die einzelnen Eigenschaften festlegen – bei der Satzart **Packstück** also etwa das Gewicht und die Packart (Paket oder Palette).

Bei der Benachrichtigung verwenden Sie die Methode **AddDPEENotification**, für die Sie nur die jeweilige E-Mail-Adresse des Empfängers angeben müssen. Im Beispiel legen wir gleich zwei E-Mail-Adressen für die Versandbestätigung an – eine für den Sender und eine für den Empfänger.

Schließlich erstellt die Prozedur eine neue Textdatei mit der Endung **.csv**

```
1234|DPEE-SHIPMENT|EPN|20150629|1.3||||AEMA|||||||||||||||||100|EUR|||||||01|||||||||||||||||||||||||||||||
|||||||||||||||||||||||
1234|DPEE-SENDER|6275498028|André Minhorst Verlag||André Minhorst|Borkhofer Str.|17||47137|Duisburg|DE||
andre@minhorst.com|0123/4567890|||||||||||||||||||||
1234|DPEE-RECEIVER|André Minhorst Verlag||||Herr Klaus Müller|Borkhofer Str.|17||47137|Duisburg|DE||
klaus@mueller.de|||||||||||||||||
1234|DPEE-ITEM|1.3|||||PK||
1234|DPEE-NOTIFICATION||andre@minhorst.com|||
1234|DPEE-NOTIFICATION||test@test.de|||
```

Listing 2: Frisch erstellte CSV-Datei mit Sendungsdaten

und schreibt den Wert der Eigenschaft **Satz** des Objekts **objDPEEMain** hinein. Diese Eigenschaft liefert den Inhalt der CSV-Datei, den die Klasse auf Basis der übergebenen Werte zusammengestellt hat. Für unser Beispiel sieht diese Datei nun genau wie in Listing 2 aus.

Das Tolle bei der Verwendung dieser Klassen ist, dass Sie nicht erst die Dokumentation nach den Namen der einzelnen Eigenschaften einer Zeile der CSV-Datei suchen oder die Position innerhalb der Zeile ermitteln müssen, sondern einfach einen Punkt hinter dem Objektbezeichner einfügen und dann alle möglichen Eigenschaften per IntelliSense präsentiert bekommen (s. Bild 1).

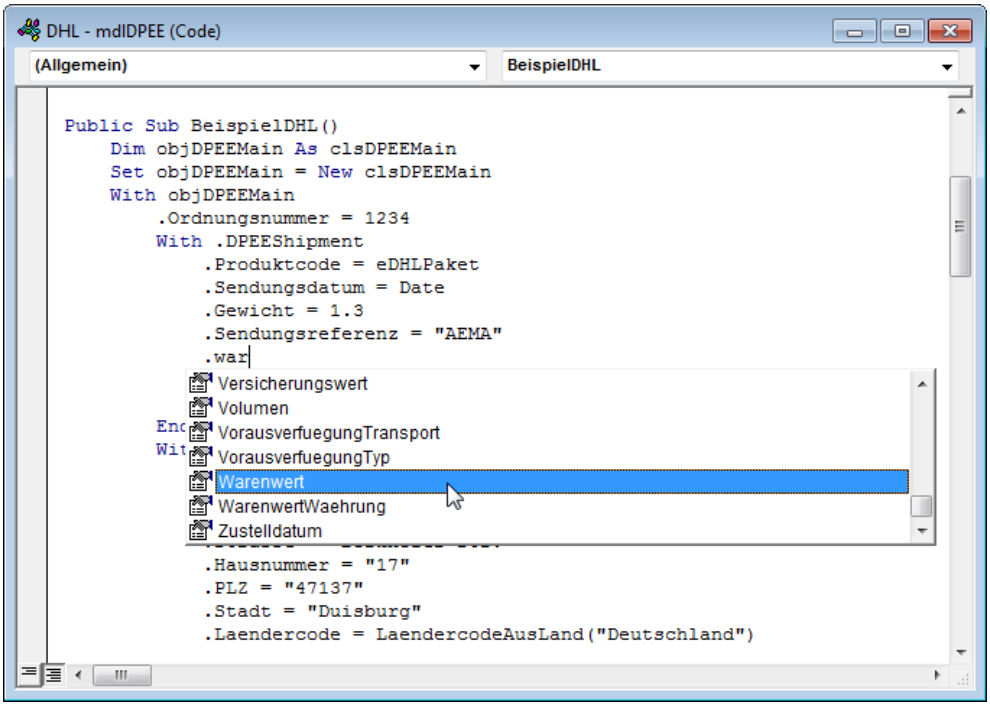

Bild 1: IntelliSense beim Programmieren der Erstellung der CSV-Datei

Versierte Access-Entwickler haben nach der Durchsicht dieses Beispielcodes schon eine Vorstellung, wie sie auf dieser Basis den Code erstellen, um die benötigten Informationen aus einigen Tabellen zusammenzusuchen und so in die CSV-Datei zu schreiben.

Klassen programmieren

Nun fehlt noch die Antwort auf die Frage: Wie kommt man von der Definition des Aufbaus der verschiedenen Satzarten aus einem 27 Seiten langen PDF-Dokument, das beispielsweise wie in Bild 2 aussieht, zu einer Reihe von Klassenmodulen, welche die Eingabe dann wie hier beschrieben erleichtern? Dazu gehört in der Tat entweder eine Menge Handarbeit oder, was wir Access-Entwickler eindeutig bevorzugen, ein paar Zeilen VBA-Code, die uns einen Großteil der Arbeit abnehmen.

Wir haben es im Detail mit fünf Tabellen zu tun, die von weniger als zehn bis zu knapp 100 Zeilen enthalten.

Jede Tabelle beschreibt eine Satzart, also benötigen wir für jede Tabelle eine eigene Klasse. Jede Zeile der Tabelle beschreibt

Fld Nr.	Beschreibung	Pflicht-feld	Feld-länge	Datentyp	Beispiel
1	Ordnungsnummer (kennzeichnet innerhalb der Auftragsdatei die Sendung). Alle zugehörigen Sätze zu einer Sendung verweisen auf diese Nummer	P	10	NUMMER	1
2	Satzart (beschreibt die Bedeutung des Satzes innerhalb der Datei)	P	30	TEXT	DPEE-SHIPMENT
3	Produktcode	P	6	TEXT	EPN
4	Sendungsdatum	P	8	DATUM	20060511
5	Summe Gewicht	A	22	NUMMER	23.23
6	Summe Volumen	O	22	NUMMER	15.25
7	Versicherungswert	O	22	NUMMER	23.23

Bild 2: Umzusetzende Beschreibung aus dem PDF

dhl_20150624.csv *

```
1       Ordnungsnummer  String  10
2       Satzart String  30
3       GewichtDesPackstueckesInKg      Currency
4       LaengeDesPackstueckesInCm       Long
5       BreiteDesPackstueckesInCm       Long
6       HoeheDesPackstueckesInCm        Long
7       PackstueckBeschreibung  String  60
8       PackartKollitraeger     String  5
9       Packstueckreferenz      String  35
10      Referenznummer  String  2
```

Bild 3: Felder im Textformat

Dummerweise können wir die Inhalte der Tabellen noch nicht einmal direkt in eine Excel- oder Access-Tabelle einfügen – zwar landet jede Zeile in einer eigenen Zeile der Zieltabelle, aber die Spalten werden alle in das erste Feld geschrieben. Also sind wir so vorgegangen: Wir haben die Tabellen zunächst in eine Text-Datei geschrieben, Tab-Zeichen zwischen die erste und zweite Spalte gebracht, in die dritte Spalte den Datentyp unter VBA eingetragen und die maximale Zeichenanzahl in die vierte Spalte geschrieben – alles jeweils durch Tabulatorzeichen voneinander getrennt. Das Zwischenergebnis sah dann wie in Bild 3 aus. Die Anzahl der manuellen Eingriffe hält sich hier noch in Grenzen.

eine Eigenschaft, also benötigen wir für jede Zeile eine private Variable in der Klasse, um den zugewiesenen Wert speichern zu können, sowie eine **Property Let**-Prozedur, um diese Eigenschaften von außerhalb der Klasse füllen zu können.

Danach haben wir den kompletten Inhalt in die Zwischenablage kopiert und dann in eine neue Tabelle namens **tblFelderDPEEItem** kopiert, die wir zuvor mit den vier benötigten Feldern ausgestattet haben. Der neue Zwischenstand ist in Bild 4 abgebildet.

Schließlich soll jede Klasse noch eine Eigenschaft namens **Satz** bereitstellen, die nach dem Zuweisen der Eigenschaften alle Eigenschaften auswertet und die Zeile der CSV-Datei für diese Satzart zusammenstellt und zurückgibt. Das wären für die fünf hier behandelten Sätze insgesamt 289 Zeilen, für die entsprechenden Eigenschaften, **Property Let**-Prozeduren und Einträge in die **Satz**-Eigenschaft vorgenommen werden müssten. Nein: Von Hand wollen wir das nicht erledigen.

tblFelderDPEEItem

ID	Feldname	Datentyp	Feldgroesse
1	Ordnungsnummer	String	10
2	Satzart	String	30
3	GewichtDesPackstueckesInKg	Currency	0
4	LaengeDesPackstueckesInCm	Long	0
5	BreiteDesPackstueckesInCm	Long	0
6	HoeheDesPackstueckesInCm	Long	0
7	PackstueckBeschreibung	String	60
8	PackartKollitraeger	String	5
9	Packstueckreferenz	String	35
10	Referenznummer	String	2
0			0

Datensatz: 1 von 10 | Kein Filter | Suchen

Bild 4: Felder in der Zieltabelle

Bestellverwaltung mit Versand

Zu einer ordentlichen Bestellverwaltung gehört die Möglichkeit, den Versand der bestellten Artikel vorzubereiten. In diesem Beitrag stellen wir das Grundgerüste einer einfachen Bestellverwaltung vor, deren Daten wir dann nutzen, um Versendungen über das DHL-Geschäftskundenportal vorzubereiten. Dazu müssen wir aus den eingegebenen Bestelldaten die Daten zu einer CSV-Datei zusammenführen, die wir dann an über das Internetportal von DHL einreichen und als Ergebnis die benötigten Versandetiketten erhalten.

Im Beitrag **Versandetiketten mit DHL-IntraShip** (**www.access-im-unternehmen.de/991**) erhalten Sie die grundlegenden Informationen über den Einsatz des Geschäftskundenportals von DHL, der Beitrag **Klasse für DHL-Intraship-CSV-Dateien** (**www.access-im-unternehmen.de/992**) liefert eine Bauanleitung für eine Klasse, mit der Sie die für die CSV-Datei benötigten Daten auf einfache Weise eingeben können.

Die Informationen dieser beiden Beiträge fließen in unsere Lösung ein, wobei wir die Daten aus Tabellen wie **tblKunden**, **tblBestellungen**, **tblBestellpositionen** und **tblArtikel** zusammensuchen.

Datenmodell

Das Datenmodell der Beispieldatenbank sieht wie in Bild 1 aus. Die enthaltenen Tabellen weisen lediglich die notwendigsten Felder auf, um Bestellungen aufzunehmen und die für den Versand notwendigen Dateien zusammenzustellen.

Dabei ist die Tabelle **tblKunden**, welche die üblichen Kundendaten enthält, per 1:n-Beziehung mit der Lookuptabelle **tblAnreden** verknüpft, welche die verschiedenen Anreden speichert. Auf die Kundentabelle wiederum verweist die Tabelle **tblBestellung** per Fremdschlüsselfeld. Auf diese Weise wird einer Bestellung der Kunde zugeordnet. Die Tabelle **tblArtikel** speichert ein paar grundlegende Artikeldaten wie den Artikelnamen oder den Einzelpreis. Für die Erstellung der Versanddatei ist es wichtig, das Gewicht der einzelnen Artikel zu kennen, daher finden Sie dort auch das Feld **Gewicht**. Schließlich nimmt das Feld **Kurzbezeichnung** eine Abkürzung für den Artikelnamen auf. Wozu wir dies benötigten, erfahren Sie weiter unten.

Die Bestellungen und die Artikel führt die Tabelle **tblBestellpositionen** zusammen. Sie ordnet die einzelnen Artikel aus der Tabelle **tblArtikel** den Bestellungen zu. Dabei sollen der aktuelle Preis und das aktuelle Gewicht des Artikels in zwei entsprechenden Feldern dieser Tabelle gespeichert werden. Damit sichern wir die zum Zeitpunkt

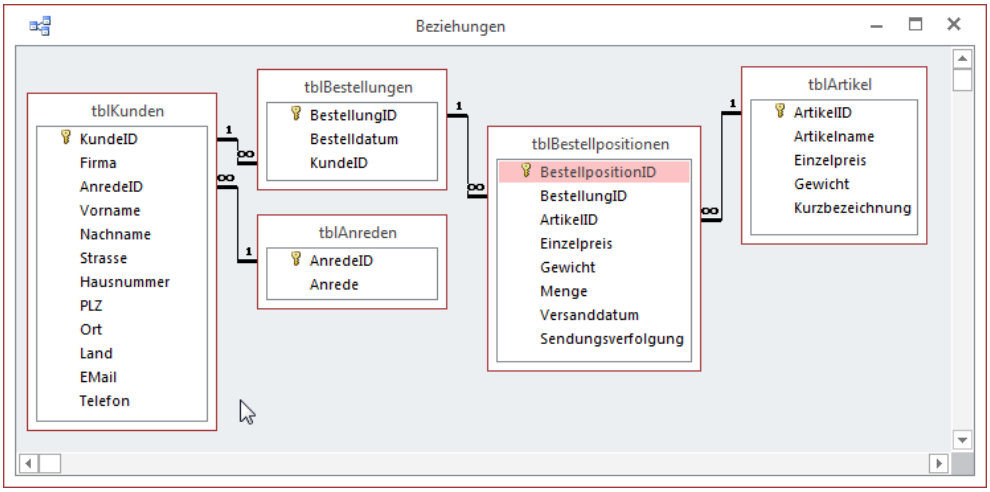

Bild 1: Datenmodell der Beispielanwendung

der Bestellung gültigen Daten, falls sich Preis oder Gewicht eines Artikels einmal ändern sollten.

Die Tabelle **tblBestellpositionen** nimmt aber auch noch ein Feld namens **Versanddatum** auf, welches bei Erstellung des Versandetiketts mit dem entsprechenden Datum gefüllt wird. Auf diese Weise erkennen wir, dass ein Artikel bereits versendet wurde. Schließlich finden Sie in dieser Tabelle das Feld **Sendungsverfolgung**. Wenn wir schon mit DHL-Paketen arbeiten, die sich ja online verfolgen lassen, wollen wir auch die dazugehörige Trackingnummer in der Tabelle speichern. Diese erhalten wir bei der Verwendung der nachfolgenden Technik zwar nur auf einem kleinen Umweg, aber immer noch schneller, als wenn wir diese von Hand eintragen.

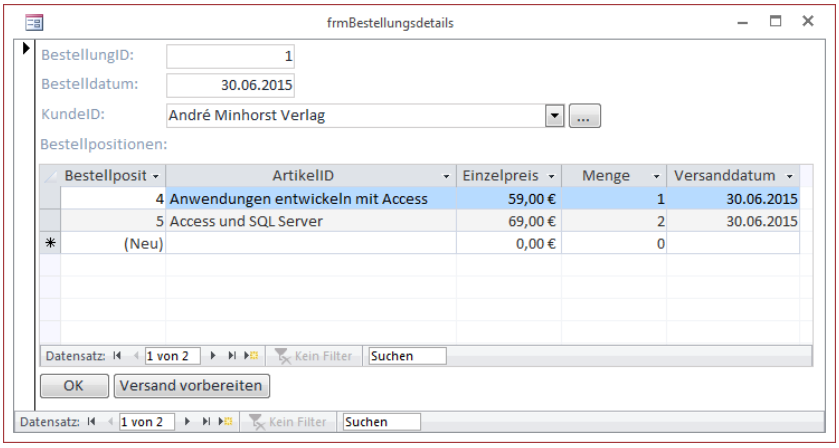

Bild 2: Das Formular **frmBestellungsdetails** in der Formularansicht

Formulare der Lösung

Das erste Formular heißt **frmBestellungsdetails** und soll die einfache Eingabe von Bestellungen ermöglichen (s. Bild 2). Dazu geben Sie das Bestelldatum im Feld **Bestelldatum** ein und wählen einen der bereits angelegten Kunden aus dem Kombinationsfeld **cboKundeID** aus. Gegebenenfalls handelt es sich um einen neuen Kunden – diesen können Sie dann über einen Klick auf die Schaltfläche mit den drei Punkten neben dem Kombinationsfeld in einem weiteren Formular hinzufügen.

Mit dieser Schaltfläche können Sie das gleiche Formular nach der Auswahl eines Kunden öffnen, um dessen Kundendaten zu bearbeiten (s. Bild 3).

Unten finden Sie ein Unterformular in der Datenblattansicht vor, mit dem Sie die Bestellpositionen zu der aktuellen Bestellung hinzufügen können. Dazu wählen Sie den gewünschten Artikel mit dem Kombinationsfeld aus.

Dadurch stellt das Unterformular automatisch die Inhalte der Felder **Einzelpreis** und **Gewicht** auf die in der zugrunde liegenden Tabelle **tblArtikel** ein und legt für das Feld **Menge** den Standardwert **1** fest. Ein Klick auf die Schaltfläche **cmdVersandVorbereiten** soll schließlich die CSV-Datei genau für diese eine Bestellung anlegen.

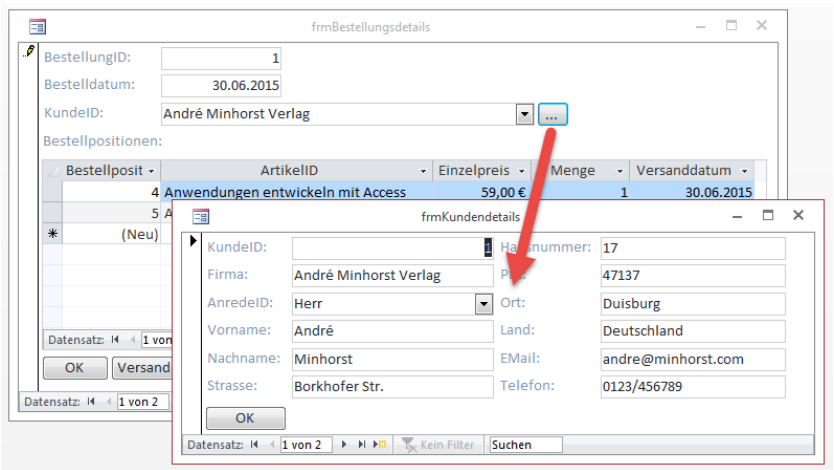

Bild 3: Bearbeiten der Kundendetails

Das zweite Formular namens **frmBestellungsdetails_Versand** ist etwas anders aufgebaut (s. Bild 4). Es soll zunächst alle Bestellpositionen, deren Gewicht größer als 0 ist, anzeigen – egal, ob diese bereits versendet wurden oder nicht. Oben bietet eine Optionsgruppe die Mög-

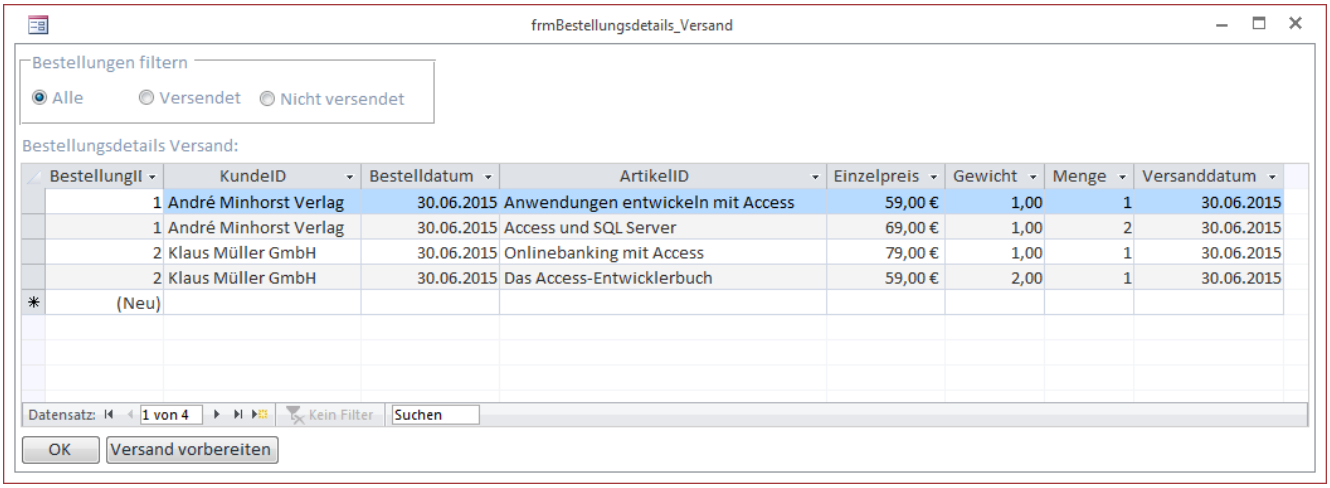

Bild 4: Das Formular **frmBestellungsdetails_Versand** in der Formularansicht

lichkeit, alternativ alle Bestellpositionen anzuzeigen, die bereits versendet wurden, oder diejenigen, die noch nicht versendet wurden.

Auch hier gibt es eine Schaltfläche namens **cmdVersand-Vorbereiten**. Diese berücksichtigt aber direkt alle noch offenen Bestellpositionen, also diejenigen, deren Feld Versanddatum noch leer ist.

Formular frmBestellungsdetails erstellen

Das Formular **frmBestellungsdetails** sieht im Entwurf wie in Bild 5 aus. Es verwendet die Tabelle **tblBestellungen** als Datenherkunft. Das Kombinationsfeld **cboKundeID** bezieht seine Daten aus der folgenden Abfrage:

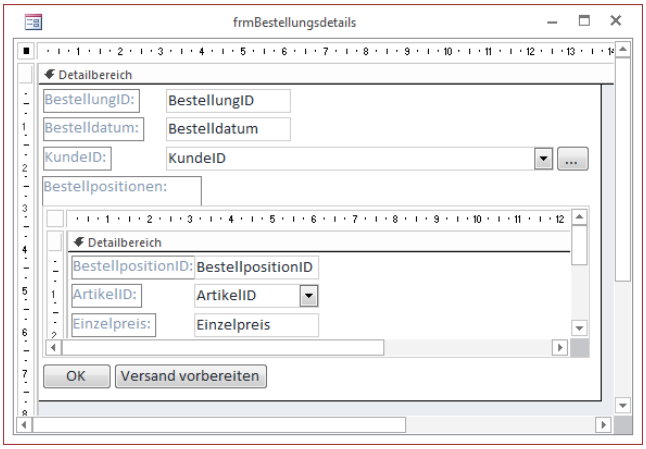

Bild 5: Das Formular **frmBestellungsdetails** in der Entwurfsansicht

```
SELECT [KundeID], [Firma] FROM tblKunden;
```

Wenn der Benutzer auf die Schaltfläche **cmdKundendetails** klickt, löst dies die Prozedur aus Listing 1 aus. Dies öffnet das Formular **frmKundendetails**. Wenn für das Kombinationsfeld **cboKundeID** noch kein Eintrag ausgewählt wurde, soll das Formular **frmKundendetails** einen neuen, leeren Datensatz anzeigen, anderenfalls den aktuell im Kombinationsfeld ausgewählten. Zu diesem Zweck wird der Primärschlüsselwert für diesen Datensatz per Öffnungsargument an das Formular **frmKundendetails** übergeben.

In beiden Fällen speichert die Prozedur nun einen Verweis auf das Formular in der Variablen **frmKundendetails**, welches im Kopf des Klassenmoduls des Formulars **frmBestellungsdetails** wie folgt deklariert wurde:

```
Dim WithEvents frmKundendetails As Form
```

Dadurch können wir in diesem Klassenmodul auch Ereignisprozeduren für dieses Formular implementieren. In diesem Fall möchten wir auf das Ereignis **Beim Entladen** reagieren, indem wir den aktuell im Formular **frmKundendetails** angezeigten Datensatz einlesen und im Kombinationsfeld **cboKundeID** auswählen. Dazu stellen wir zunächst die Eigenschaft **OnUnload** von **frmKundende-**

```
Private Sub cmdKundendetails_Click()
    If IsNull(Me!cboKundeID) Then
        DoCmd.OpenForm "frmKundendetails", DataMode:=acFormAdd
    Else
        DoCmd.OpenForm "frmKundendetails", DataMode:=acFormEdit, OpenArgs:=Me!cboKundeID
    End If
    Set frmKundendetails = Forms!frmKundendetails
    With frmKundendetails
        .Modal = True
        .OnUnload = "[Event Procedure]"
    End With
End Sub
```

Listing 1: Öffnen des Formulars zum Bearbeiten der Kundendetails oder zum Erstellen eines neuen Kunden

tails auf den Wert **[Event Procedure]** ein. Dieses Ereignis programmieren wir dann wie folgt:

```
Private Sub frmKundendetails_Unload(Cancel As Integer)
    If Not IsNull(frmKundendetails!KundeID) Then
        Me!cboKundeID.Requery
        Me!cboKundeID = frmKundendetails!KundeID
    End If
End Sub
```

Dies sorgt dafür, dass wenn der Benutzer das Formular **frmKundendetails** schließt, vorher noch das Ereignis **frmKundendetails_Unload** im Klassenmodul des Formulars **frmBestellungsdetails** ausgeführt wird und zweierlei erledigt: die Datensatzherkunft des Kombinationsfeldes **cboKundeID** zu aktualisieren und den aktuell im Formular angezeigten Datensatz dort auszuwählen.

Kundendetails bearbeiten

Damit das Formular **frmKundendetails** gleich beim Öffnen den richtigen Datensatz anzeigt, legen Sie die folgende Prozedur für das Ereignis **Beim Öffnen** des Formulars an:

```
Private Sub Form_Open(Cancel As Integer)
    If Not IsNull(Me.OpenArgs) Then
        Me.Recordset.FindFirst "KundeID = " & Me.OpenArgs
    End If
End Sub
```

Diese prüft, ob das Öffnungsargument einen Wert enthält, und stellt das Formular in diesem Fall auf den betroffenen Datensatz ein.

Versand vorbereiten

Die Schaltfläche **cmdVersandVorbereiten** im unteren Bereich löst die folgende Prozedur aus:

```
Private Sub cmdVersandVorbereiten_Click()
    Dim strSatz As String
    Dim strDateiname As String
    strSatz = SatzErstellen(Me!BestellungID)
    strDateiname = CurrentProject.Path & "\DHL_" _
        & Format(Date, "yyyymmdd") & ".csv"
    Open strDateiname For Output As #1
    Print #1, strSatz
    Close #1
    Call ShellExecute(Me.hWnd, "open", strDateiname, "", _
        "", SW_NORMAL)
End Sub
```

Diese ruft zunächst die Funktion **SatzErstellen** auf und übergibt dabei den Primärschlüsselwert der aktuell angezeigten Bestellung als Parameter. Diese Funktion erstellt die CSV-Datei auf Basis der Daten zu den Bestellpositionen dieser Bestellung und liefert den Inhalt im **String**-Format an die Variable **strSatz** zurück (siehe weiter unten). Die folgende Anweisung stellt einen Dateinamen zusammen, der aus dem Verzeichnis der aktuellen Datenbank, der Zeichenfolge **DHL_**, dem Datum im Format **yyyymmdd** und der Dateiendung **.csv** besteht, also etwa **DHL_20150630.csv**. Dann öffnet sie eine neue Textdatei unter diesem Namen, schreibt den Inhalt der Variablen **strSatz** hinein und schließt die Datei wieder. Schließlich öffnet die Prozedur die neu erstellte Datei noch mit der dafür vorgesehenen Anwendung – in diesem Fall

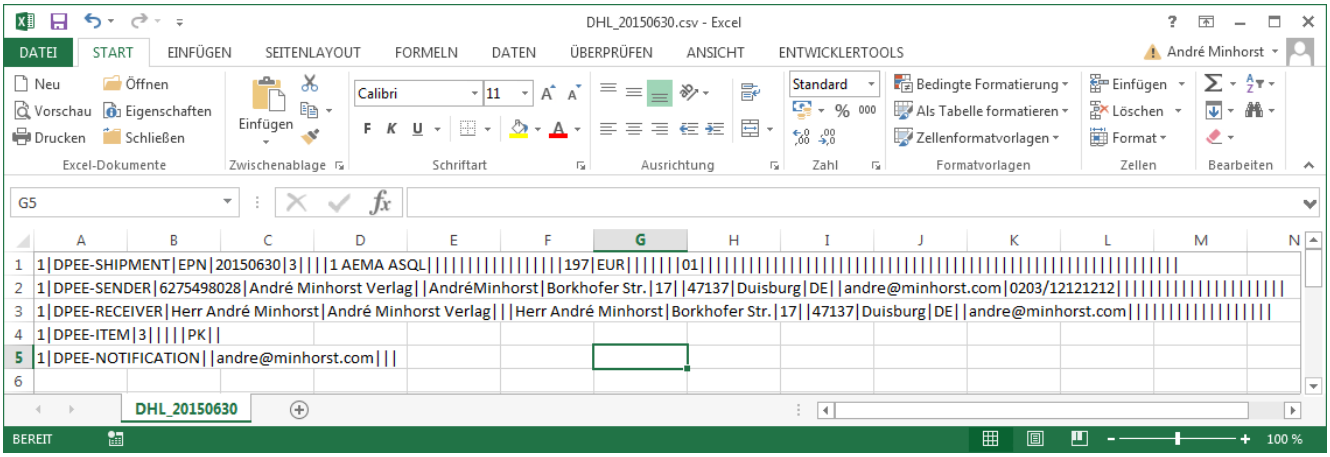

Bild 6: Frisch erstellte Versanddatei in Excel

mit Excel. Das Ergebnis sieht dann beispielsweise wie in Bild 6 aus.

Unterformular

Damit das Unterformular gleich den Einzelpreis, das Gewicht und die Menge für die Bestellposition einstellt, legen wir für das Ereignis **Vor Aktualisierung** des Kombinationsfeldes **cboArtikelID** noch die folgende Ereignisprozedur an:

```
Private Sub cboArtikelID_BeforeUpdate(Cancel As Integer)
    Me!Einzelpreis = DLookup("Einzelpreis", _
        "tblArtikel", "ArtikelID = " & Nz(Me!cboArtikelID))
    Me!Gewicht = DLookup("Gewicht", "tblArtikel", _
        "ArtikelID = " & Nz(Me!cboArtikelID))
    Me!Menge = 1
End Sub
```

Die Prozedur holt sich die entsprechenden Werte für den gewählten Artikel aus der Tabelle **tblArtikel**. Werden diese nun im Rahmen dieser Bestellung angepasst, hat das keine Auswirkungen auf den Originalpreis.

Sätze für die CSV-Datei erstellen

Die Prozedur **SatzErstellen** erwartet mit dem Parameter **lngBestellungID** den Primärschlüsselwert der Bestellung, deren Bestellpositionen für die Erstellung der Sätze für eine Verwendung herangezogen werden sollen (s. Listing 2).

Sie greift zunächst auf über das Recordset **rstOptionen** auf eine Tabelle namens **tblOptionen** zu, welche die Informationen zum Versender enthält. Wichtig ist hier vor allem das Feld **DHLKundennummer**, welches Sie mit der von DHL gelieferten Kundennummer füllen (s. Bild 7).

Ein weiteres Recordset namens **rstBestellungEmpfaenger** greift auf den Datensatz der Abfrage **qryBestellungEmpfaenger** zu, der dem per Parameter übermittelten Primärschlüsselwert entspricht (s. Bild 8).

Das dritte Recordset heißt **rstBestellpositionen** und greift auf die Abfrage **qryBestellpositionenNichtVersendet** zu. Diese liefert die Daten der Tabelle **tblBestellpositionen** plus das Feld **Kurzbezeichnung** der Tabelle **tblArtikel**, wobei nur solche Datensätze herangezogen werden, deren Feld **Versanddatum** noch leer ist. Außerdem muss der Wert des Feldes **Gewicht** größer als **0** sein (so werden beispielsweise eBooks oder sonstige digitale Artikel nicht berücksichtigt) und die **BestellungID** muss der aktuell bearbeiteten Bestellung entsprechen.

Die Funktion speichert dann zunächst den Primärschlüsselwert der Bestellung als Sendungsreferenz in der Variablen **strSendungsreferenz**. Anschließend durchläuft sie alle Datensätze des Recordsets **rstBestellpositionen**, um den Gesamtpreis und das Gesamtgewicht aller zu versendenden Artikel dieser Bestellung zu ermitteln. Und

Onlinebanking per Webservice III

In den ersten beiden Teilen dieser Beitragsreihe haben wir uns die Funktionen für das Einlesen von Informationen und das Konvertieren von Kontonummern in IBAN sowie das Ermitteln des Kontostandes angesehen. Im abschließenden, letzten Teil wird es spannend: Wir fügen Funktionen hinzu, mit denen Sie die Kontoumsätze abrufen und Überweisungen tätigen können. Damit ist auch der aktuelle Leistungsumfang des hier abgebildeten Webservice der Firma B+S Banksysteme AG beschrieben. Die Nutzung ist weiterhin für private Zwecke kostenlos!

Umsätze abrufen

Diese Funktion ist natürlich etwas umfangreicher als das Einlesen des Kontostandes, da wir ja nicht nur einen einzelnen Wert einlesen, sondern gleich eine ganze Reihe von Werten. Dazu benötigen wir natürlich eine eigene Tabelle, die wir **tblUmsaetze** nennen, und welche im Entwurf wie in Bild 1 aussieht.

Auf die einzelnen Felder gehen wir gleich bei der Beschreibung des Zugriffs auf den Webservice ein. Wichtig ist an dieser Stelle, dass die Tabelle nicht mit der Tabelle **tblKonten** verknüpft ist. Stattdessen enthält sie die beiden Felder **BankCode** und **Accountnumber**, welche die Bank und das Konto, über das die Transaktionen ausgeführt wurden, eindeutig identifiziert.

Banken fusionieren ja heutzutage gern mal oder ändern ihre Bankleitzahl aus anderen Gründen. Bei der hier durchgeführten Methode behalten Sie auf jeden Fall die Originalbankdaten bei. Es steht Ihnen natürlich frei, die Datensätze der Tabelle **tblUmsaetze** über ein Fremdschlüsselfeld mit der Tabelle **tblKonten** zu verknüpfen.

Weiterhin ist hier zu beachten, dass wir für das Feld **HashValue** einen eindeutigen Index definiert haben. Damit stellen wir sicher, dass ein bereits eingelesener Umsatz nicht nochmals eingelesen wird.

Formularelemente zum Abruf der Umsätze

Nun benötigen wir noch ein Element in der Benutzeroberfläche, um die Umsätze abzurufen und auch anzuzeigen. Dazu erweitern wir einfach das Formular **frmKonten**, und zwar um eine Schaltfläche namens **cmdUmsaetzeEinlesen**, zwei Textfelder namens **txtStartdatum** und **txtEnddatum**, mit denen Sie den Zeitraum für die einzulesenden Umsätze einstellen können, sowie ein Unterformular zur Anzeige der Umsätze (s. Bild 2).

Das Unterformular verwendet die Tabelle **tblUmsaetze** als Datenherkunft und zeigt davon die Felder **Amount**, **Valuta**, **Purpose**, **RecBankID**, **RecAccountNr** und **RecName** an.

tblUmsaetze

Feldname	Felddatentyp	Beschreibung (optional)
UmsatzID	AutoWert	Primärschlüsselfeld der Tabelle
BankCode	Kurzer Text	Bankleitzahl
Accountnumber	Kurzer Text	Kontonummer
BusinessTransactionCode	Kurzer Text	Transaktioncode
HashValue	Kurzer Text	Eindeutiger Wert für diesen Umsatz
Amount	Währung	Betrag
StatementNr	Zahl	Nummer
BookingDate	Datum/Uhrzeit	Buchungsdatum
Currency	Kurzer Text	Währung
Valuta	Datum/Uhrzeit	Valuta
BookingRef	Kurzer Text	Buchungsreferenz
Purpose	Langer Text	Verwendungszweck
RecBankID	Kurzer Text	Zielbank
RecAccountNr	Kurzer Text	Zielkonto
RecName	Kurzer Text	Zielname

Bild 1: Tabelle zum Speichern der Umsätze

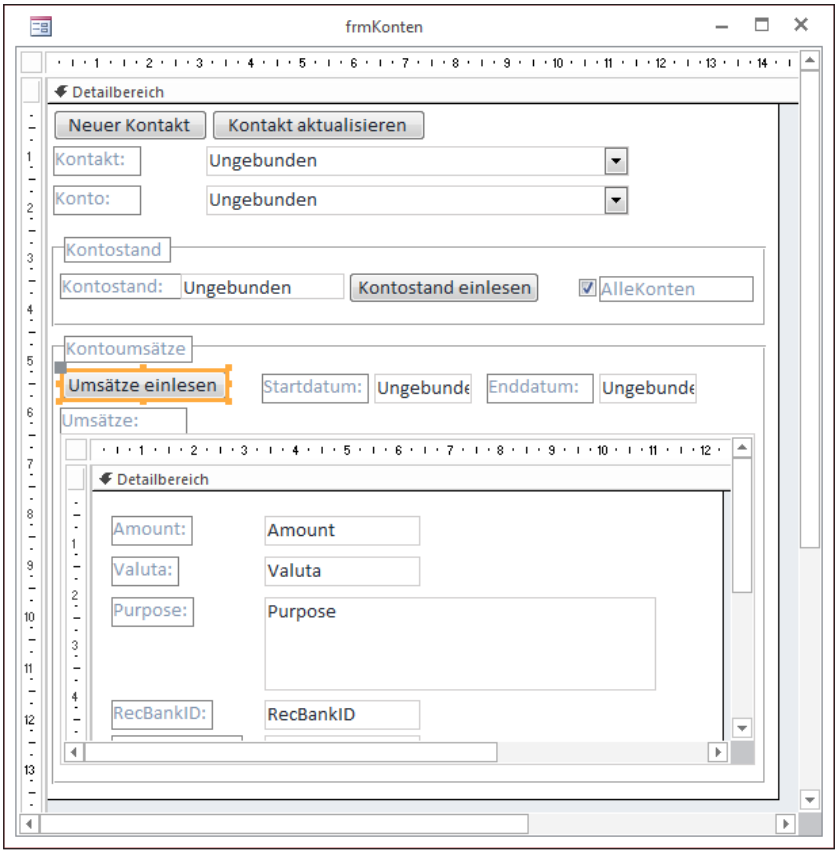

Bild 2: Anpassungen des Formulars **frmKonten**

Umsätze abrufen

Das Abrufen der Umsätze startet der Benutzer mit einem Klick auf die Schaltfläche **cmdUmsaetzeEinlesen**. Diese Prozedur sieht wie in Listing 1 aus und prüft zunächst, ob der Benutzer für das Kombinationsfeld **cboKontakte** überhaupt einen Wert ausgewählt hat.

Anderenfalls erscheint eine entsprechende Meldung und die Prozedur wird beendet.

Danach ermittelt die Prozedur den Wert des Feldes **ContactData** für den aktuellen Kontakt. Während die folgenden Werte direkt aus den weiteren Spalten des Kombinationsfeldes **cboKontakte** ausgelesen werden, entnehmen wir diesen Wert per **DLookup**-Funktion aus der Tabelle **tblKontakte**. Der Grund ist einfach: Jede Spalte des Kombinationsfeldes kann nur 255 Zeichen aufnehmen, **ContactData** ist aber länger.

Danach liest die Prozedur die Werte der Felder **AccountID** und Accountnumber der Tabelle **tblKontakte** ein, die wir aber bereits in den folgenden Spalten des aktuellen Eintrags des Kombinationsfeldes **cboKonten** gespeichert haben.

Dann setzt die Prozedur einen Aufruf der Funktion **StatementRequest** ab, welche ein XML-Dokument mit den angefragten Umsatz-Informationen zurückliefert. Die Funktion erwartet die Werte der Felder **ContactData** und **AccountID**, zwei Parameter, mit denen eventuelle Fehlerinformationen zurückgeliefert werden können, sowie das Start- und das Enddatum, wenn nur die Umsätze eines begrenzten Zeitraums eingelesen werden sollen.

Das zurückgelieferte XML-Dokument wertet die Prozedur dann gleich aus – zunächst, indem sie prüft, ob das Ergebnisdokument ein Element namens **SuccessText** enthält. Ist dies der Fall, war der Aufruf des Webservice erfolgreich und wir können uns an die Auswertung begeben. Das bedeutet in diesem Fall, dass wir den Inhalt des XML-Dokuments zusammen mit dem Wert von **strAccountnumber** an die Routine **Umsaetze-Verarbeiten** übergeben. Danach brauchen wir nur noch das Unterformular **sfmUmsaetze** zu aktualisieren, damit die neu eingelesenen Umsatzpositionen dort angezeigt werden.

Umsätze vom Webservice holen

Die Funktion **StatementRequest** aus Listing 2 erwartet die bereits erwähnten Parameter. Sie fragt als Erstes per **InputBox** den PIN für den Zugriff auf das Konto ab und speichert diesen in der Variablen **strPIN**.

Dann beginnt sie, den Request zusammenzusetzen. Dieses sieht anschließend beispielsweise wie folgt aus:

```
Private Sub cmdUmsaetzeEinlesen_Click()
    Dim strContactData As String
    Dim strAccountId As String
    Dim strAccountnumber As String
    Dim strErrorCode As String
    Dim strErrorText As String
    Dim strXML As String
    If IsNull(Me!cboKontakte) Then
        MsgBox "Wählen Sie einen Kontakt aus."
        Me!cboKontakte.SetFocus
        Exit Sub
    End If
    strContactData = DLookup("ContactData", "tblKontakte", "KontaktID = " _
        & Me!cboKontakte) 'wegen Längenbeschränkung auf 255 Zeichen
    strAccountId = Me!cboKonten.Column(5)
    strAccountnumber = Me!cboKonten.Column(2)
    strXML = StatementRequest(strContactData, strAccountId, strErrorCode, strErrorText, Nz(Me!txtStartdatum, 0), _
        Nz(Me!txtEnddatum, 0))
    If Len(GetXMLElement(strXML, "//SuccessText")) > 0 Then
        UmsaetzeVerarbeiten strXML, strAccountnumber
        Me!sfmUmsaetze.Form.Requery
    Else
        MsgBox "Fehler:" & vbCrLf & strErrorCode & vbCrLf & strErrorText
    End If
End Sub
```

Listing 1: Start des Einlesevorgangs der Umsätze

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.
org/soap/envelope/" xmlns:ser="http://service.ddbac.de/">
    <soapenv:Body>
        <ser:StatementRequest>
                <ser:StatementRequestData>
                    <ser:ContactData>GgxGb1115NitZ...
                        </ser:ContactData>
                    <ser:AccountId>5E3A4E4882B730A...
                        </ser:AccountId>
                </ser:StatementRequestData>
                <ser:Pin>12345</ser:Pin>
        </ser:StatementRequest>
    </soapenv:Body>
</soapenv:Envelope>
```

Wenn der Benutzer ein Start- oder Enddatum angegeben hat, wird die Anfrage noch entsprechend erweitert:

```
...
<ser:StartDate>2015-01-01</ser:StartDate>
<ser:EndDate>2015-05-31</ser:EndDate>
...
```

Danach ruft die Funktion die Routine **Request** auf und übergibt ihr den Request als ersten und ein leeres **DOM-Document**-Objekt als zweiten Parameter. Letzterer soll die Antwort des Webservice auf den Request aufnehmen.

Das Ergebnis dieses Aufrufs (die Routine **Request** haben wir bereits im ersten Teil der Beitragsreihe beschrieben) speichert die Prozedur dann in der String-Variablen **strResponse**. Die im Modul **mdlWebservice** befindliche Funktion **GetXMLElement** ermittelt aus dieser Antwort den Inhalt der Elemente **Result/Error/ErrorCode** und **Result/Error/ErrorText** und speichert diese, soweit gefüllt, in

```
Public Function StatementRequest(strContactData As String, strAccountId As String, strErrorCode As String, _
        strErrorText As String, Optional datStart As Date, Optional datEnde As Date) As String
    Dim objXMLResponse As MSXML2.DOMDocument
    Dim strRequest As String, strResponse As String, strPIN As String
    Dim strStartdate As String, strEnddate As String, strFunction As String
    strFunction = "StatementRequest"
    strPIN = InputBox("PIN:")
    strRequest = "                <ser:StatementRequestData>" & vbCrLf
    strRequest = strRequest & "                    <ser:ContactData>" & strContactData & "</ser:ContactData>" & vbCrLf
    strRequest = strRequest & "                    <ser:AccountId>" & strAccountId & "</ser:AccountId>" & vbCrLf
    If datStart > 0 Then
        strStartdate = Format(datStart, "yyyy-mm-dd")
        strRequest = strRequest & "                <ser:StartDate>" & strStartdate & "</ser:StartDate>" & vbCrLf
    End If
    If datEnde > 0 Then
        strEnddate = Format(datEnde, "yyyy-mm-dd")
        strRequest = strRequest & "                <ser:EndDate>" & strEnddate & "</ser:EndDate>" & vbCrLf
    End If
    strRequest = strRequest & "                </ser:StatementRequestData>" & vbCrLf
    strRequest = strRequest & "                <ser:Pin>" & strPIN & "</ser:Pin>"
    strRequest = CreateSoapRequest(strRequest, strFunction)
    Request strRequest, objXMLResponse
    strResponse = objXMLResponse.XML
    strErrorCode = GetXMLElement(strResponse, "//" & strFunction & "Result/Error/ErrorCode")
    strErrorText = GetXMLElement(strResponse, "//" & strFunction & "Result/Error/ErrorText")
    StatementRequest = FormatXML(objXMLResponse.XML)
End Function
```

Listing 2: Zusammensetzen des Requests und Zurückgeben des Ergebnisses

```
<soap:Envelope ...>
    <soap:Body>
        <StatementRequestResponse xmlns="http://service.ddbac.de/">
            <StatementRequestResult>
                <Error>
                    <ErrorType>BUSINESS</ErrorType>
                    <ErrorCode>9800</ErrorCode>
                    <ErrorText>Der Dialog wurde abgebrochen. Bitte melden Sie sich erneut an. (9800);...</ErrorText>
                    <ErrorCustomerText>Der Dialog wurde abgebrochen. Bitte melden Sie sich erneut an. (9800);
                        Ungültige Dialogkennung. Bitte melden Sie sich erneut an. (9010)...</ErrorCustomerText>
                    <NoFurtherRequestsPreferred>false</NoFurtherRequestsPreferred>
                </Error>
            </StatementRequestResult>
        </StatementRequestResponse>
    </soap:Body>
</soap:Envelope>
```

Listing 3: Rückgabe für eine fehlerhafte Anfrage

```
Public Sub UmsaetzeVerarbeiten(strXML As String, strAccountnumber As String)
    Dim objXML As MSXML2.DOMDocument
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim objBACStatementLine As MSXML2.IXMLDOMNode
    Dim strBankCode As String
    Dim strCustomerID As String
    Set objXML = New MSXML2.DOMDocument
    objXML.loadXML strXML
    strBankCode = objXML.selectSingleNode("//CustomerData/BankCode").nodeTypedValue
    strCustomerID = objXML.selectSingleNode("//CustomerData/CustomerId").nodeTypedValue
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM tblUmsaetze", dbOpenDynaset)
    For Each objBACStatementLine In objXML.selectNodes("//BACStatementLine")
        rst.AddNew
        rst!BankCode = strBankCode
        rst!AccountNumber = strAccountnumber
        rst!BusinessTransactionCode = objBACStatementLine.selectSingleNode("BusinessTransactionCode").nodeTypedValue
        rst!HashValue = objBACStatementLine.selectSingleNode("HashValue").nodeTypedValue
        rst!Amount = Eval(objBACStatementLine.selectSingleNode("Amount").nodeTypedValue)
        rst!StatementNr = objBACStatementLine.selectSingleNode("StatementNr").nodeTypedValue
        rst!BookingDate = DatumAusXML(objBACStatementLine.selectSingleNode("BookingDate").nodeTypedValue)
        rst!Currency = objBACStatementLine.selectSingleNode("Currency").nodeTypedValue
        rst!Valuta = DatumAusXML(objBACStatementLine.selectSingleNode("Valuta").nodeTypedValue)
        rst!BookingRef = objBACStatementLine.selectSingleNode("BookingRef").nodeTypedValue
        On Error Resume Next
        rst!Purpose = objBACStatementLine.selectSingleNode("Purpose").nodeTypedValue
        rst!RecBankId = objBACStatementLine.selectSingleNode("RecBankId").nodeTypedValue
        rst!RecAccountNr = objBACStatementLine.selectSingleNode("RecAccountNr").nodeTypedValue
        On Error GoTo 0
        rst!RecName = objBACStatementLine.selectSingleNode("RecName").nodeTypedValue
        On Error Resume Next
        rst.Update
        Select Case Err.Number
            Case 0, 3022
            Case Else
                MsgBox "Fehler " & Err.Number & ", " & Err.Description
        End Select
        On Error GoTo 0
    Next objBACStatementLine
End Sub
```

Listing 4: Auslesen und speichern des Inhalts des XML-Dokuments mit den Umsätzen

den beiden Variablen **strErrorCode** und **strErrorText**. Ein **Response**-Dokument mit diesen Fehlerinformationen sieht beispielsweise wie in Listing 3 aus. Sie können diesen Fehler beispielsweise hervorrufen, indem Sie im Testsystem als PIN einen Wert eingeben, der mit **0** beginnt (also etwa **01111**). Die Prozedur gibt neben eventuell

vorhandenen Fehlerinformationen das zurückgelieferte XML-Dokument als Funktionswert an die aufrufende Prozedur zurück.

Umsätze verarbeiten

Die Prozedur **UmsaetzeVerarbeiten** aus Listing 4 erwartet das XML-Dokument, das die Funktion **StatementRequest** geliefert hat, sowie die Kontonummer als Parameter.

Sie erstellt zunächst ein neues **DOMDocument**-Objekt und füllt dieses mit dem Inhalt aus **strXML**, der etwa wie in Listing 5 aussieht. In diesem Dokument interessieren uns die einzelnen Elemente unterhalb des Elements **StatementLines**. Jedes Element namens **BACStatementLine** enthält nämlich genau einen Buchungssatz. Die einzelnen dort enthaltenen Elemente liefern jeweils eine Information zur Buchung, zum Beispiel den Betrag, die Währung oder das Buchungsdatum. Außerdem liefert das Dokument im Element **CustomerData** (hier zur Platzersparnis weggelassen) noch die Informationen zum Bankkonto.

Die Prozedur liest per **selectSingleNode** beispielsweise das Element **CustomerData/BankCode** ein und speichert

```
<soap:Envelope ...">
    <soap:Body>
        <StatementRequestResponse xmlns="http://service.ddbac.de/">
            <StatementRequestResult>
                <CustomerData>...</CustomerData>
                <ContactData>GgxGb1115Nit...</ContactData>
                <AccountId>C29990EF1F05175C1E952DF2005498CC79829427</AccountId>
                <StatementLines>
                    <BACStatementLine>
                        <BusinessTransactionCode>51</BusinessTransactionCode>
                        <HashValue>62F619DF4221AD05958F6AF06957438D4E6816C8</HashValue>
                        <Amount>1</Amount>
                        <StatementNr>54</StatementNr>
                        <BookingDate>2015-06-26T00:00:00</BookingDate>
                        <Currency>EUR</Currency>
                        <Valuta>2015-06-26T00:00:00</Valuta>
                        <BookingRef>NONREF</BookingRef>
                        <BookingText>Überweisungseingang</BookingText>
                        <Purpose>Testlastschrift</Purpose>
                        <RecBankId>DDBADEMM002</RecBankId>
                        <RecAccountNr>DE95700009972000735160</RecAccountNr>
                        <RecName>USER 735160</RecName>
                    </BACStatementLine>
                </StatementLines>
                <SuccessText>Die Nachricht wurde entgegengenommen. (0010); Der Auftrag wurde ausgeführt. (0020)
                    </SuccessText>
            </StatementRequestResult>
        </StatementRequestResponse>
    </soap:Body>
</soap:Envelope>
```

Listing 5: XML-Dokument mit den Umsatzdaten, hier nur für eine Umsatzposition