HAUFE.
Ausgabe 03/2025

ACCESS
IM UNTERNEHMEN

SQL SERVER-TABELLEN EINFACH VERKNÜPFEN

Nutzen Sie unser Add-In, um Tabellenverknüpfungen zum SQL Server blitzschnell zu aktualisieren (ab Seite 60).

In diesem Heft:

ANLAGEFELDER ZUM SQL SERVER

Lernen Sie, wie Sie Access-Anlagefelder zum SQL Server migrieren.

SEITE 38

SPALTENBREITEN UND -POSITIONEN SPEICHERN

Wie können Sie benutzerdefinierte Anpassungen eines Datenblatts sogar in .accde-Dateien speichern?

SEITE 10

MARKIERTE ZEILEN IM DATENBLATT AUSLESEN

Das Auslesen der Datenblattmarkierung ist kein Hexenwerk – mit den Tricks aus diesem Beitrag.

SEITE 2

Access neu erleben

Es sind oft die kleinen Dinge, die den großen Unterschied machen. Gerade bei einem Werkzeug wie Access, das viele von uns täglich einsetzen, passiert es schnell, dass man sich mit dem zufrieden gibt, was „immer schon funktioniert hat“. Dabei bietet Access in seinen Details so viel mehr – man muss nur wissen, wo man hinschauen muss. Diese Ausgabe lädt Sie dazu ein, bekannte Funktionalität einmal mit frischem Blick zu betrachten. Nehmen wir zum Beispiel die Datenblattansicht: Sie gilt als nüchternes Arbeitsmittel, um schnell Daten durchzusehen. Aber was, wenn Ihre Anwendung sich merkt, welche Spalten der Benutzer ausgeblendet hat, welche Breiten diese Spalten hatten und in welcher Reihenfolge sie angezeigt wurden? Und was, wenn Sie all das auch in einer ACCDE-Datei erhalten könnten?

Genau das zeigen wir im Beitrag **Spaltenbreiten und -position in ACCDEs speichern** ab Seite 10. Eine Komfortfunktion, die auch Ihre Nutzer begeistern wird.

Effizienzgewinne gibt’s aber auch über die Tastatur: In der Registerkartenansicht kann man zwar mit der Maus zwischen Formularen wechseln – aber das geht auch schneller. Wir zeigen Ihnen, wie Sie mit nur wenigen Zeilen Code die Tasten **Alt + 1** bis **Alt + 9** belegen, um blitzschnell von Formular zu Formular zu springen – wie in einer modernen Webanwendung. Mehr dazu im Beitrag **Access-Formulare per Tastenkombination wechseln** ab Seite 22.

Wer regelmäßig mit der Datenblattansicht arbeitet, kennt das Problem: Mehrere Einträge markieren – und dann? Wie lässt sich diese Auswahl weiterverarbeiten? Im Beitrag **Mehrere markierte Datensätze aus dem Datenblatt einlesen** (ab Seite 2) finden Sie eine kompakte Lösung, mit der Sie nicht nur die Auswahl auslesen, sondern auch gezielt auf die markierten Datensätze zugreifen können – sei es zur Verarbeitung oder Ausgabe.

Und dann wäre da noch ein Thema, das aus der Welt der Webformulare kommt, aber auch in Access-Formularen für mehr Benutzerfreundlichkeit sorgt: Platzhaltertexte direkt im Eingabefeld. »Bitte Kundennummer eingeben« oder »PLZ, z. B. 12345« – solche Hinweise machen das Interface klarer und sparen Platz. Wie Sie diese Funktion auch in Access realisieren – und das sogar dynamisch – zeigt Ihnen der Beitrag **Platzhalter für Textfelder in Formularen** ab Seite 29.

Auch der SQL Server kommt wieder zu seinem Recht: Wir zeigen Ihnen im Beitrag **Anlagefelder mit mehreren Dateien zum SQL Server** ab Seite 51, wie Sie mehrere Dateien aus einem Anlagefeld korrekt in die SQL Server-Welt überführen – eine Herausforderung, die bei vielen Migrationsprojekten auftaucht.

Dazu passend: **SQL Server-Tabellenverknüpfungsassistent, Teil 2** ab Seite 60. In dieser Fortsetzung unseres beliebten Add-Ins zeigen wir, wie Sie nicht nur Tabellen verknüpfen, sondern dabei auch bestehende Verknüpfungen intelligent erkennen, Namen flexibel vergeben und die Funktionalität für wechselnde Serververbindungen verbessern. Ideal für Projekte, bei denen Automatisierung und Wiederverwendbarkeit gefragt sind.

Wenn Sie also gedacht haben, dass Sie mit Access schon alles gesehen haben – dann wird Ihnen diese Ausgabe hoffentlich zeigen, dass es sich lohnt, gewohnte Wege zu hinterfragen.

Viel Freude beim Lesen, Entdecken und Umsetzen wünscht Ihnen

Ihr André Minhorst

Markierte Datensätze aus dem Datenblatt einlesen

Die Datenblattansicht ist sehr hilfreich, wenn man die Daten aus Tabellen oder Abfragen in tabellarischer Form darstellen möchte. Sie bietet außerdem Funktionen zum Anpassen der Spalten, der Sortierung und auch der Filter. Zusätzlich können wir damit nicht nur einen Datensatz markieren, sondern gleich mehrere. Wir erhalten zwar nicht den Komfort wie im Listenfeld, wo wir nicht nur zusammenhängende, sondern auch einzelne Einträge selektieren können, aber immerhin ist es grundsätzlich möglich, bei gedrückter Maustaste mehr als einen Eintrag auszuwählen. Die Frage ist nur: Was machen wir mit diesem selektierten Bereich? Das Listenfeld bietet eigene Eigenschaften, mit denen wir auf die markierten Elemente zugreifen können. Das ist in der Datenblattansicht nicht der Fall. Das soll uns jedoch nicht davon abhalten, die gewünschten Daten weiterzuverarbeiten.

Wenn wir uns eine Datenblattansicht wie in Bild 1 ansehen, können wir verschiedene Arten von Selektionen durchführen. Die einfachste Selektion ist, in ein Feld zu klicken. Dies fügt an der Stelle des Klicks die Einfügemarke ein. Den Wert dieses Feldes können wir beispielsweise vom Direktbereich des VBA-Editors aus mit dem folgenden Ausdruck ermitteln:

```
Debug.Print Screen.ActiveControl.Value
  Knuth
```

Wir können auch den Namen des aktuellen Steuerelements abrufen:

```
Debug.Print Screen.ActiveControl.Name
                                txtVorname
```

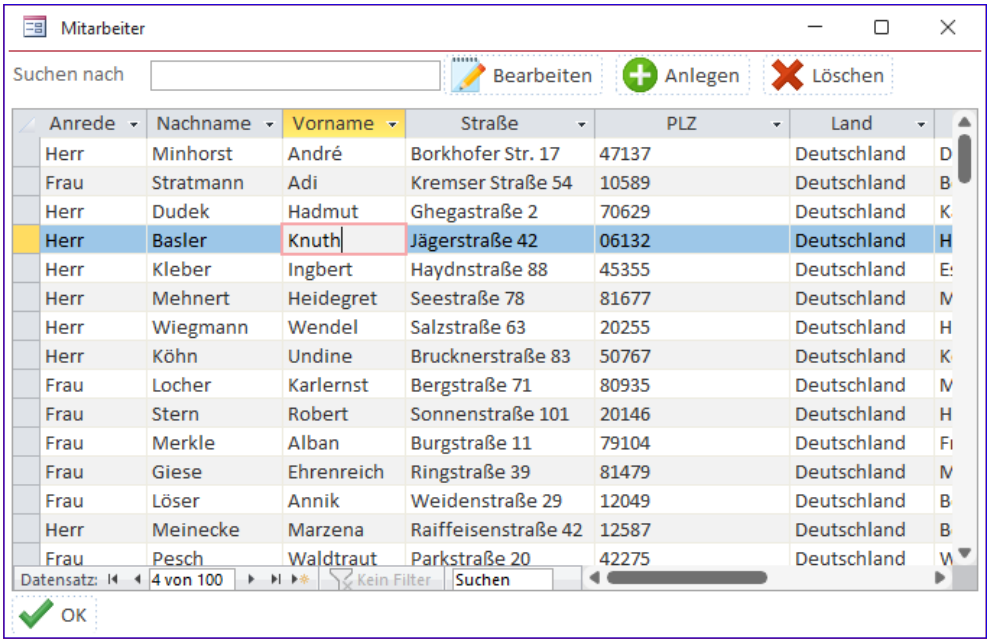

Bild 1: Selektieren eines einzelnen Feldes

Werte der anderen Felder des markierten Datensatzes auslesen

Auch wenn sich die Einfügemarke nur in einem Feld befindet, so wird doch der vollständige Datensatz markiert.

Aus dem Direktbereich heraus können wir wie folgt auf einen Wert eines anderen Steuerelements dieses Datensatzes zugreifen und diesen beispielsweise dort ausgeben:

```
Debug.Print Screen.Active-
Form!sfmMitarbeiterUeber-
sicht!MitarbeiterID
 8
```

Direkt vom Klassenmodul des Unterformulars **sfmMitarbeiterUebersicht** greifen wir noch einfacher auf diesen Wert zu:

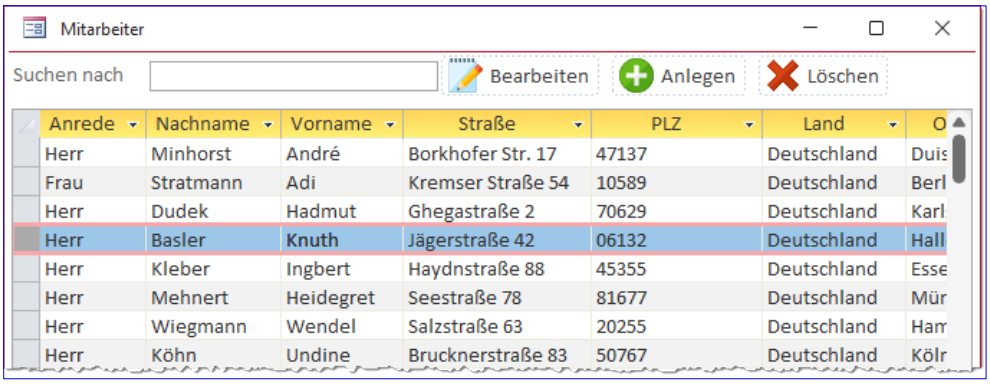

Bild 2: Markieren eines vollständigen Datensatzes

```
Debug.Print Me!MitarbeiterID
```

Auf die gleiche Weise können wir auch auf die Inhalte der übrigen Felder per Feldname oder Steuerelementname zugreifen.

Markieren eines kompletten Datensatzes

Auch wenn das Anklicken eines einzigen Feldes bereits die gesamte Zeile markiert hat, gibt es noch eine weitere Abstufung. Wenn wir auf den grauen Bereich links vom Datensatz klicken, den sogenannten Datensatzmarkierer, markieren wir den vollständigen Datensatz (siehe Bild 2).

Der Unterschied ist, dass nun kein einzelnes Steuerelement mehr die Einfügemarke anzeigt, sondern alle Felder markiert und von einem Rahmen umgeben werden.

Screen.ActiveControl.Name liefert nun den Namen des ersten sichtbaren Steuerelements der Zeile.

Wenn wir auf die Daten des aktuell markierten Datensatzes zugreifen wollen, können wir einfach die einzelnen Steuerelemente auslesen– diese liefern die Werte für den aktuellen Datensatz.

Markieren mehrerer Datensätze

Damit kommen wir zum Kern des Beitrags, dem Markieren zweier oder mehrerer zusammenhängender Datensätze gleichzeitig. Das gelingt beispielsweise auf die folgenden Arten:

- mit der Maus durch Markieren des Datensatzmarkierers des obersten oder untersten zu markieren Daten-

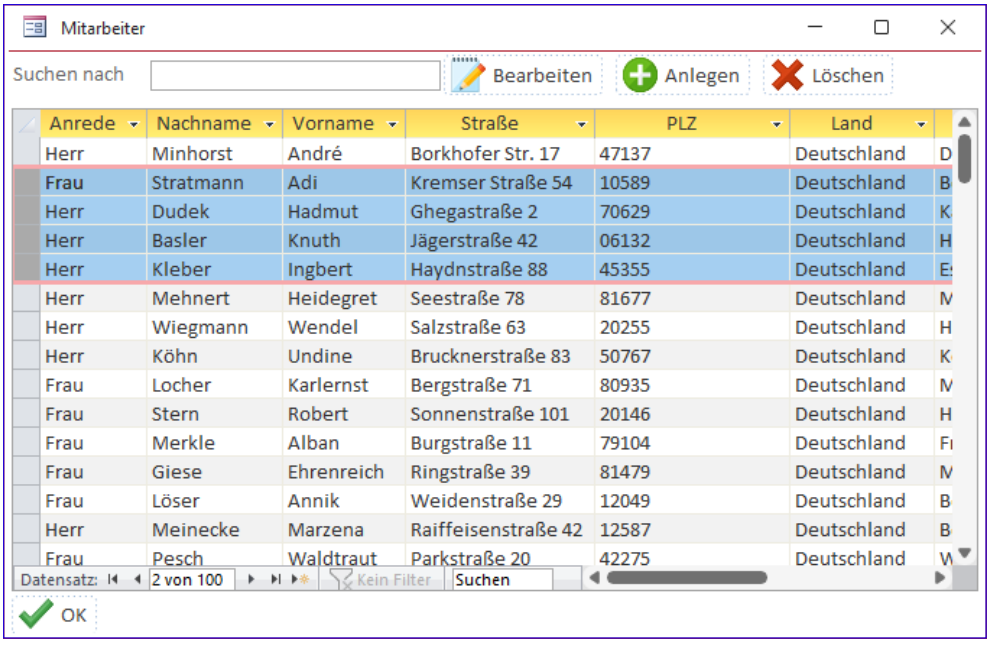

Bild 3: Markieren mehrerer Datensätze in der Datenblattansicht

satzes und Ziehen bis zum Datensatzmarkierer am anderen Ende der Markierung,

- durch Anklicken des Datensatzmarkierers des ersten zu markierenden Datensatzes und Anklicken des Datensatzmarkierers eines anderen Datensatzes bei gedrückter Umschalttaste oder

- durch Anklicken des Datensatzmarkierers des ersten zu markierenden Datensatzes und Erweitern der Markierung mit den Tastenkombinationen **Umschalt + Nach oben** oder **Umschalt + Nach unten**.

Wir klicken mit der Maus auf den Datensatzmarkierer des obersten oder untersten zu markierenden Datensatzes und ziehen die Maus bei gedrückter Maustaste nach oben oder unten, je nachdem in welche Richtung wir die Markierung erweitern wollen.

Ein Datenblatt mit mehreren markierten Datensätzen sehen wir in Bild 3.

Der folgende Ausdruck liefert nun den Wert des linken, oberen Feldes des Datensatzes, der als Erstes markiert wurde:

```
Debug.Print Screen.ActiveControl.Value
```

Sollten wir also einen Datensatz markieren und die Markierung nach unten erweitern, werden die Daten des obersten Datensatzes geliefert. Wenn wir die Markierung nach oben ziehen, erhalten wir die Daten des untersten Datensatzes der Markierung.

Wenn wir wie folgt gezielt auf ein Steuerelement zugreifen wollen, erhalten wir ebenfalls die Werte der Steuerelemente der Zeile der Markierung, die als erste selektiert wurde:

```
Debug.Print Forms!frmMitarbeiterUebersicht!sfmMitarbeiterUebersicht!Vorname
```

Auslesen aller Zeilen der Markierung

Die eingebauten Werkzeuge zum Auslesen der Markierung sind zwar gegeben, allerdings kommen wir nicht ohne Weiteres an die in den markierten Zeilen enthaltenen Daten heran.

Die Markierung jedoch können wir mit den folgenden Eigenschaften auslesen:

- **SelLeft**: Index der linken Spalte der Selektion inklusive der ausgeblendeten Spalten, beginnt immer mit dem Wert 2

- **SelTop**: Index der obersten Spalte der Selektion

- **SelWidth**: Anzahl der Spalten der Selektion, inklusive der ausgeblendeten

- **SelHeight**: Anzahl der Zeilen der Selektion

Diese Daten lassen wir uns mit der folgenden Prozedur für das im Formular **frmMitarbeiterUebersicht** enthaltene Unterformular **sfmMitarbeiterUebersicht** ausgeben:

```
Public Sub MarkierungErfassen()
    Dim sfm As Form
    Set sfm = Forms!frmMitarbeiterUebersicht!
                                sfmMitarbeiterUebersicht.Form
    With sfm
        Debug.Print "SelLeft:   " & .SelLeft
        Debug.Print "SelTop:    " & .SelTop
        Debug.Print "SelWidth:  " & .SelWidth
        Debug.Print "SelHeight: " & .SelHeight
    End With
End Sub
```

In Bild 4 sehen wir die beispielhafte Ausgabe dieser Werte für die angezeigte Markierung. **SelLeft** liefert unerwarteterweise den Wert 2, obwohl bereits die erste Spalte markiert ist. Die Werte für die anderen Eigenschaften sind nachvollziehbar.

Auslesen der Daten der Selektion

Wir wollen nun die Primärschlüsselwerte der selektierten Zeilen ermitteln.

Wie oben erwähnt, gelingt uns das durch Referenzieren des Feldes **MitarbeiterID** nur für den ersten Datensatz der Selektion – und hier erscheint auch noch der Wert des Datensatzes, mit dem die Markierung gestartet wurde.

Im Beispiel wäre das, wenn wir die Markierung ausgehend von der oberen Zeile aus angelegt hätten, der Wert **6**. Wenn wir die Markierung jedoch unten begonnen hätten, würde **sfm!MitarbeiterID** den Wert **9** liefern.

Wie kommen wir nun an die **MitarbeiterID**-Werte der markierten Datensätze heran? Hier nutzen wir eine Kombination aus den zuvor ermittelten Koordinaten für die Markierung und dem **RecordsetClone**-Objekt.

Mit dem **RecordsetClone**-Objekt eines Recordsets, in diesem Fall des zu untersuchenden Formulars, erstellen wir einen Klon des Recordsets. Es ist kein Verweis auf das bestehende Recordset, sondern eine eigene Kopie.

In dieser können wir verschiedene Dinge tun, zum Beispiel darin navigieren. Die Idee ist, dass wir das **RecordsetClone**-Objekt nutzen, um darin zunächst die Position der ersten Zeile der Markierung einzunehmen und dann durch die Anzahl der markierten Zeilen zu navigieren und die jeweiligen Primärschlüsselwerte zu ermitteln.

Der Code sieht wie in Listing 1 aus.

In der Prozedur deklarieren wir neben dem Formular-Objekt **sfm** (für das Unterformular) ein **Recordset**-Objekt für das **RecordsetClone**-Objekt sowie eine Laufvariable namens **i** und zwei Variablen zum Speichern der relevanten Markierungskoordinaten.

Schließlich benötigen wir noch ein **Variant**-Array namens **arrBookmarks**, in dem wir Bookmarks der betroffenen Datensätze des **RecordsetClone**-Methode speichern.

Die Prozedur referenziert das zu untersuchende Formular in der Datenblattansicht mit der Variablen **sfm**. Dann weist sie **rstClone** das **RecordsetClone**-Objekt des Formulars zu.

Die folgenden Anweisungen speichern den Index der ersten Zeile und die Höhe des selektierten Bereichs in den Variablen **lngSelTop** und **lngSelHeight**.

Wenn **lngSelHeight** größer ist als **0**, was Voraussetzung für das Auslesen von Markierungen ist, dimensionieren wir das Array für die Bookmarks von **1** bis zur Anzahl der Zeilen aus **lngSelHeight**.

Im **RecordsetClone** springen wir auf den ersten Datensatz und bewegen uns in einer ersten Schleife solange weiter, bis wir zum ersten markierten Datensatz gelangen.

In einer zweiten Schleife springen wir mit **MoveNext** solange jeweils einen Datensatz weiter, bis wir die Anzahl

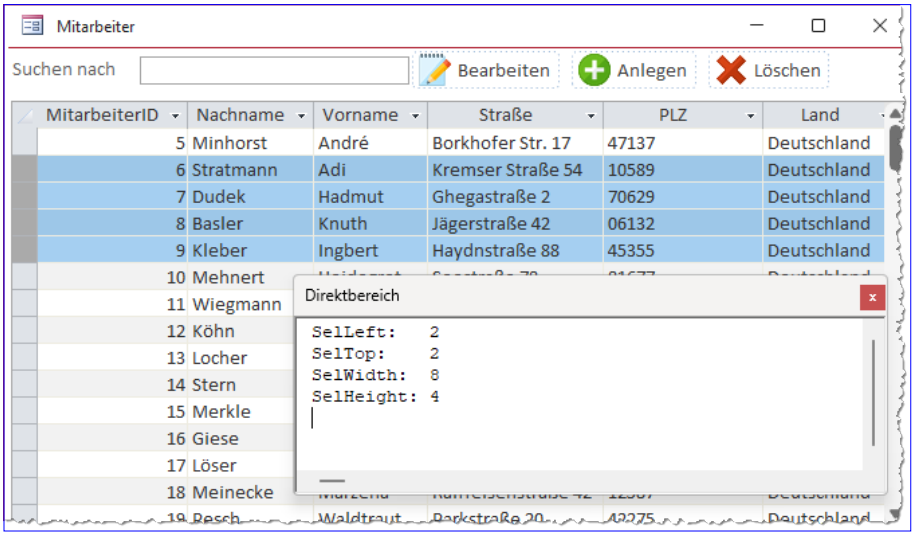

Bild 4: Ausgabe der Selektion in Zahlen

aus **lngSelHeight**, also die Höhe der Markierung, erreicht haben.

Dies geschieht normalerweise erfolgreich in einer einfachen **For...Next**-Schleife von **1** bis zum Wert **lngSel-**

```
Public Sub DatenDerMarkierungErmitteln()
    Dim sfm As Form
    Dim rstClone As DAO.Recordset
    Dim i As Long
    Dim lngSelTop As Long
    Dim lngSelHeight As Long
    Dim lngCount As Long
    Dim arrBookmarks() As Variant
    Set sfm = Forms!frmMitarbeiterUebersicht!sfmMitarbeiterUebersicht.Form
    Set rstClone = sfm.RecordsetClone

    lngSelTop = sfm.SelTop
    lngSelHeight = sfm.SelHeight

    If lngSelHeight > 0 Then
        ReDim arrBookmarks(1 To lngSelHeight)

        rstClone.MoveFirst

        For i = 1 To lngSelTop - 1
            rstClone.MoveNext
        Next i

        lngCount = lngSelHeight

        For i = 1 To lngSelHeight
            If rstClone.AbsolutePosition = -1 Then
                lngCount = lngSelHeight - 1
                Exit For
            End If
            arrBookmarks(i) = rstClone.Bookmark
            rstClone.MoveNext
        Next i

        For i = 1 To lngCount
            rstClone.Bookmark = arrBookmarks(i)
            Debug.Print "ID: " & rstClone!MitarbeiterID
        Next i
    Else
        MsgBox "Bitte mindestens einen Datensatz markieren."
    End If
End Sub
```

Listing 1: Anpassung der Prozedur für das Vergrößern und Verkleinern

Spaltenbreiten und -position in ACCDE speichern

Wenn Sie eine Datenbank an die Benutzer weitergeben, verwenden Sie je nach Zielgruppe eine .accde-Datenbank, also eine Datenbank, in welcher der Entwurf von Tabellen, Abfragen, Formularen, Berichten und VBA nicht mehr möglich ist. Damit schützen wir auch den wertvollen VBA-Code vor den Blicken anderer Menschen. Es bringt aber auch mit sich, dass Änderungen an Eigenschaften wie Spaltenbreiten, Spaltenreihenfolge oder die Sichtbarkeit der Spalten nicht gespeichert werden können, da es sich dabei um Änderungen am Design des Formulars in der Datenblattansicht handelt. Gerade wenn Benutzer die Datenblätter nach ihren eigenen Vorlieben umgestalten und dies bei jedem Öffnen eines Formular erneut erledigen müssen, macht sich schnell Frustration breit. Daher zeigen wir in diesem Beitrag, wie Sie eine Funktion zum Speichern und Wiederherstellen der Spalteneigenschaften von Formularen in der Datenblattansicht programmieren können.

Datenblattansichten gibt es in den meisten Datenbanken – wie hier zum Beispiel zur Anzeige der Daten einer Mitarbeitertabelle (siehe Bild 1).

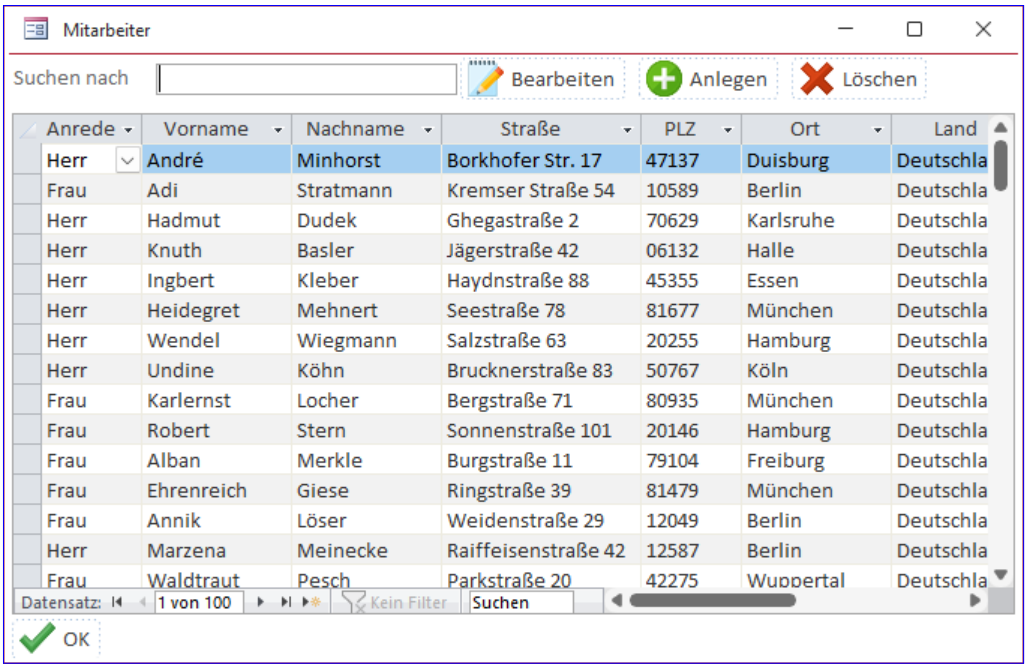

Bild 1: Beispiel für eine Datenblattansicht

Sie bieten viele Vorteile, wenn wir die Daten einer Tabelle oder Abfrage als Liste anzeigen wollen.

Dazu gehört die Möglichkeit, die Spaltenbreiten anzupassen, die Spalten nach unseren Wünschen zu arrangieren und Spalten ein- oder auszublenden.

Mögliche Anpassungen der Datenblattansicht

Wir können also folgende Änderungen an der Datenblattansicht durchführen:

- Einstellen der Spaltenbreite durch Ziehen des rechten Randes des Spaltenkopfes
- Ändern der Reihenfolge der Spalten, indem wir die zu verschiebende Spalte am Spaltenkopf mit der Maus anfassen und an die gewünschte Position ziehen

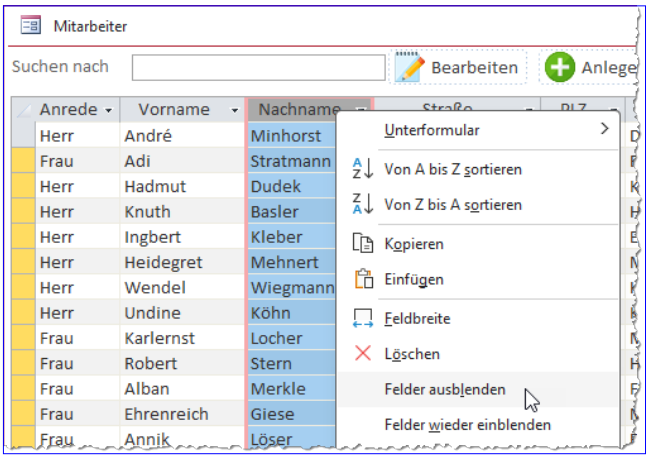

Bild 2: Ausblenden einer Spalte

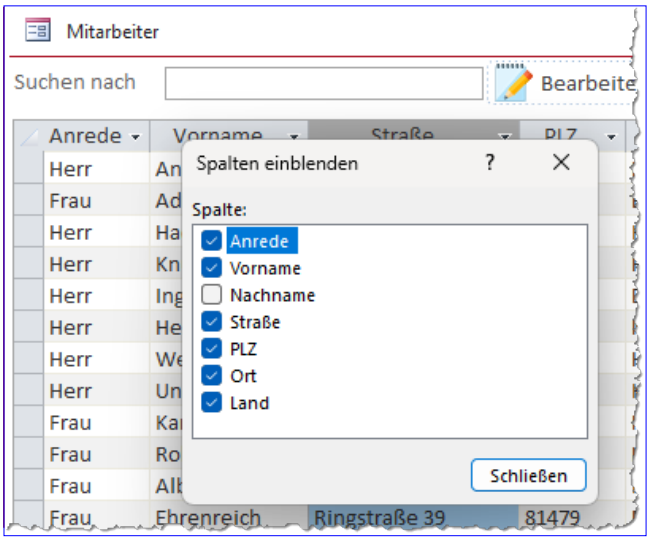

Bild 3: Erneutes Einblenden von Spalten

- Ausblenden der aktuellen Spalte per Rechtsklick und Auswahl des Befehls **Felder ausblenden** (siehe Bild 2). Hier können wir auch mehrere zusammenhängende Felder gleichzeitig ausblenden.

- Wiedereinblenden von Spalten durch Betätigen des Kontextmenübefehls **Felder wieder einblenden** und Auswahl der einzublendenden Felder im Dialog **Spalten einblenden** (siehe Bild 3).

- Fixieren von Feldern durch Auswahl eines oder mehrerer Felder und anschließendes Betätigen des Kontextmenübefehls **Felder fixieren** des Spaltenkopfes, wodurch alle Felder links des rechten markierten Feldes inklusive dieses Feldes fixiert werden. Scrollen wir nun nach rechts, bleiben die fixierten Felder immer sichtbar.

- Aufhebung der Fixierung aller Felder durch den Kontextmenübefehl **Fixierung aller Felder aufheben**

Verhalten in der .accdb-Version der Datenbank

Wenn wir in der **.accdb**-Version der Datenbank arbeiten und eine oder mehrere dieser Änderungen vorgenommen haben, können wir das Formular schließen und wieder öffnen und finden alle Einstellungen vor, wie sie zuletzt eingestellt waren. Wenn wir eine solche Änderung vornehmen, stellen wir sogar fest, dass diese Änderungen noch nicht einmal die Rückfrage auslöst, ob Änderungen an diesem Formular gespeichert werden sollen. Selbst nach dem Schließen und erneutem Öffnen der Datenbank finden wir die letzten Einstellungen erneut vor.

Speichern der Datenbank als .accde-Datenbank

Nun wollen wir die Datenbank als **.accde**-Datenbank speichern und somit in einem Format, das keine Änderungen mehr an dem Entwurf der Objekte zulassen sollte.

Dazu klicken wir in Access auf den Reiter **Datei** und dann auf **Speichern unter**. Damit blenden wir den Bereich **Speichern unter** ein, wo wir im rechten Bereich den Befehl **ACCDE erstellen** vorfinden (siehe Bild 4).

Klicken wir diesen doppelt an, erscheint ein Dialog, mit dem wir den Namen für die zu erstellende Datei angeben können. Access schlägt hierzu den gleichen Dateinamen vor, allerdings mit der Dateiendung **.accde**. Diesen Dateinamen behalten wir bei und speichern die neue Datei.

Experimente mit der .accde-Version

Nun öffnen wir die Datenbank in der **.accde**-Version. Das Formular **frmMitarbeiterUebersicht** erscheint genauso wie in der **.accdb**-Version. Dass es sich überhaupt um

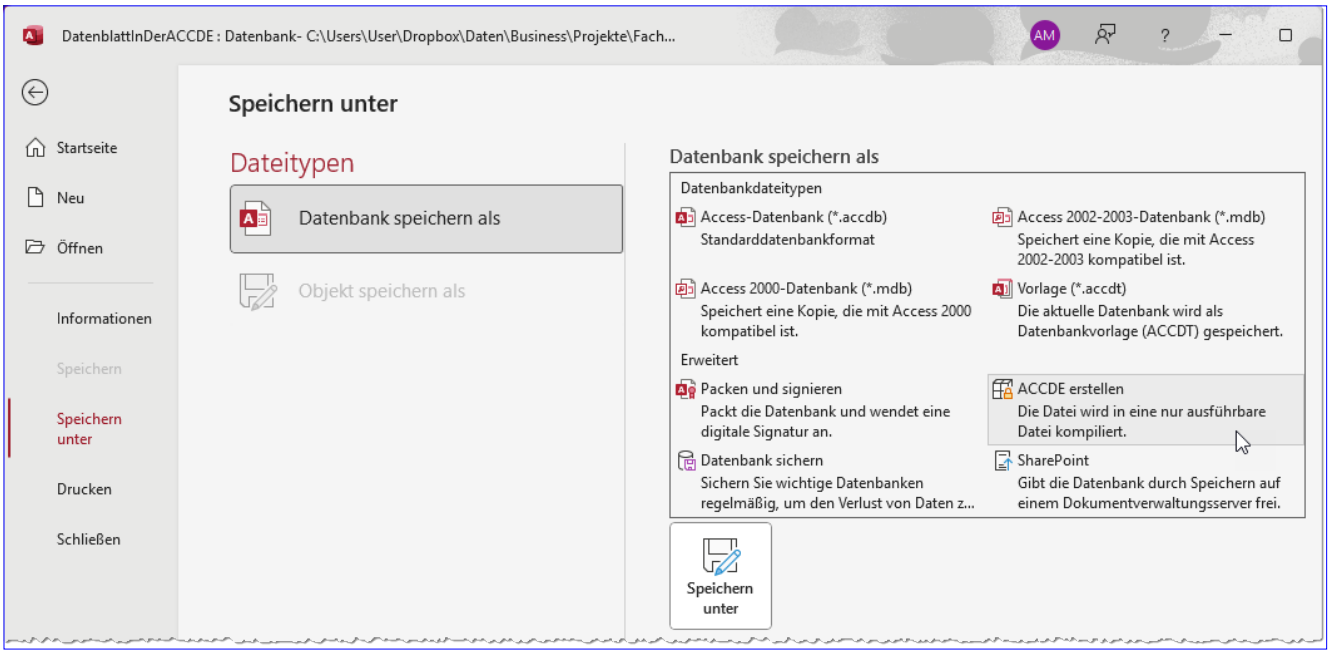

Bild 4: Speichern der Datenbank als **.accde**-Datenbank

die **.accde**-Version handelt, können wir leicht feststellen, indem wir versuchen, die Entwurfsansicht des Formulars anzuzeigen. Dies gelingt nicht, weil die entsprechende Schaltfläche im Formular gar nicht verfügbar ist (siehe Bild 5).

Nun nehmen wir einige Änderungen vor – wir ändern die Reihenfolge der Spalten, passen ihre Breite an, blenden eine Spalte aus und fixieren eine am linken Rand.

Das gelingt alles ohne Probleme. Wenn wir das Formular jedoch schließen und erneut öffnen, finden wir genau die gleiche Ansicht vor, wie wir sie in der Originaldatenbank im **.accdb**-Format vor dem Erstellen der **.accde**-Datenbank eingestellt hatten.

Spalteneinstellungen manuell speichern und wiederherstellen

Was nun? Zum Glück können wir die Eigenschaften der Spalten der Datenblattansicht per VBA auslesen.

Das bedeutet, dass wir sie auch wiederherstellen können. Alles, was wir benötigen, ist eine Tabelle, in der wir die

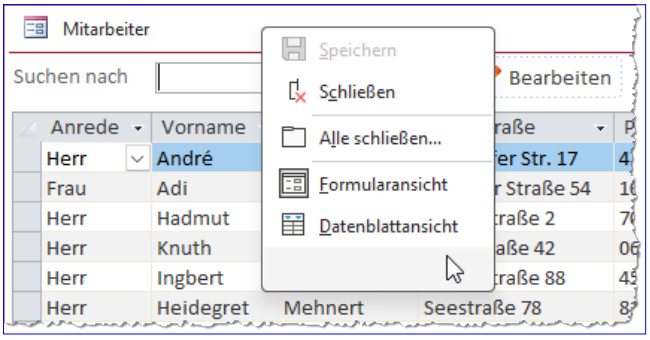

Bild 5: In der **.accde**-Version kann die Entwurfsansicht nicht mehr geöffnet werden.

Einstellungen speichern können, während das Formular oder die Anwendung geschlossen werden, und aus der wir die Einstellungen wieder auslesen und wiederherstellen können.

Ermitteln der Spalteneinstellungen des Formulars

Wir starten mit dem Auslesen der Einstellungen der Spalten des Formulars. Dazu öffnen wir zunächst wieder die **.accdb**-Version der Datenbank – in der **.accde**-Datenbank können wir schließlich erst gar nicht auf den VBA-Code zugreifen.

Spalteneigenschaften unter VBA

Für das Ermitteln der Positionen, Sichtbarkeit und Breite sowie der fixierten Spalten können wir einige VBA-Eigenschaften verwenden.

Fixierte Spalten per VBA ermitteln

Zunächst einmal verwenden wir eine Eigenschaft des Formulars in der Datenblattansicht namens **FrozenColumns**. Diese liefert uns Informationen darüber, wie viele Spalten fixiert sind. Dabei ist der zurückgegebene Wert etwas irreführend:

- **1**: keine Spalte ist fixiert
- **2**: es ist eine Spalte fixiert
- **3**: es sind zwei Spalten fixiert
- und so weiter

Beispiel für das Hauptformular **frmMitarbeiterUebersicht** mit dem Unterformular **sfmMitarbeiterUebersicht**:

```
Debug.Print Forms!frmMitarbeiterUebersicht!
        sfmMitarbeiterUebersicht.Form.FrozenColumns
```

Reihenfolge der Spalten per VBA ermitteln

Zum Ermitteln der Reihenfolge können wir die Eigenschaft **ColumnOrder** der einzelnen Steuerelemente nutzen. Diese liefert uns für jede Spalte den Index, den die aktuelle Spalte in der Reihenfolge hat. Die linke Spalte liefert den Wert **1**, die zweite von links den Wert **2** und so weiter.

Beispiel:

```
Debug.Print Forms!frmMitarbeiterUebersicht!
        sfmMitarbeiterUebersicht!cboAnredeID.ColumnOrder
```

Spaltenbreiten ermitteln

Die Eigenschaft zum Ermitteln der Spaltenbreite lautet **ColumnWidth**. Sie liefert die Breite in Twips:

```
Debug.Print Forms!frmMitarbeiterUebersicht!
        sfmMitarbeiterUebersicht!cboAnredeID.ColumnWidth
```

Sichtbarkeit einer Spalte ermitteln

Schließlich fehlt noch die Sichtbarkeit. Diese ermitteln wir im Gegensatz zu Steuerelementen in Formularen in der Formularansicht nicht mit der **Visible**-Eigenschaft, sondern mit der Eigenschaft **ColumnHidden**. Sie liefert den Wert **False**, wenn das Steuerelement sichtbar ist, und **True**, falls es ausgeblendet ist.

Beispiel:

```
Debug.Print Forms!frmMitarbeiterUebersicht!
        sfmMitarbeiterUebersicht!cboAnredeID.ColumnHidden
```

Keine Steuerelementeigenschaften

Die Eigenschaften **ColumnOrder**, **ColumnWidth** und **ColumnHidden** sind keine allgemeinen Eigenschaften von Steuerelementen, werden also von IntelliSense nicht angezeigt, wenn wir über den Typ **Control** darauf zugreifen wollen.

Wir finden die Eigenschaften erst für explizit angegebene Steuerelementtypen, die wir beispielsweise im Objektkatalog sehen (siehe Bild 6).

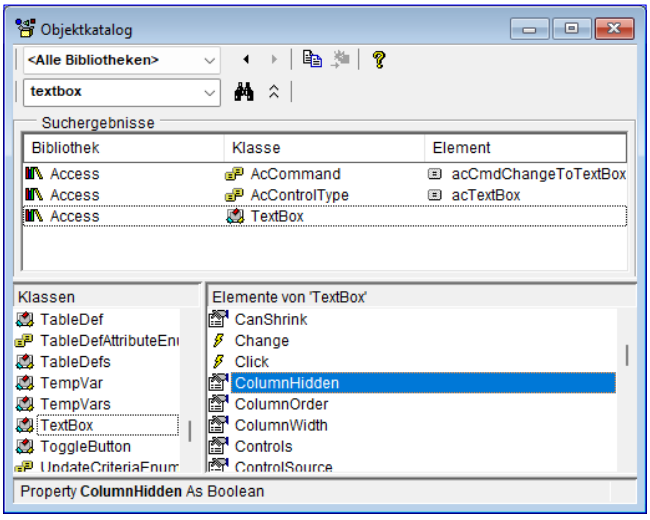

Bild 6: Die Spalteneigenschaften im Objektkatalog

Wir können diese aber auch über den Datentyp **Control** nutzen, vorausgesetzt das referenzierte Steuerelement verfügt über die entsprechende Eigenschaft. Bei Beschriftungsfeldern ist das beispielsweise nicht der Fall.

Durchlaufen der Spalten und Ermitteln der Eigenschaften

In der Prozedur aus Listing 1 geben wir zunächst einmal alle Eigenschaften der gebundenen Steuerelemente des Unterformulars **sfmMitarbeiterUebersicht** aus.

Dabei referenzieren wir mit der Variablen **frm** das Hauptformular **frmMitarbeiterUebersicht** und mit **sfm** das im Unterformularsteuerelement **sfmMitarbeiterUebersicht** enthaltene Formular.

Bevor wir die Steuerelemente durchlaufen, geben wir die Eigenschaft **FrozenColumns** des Unterformulars aus. Dann durchlaufen wir alle Steuerelemente der Auflistung **Controls** des Unterformulars und referenzieren diese jeweils mit der Variablen **ctl** mit dem Datentyp **Control**.

In der Schleife prüfen wir, ob es sich bei dem aktuellen Steuerelement um ein gebundenes Steuerelement handelt.

Das ist nötig, weil dieses ja auch Bezeichnungsfelder enthält. Dazu versuchen wir bei deaktivierter Fehlerbehandlung, den Wert der Eigenschaft **ControlSource** (**Steuerelementinhalt**) auszulesen und speichern diesen in der Variablen **strControlSource**.

Ist **strControlSource** dann nicht leer, handelt es sich um ein gebundenes Steuerelement. Dann geben wir den Namen des Steuerelements und die Werte der Eigenschaften **ColumnHidden**, **ColumnWidth** und **ColumnOrder** im Direktbereich des VBA-Editors aus. Das Ergebnis für ein Datenblatt mit einigen Anpassungen sieht wie in Bild 7 aus.

Felder fixieren ändert gegebenenfalls Reihenfolge

Es fehlt noch ein Hinweis darauf, dass wir mit dem Fixieren von Spalten eventuell die Reihenfolge der Spalten

```
Public Sub DatenblattAuslesen()
    Dim frm As Form
    Dim sfm As Form
    Dim ctl As Control
    Dim strControlSource As String
    Set frm = Forms!frmMitarbeiterUebersicht
    Set sfm = frm!sfmMitarbeiterUebersicht.Form
    Debug.Print "Fixierte Spalte: " & sfm.FrozenColumns
    For Each ctl In sfm.Controls
        strControlSource = ""
        On Error Resume Next
        strControlSource = ctl.ControlSource
        On Error GoTo 0
        If Not Len(strControlSource) = 0 Then
            Debug.Print strControlSource, ctl.ColumnHidden, ctl.ColumnWidth, ctl.ColumnOrder
        End If
    Next ctl
End Sub
```

Listing 1: Auslesen der Spalteneigenschaften für die Steuerelemente unseres Beispielformulars

Access-Formulare per Tastenkombination wechseln

In modernen Browsern wie Chrome oder Edge ist es längst selbstverständlich: Mit Tastenkombinationen wie Alt + 1, Alt + 2, Alt + 3 kann man blitzschnell zwischen geöffneten Tabs springen. Hat man diesen Shortcut einmal verinnerlicht, will man nicht mehr ohne arbeiten. Die Tastenkombinationen sind viel schneller, als wenn man mit der Maus auf den jeweiligen Reiter klickt. Man könnte zwar auch per Strg + Tab zwischen den Seiten wechseln, aber warum nicht direkt zur gesuchten Information springen? Da wir auch in Access standardmäßig eine Ansicht vorfinden, die über entsprechende Reiter angesteuert werden kann, stellt sich die Frage: Können wir nicht auch hier solche Tastenkombinationen nutzen? Genau das untersuchen wir in diesem Beitrag.

Alt + 1, Alt + 2, Alt + 3 für Formulare in Registerkartenansicht

Access zeigt bei neu angelegten Datenbanken zunächst alle neu erstellten oder geöffneten Elemente als Tab an, der den vollständigen Arbeitsbereich des Access-Fensters einnimmt. Dem einen mag das gefallen, andere bevorzugen die Anzeige als überlappende Fenster. Bevor wir in unsere Lösung einsteigen, schauen wir uns kurz an, wie wir standardmäßig verwendete Einstellungen ändern können.

Dazu benötigen wir den Dialog mit den Access-Optionen, den wir über **Datei|Optionen** öffnen.

Hier wechseln wir zum Bereich **Aktuelle Datenbank** und finden dort die Option **Dokumentfensteroptionen** vor (siehe Bild 1).

Der Bereich, in dem sich diese Option befindet, deutet bereits darauf hin, dass es sich um eine Einstellung handelt, die individuell je Datenbank vorgenommen werden kann.

Die aktuelle Standardeinstellung lautet **Dokumente im Registerkartenformat** und zusätzlich ist die Option **Dokumentregisterkarten anzeigen** aktiviert.

Dies führt zu der Anzeige, auf die sich dieser Beitrag bezieht (siehe Bild 2). Wenn wir hier die Option **Dokumentregisterkarten anzeigen** deaktivieren, blendet Access die Registerreiter aus und wir sehen nur das aktuell aktive Formular.

Sollten wir jedoch die Option **Dokumentfensteroptionen** auf **Überlappende Fenster** einstellen, sehen wir erst-

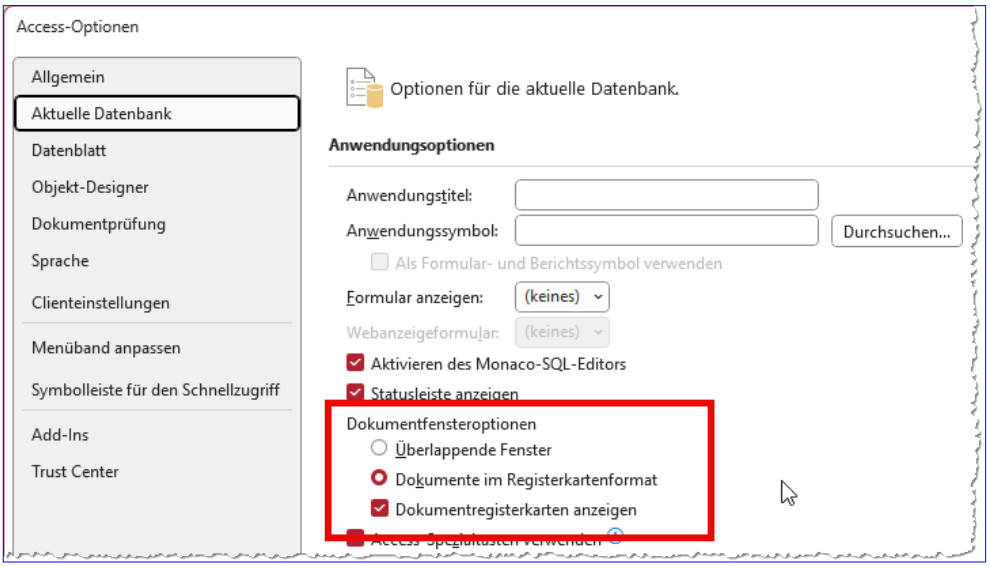

Bild 1: Optionen zum Einstellen der Anzeige von Fenstern innerhalb von Access

ens keine Registerreiter für die Fenster.

Wir können die geöffneten Fenster zwar nach wie vor maximieren, aber auch als einzelne Fenster innerhalb des Arbeitsbereichs von Access positionieren. Anwendungen, die das gleichzeitige Anzeigen mehrerer Formulare erfordern, sind mit dieser Einstellung am besten bedient.

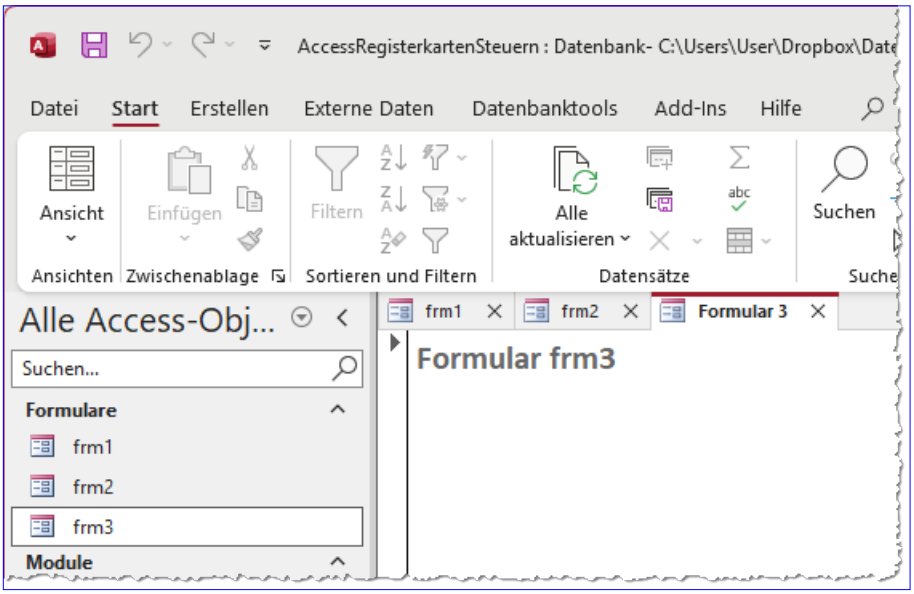

Bild 2: Registerkarten einer Access-Anwendung

Wechseln zwischen den Registerkarten

Wir wollen uns jedoch auf die Einstellung im Registerkartenformat mit Registerkartenreitern konzentrieren. Die üblicherweise verwendeten Methoden zum Wechseln des aktuell im Vordergrund befindlichen Fensters sind die Folgenden:

- Anklicken des gewünschten Registerreiters mit der Maus
- Betätigen der Tastenkombination **Strg + Tab**, um vom aktiven zum nächsten Register zu wechseln. Man kann übrigens nicht mit **Umschalt + Strg + Tab** in die entgegengesetzte Richtung wechseln, wie es bei anderen Anwendungen möglich ist. Diese Funktion können wir aber auch noch nachreichen.

Es gibt keine andere eingebaute Methode, Tastenkombinationen wie **Strg + 1** et cetera betätigen wir ohne jeden Effekt.

Ziel: Alt + 1 bis Alt + 9 aktivieren

Unser Ziel ist es, **Alt + 1** bis **Alt + 9** so zu programmieren, dass bei Betätigen dieser Tastenkombination das entsprechende Formular aktiviert wird – genau wie im Browser.

Unser erster Ansatz ist, jedem Formular eine Ereignisprozedur hinzuzufügen, die auf das Betätigen der entsprechenden Tastenkombination reagiert und den Fokus auf die entsprechende Registerkarte verschiebt.

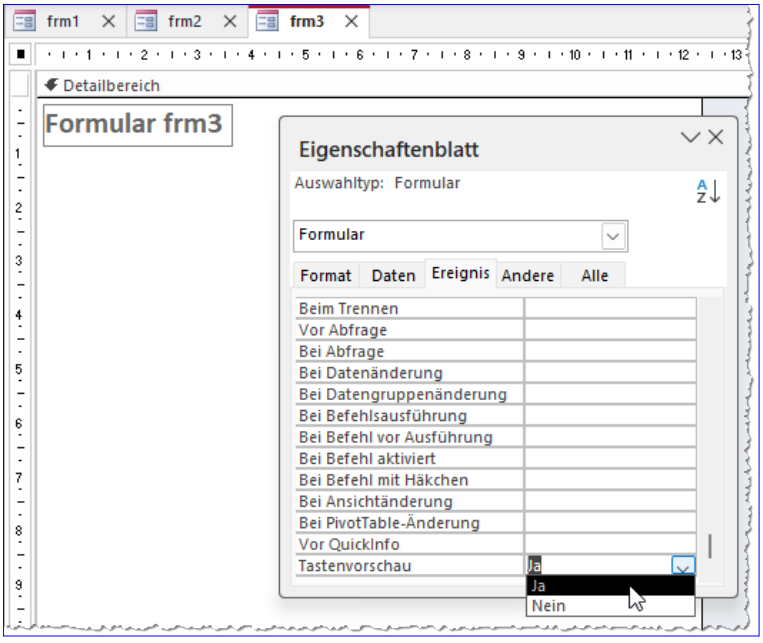

Bild 3: Einstellen der Eigenschaft **Tastenvorschau** auf den Wert **Ja**

Vorbereitung

Damit Access auf Tastatureingaben reagiert, müssen wir die Eigenschaft **Tastenvorschau** aller beteiligten Formulare auf **Ja** einstellen. Das bewirkt, dass alle Tastaturereignisse zuerst an das entsprechende Ereignis des Formulars gesendet werden und nicht etwa an ein Steuerelement, das gerade den Fokus hat.

Diese Einstellung führen wir wie in Bild 3 durch.

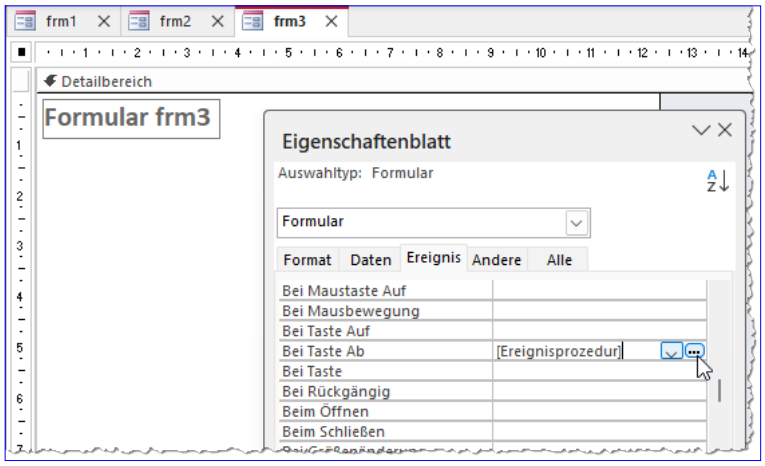

Bild 4: Anlegen der Ereignisprozedur

Wie kann ich die Tabs gezielt aktivieren?

Bevor wir überhaupt Code schreiben, der die aktive Registerseite aktivieren soll, müssen wir uns überlegen, wie man eine Registerseite aktiviert. Wir gehen hier davon aus, dass wir die Technik nur für Formulare einsetzen.

Hinter jedem Tab steckt also ein Formular, was bedeutet, dass wir diese über die **Forms**-Auflistung aktivieren können. Probieren wir das aus, sehen wir:

Forms(0).SetFocus aktiviert das erste Registerblatt, **Forms(1).SetFocus** das zweite und so weiter. Die Formulare werden also entsprechend der Reihenfolge der Registerseiten indiziert, sodass wir über den 0-basierten Index auf diese zugreifen können.

Anlegen der Ereignisprozedur

Nun wollen wir dafür sorgen, dass beim Betätigen einer der Tastenkombinationen **Alt + 1** bis **Alt + 9** die entsprechende Registerseite aktiviert wird.

Dazu hinterlegen wir zuerst für das Formular **frm1** eine Ereignisprozedur, die durch das Ereignis **Bei Taste ab** des Formulars ausgelöst wird.

Dazu wählen wir für die entsprechende Eigenschaft den Eintrag **[Ereignisprozedur]** aus und klicken auf die Schaltfläche mit den drei Punkten (siehe Bild 4).

Dies zeigt die automatisch angelegte, noch leere Ereignisprozedur im VBA-Editor an, die wir wie in Bild 5 ergänzen.

Der Code erledigt im Einzelnen die folgenden Schritte:

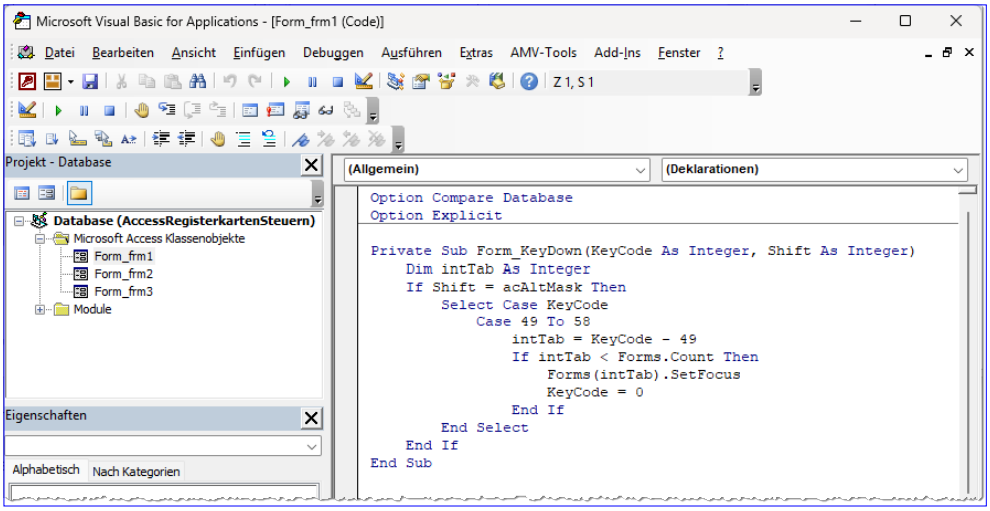

Bild 5: Die fertige Ereignisprozedur

Platzhalter für Textfelder in Formularen

Textfelder kommen standardmäßig mit Bezeichnungsfeldern. Das reicht eigentlich aus, damit der Benutzer weiß, welche Informationen er in die Textfelder eintragen soll. Moderne Benutzeroberflächen kommen aber teilweise vollständig ohne diese Bezeichnungsfelder aus und zeigen einfach solange, bis der Benutzer etwas eingibt, einen Platzhalter im Textfeld an, der den einzufügenden Wert beschreibt. Das liegt möglicherweise am geringeren Platzbedarf, was gerade auf Smartphones ein Vorteil ist. Es gibt aber noch andere Gründe, warum man Textfelder mit Platzhaltertexten ausstatten sollte. Welche das sind und wie die Anzeige von Platzhaltern realisiert werden kann, zeigen wir in diesem Beitrag.

Platzhaltertexte in Access-Formularen: Mehr als nur ein Design-Trick

Platzhaltertexte in Textfeldern sind in modernen Webanwendungen längst Standard – und auch in Microsoft Access können sie eine echte Bereicherung sein.

Ob zur Platzersparnis, besseren Benutzerführung oder als Alternative zu klassischen Beschriftungen: Auf den folgenden Seiten zeigen wir die vielseitigen Einsatzmöglichkeiten und geben praktische Beispiele für den Einsatz von Platzhaltern in Access-Formularen. Wie das aussehen wird, sehen Sie in Bild 1.

Hier sind unsere Top-Gründe für den Einsatz dieser Platzhalter:

- **Ersatz für klassische Beschriftungsfelder**: Platzhaltertexte erlauben es, auf separate Beschriftungsfelder zu verzichten. Dadurch wird das Formular kompakter und aufgeräumter. Besonders bei vielen Eingabefeldern spart das nicht nur Platz, sondern erleichtert auch die visuelle Orientierung.

- **Zusätzliche Hinweise zur Dateneingabe**: Platzhalter können Hinweise zur erwarteten Eingabe geben, ohne dass dafür zusätzliche Tooltips oder Hilfetexte nötig sind. Beispiele: **JJJJ-MM-TT** bei Datumsfeldern, **Privat, Dienstlich** bei einer Kategorieauswahl oder **Nur Zahlen erlaubt** bei numerischen Feldern.

- **Verdeutlichung bei leeren Feldern**: Ein leerer Wert wird durch einen Platzhalter besser verständlich. Nutzer sehen sofort, dass ein Feld leer, aber beabsichtigt leer ist – etwa bei optionalen Angaben. Beispiel: **Optional: Bemerkung für Rechnung**

- **Unterstützung beim Onboarding**: Gerade neue Benutzer profitieren von klaren Hinweisen in der Eingabemaske. Platzhalter übernehmen hier eine Art Schulungsfunktion – ohne separate Hilfe anzeigen zu müssen.

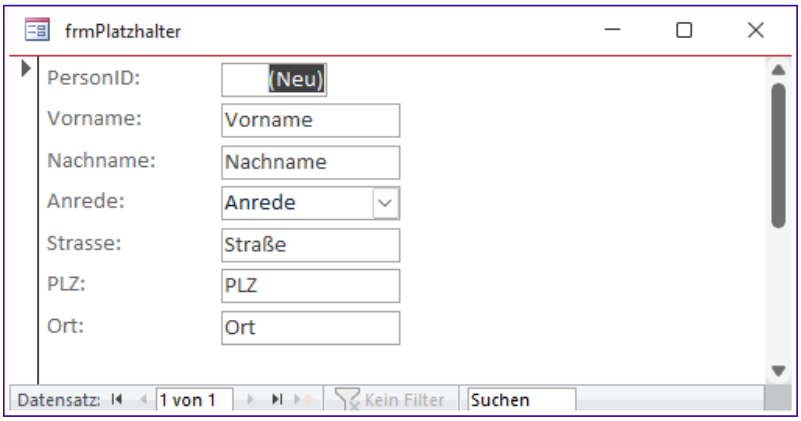

Bild 1: Dies sind keine Werte, sondern Platzhalter

- **Einheitliches Design ohne Label-Fummelei**: Wer Access-Formulare im einheitlichen Stil gestalten will, wird Platzhalter schnell zu schätzen wissen. Labels mit unterschiedlicher Länge, Position oder Ausrichtung gehören der Vergangenheit an.

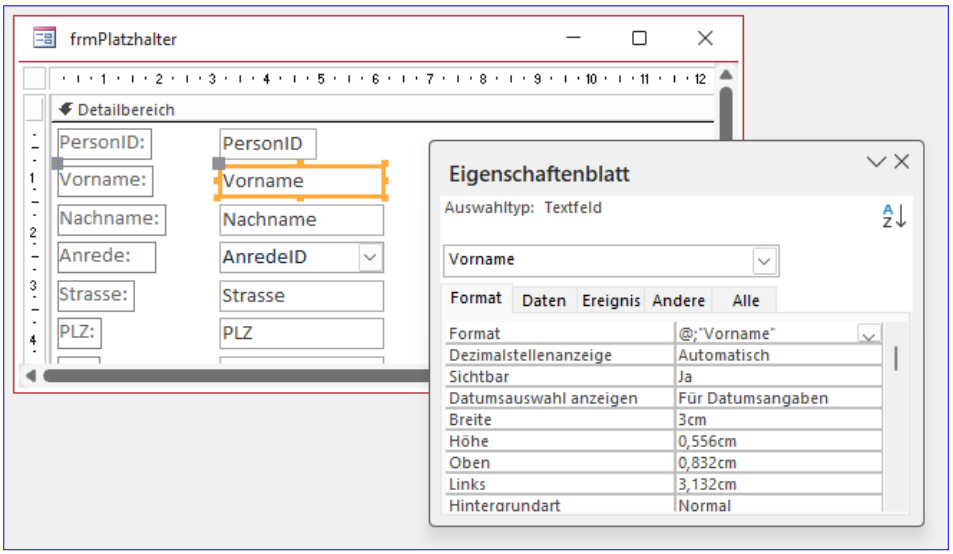

Bild 2: Einstellen des Formats für die Anzeige eines Platzhalter-Textes

- **Ersatz für Standardwerte**: Der Platzhalter ist wie ein Platzhalter kein gespeicherter Wert. Damit lassen sich scheinbare Standardwerte anzeigen, ohne sie wirklich zu übernehmen. Hier muss man allerdings unterscheiden, ob man nicht doch einen Standardwert nutzen möchte, da ein Platzhalter im Gegensatz zu einem Standardwert beim Bearbeiten des Datensatzes nicht automatisch übernommen und später gespeichert wird.

Einfache Umsetzung per Format-Eigenschaft

Die Umsetzung ist sehr einfach. Wir müssen lediglich die **Format**-Eigenschaft des jeweiligen Textfeldes anpassen.

Dazu wechseln wir in die Entwurfsansicht des entsprechenden Formulars und aktivieren das Eigenschaftenblatt, sofern es nicht bereits angezeigt wird. Hier finden wir auf der Registerseite **Format** den folgenden Wert:

```
@;"Vorname"
```

Das bedeutet schlicht und einfach: Wenn der angezeigte Wert **Null** ist, zeige den Text aus dem zweiten Parameter an (siehe Bild 2). Das **@**-Zeichen ist ein Platzhalter für eine Zeichenkette. Sobald wir diesen Ausdruck für die Eigenschaft **Format** eingetragen haben und wieder zur Formularansicht wechseln, erscheint dieser Wert in dem Textfeld zur Eingabe des Vornamens – aber nur, wenn dieses noch leer ist.

Klicken wir dieses Feld an und fügen so die Einfügemarke dort ein, verschwindet der Platzhalter, sodass wir direkt sehen, dass hier etwas eingegeben werden muss (siehe Bild 3).

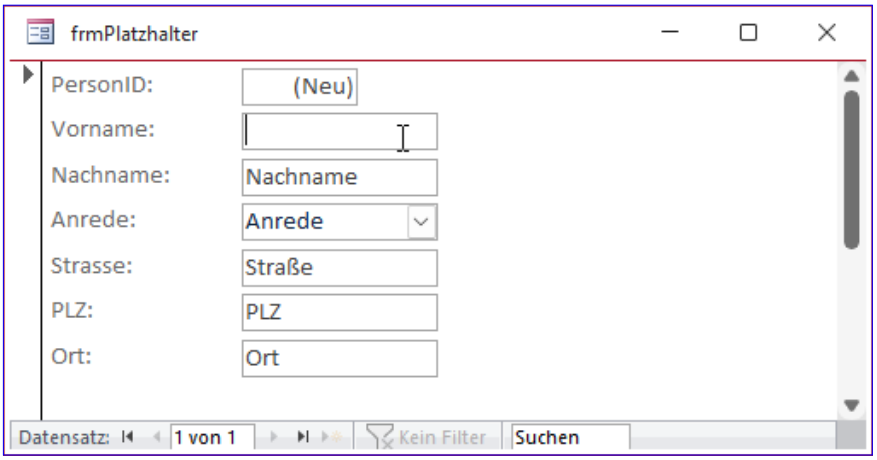

Bild 3: Der Platzhalter verschwindet beim Anklicken des Textfeldes.

Wenn wir auf andere Weise zwischen diesen Steuerelementen navigieren, erhalten wir allerdings ein anderes Verhalten, das von der Einstellung der Option **Cursorverhalten bei Eintritt in Feld abhängt**. Diese stellen wir ein, indem wir mit **DateiOptionen** den **Optionen**-Dialog öffnen, dort zum Bereich **Clienteinstellungen** wechseln und hier die entsprechende Option auf den gewünschten Wert setzen (siehe Bild 4).

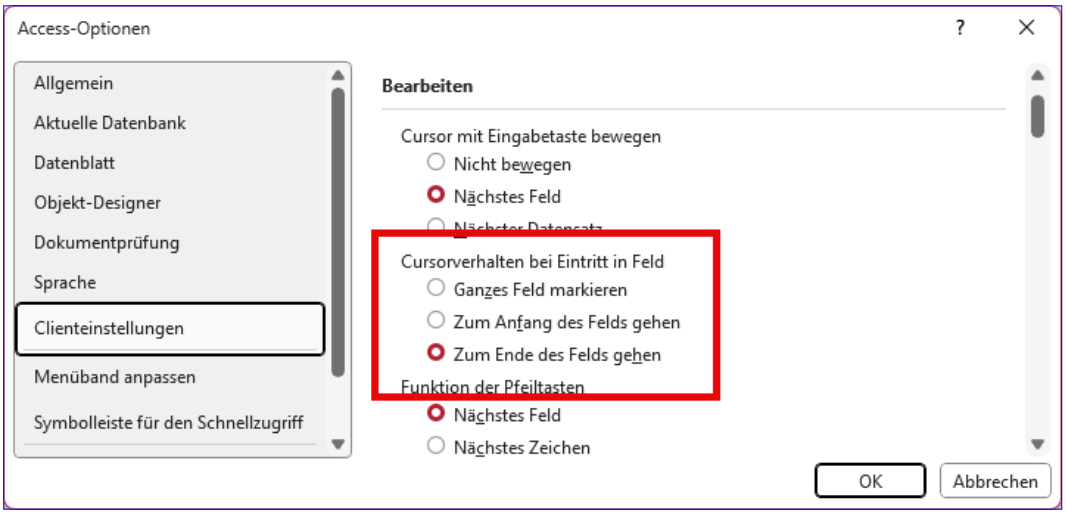

Bild 4: Einstellungen für das Verhalten bei Eintreten in ein Feld

Bei **Ganzes Feld markieren** wird das ganze Feld markiert, aber der Platzhaltertext verschwindet nicht. Bei den beiden anderen Einstellungen wird der Platzhaltertext beim Eintritt in das Feld automatisch ausgeblendet.

Platzhaltertexte einfach angeben

Die Eingabe der Platzhaltertexte ist zwar einfach, aber wenn man dies für viele Steuerelemente machen möchte, ist die Arbeit ungefähr genauso aufwendig wie das Erstellen entsprechender Textfelder.

Also versuchen wir, das Ganze etwas zu vereinfachen. Zum Beispiel so, dass wir nicht alle Steuerelemente einzeln mit den entsprechenden Platzhaltern versehen müssen. Dies können wir nämlich auch per VBA erledigen.

Dazu benötigen wir lediglich eine Ereignisprozedur, die beim Öffnen des Formulars ausgeführt wird. Diese hinterlegen wir in diesem Fall für das Ereignis **Beim Laden** des Formulars:

```
Private Sub Form_Load()
    Me.txtVorname.Format = "@,""Vorname"""
    Me.txtNachname.Format = "@,""Nachname"""
    Me.txtStrasse.Format = "@,""Straße"""
    Me.txtPLZ.Format = "@,""PLZ"""
    Me.txtOrt.Format = "@,""Ort"""
End Sub
```

Wir weisen also jeweils der Eigenschaft **Format** einen Ausdruck zu, der im Wesentlichen dem entspricht, den wir auch in das Eigenschaftenblatt eingetragen haben.

Mit dem Unterschied, dass wir diesen in Anführungszeichen einfassen müssen – und den Text, der bereits in Anführungszeichen steht, mit doppelten Anführungszeichen versehen:

```
Me.txtVorname.Format = "@,""Vorname"""
```

Wenn wir das Formular nun in der Formularansicht öffnen, erhalten wir genau die gleiche Ansicht wie zuvor.

Code per Code erstellen

Nun ist auch noch die Aufgabe, für jedes Textfeld eine solche Zeile zu schreiben, ein gewisser Aufwand.

Auch diesen können wir noch angenehmer machen, indem wir eine kleine Hilfsprozedur verwenden. Diese sieht wie in Listing 1 aus.

```
Public Sub Platzhaltersteuerelemente()
    Dim frm As Form
    Dim ctl As Control
    Dim strSteuerelementinhalt As String
    Set frm = Screen.ActiveForm
    For Each ctl In frm.Controls
        strSteuerelementinhalt = ""
        On Error Resume Next
        strSteuerelementinhalt = ctl.ControlSource
        On Error GoTo 0
        If Not Len(strSteuerelementinhalt) = 0 Then
            Select Case ctl.ControlType
                Case acTextBox
                    Debug.Print "Me." & ctl.Name & ".Format = ""@;""""""" & ctl.ControlSource & """""""""
                Case Else
                    'Debug.Print ctl.ControlType
            End Select
        End If
    Next ctl
End Sub
```

Listing 1: Prozedur zum Erstellen des Codes für die **Format**-Eigenschaft der Steuerelemente

Die Prozedur referenziert das aktuell geöffnete Formular mit der Variablen **frm**. Um den Code für ein bestimmtes Formular zu erstellen, müssen wir dieses also öffnen und es muss sich im Fokus befinden. Danach durchläuft die Prozedur alle Steuerelemente der **Controls**-Auflistung des Formulars in einer **For Each**-Schleife.

Hier prüfen wir zuerst, ob es sich bei dem Steuerelement um ein gebundenes Steuerelement handelt, also eines, das für die Eigenschaft **Steuerelementinhalt** einen Wert enthält. Für diese Prüfung leeren wir zuerst eine **String**-Variable namens **strSteuerelementinhalt**. Dann versuchen wir, die Eigenschaft **ControlSource** bei deaktivierter Fehlerbehandlung auszulesen.

Warum bei deaktivierter Fehlerbehandlung? Weil nicht alle Steuerelemente wie beispielsweise Bezeichnungsfelder diese Eigenschaft aufweisen. Wenn wir dennoch versuchen, für ein solches Steuerelement auf **ControlSource** zuzugreifen, löst dies einen Fehler aus, den wir hier verhindern wollen. Andererseits erhalten wir so genau das gewünschte Verhalten, nämlich dass **strSteuerelementinhalt** nur gefüllt wird, wenn die Eigenschaft **ControlSource** vorhanden ist.

Also können wir danach einfach prüfen, ob **strSteuerelementinhalt** eine Zeichenkette mit einer Länge ungleich **0** enthält. In diesem Fall wollen wir noch den Steuerelementtyp in einer **Select Case**-Bedingung untersuchen, denn wir wollen die Formatierung nur für Textfelder anwenden. Also prüfen wir **ctl.ControlType** auf den Wert **acTextBox**.

Ist auch diese Bedingung erfüllt, können wir die benötigte VBA-Zeile mit der Zuweisung des Ausdrucks für die **Format**-Eigenschaft erstellen. Da wir nicht genau wissen können, welcher Platzhaltertext verwendet werden soll, wählen wir hier erst einmal den Inhalt der Eigenschaft Steuerelementinhalt, also in der Regel den Namen des Feldes aus der zugrunde liegenden Tabelle.

Diese geben wir für alle betroffenen Steuerelemente im Direktbereich aus und erhalten so eine Ausgabe wie die folgende:

Anlage-Feld zum SQL Server migrieren

Anlage-Felder erlauben es seit Access 2010, Bilder und andere Dateien direkt in Access-Tabellen zu speichern. Sie bieten ein praktisches Popup-Menü, mit dem man die enthaltenen Dateien verwalten kann. Enthält ein Anlage-Feld Bilddateien, können diese mit dem Bild-Steuerelement von Access direkt in Formularen und Berichten angezeigt werden. Wenn man seine Tabellen zum SQL Server migrieren möchte, wird dies jedoch zu einem Problem: Im SQL Server gibt es nämlich kein Pendant zum Anlage-Feld, sondern nur Alternativen wie beispielsweise den varbinary(max)-Datentyp. Damit können wir zwar die Dateien im binären Format im SQL Server speichern, aber rundherum ergeben sich einige Fragen: Wie zeige ich Bilddateien an, die in einem solchen Feld gespeichert sind? Wie gehe ich mit Anlage-Feldern um, die nicht nur eine Anlage enthalten? All dies erläutern wir in diesem und weiteren Beiträgen.

Anlage-Felder sind flexibel einsetzbar. Grundsätzlich kann man in ihnen Dateien speichern, und zwar so viele, wie man möchte. Der Grund dafür ist, dass Access im Hintergrund eine eigene Tabelle für die einzelnen Anlagen eines Anlage-Feldes bereithält.

Das dies der Fall ist, werden wir später sehen, wenn wir mit zwei verschachtelten Recordsets auf die einzelnen Datensätze eines Anlage-Feldes zugreifen.

Aber man kann nicht nur einfach Dateien in Anlage-Feldern speichern. Wenn es sich bei den Dateien um Bilddateien handelt, können wir diese sogar beispielsweise im Anlage-Steuerelement oder im Bild-Steuerelement (mit wesentlich mehr Aufwand) anzeigen.

Um die in einem Anlage-Feld enthaltenen Dateien in einem Formular zu verwalten, brauchen wir nur das Anlagefeld in den Tabellenentwurf zu ziehen. Klicken wir dieses dann an, sehen wir ein kleines Popup-Menü mit drei Befehlen, von denen die ersten beiden zum Navigieren zwischen den enthaltenen Dateien und die dritte zum Öffnen des Dialogs zum Verwalten der enthaltenen Dateien verwendet wird (siehe Bild 1).

Wir gehen an dieser Stelle davon aus, dass wir bereits eine Beispieltabelle mit den drei Feldern **AnlageID** (Primärschlüsselfeld), **Bezeichnung** (Textfeld) und **Anlagefeld** (Anlage-Feld) erstellt haben, die wir als Datensatzquelle des Formulars verwenden.

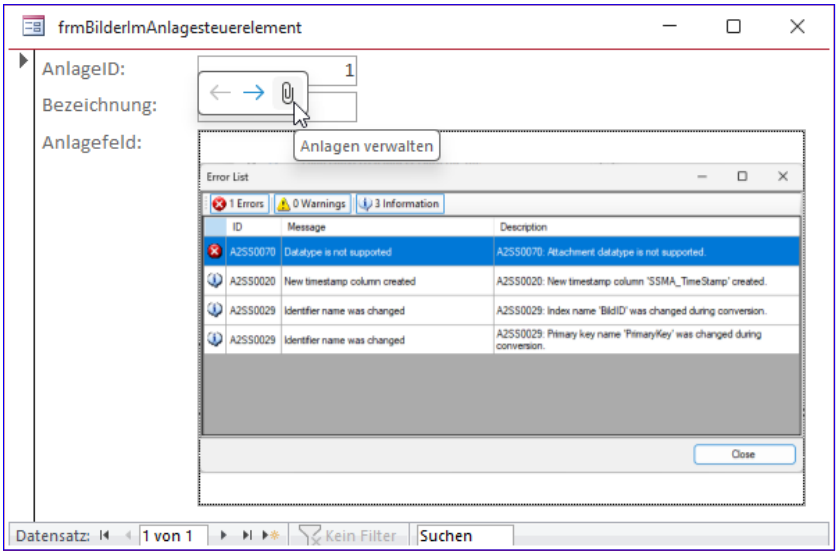

Bild 1: Anzeigen von Bildern aus einem Anlage-Feld

Anlage-Felder im SQL Server?

Zum Problem werden Anlage-Felder, wenn wir sie zum SQL Server migrieren wollen. Im Beitrag **Bilder im SQL Server mit varbinary(max)** (**www.access-im-unternehmen.de/1538**) zeigen wir, wie man Bild-Dateien in einem **varbinary(max)**-Feld des SQL Servers speichert und wie man die Bilder daraus in einem Bild-Steuerelement in Access anzeigt.

Im vorliegenden Artikel schauen wir uns genauer an, wie eine Tabelle mit Anlage-Feld zu einer entsprechenden SQL Server-Tabelle migriert werden kann. Dabei ist auch hier unser Ziel, ein entsprechendes **varbinary(max)**-Feld zu erstellen und die Inhalte unseres Anlage-Feldes automatisiert zum SQL Server zu übertragen.

Zuvor müssen wir jedoch ein wenig planen.

Migration mit dem SQL Server Migration-Assistant: Wie Anlage-Felder berücksichtigen?

Der SQL Server Migration Assistant ist das beste Werkzeug, wenn es um eine weitgehend automatisierte Migration der Tabellen von Access zu einer SQL Server-Datenbank geht.

Es werden mögliche Probleme angezeigt und es können alle Tabellen, Felder, Beziehungen, weitere Eigenschaften und auch noch die Daten migriert werden.

Leider bietet der SQL Server Migration Assistant keine Möglichkeit, ein Feld mit dem Datentyp **Anlage** in ein entsprechendes Feld zu migrieren. Der Workaround des SQL Server Migration Assistant ist es, eine Meldung anzuzeigen, dass der **Attachment**-Datentyp nicht migriert werden kann, und stattdessen immerhin das Feld als **varchar(8000)**-Feld anzulegen.

Hier haben wir nun zwei Möglichkeiten:

- Wir können die Situation so aufgreifen, wie sie ist, und für alle Anlage-Felder in der Quelldatenbank das dafür erstellte **varchar(8000)**-Feld in ein **varbinary(max)**-Feld umwandeln und die entsprechenden Dateien in dieses Feld übertragen.
- Oder wir lassen die Tabelle bei der Migration aus und migrieren sie komplett von Hand. Das würde jedoch bedeuten, dass wir auch die Beziehungen, an denen diese Tabelle beteiligt ist, nachträglich hinzufügen müssen. Das scheint auf jeden Fall die aufwendigere Variante zu sein, vor allem deshalb, weil die Migration die übrigen Felder der Tabelle korrekt durchführt.

Außerdem müssen wir noch beachten, dass ein Anlage-Feld auch immer mehr als eine Datei enthalten kann. Wie oben erwähnt, befindet sich hinter einem Anlage-Feld immer eine intern gepflegte Tabelle, die für jede enthaltene Datei einen eigenen Datensatz anlegt.

Wenn wir dies korrekt im SQL Server abbilden wollen, brauchen wir also eigentlich sogar eine eigene Tabelle für die Dateien.

Da es jedoch auch Fälle gibt, in denen die Anzahl der enthaltenen Dateien auf eine begrenzt ist, schauen wir uns zunächst diese einfachere Methode an.

Nochmal zusammengefasst die Ausgangssituation: Wir haben eine Migration durchgeführt und es fehlen nur noch die Anlage-Felder und ihre Daten, die als **varbinary(max)**-Felder angelegt werden sollen.

Also lauten die Voraussetzungen:

- Wir haben eine Access-Tabelle mit einem Anlage-Feld.
- Wir haben diese Tabelle bereits zum SQL Server migriert, wobei das Feld mit dem Datentyp **varchar(8000)** angelegt wurde.

Wir werden also zunächst eine Prozedur erstellen, der wir den Namen der Tabelle und des Feldes auf Access-Seite

übergeben sowie den Namen der Tabelle und des Feldes in der SQL Server-Datenbank.

Diese Prozedur soll nun folgende Schritte ausführen:

- Das vorhandene Feld mit dem Datentyp **varchar(8000)** löschen.
- Ein neues Feld mit dem Datentyp **varbinary(max)** anlegen.
- Den Inhalt des Anlage-Feldes für alle Datensätze der Quelltabelle in das neue **varbinary(max)**-Feld übertragen.

Vorbereitung

Wir benötigen einen Verweis auf die Bibliothek **Microsoft ActiveX Data Objects 6.1 Library**, den wir über den **Verweise**-Dialog des VBA-Editors hinzufügen können.

Prozeduren zum Anlegen und Übertragen erstellen

Aus Gründen der Wartbarkeit wollen wir die Funktion auf mehrere Routinen aufteilen. Die erste, die wir uns anschauen, soll die Voraussetzungen liefern, um überhaupt Bilddaten in die SQL Server-Tabelle schreiben zu können.

Zuallererst legen wir jedoch die Verbindungszeichenfolge fest, die wir für den Zugriff per ADODB auf die Datenbank benötigen. Sie sieht wie folgt aus und erfordert noch das Ersetzen der Platzhalter **[SERVER]** und **[DATABASE]** durch die bei Ihnen vorliegenden Werte:

```
Public Const cStrConnection As String = "Provider=MSO-
LEDBSQL;Data Source=[SERVER];Initial Catalog=[DATABA-
SE];Integrated Security=SSPI;"
```

Danach erstellen wir die erste Funktion namens **VarbinaryFeldAnlegenOderAnpassen** (siehe Listing 1).

Sie erwartet die folgenden Parameter:

- **cnn**: **ADODB.Connection**-Objekt für die zu verwendende Verbindung
- **strSQLTabelle**: Name der zu verwendenden Tabelle auf dem SQL Server
- **strSQLFeld**: Name des anzupassenden/zu erstellenden Bildfeldes auf dem SQL Server

Die Funktion liefert den Wert **True** zurück, wenn das Feld vorhanden ist (gegebenenfalls durch Anpassung oder Erstellung).

Die Funktion erstellt als Erstes ein neues **ADODB.Recordset** und referenziert es mit der Variablen **rst**.

Dann stellt es in der Variablen **strSQL** eine Abfrage zusammen, die in unserem Beispiel wie folgt aussieht:

```
SELECT DATA_TYPE, CHARACTER_MAXIMUM_LENGTH FROM INFORMA-
TION_SCHEMA.COLUMNS WHERE TABLE_NAME = 'tblBilder' AND
COLUMN_NAME = 'Bild'
```

Diese Abfrage ermittelt den Datentyp und die Anzahl Zeichen des Feldes, das noch den Datentyp **varchar(8000)** hat und das wir ersetzen wollen.

Diese Abfrage öffnen wir mit der **Open**-Methode des Recordsets und prüfen in einer **If...Then**-Bedingung, ob überhaupt ein entsprechender Datensatz gefunden wurde, sprich: ob das Feld überhaupt vorhanden ist.

Falls ja, prüft die zweite **If...Then**-Bedingung, ob der Datentyp des Feldes ungleich **varbinary** ist oder ob die maximale Zeichenanzahl für dieses Feld ungleich **-1** ist (was MAX entspricht). In diesem Fall erscheint eine Meldung, die fragt, ob der Datentyp des Feldes geändert werden soll.

Antwortet der Benutzer mit **Ja**, führt die Funktion bei deaktivierter Fehlerbehandlung zwei SQL-Anweisungen aus.

```
Public Function VarbinaryFeldAnlegenOderAnpassen(cnn As ADODB.Connection, strSQLTabelle As String, _
        strSQLFeld As String, Optional lngErrNumber As Long, Optional strErrDescription As String) As Boolean
    Dim rst As ADODB.Recordset
    Dim strSQL As String
    Dim intResult As VbMsgBoxResult

    Set rst = New ADODB.Recordset
    strSQL = "SELECT DATA_TYPE, CHARACTER_MAXIMUM_LENGTH FROM INFORMATION_SCHEMA.COLUMNS " & _
             "WHERE TABLE_NAME = '" & strSQLTabelle & "' AND COLUMN_NAME = '" & strSQLFeld & "'"
    rst.Open strSQL, cnn, adOpenStatic, adLockReadOnly

    If Not rst.EOF Then
        If rst.Fields("DATA_TYPE").Value <> "varbinary" Or rst.Fields("CHARACTER_MAXIMUM_LENGTH").Value <> -1 Then
            intResult = MsgBox("Das Feld " & strSQLFeld & " ist kein varbinary(max). Soll es geändert werden?", _
                vbYesNo + vbQuestion, "Feld anpassen?")
            If intResult = vbYes Then
                On Error Resume Next
                cnn.Execute "ALTER TABLE " & strSQLTabelle & " DROP COLUMN " & strSQLFeld
                cnn.Execute "ALTER TABLE " & strSQLTabelle & " ADD " & strSQLFeld & " varbinary(max)"
                If Not Err.Number = 0 Then
                    lngErrNumber = Err.Number
                    strErrDescription = Err.Description
                    Exit Function
                End If
                On Error GoTo 0
            End If
        End If
    Else
        intResult = MsgBox("Das Feld '" & strSQLFeld & "' ist noch nicht vorhanden. Soll es mit dem " _
            & "Datentyp varbinarx(max) angelegt werden?", vbYesNo, "Feld nicht vorhanden")
        If intResult = vbYes Then
            On Error Resume Next
            cnn.Execute "ALTER TABLE " & strSQLTabelle & " ADD " & strSQLFeld & " varbinary(max)"
            If Not Err.Number = 0 Then
                strErrDescription = Err.Description
                lngErrNumber = Err.Number
                Exit Function
            End If
            On Error GoTo 0
        End If
    End If

    VarbinaryFeldAnlegenOderAnpassen = True
    rst.Close

End Function
```

Listing 1: Funktion zum Anlegen von **varbinary(max)**-Feldern oder Umwandeln von anderen Felddatentypen in **varbinary(max)**

- Die erste Anweisung löscht das bestehende Feld mit dem angegebenen Namen.

- Die zweite fügt das neue Feld mit dem Datentyp **varbinary(max)** hinzu.

Löst dies einen Fehler aus, werden die beiden Rückgabeparameter **lngErrNumber** und **strErrDescription** mit den entsprechenden Werten aus der **Err**-Klasse gefüllt und die Funktion wird verlassen.

Es kann auch geschehen, dass das Feld, dessen Name wir mit **strSQLFeld** übergeben haben, noch nicht vorhanden ist. Dann erscheint eine Meldung, die den Benutzer fragt, ob es mit den Datentyp **varbinary(max)** angelegt werden soll. Antwortet der Benutzer mit **Ja**, wird dieses Feld zur Tabelle aus **strSQLTabelle** hinzugefügt. Tritt hier ein Fehler auf, verfahren wir mit den Fehlerinformationen wie oben.

Damit haben wir bereits das benötigte Feld mit dem Datentyp **varbinary(max)** vorbereitet.

Kopieren der Daten von Anlagefeld zum varbinary(max)-Feld

Hier stellt sich die Frage, wie wir im Detail beim Kopieren vorgehen wollen. Der Prozess ist ohnehin individuell, weil jedes Bild einzeln vom Anlage-Feld in ein Byte-Array übertragen und dann in das **varbinary(max)**-Feld eingetragen werden muss.

Wir können nun verschiedene Möglichkeiten wählen. Die erste Frage ist:

- Soll die Prozedur nur die Inhalte der Anlage-Felder in die entsprechenden **varbinary(max)**-Felder kopieren?

- Oder soll diese direkt den vollständige Datensatz kopieren?

Es wird vermutlich im Laufe der Zeit mal den einen, mal den anderen Fall geben. Weiter oben haben wir bereits diskutiert, dass es vermutlich oft vorkommt, dass das vollständige Datenmodell bis auf die Anlage-Felder mit Tools wie dem SQL Server Migration Assistant zum SQL Server übertragen wird.

Wir tun uns also in den meisten Fällen einen Gefallen, indem wir den Teil, der das Anlage-Feld in das **varbinary(max)**-Feld überträgt, in einer eigenen Funktion realisieren.

Wir können dann beispielsweise die folgenden Varianten für aufrufende Funktionen wählen:

- Eine Prozedur, die alle Datensätze der Access-Tabelle mit dem Anlage-Feld durchläuft und zunächst alle Daten außer dem Anlage-Feld in die SQL Server-Tabelle kopiert. Für jeden aktuellen Datensatz wird dann die oben beschriebene Funktion zum Übertragen der Daten aus dem Anlage-Feld aufgerufen.

- Oder wir führen das Kopieren der einfachen Daten in einer **INSERT INTO**-Anweisung für alle Datensätze gleichzeitig durch und durchlaufen anschließend eine Schleife mit allen Datensätzen, wo wir die Inhalte der Anlage-Felder nachziehen.

In beiden Fällen benötigen wir eine eigene Prozedur, die davon ausgeht, dass der eigentliche Datensatz bereits kopiert wurde und nur noch der Inhalt der Anlage-Felder übertragen werden muss.

Funktion zum Übertragen der Anlage-Felder

Daher programmieren wir die Funktion namens **CopyAttachmentToVarBinaryMax** so, dass wir mit den Parametern die folgenden Informationen übergeben:

- **strAccessTable**: Name der verwendeten Quelltabelle in Access

- **strAccessField**: Name des Anlage-Feldes in der Access-Tabelle

Anlagefelder mit mehreren Dateien zum SQL Server

Im Beitrag »Anlage-Feld zum SQL Server migrieren« (www.access-im-unternehmen.de/1542) haben wir gezeigt, wie Inhalte von Anlagefeldern einer Access-Datenbank in ein varbinary(max)-Feld einer SQL Server-Datenbank übertragen werden können. Dort sind wir davon ausgegangen, dass jedes Anlagefeld immer nur eine Datei enthält, was in vielen Fällen ausreichend ist. Was aber, wenn der Entwickler das Anlagefeld für mehrere Dateien vorgesehen hat – beispielsweise, um ein oder mehrere Produktbilder zu einem Produkt zu speichern? In diesem Fall müssen wir umdenken, denn wir können optimalerweise immer nur eine Datei in einem varbinary(max)-Feld speichern. Zum Speichern mehrerer Dateien müssen wir uns also einen Workaround überlegen. Wie dieser aussieht, schauen wir uns im vorliegenden Beitrag an.

Anlagefelder mit einem oder mehreren Dateien

Anlagefelder speichern ihre Anlagen in einer internen Tabelle, die von außen erst einmal nicht sichtbar ist. Deshalb ist es auch möglich, dass für einen einzelnen Datensatz mehr als eine Anlage hinterlegt werden kann.

Im Beispiel verwenden wir die Tabelle **tblProdukte**, die in der Entwurfsansicht wie in Bild 1 aussieht.

In der Datenblattansicht haben wir der Tabelle drei Bilder zum Anlagefeld hinzugefügt (siehe Bild 2).

Dass es in einem Anlagefeld ein Objekt wie eine Tabelle geben muss, sehen wir, wenn wir eine Tabelle mit einem Anlagefeld in eine neue, leere Abfrage ziehen. Hier fügen wir einmal alle Felder der Tabelle einschließlich der Felder des Anlagefeldes zum Entwurfsraster hinzu (siehe Bild 3).

Wechseln wir hier in die Datenblattansicht, sehen wir, dass die Idee, dass wir es hier mit einer internen Untertabelle zu tun haben, nicht allzu abwegig ist. Aus einem Datensatz werden nun

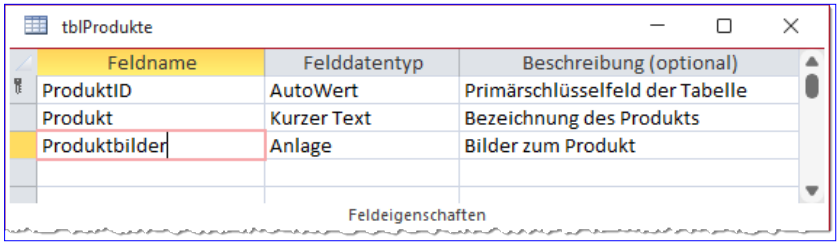

Bild 1: Tabelle zum Speichern von Produkten mit Bildern

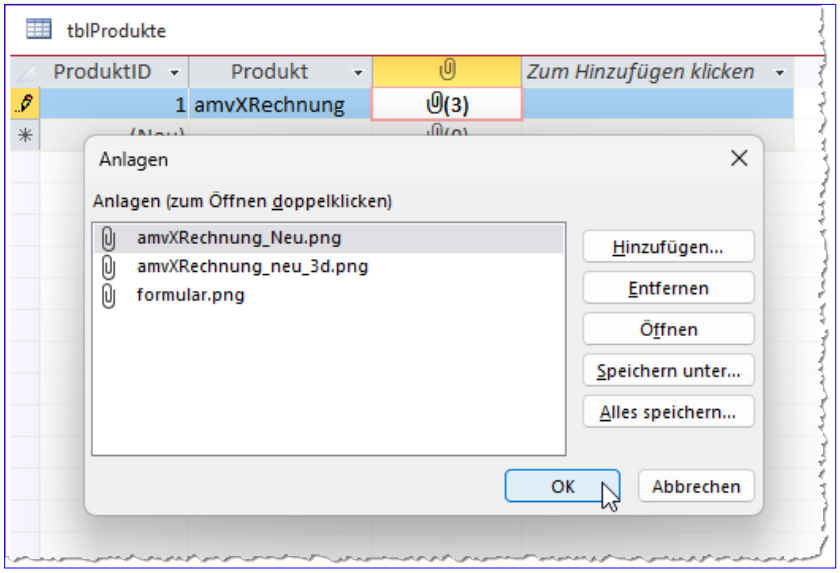

Bild 2: Tabelle mit mehreren Anlagen

direkt drei Datensätze, was sonst nur geschieht, wenn wir eine verknüpfte Tabelle mit in den Abfrageentwurf ziehen (siehe Bild 4).

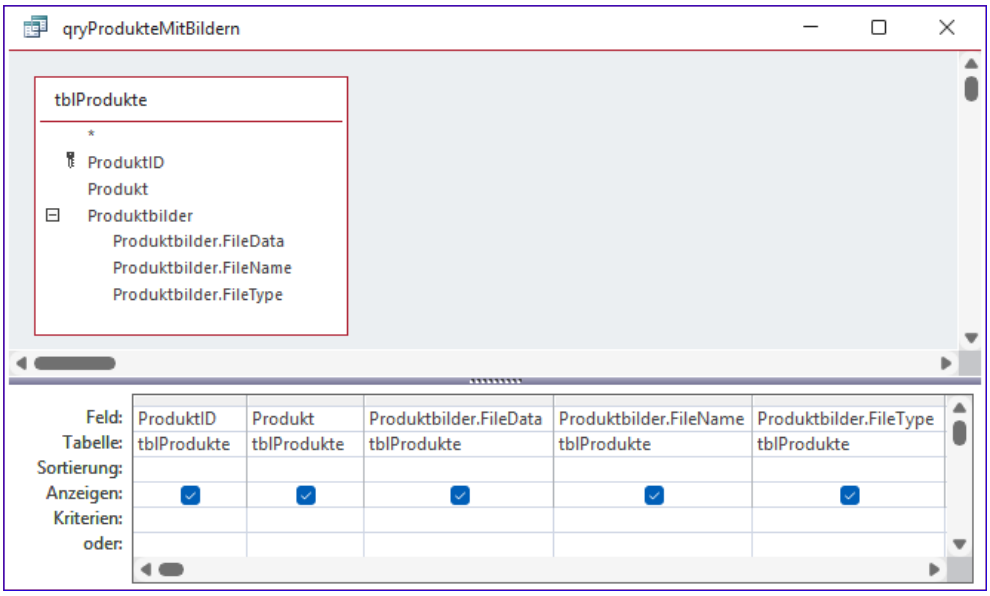

Bild 3: Die Untertabelle in einem Anlagefeld

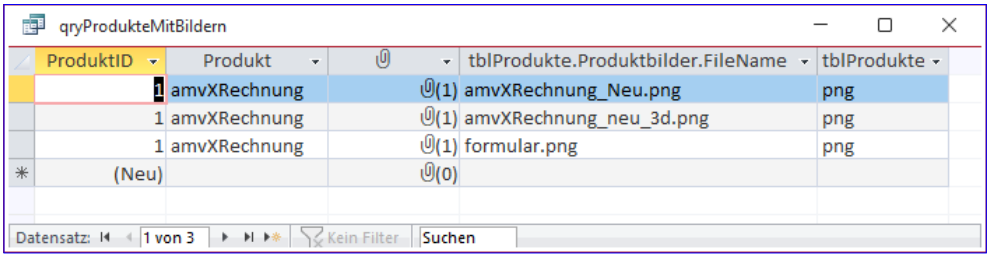

Bild 4: Die Felder eines Anlagefeldes in der Datenblattansicht

Anlagen mit mehreren Dateien zum SQL Server migrieren

Betrachten wir nun die Aufgabe, die Inhalte einer solchen Tabelle in eine SQL Server-Datenbank zu übertragen, sehen wir erst einmal kein derartiges Konstrukt, indem wir mehrere Datensätze pro Feld speichern könnten. Letztlich ist das, was wir hier abgebildet sehen, nichts anderes als zwei Tabellen, die per 1:n-Beziehung miteinander verknüpft sind. Und das lässt sich wiederum auch im SQL Server abbilden – wenn auch mit Unterstützung durch eine weitere Tabelle.

Wir erstellen also im SQL Server zwei Tabellen, um die Daten dieser einen Tabelle mit Anlagefeld abzubilden. Diese sehen wie folgt aus:

- **tblProdukte**: **ProduktID** (Primärschüsselfeld), **Produkt** (Textfeld)
- **tblProduktbilder**: **ProduktbildID** (Primärschlüsselfeld), **ProduktID** (Fremdschlüsselfeld zur Tabelle **tblProduktbilder**), **Produktbild** (Datentyp **varbinary(max)**)

Um die Tabellen schnell zu erstellen, können wir das folgende Skript nutzen:

```
CREATE TABLE tblProdukte (
    ProduktID INT IDENTITY(1,1) PRIMARY KEY,
    Produkt NVARCHAR(255) NOT NULL
);

CREATE TABLE tblProduktbilder (
    ProduktbildID INT IDENTITY(1,1)
        PRIMARY KEY,
    ProduktID INT NOT NULL,
    Produktbild VARBINARY(MAX) NOT NULL,
    CONSTRAINT FK_Produktbilder_Produkte
        FOREIGN KEY (ProduktID)
        REFERENCES tblProdukte (ProduktID)
            ON DELETE CASCADE
);
```

Danach können wir uns an die Erstellung des Codes zum Übertragen der Daten samt Bilddaten begeben.

Notwendiger Parameter zum Übertragen der Tabelle inklusive Anlagefeld

Im Beitrag **Anlage-Feld zum SQL Server migrieren** (**www.access-im-unternehmen.de/1542**) haben wir bereits eine Funktion programmiert, die wir komplett über Parameter steuern können.

Im vorliegenden Beitrag benötigen wir eine ähnliche Prozedur, die allerdings noch einige weitere Parameter benötigt. Immerhin sollen die Daten aus ursprünglich einer Tabelle nun auf zwei Tabellen aufgeteilt werden.

Zuvor legen wir fest, wie wir hier vorgehen wollen. Möchten wir eine ähnliche Prozedur schreiben, wie wir sie im oben genannten Beitrag angelegt haben, die für eine bereits bestehende Migration inklusive Daten nur noch die Inhalte der Anlagefelder hinterherkopiert? Dies ist in den Fällen interessant, wo man die übrige Migration bereits vollständig durchgeführt hat, beispielsweise mit einem Tool wie SQL Server Migration Assistant.

In diesem Beitrag wollen wir jedoch eine Lösung kreieren, welche die Inhalte der Tabellen vollständig überträgt.

Wir gehen also von den leeren Tabellen **tblProdukte** und **tblProduktbilder** auf dem SQL Server aus und wollen alle enthaltenen Informationen dorthin übertragen.

Wir werden also in den folgenden Schritten vorgehen:

- Übertragen eines Datensatzes der Tabelle **tblProdukte** zur gleichnamigen SQL Server-Tabelle (mit Ausnahme des Anlagefeldes)
- Merken des neuen Primärschlüsselwertes
- Durchlaufen der Anlagen und schrittweises Übertragen der Anlagen in die Tabelle **tblProdukteBilder**, wobei wir den gemerkten Primärschlüsselwert als Fremdschlüsselwert nutzen

Für die Parametrisierung benötigen wir also die folgenden Daten:

- **strAccessTable**: Name der Haupttabelle
- **strAccessField**: Name des Anlagefeldes der Haupttabelle
- **strAccessPKField**: Name des Primärschlüsselfeldes der Haupttabelle
- **lngAccessPKID**: Wert des Primärschlüsselfeldes des aktuellen Datensatzes
- **strSQLTable**: Name der Haupttabelle im SQL Server
- **strSQLPKField**: Name des Primärschlüsselfeldes der Haupttabelle im SQL Server
- **strSQLAttachmentTable**: Name der Anlagentabelle im SQL Server
- **strSQLAttachmentPKField**: Name des Primärschlüsselfeldes der Anlagentabelle im SQL Server
- **strSQLVarBinaryMaxField**: Name des **varbinary(max)**-Feldes in der Anlagentabelle der SQL Server-Datenbank
- **strSQLAttachmentFKField**: Name des Fremdschlüsselfeldes der Anlagentabelle im SQL Server
- **cnn**: Verbindung zum SQL Server
- **db**: Verweis auf das aktuelle **Database**-Objekt
- **strErrorMessage**: Rückgabeparameter für Fehlermeldungen

Aufruf der Funktionen

Den Start machen wir mit der Prozedur **TabelleMitMehrfachanlagenZumSQLServer** (siehe Listing 1). Diese erstellt einen Verweis auf das aktuelle **Database**-Objekt sowie eine Connection auf die entsprechende SQL Server-Datenbank.

```
Public Sub TabelleMitMehrfachanlagenZumSQLServer()
    Dim db As DAO.Database
    Dim cnn As ADODB.Connection
    Dim rst As DAO.Recordset
    Dim strErrorMessage As String
    Dim strSQL As String
    Dim cmd As ADODB.Command
    Dim rstResult As ADODB.Recordset
    Dim lngID As Long
    Dim bolKopiert As Boolean
    Set db = CurrentDb
    Set cnn = New ADODB.Connection
    cnn.Open cStrConnection
    Set rst = db.OpenRecordset("SELECT * FROM tblProdukte", dbOpenDynaset)
    Do While Not rst.EOF
        strSQL = "INSERT INTO tblProdukte(Produkt) OUTPUT INSERTED.ProduktID  VALUES('" & rst!Produkt & "');"
        Set cmd = New ADODB.Command
        cmd.ActiveConnection = cnn
        cmd.CommandText = strSQL
        cmd.CommandType = adCmdText
        Set rstResult = cmd.Execute()
        If Not rstResult.EOF Then
            lngID = rstResult!ProduktID
            bolKopiert = CopyAttachmentToVarBinaryMax_Multi("tblProdukte", "Produktbilder", "ProduktID", _
                rst!ProduktID, lngID, "tblProduktbilder", "Produktbild", "ProduktID", cnn, db, strErrorMessage)
            If bolKopiert = True Then
                MsgBox "Datensatz " & rst!ProduktID & " erfolgreich kopiert."
            Else
                MsgBox "Datensatz " & rst!ProduktID & " nicht kopiert." & vbCrLf & vbCrLf & strErrorMessage, _
                    vbCritical + vbOKOnly, "Fehler beim Kopieren"
            End If
        End If
        rst.MoveNext
    Loop
    rst.Close
    Set rst = Nothing
    Set db = Nothing
    cnn.Close
    Set cnn = Nothing
End Sub
```

Listing 1: Aufruf der Funktionen zum Kopieren von Tabellen mit Anlagefeldern mit mehreren Dateien

SQL Server-Tabellenverknüpfungsassistent, Teil 2

Der SQL Server-Tabellenverknüpfungsassistent, den wir im Beitrag »SQL Server-Tabellenverknüpfungsassistent« (www.access-im-unternehmen.de/1540) vorgestellt haben, funktioniert in vielen Settings bereits recht gut. Die generelle Praxistauglichkeit zeigt sichjedoch erst, wenn die Anwendung einmal auf andere Benutzer und/oder Verwendungszwecke stößt. Da dies nun geschehen ist, wollen wir im zweiten Teil der Beitragsreihe zum Thema SQL Server-Tabellenverknüpfungsassistent einige Änderungen und Erweiterungen nachreichen. Dabei geht es im Wesentlichen um die Benennung der Tabellenverknüpfungen, die von dem Assistenten entweder initial angelegt oder aktualisiert werden sollen.

Probleme beim Einsatz des Tabellenverknüpfungsassistenten

Das Hauptproblem, das uns zurückgemeldet wurde, ist die Inflexibilität bei der Verwendung der Bezeichnungen für die Tabellenverknüpfungen. Im ersten Teil gingen wir davon aus, dass wir allein mit dem Namen der Tabelle auskommen – so sah auch das Formular zum Verwalten der zu verknüpfenden Tabellen aus (siehe Bild 1).

Das Add-In erledigt die folgenden Aufgaben:

- Wenn die Access-Anwendung erstmals mit den Tabellen einer SQL Server-Datenbank verknüpft werden soll, legt es diese einfach unter dem gleichen Namen an, unter dem diese auch in der SQL Server-Datenbank gespeichert sind – ohne Berücksichtigung von **dbo** oder anderen Schemabezeichnungen.

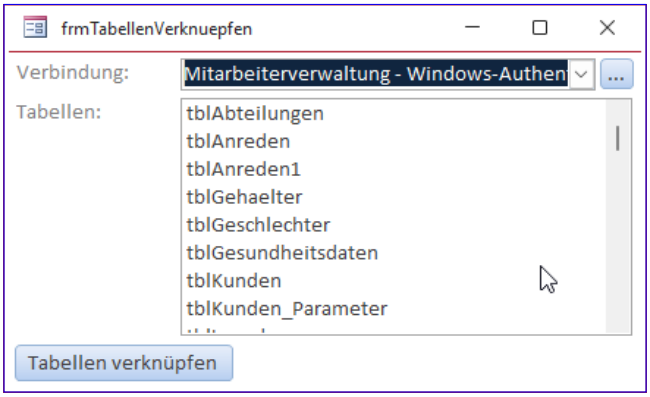

Bild 1: Hauptformular Access-Add-Ins

- Wenn die Tabellenverknüpfungen aktualisiert werden sollen, beispielsweise wenn sich der Name der Datenbank oder sich Name/Adresse des SQL Servers geändert haben, werden auch die Tabellennamen aus dem SQL Server verwendet. Wenn bereits eine Tabelle oder Tabellenverknüpfung gleichen Namens vorhanden ist, erscheint eine Inputbox, die dem Benutzer die Möglichkeit bietet, einen anderen Namen zu wählen oder die vorhandene Tabelle oder Tabellenverknüpfung zu überschreiben.

Das initiale Verknüpfen ist somit normalerweise immer funktional, außer man möchte, dass die Tabellenverknüpfungen anders benannt werden als die Tabellen auf dem SQL Server.

Das Aktualisieren der Tabellenverknüpfungen offenbart allerdings weitere Schwachstellen:

- Je nachdem, mit welcher Methode man initial die Tabellenverknüpfungen erstellt hat, wird eine Tabellenverknüpfung beispielsweise für eine Tabelle wie **tblKunden** auch einmal **dbo_tblKunden** genannt. Damit kommt das Add-In grundsätzlich nicht zurecht:

Es soll die Tabelle **tblKunden** als Tabellenverknüpfung angelegt und geprüft werden, ob bereits eine Verknüpfung namens **tblKunden** vorhanden ist. Falls nicht, wird die Tabellenverknüpfung unter dem Namen **tblKunden** hinzugefügt. Das ist natürlich nicht zielführend, denn dann gibt es zwei Tabellenverknüpfungen für die gleiche Tabelle, nämlich **dbo_tblKunden** und **tblKunden**. Da Abfragen, Formulare, Berichte und VBA in diesem Fall auf die Tabellenverknüpfung **dbo_tblKunden** eingestellt sind, greifen diese weiterhin auf die alte Tabellenverknüpfung zu, die nicht geändert wurde und gegebenenfalls auf die falsche Datenbank oder den falschen Server verweist.

- Eines der Probleme hierbei ist, dass das Add-In die vorhandenen Tabellenverknüpfungen nicht den Tabellen der SQL Server-Datenbank zuordnen kann, da es nur den Namen der Tabellenverknüpfung mit dem Namen der Tabelle vergleicht.

- Außerdem prüft das Add-In in der Version aus dem ersten Teil der Beitragsreihe immer nur, ob es bereits eine Tabelle oder Tabellenverknüpfung mit dem Namen der anzulegenden Tabellenverknüpfung gibt. Das ist eigentlich nur sinnvoll, wenn es bereits Tabellen gleichen Namens gibt, da diese gegebenenfalls nicht überschrieben werden sollen. Wenn es eine Verknüpfung gleichen Namens gibt, sollte diese eigentlich direkt überschrieben werden.

Diese Probleme adressieren wir in der Überarbeitung des Assistenten im vorliegenden Beitrag.

Ziele der Anpassung

Wir wollen also folgende Änderungen erreichen:

- Das Add-In soll zuverlässig erkennen, ob eine Tabellenverknüpfung sich auf eine in der SQL Server-Datenbank enthaltene Tabelle bezieht und diese dann einfach ersetzen.

- Dabei soll dann der Name der Tabellenverknüpfung übernommen werden, den die zu ersetzende Verknüpfung hatte.

- Außerdem wollen wir dem Benutzer die Möglichkeit geben, beim initialen Erstellen einer Tabellenverknüpfung festzulegen, wie die Tabellenverknüpfungen benannt werden sollen. Dazu stehen die beiden Platzhalter **Schema** und **Name** zur Verfügung, die man in eckigen Klammern neben weiteren Zeichen angeben kann.

Erster Schritt: Tabelle zum Speichern der Tabelleninformationen

Da wir nun nicht mehr nur mit den Tabellennamen arbeiten wollen, wie sie im SQL Server vorkommen, sondern auch noch die Namen der vorhandenen Tabellenverknüpfungen berücksichtigen wollen, nutzen wir zur Übersicht eine Tabelle namens **tblTablesTemp**. Diese ist wie in Bild 2 aufgebaut. Hier speichern wir neben dem Namen der Tabelle im SQL Server auch den Namen der Tabellen-

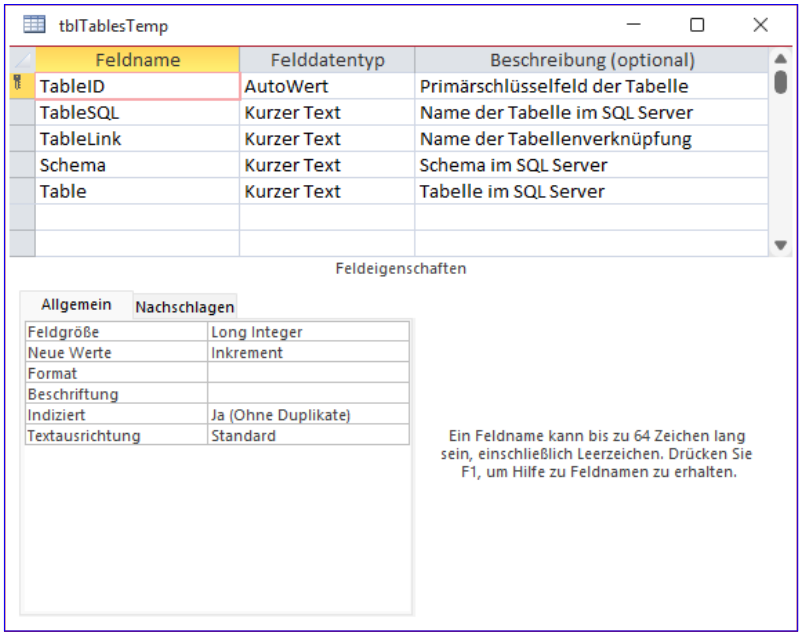

Bild 2: Tabelle zum Speichern der Tabelleninformationen

verknüpfung in der Access-Datenbank. Außerdem wollen wir auch noch das Schema (zum Beispiel **dbo**) und den Tabellennamen ohne Schema in eigenen Feldern speichern.

Füllen der Tabelle tblTempTables

Beim Laden des Formulars wird in der Ereignisprozedur **Form_Load** die Methode **cboVerbindung_AfterUpdate** aufgerufen. Diese wird auch als Ereignis ausgelöst, wenn der Benutzer eine neue Verbindung über das Kombinationsfeld **cboVerbindung** auswählt.

Die Prozedur **cboVerbindung_AfterUpdate** haben wir so angepasst, dass alle für die Tabelle **tblTablesTemp** benötigten Daten eingelesen werden (siehe Listing 1).

Wir deklarieren und initialisieren zwei **DAO.Database**-Variablen namens **dbc** und **db**, mit denen wir das **Database**-Objekt der Add-In-Datenbank und der geöffneten Datenbank referenzieren.

Im ersten Schritt leeren wir die Tabelle **tblTablesTemp**. Dann folgen die bereits bekannten Schritte, um den aktuellen Benutzer und sein Kennwort zu ermitteln, die wir zum Testen der Verbindung mit der Funktion **Verbindung-Testen** benötigen.

Ist dieser Aufruf erfolgreich, haben wir in **strVerbindungszeichenfolge** eine funktionierende Zeichenkette mit den Verbindungsdaten.

Zum Aufbau der Verbindung erstellen wir nun mit **CreateQueryDef** eine temporäre Abfrage, die wir mit **qdf** referenzieren.

Diese versehen wir über die **SQL**-Eigenschaft mit einer Abfrage, die uns die Felder **TABLE_SCHEMA**, **TABLE_NAME** und den mit dem Schema versehenen Tabellennamen (**CONCAT(TABLE_SCHEMA, '.', TABLE_NAME**) für das Feld **TABLE_SCHEMA_NAME** aus der Systemtabelle **INFORMATION_SCHEMA.TABLES** des SQL Servers, bei denen der Tabellentyp über das Kriterium **TABLE_TYPE IN ('BASE TABLE', 'VIEW')** als Tabelle oder View erkannt wird.

Die Verbindungszeichenfolge stellen wir auf den Inhalt der Variablen **strVerbindungszeichenfolge** ein und legen mit **.ReturnsRecords = True** fest, dass diese Abfrage Datensätze zurückliefern soll.

Auf Basis dieser Abfrage erstellen wir mit **OpenRecordset** ein Recordset, das wir mit **rstTablesSQL** referenzieren.

Danach erstellen wir noch ein zweites Recordset namens **rstTablesLocal**, das wir mit den Daten der Access-Systemtabelle **MSysObjects** füllen, bei denen das Feld **Connect** gefüllt ist.

Dies weist darauf hin, dass es sich um eine Tabellenverknüpfung handelt. Die Abfrage enthält einen weiteren Parameter, nämlich **NOT Name Like '~*'**.

Er sorgt für den Fall vor, dass wir in der aktuellen Session, also seit dem Öffnen der Access-Datenbank, bereits manuell Tabellenverknüpfungen gelöscht haben.

Diese werden nicht direkt vollständig gelöscht, sondern sind noch unter einem temporären Namen wie **~TMPCLP215381** in der Datenbank vorhanden.

Sie enthalten wie die gelöschte Tabellenverknüpfung noch die Eigenschaft **Connect** und werden so zu der entsprechenden SQL Server-Tabelle zugeordnet. Damit diese in der Liste nicht angezeigt werden, geben wir den entsprechenden zusätzlichen Filter an.

In der folgenden **Do While**-Schleife durchlaufen wir alle Datensätze des Recordsets **rstTableSQL** und schreiben dort den Namen der Tabelle in die Variable **strTableSQL**.

Damit durchlaufen wir in einer inneren **Do While**-Schleife alle Datensätze des Recordsets mit den Tabellenverknüpfungen. Hier lesen wir aus der Eigenschaft **SourceTable-**

```
Public Sub cboVerbindung_AfterUpdate()
    ' ... Deklarationen ...
    Set dbc = CodeDb
    Set db = CurrentDb
    dbc.Execute "DELETE FROM tblTablesTemp", dbFailOnError
    If Len(strBenutzername) = 0 Then
        strBenutzername = Nz(dbc.OpenRecordset("SELECT Benutzername FROM tblVerbindungszeichenfolgen " _
            & "WHERE VerbindungszeichenfolgeID = " & Me!cboVerbindung).Fields(0), "")
    End If
    If Len(strKennwort) = 0 Then
        strKennwort = Nz(dbc.OpenRecordset("SELECT Kennwort FROM tblVerbindungszeichenfolgen " _
            & "WHERE VerbindungszeichenfolgeID = " & Me!cboVerbindung).Fields(0), "")
    End If
    If VerbindungTesten(Me!cboVerbindung, strVerbindungszeichenfolge, lngErrorNumber, strErrorDescription) = True Then
        Set qdf = db.CreateQueryDef("")
        With qdf
            .SQL = "SELECT TABLE_SCHEMA, TABLE_NAME, CONCAT(TABLE_SCHEMA, '.', TABLE_NAME) AS TABLE_SCHEMA_NAME " _
                & "FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE IN ('BASE TABLE', 'VIEW') ORDER BY TABLE_TYPE, TABLE_NAME"
            .Connect = strVerbindungszeichenfolge
            .ReturnsRecords = True
        End With
        Set rstTablesSQL = qdf.OpenRecordset
        Set rstTablesLocal = db.OpenRecordset("SELECT Name FROM MSysObjects WHERE NOT Connect IS NULL " _
            & "AND NOT Name LIKE '~*'", dbOpenDynaset)
        Do While Not rstTablesSQL.EOF
            strTableSQL = rstTablesSQL!TABLE_SCHEMA_NAME
            Do While Not rstTablesLocal.EOF
                strTableSource = db.TableDefs(rstTablesLocal!Name).SourceTableName
                strTableLink = ""
                If rstTablesSQL!TABLE_SCHEMA_NAME = strTableSource Then
                    strTableLink = rstTablesLocal!Name
                    Exit Do
                End If
                rstTablesLocal.MoveNext
            Loop
           dbc.Execute "INSERT INTO tblTablesTemp(TableSQL, TableLink, [Schema], [Table]) VALUES('" & strTableSQL & "', '" _
              & strTableLink & "', '" & rstTablesSQL!TABLE_SCHEMA & "', '" & rstTablesSQL!TABLE_NAME & "')", dbFailOnError
            If Not rstTablesLocal.BOF Then
                rstTablesLocal.MoveFirst
            End If
            rstTablesSQL.MoveNext
        Loop
        Me.lstTabellen.RowSource = "SELECT TableSQL, TableLink, Schema, Table FROM tblTablesTemp ORDER BY TableSQL, TableLink"
        For i = 0 To Me.lstTabellen.ListCount
            If Not Len(Me.lstTabellen.Column(1, i)) = 0 Then
                Me.lstTabellen.Selected(i) = True
            End If
        Next i
    Else
        MsgBox "Keine gültige Verbindung." & vbCrLf & vbCrLf & strErrorDescription
    End If
End Sub
```

Listing 1: Einlesen der Tabellendaten beim Laden des Formulars

Name den Namen der Tabelle ein, die als Quelle für die Tabellenverknüpfung dient. Dazu greifen wir über die **TableDefs**-Auflistung auf die entsprechende Tabellendefinition zu und speichern das Ergebnis in der Variablen **strTableSource**.

Wenn nun der Wert der in der äußeren **Do While**-Schleife ermittelten SQL Server-Tabelle mit dem Namen der Quelltabelle mit der in der inneren Schleife durchlaufenen Tabelle übereinstimmt, haben wir einen Treffer: Dann tragen wir in die Variable **strTableLink** (für die Tabellenverknüpfung) den Namen der Tabellenverknüpfung ein und verlassen die innere Schleife.

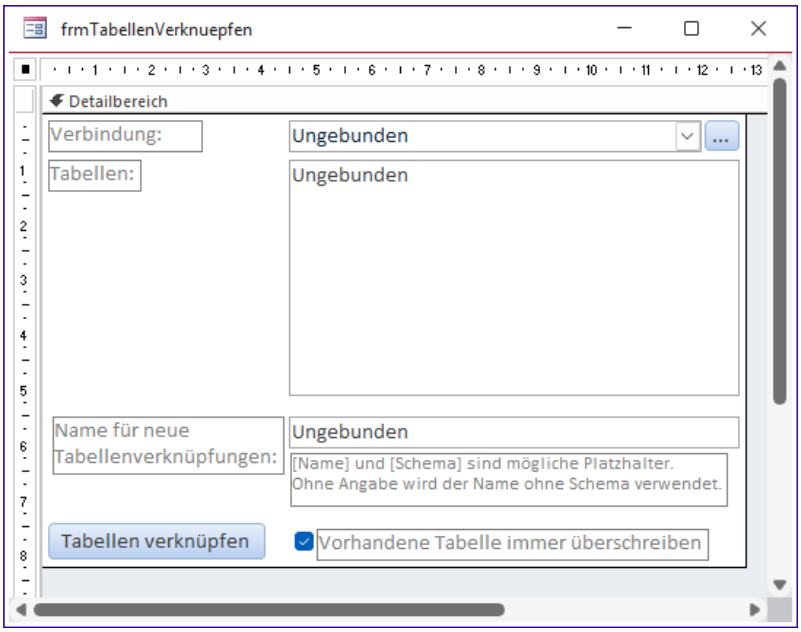

Bild 3: Neuer Entwurf des Formulars **frmTabellenVerknuepfen**

Anderenfalls durchlaufen wir die lokalen Tabellen solange, bis wir eine Übereinstimmung finden oder alle Tabellen durchlaufen wurden. Damit können wir einen neuen Datensatz in die Tabelle **tblTablesTemp** anlegen – unabhängig davon, ob wir eine passende Tabellenverknüpfung gefunden haben.

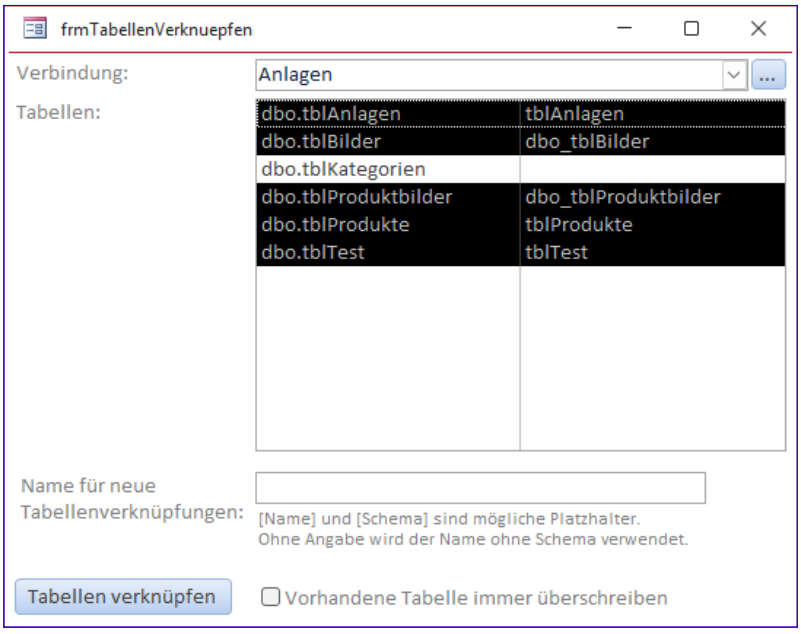

Bild 4: Formularansicht des Formulars **frmTabellenVerknuepfen**

Es kann schließlich auch vorkommen, dass eine Tabelle der SQL Server-Datenbank noch nicht als Tabellenverknüpfung in der Access-Datenbank repräsentiert ist.

Wir tragen also den Namen der SQL Server-Tabelle, den Namen der Tabellenverknüpfung, das Schema und den reinen Tabellennamen in die Tabelle **tblTablesTemp** ein.

Damit stellen wir das Recordset für die innere Schleife wieder auf den ersten Datensatz zurück und bewegen den Datensatzzeiger für das Recordset der äußeren Schleife zum nächsten Datensatz.

SQL Server-Tabellen und Tabellenverknüpfungen im Listenfeld anzeigen

Bevor wir den Rest der Prozedur betrachten, schauen wir uns die notwendigen Änderungen am Listenfeld **lstTabellen** zur Anzeige der Tabellen im Formular **frmTabellenverknuepfungen** an (siehe Bild 3).

Hier haben wir zunächst eine Datensatzherkunft zugewiesen:

```
SELECT TableSQL, TableLink, Schema, Table
FROM tblTablesTemp
ORDER BY TableSQL, TableLink;
```

Damit nur die beiden Felder **TableSQL** und **TableLink** angezeigt werden, haben wir die Eigenschaften **Spaltenanzahl** auf **4** und **Spaltenbreiten** auf **;;0cm;0cm** eingestellt. Die übrigen beiden Felder **Schema** und **Table** landen unsichtbar im Listenfeld, wodurch wir sie zwar nicht sehen, aber dennoch auf diese zugreifen können, sollte dies eventuell zum Verknüpfen erforderlich sein.

In der Formularansicht sieht das nun beispielsweise wie in Bild 4 aus.

Es fehlt noch der Teil des Codes, der dafür sorgt, dass solche Tabellen, für die bereits eine Verknüpfung vorliegt, automatisch markiert werden.

Dazu durchlaufen wir die Elemente des Listenfeldes in einer **For...Next**-Schleife über die Werte **0** bis zur Anzahl der Listenelemente minus **1**.

Wir prüfen, ob die erste Spalte der aktuellen Zeile einen Eintrag mit einer Länge größer als **0** enthält.

Ist das der Fall, wird der Eintrag durch Einstellen der **Selected**-Auflistung für die Zeile mit dem entsprechenden Index auf **True** eingestellt.

Damit haben wir den Teil zum Darstellen der vorliegenden SQL Server-Tabellen und der entsprechenden Tabellenverknüpfungen bereits erledigt.

Erweiterte Optionen zum Verknüpfen von Tabellen

Wie oben beschreiben, wollen wir nun das Verknüpfen von Tabellen optimieren. Nochmal kurz zusammengefasst:

- Wenn eine Verknüpfung vorhanden ist, soll diese je nach den Vorgaben des Benutzers entweder neu erstellt oder beibehalten werden.
- Wenn eine Verknüpfung noch nicht vorhanden ist, soll die Verknüpfung nach bestimmten Regeln neu erstellt werden. Diese Regeln beziehen sich auf den Namen der neuen Tabellenverknüpfung.

Damit der Benutzer Regeln für die zu verwendenden Namen aufstellen kann, haben wir dem Formular **frmTabellenVerknuepfen** weitere Steuerelemente hinzugefügt.

Hier sehen wir als Erstes ein Textfeld mit der Beschriftung **Name für neue Tabellenverknüpfungen**. Hier können wir beliebige Texte einfügen, aber wir sollten die Platzhalter **[Name]** und gegebenenfalls zusätzlich **[Schema]** hinzufügen.

Wichtig: Wenn wir hier keinen Wert angeben, wird der Name der Tabelle ohne Schema verwendet, bei **dbo.tblAnlagen** also beispielsweise nur **tblAnlagen**.

Beispiele für die Benennung der neuen Tabellenverknüpfungen

Hier sind einige Beispiele für die Ausdrücke, die im Feld **Name für neue Tabellenverknüpfungen** angegeben werden können:

- **[Name]**: Legt nur den Namen der SQL Server-Tabelle ohne Schema als Bezeichnung der Tabellenverknüpfung fest.
- **Keine Angabe**: Legt ebenfalls den Namen der SQL Server-Tabelle ohne Schema als Bezeichnung der Tabellenverknüpfung fest.
- **[Schema]_[Name]**: Legt die SQL Server-Tabelle **dbo.tblAnlagen** unter dem Namen **dbo_tblAnlagen** als Tabellenverknüpfung an.

- **sql_[Name]**: Legt die SQL Server-Tabelle **dbo.tblAnlagen** unter dem Namen **sql_tblAnlagen** als Tabellenverknüpfung an.

Beachten Sie, dass die Regeln für die Benennung von Access-Objekten hier berücksichtigt werden – also nur alphanumerische Zeichen und der Unterstrich. Es sind zwar auch andere Zeichen möglich, aber ungünstig.

Option »Vorhandene Tabellen immer überschreiben«

Diese Option sorgt dafür, dass alle Tabellenverknüpfungen, die im Listenfeld markiert sind, gelöscht und neu angelegt werden. Das führt auch dazu, dass die neuen Tabellenverknüpfungen unter Berücksichtigung der angegebenen Benennungskonvention erstellt werden.

Wenn man also beispielsweise eine Migration der Datenbank mit dem SQL Server Migration Assistant durchgeführt hat, wurden die Tabellenverknüpfungen üblicherweise mit **dbo_Tabellenname** benannt.

Dann sollte man, wenn man die Tabellen mit der hier vorgestellten Lösung neu verknüpft, auch das Format **[Schema]_[Name]** einstellen.

Durchführung der Verknüpfung

Nachdem wir festgelegt haben, welche Tabellen verknüpft werden sollen und nach welchem Schema, können wir auf die Schaltfläche **cmdTabellenVerknuepfen** klicken. Diese ruft die Prozedur aus Listing 2 auf.

Die Prozedur deklariert drei Variablen für die Benennung von Tabellen:

- **strTableSQL**: Name der Tabelle im SQL Server (siehe erste Spalte im Listenfeld)

- **strTableLocal**: Name der Tabellenverknüpfung, die mit der Tabelle aus **strTableSQL** verknüpft ist – sofern vorhanden, sonst leer.

- **strTableNew**: Neuer Name für die Tabellenverknüpfung mit der Tabelle aus **strTableSQL**. Kann mit **strTableLocal** übereinstimmen, muss es aber nicht.

Als Erstes prüft die Prozedur, ob die Bedingungen für das Ersetzen der Tabellenverknüpfungen erfüllt sind. Wenn der Benutzer im Textfeld **txtNeuerName** nicht den Platzhalter **[Name]** einträgt, der den Tabellennamen aus dem SQL Server widerspiegelt und dieses Textfeld auch nicht leer lässt, kann er nur eine Verknüpfung neu erstellen.

Der Grund ist, dass mit dem statischen Namen für die neue Verknüpfung sonst mehrere neue Verknüpfungen den gleichen Namen erhalten würden, was beim Anlegen des ersten doppelten Objekts zu einem Fehler führen würde.

Also prüfen wir zuerst, ob **txtNeuerName** nicht leer ist. Ist das Textfeld leer, verwendet das Formular automatisch den Namen der SQL Server-Tabelle für die Tabellenverknüpfung und wir haben individuelle Namen je Tabellenverknüpfung.

Ist es nicht leer, prüft die Prozedur, ob der Benutzer den Platzhalter **[Name]** in den Ausdruck für den Namen der neuen Tabellenverknüpfungen integriert hat.

Anderenfalls prüft sie, ob der Benutzer mehr als einen Eintrag der Liste zum Aktualisieren der Verknüpfung ausgewählt hat. Ist das der Fall, erscheint eine Meldung, dass dies nicht möglich ist und die Prozedur wird abgebrochen.

Ist das nicht der Fall, prüft die Prozedur mit einem Aufruf der Funktion **VerbindungTesten** wieder, ob die Verbindung funktioniert, und liefert mit **strVerbindungszeichenfolge** die aktuelle Verbindungszeichenfolge zurück.

Dann durchläuft die Prozedur in einer **For...Next**-Schleife alle Elemente des Listenfeldes **lstTabellen**. Dabei werden die Werte von **0** bis zur Anzahl der Listenelement minus **1** verwendet.