HAUFE.
Ausgabe 02/2024

ACCESS
IM UNTERNEHMEN

FORMULARE GENERIEREN

Konfigurieren und erstellen Sie Formulare mit wenigen Mausklicks (ab Seite 52).

In diesem Heft:

ÜBERSICHTS- UND DETAILFORMULARE

Entwickeln Sie Formulare für die Ansicht aller Datensätze einer Tabelle und zur Bearbeitung einzelner Datensätze.

SEITE 26

INDIVIDUELLE ICONS IN FORMULAREN

Fügen Sie Formularen individuelle Icons direkt aus der Tabelle MSysResources hinzu – ohne Umweg über das Dateisystem.

SEITE 2

VON DER ABFRAGEN ZUR STORED PROCEDURE

Übersetzen Sie Access-Abfragen in gespeicherte Prozeduren und gewinnen Sie an Performance.

SEITE 39

Formularerstellung automatisieren

Es gibt Aufgaben im Leben eines Access-Entwicklers, die immer wieder in ähnlichen Varianten auftauchen. Eine davon ist das Erstellen einfacher Formulare, die lediglich die Felder eines Datensatzes einer Tabelle darstellen und Schaltflächen zum Abschließen oder Verwerfen der Eingabe anbieten. Ausgehend von einem neuen, leeren Formulare in der Entwurfsansicht wiederholen sich immer wieder die gleichen Schritte – der einzige Unterschied ist die Tabelle oder Abfrage, deren Felder im Formular angezeigt werden. In dieser Ausgabe schauen wir uns an, wie ein solches Formular manuell erstellt wird. Wir gehen aber auch noch einen Schritt weiter und liefern eine Lösung, mit der Sie solche Formulare von nun an per Mausklick erstellen können.

Im ersten Beitrag dieser Ausgabe legen wir noch einmal selbst Hand an. Unter dem Titel **Tabellendaten mit Übersicht und Details anzeigen** zeigen wir ab Seite 26, wie wir ein Übersichtsformular und ein Detailformular von Hand erstellen. Das Übersichtsformular zeigt alle Daten der zugrunde liegenden Tabelle in einem Unterformular in der Datenblattansicht an. Mit Steuerelementen im Hauptformular können wir die angezeigten Datensätze im Unterformular filtern. Mit verschiedenen Schaltflächen lassen sich vorhandene Datensätze zum Bearbeiten öffnen oder löschen. Eine weitere Schaltfläche erlaubt das Anlegen neuer Datensätze. Zum Anlegen oder Bearbeiten von Daten erstellen wir ein Detailformular, das die Daten eines einzigen Datensatzes liefert und ihre Bearbeitung erlaubt.

Allein das Anlegen des Detailformulars ist eine Menge Arbeit. Diese können wir uns allerdings sparen, wenn wir uns auf gewisse Standards bei der Erstellung dieses Formulartyps einigen. Wie das gelingt, zeigen wir in der Lösung aus dem Beitrag **Detailformular per Mausklick erstellen** ab Seite 52. Die Lösung ist so flexibel, dass wir in einem Konfigurationsformular beliebige Tabellen oder Abfragen auswählen und auf der Basis der ausgewählten Felder ein entsprechendes Detailformular anlegen können. Nachdem wir einige wichtige Einstellungen vorgenommen oder übernommen haben, können wir per Mausklick das Formular anlegen lassen. Das Beste ist: Wir erstellen sogar noch ein Add-In, das die Funktion direkt über die Benutzeroberfläche von Access verfügbar macht.

Hübsche Icons geben einer Anwendung das gewisse Etwas. Noch schöner ist es, wenn wir einzelne Formulare mit eigenen Icons ausstatten können. Die grundlegende Technik dazu haben wir bereits einmal vorgestellt. Im Beitrag Individuelle **Formular-Icons ohne Zusatzdateien** zeigen wir jedoch ab Seite 2 noch, wie man dies realisiert, ohne dazu Icons im Dateisystem speichern zu müssen.

Gegentlich liefert Access unerwartete Ergebnisse – zum Beispiel, indem Datensätze beim Schließen eines Formulars einfach nicht gespeichert werden. Woran dies liegt und wie wir es vermeiden, beschreiben wir im Beitrag **Formulare: Datensatz wird nicht gespeichert** ab Seite 11.

Einen neuen Bug haben wir auch noch gefunden. Das TreeView-Steuerelement verarbeitet unter bestimmten Umständen Einheiten nicht richtig. Mehr dazu unter **TreeView: Bug durch falsche Einheiten** ab Seite 16.

Nach der Migration von Access zum SQL Server performan manche Abfragen nicht mehr so wie vorher. Im Beitrag **SQL Server: Von der Abfrage zur Stored Procedure** zeigen wir ab S. 39, wie wir solche Abfragen durch die Umwandlung in gespeicherte Prozeduren beschleunigen.

Viel Spaß beim Lesen!

Ihr André Minhorst

Individuelle Formular-Icons ohne Zusatzdateien

Im Beitrag »Icons in Access-Formularen und Berichten« (www.access-im-unternehmen.de/1235) haben wir schon einmal eine Möglichkeit aufgezeigt, wie man Formulare und Berichte in Access mit individuellen Icons ausstatten kann. So kann man beispielsweise ein Formular zum Bearbeiten eines Kunden mit dem gleichen Icon ausstatten, das man auch für die Schaltfläche zum Öffnen dieses Formulars im Ribbon untergebracht hat. Der Benutzer kann so noch besser erkennen, worum es im Formular geht. In der vorherigen Fassung der Lösung hatten wir allerdings noch das Problem, dass wir die Icons, die links oben in Formularen und Berichten erscheinen sollten, noch im Dateisystem speichern mussten. Das kann aus diversen Gründen zu Problem führen und daher sind wir froh, hier den nächsten Schritt gehen zu können: Das direkte Einlesen der Icons aus der Tabelle »MSysResources« und anschließendes Anzeigen in Formulare und Berichten.

Ziel: Formular-Icons ohne externe Dateien

Wie bereits erwähnt, sind wir bereits in der Lage, Formulare mit einem eigenen Icon auszustatten (siehe Bild 1). Das Problem ist jedoch bisher gewesen, dass die entsprechende Icon-Datei sich auf der Festplatte befinden muss, damit wir diese laden und per API dem Formular als Icon zuweisen konnten. Damit wir nicht für jede Anwendung einen eigenen Ordner bereitstellen mussten, der die benötigten Icon-Dateien enthält oder sogar die Icon-Dateien lose im Verzeichnis der Access-Datenbank gespeichert haben, befanden sich die **.ico**-Dateien bis zur Verwendung in der Tabelle **USysResources** und wurden erst dann zur Anwendung exportiert.

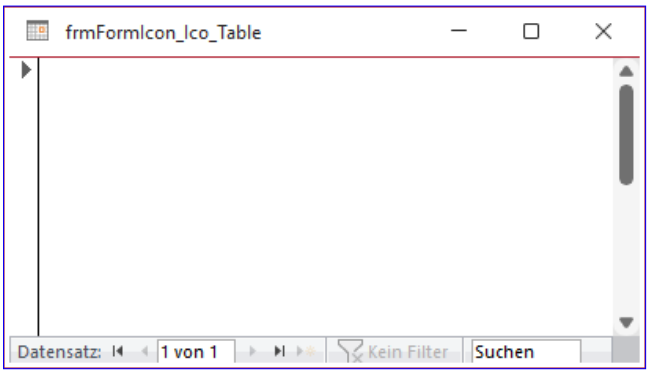


Bild 1: Formular mit individuellem Icon

Nunmehr haben wir den noch fehlenden Schritt realisiert: Wir können nun die Icon-Dateien direkt aus dem Anlagefeld der Ressourcen-Tabelle auslesen und dem jeweiligen Formular zuweisen. Wie das gelingt, lesen Sie in diesem Beitrag.

Code zum Anzeigen eines Formularicons

Der Code sieht im Wesentlichen wie in Listing 1 aus. Im oberen Teil finden wir einige Konstanten-Deklarationen, die wir im weiteren Verlauf benötigen.

Darunter sehen wir die Deklaration von zwei API-Funktionen, die uns in der folgenden Prozedur unterstützen.

Schließlich folgt die Funktion **SetFormIconFromMSysResources**, welche die eigentliche Arbeit übernimmt. Die Funktion hat zwei Parameter:

- **hWnd**: Das Handle des Fensters, für das das Icon gesetzt werden soll. Das ist normalerweise das Handle des Access-Formulars oder -Berichts.
- **strIconName**: Der Name des Icons, dessen ID in der **MSysResources**-Tabelle gesucht wird.

Die Funktion verwendet die folgenden Variablen:

- **hIcon**: Speichert das Handle des eingelesenen Icons.
- **bData**: Ein Byte-Array, das die Binärdaten des Icons aus der **MSysResources**-Tabelle aufnimmt.
- **dwOffset**: Speichert das Offset des Icons.
- **dwSize**: Speichert die Größe des Icon-Ressourcenblocks.
- **lngIndex**: Index des Icons in der **.ico**-Datei. Eine **.ico**-Datei kann mehrere Icons enthalten.

```
Option Compare Database
Option Explicit

Public Const C_IMAGE_ICON = 1
Public Const C_LR_LOADFROMFILE = &H10
Public Const C_WM_SETICON = &H80
Public Const ICRESVER As Long = &H30000
Public Const LR_DEFAULTSIZE As Long = &H40
Public Const WM_SETICON As Long = &H80
Public Const ICON_SMALL As Long = 0

Public Declare PtrSafe Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hWnd As LongPtr, _
    ByVal wMsg As Long, ByVal wParam As LongPtr, lParam As Any) As LongPtr
Public Declare PtrSafe Function CreateIconFromResourceEx Lib "user32.dll" (presbits As Any, dwResSize As Any, _
    ByVal fIcon As Long, ByVal dwVer As Long, ByVal cxDesired As Long, ByVal cyDesired As Long, _
    ByVal flags As Long) As LongPtr

Public Function SetFormIconFromMSysResources(ByVal hWnd As Long, ByVal strIconName As String) As Boolean
    Dim hIcon As LongPtr
    Dim bData() As Byte
    Dim dwOffset As LongPtr
    Dim dwSize As LongPtr
    Dim lngIndex As Long
    Dim lngID As Long
    lngIndex = 0
    lngID = Nz(DLookup("id", "MSysResources", "name='" & strIconName & "'"), 0)
    If Not lngID = 0 Then
        bData() = BLOB2Binary0710("MSysResources", "Data", "Id", lngID, True)
        dwSize = VarPtr(bData(16& * lngIndex + 14&))
        dwOffset = VarPtr(bData(bData(16& * lngIndex + 18&)))
        hIcon = CreateIconFromResourceEx(ByVal dwOffset, ByVal dwSize, 1, ICRESVER, 0&, 0&, LR_DEFAULTSIZE)
        If hIcon Then
            SendMessage hWnd, WM_SETICON, ICON_SMALL, ByVal hIcon
        End If
    Else
        MsgBox "Icon '" & strIconName & "' fehlt in der Tabelle MSysResources", vbExclamation + vbOKOnly, "Icon fehlt"
    End If
End Function
```

Listing 1: Code zum Bereitstellen von Icons in Formularen und Berichten

- **lngID**: Primärschlüsselwert des Eintrags der Tabelle **MSysResources** mit dem gesuchten Icon.

Die Funktion setzt den Index für das aus der **.ico**-Datei zu verwendende Icon mit **lngIndex** auf **0**. Der Parameter **strIconName** der Funktion liefert den Wert des Feldes **Name** der Tabelle **MSysResources**, dessen Anlage aus dem Feld **Data** als Icon verwendet werden soll.

Die Funktion holt per **DLookup** den Wert des Feldes **id** für diesen Datensatz und speichert diesen in der Variablen **lngID**.

Die folgende **If...Then**-Bedingung prüft, ob der gesuchte Datensatz gefunden werden konnte, anderenfalls erscheint eine entsprechende Meldung.

Anschließend folgt die Extraktion der Icon-Daten aus dem Feld **Data**. Das erledigen wir mit einem Aufruf der Funktion **BLOB2Binary0710** aus dem Modul **mdlImages**, der wir den Namen der Tabelle, das Anlagefeld, das Feld mit dem Primärschlüssel und den Primärschlüsselwert übergeben. Die Icon-Daten landen danach in einem **Byte**-Array namens **bData**. Für dieses Array ermitteln wir nun die Größe und das Offset im Speicher und speichern beides in den Variablen **dwSize** und **dwOffset**.

Schließlich folgt der Einsatz der API-Funktionen. Die erste heißt **CreateIconFromResourceEx** und liefert uns ein Handle auf das Icon. Dieses Handle setzen wir schließlich mit der Funktion **SendMessage** als Icon des Formulars oder Berichts mit dem Handle aus **hWnd** ein.

Einsatz der Funktion

Den Aufruf der Funktion **SetFormIconFromMSysResources** fügen wir nun in die Ereignisprozedur **Form_Load** des Formulars oder Berichts ein, der ein Icon aus der Tabelle **MSysResources** anzeigen soll. Das sieht beispielsweise wie folgt aus:

```
Private Sub Form_Open(Cancel As Integer)
    Call SetFormIconFromMSysResources(Me.hWnd, "alarmclock")
End Sub
```

Dazu muss nun nur noch die Tabelle **MSysResources** einen Datensatz enthalten, der für das Feld **Name** den Wert **alarmclock** aufweist und dessen Feld **data** eine entsprechende **.ico**-Datei enthält. Das ist in unserer Datenbank der Fall (siehe Bild 2).

Voraussetzungen für die .ico-Dateien

Nun sind allerdings leider nicht alle **.ico**-Dateien für die Anzeige als Icon eines Formulars oder Berichts geeignet. Wir benötigen eine bestimmte Auflösung, sonst wird das Icon nicht korrekt angezeigt. Dazu gehören beispielsweise Auflösungen mit 8- oder 24-Bit. 4-Bit reichen nicht aus.

Hinzufügen der .ico-Dateien

Leider können wir die **.ico**-Dateien nicht auf die gleiche einfache Weise wie **.png**-Dateien zur Tabelle **MSysResources** hinzufügen. Dazu brauchten wir einfach nur ein Formular in der Entwurfsansicht zu öffnen und dann ein Bild zum Formular oder zu einer Schaltfläche hinzuzufügen.

Die dabei ausgewählten Dateien werden automatisch als **.png**-Dateien in der Tabelle **MSysResources** hinterlegt und können dann in den entsprechenden Eigenschaften der Steuerelemente ausgewählt werden. Wenn wir hier **.ico**-Dateien auswählen, werden diese als **.png**-Dateien importiert. **.png**-Dateien können wir leider nicht als Icon von Formularen und Berichten einsetzen.

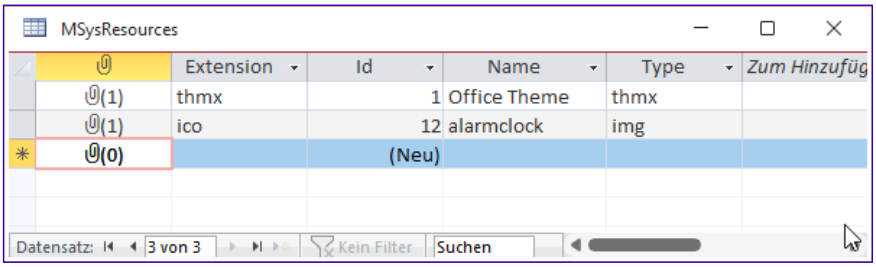


Bild 2: Das anzuzeigende Icon in der Tabelle **MSysResources**

.ico-Dateien manuell hinzufügen

Also haben wir zwei Möglichkeiten. Die erste ist, die **.ico**-Dateien manuell zur Tabelle **MSysResources** hinzuzufügen. Das gelingt mit den folgenden Schritten:

- Wir öffnen die Tabelle **MSysResources** in der Datenblattansicht.
- Wir klicken für einen neuen, leeren Datensatz doppelt auf das Anlage-Feld.
- Im Dialog **Anlegen** klicken wir auf **Hinzufügen...** und wählen die gewünschten **.ico**-Datei aus, zum Beispiel **apple.ico**.
- Danach tragen wir für das Feld **Extension** den Wert **ico**, für **Name** den Wert **apple** und für **Type** den Text **img** ein.

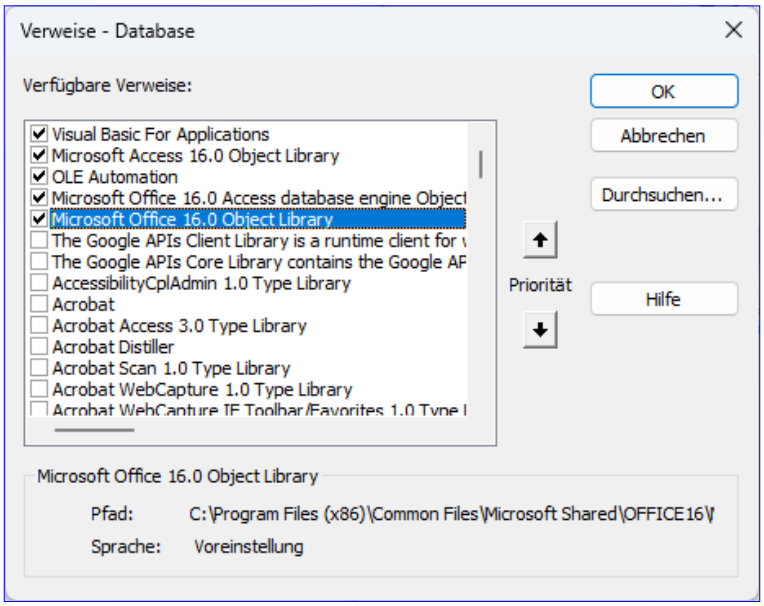


Bild 3: Verweis auf die Office-Bibliothek

Danach können wir die Funktion **SetFormIconFromMSysResources** für dieses Icon aufrufen:

```
Call SetFormIconFromMSysResources(Me.hWnd, "apple")
```

Diese Vorgehensweise ist allerdings nicht besonders komfortabel, wenn wir mehrere **.ico**-Dateien zur Tabelle **MSysResources** hinzufügen wollen. Also schreiben wir uns schnell Code, um den Vorgang zu automatisieren.

Mehrere .ico-Dateien gleichzeitig in die Tabelle MSysResources einlesen

Was benötigen wir dazu als Erstes? Einen Dateiauswahl-Dialog, mit dem wir mehrere Dateien aus einem Verzeichnis auswählen können. Die notwendigen Elemente finden wir in der Office-Bibliothek, die wir wie in Bild 3 per Verweis referenzieren.

.ico-Dateien auswählen

Danach können wir uns eine Funktion namens **SelectICOFiles** wie in Listing 2 zusammenstellen. Diese nimmt mit dem Parameter **strFolder** ein Verzeichnis entgegen, das als Startverzeichnis des Dateiauswahl-Dialogs verwendet wird. Zuerst werden das Objekt **objFiledialog** und zwei Variablen für die Verarbeitung der Pfade zu den **.ico**-Dateien deklariert. Mit der Variablen **objFiledialog** referenzieren wir den Dateiauswahl-Dialog, den wir mit der Methode **Filedialog** des **Application**-Objekts initialisieren. Dabei übergeben wir den Wert **msoFileDialogFilePicker** als Parameter.

Für den Dialog legen wir mit **AllowMultiSelect** fest, das mehrere Dateien gleichzeitig ausgewählt werden können. Außerdem stellen wir den Startordner mit **InitialFileName** ein. Wir leeren die Filter mit der **Clear**-Methode der **Filters**-Auflistung und fügen einen neuen Filter für **.ico**-Dateien hinzu. Dann zeigen wir den Dialog mit der **Show**-Methode an. Liefert diese den Wert **True** zurück, hat der Benutzer den Dialog geschlossen und wir werten die gewählten Einträge in einer **For Each**-Schleife über die Elemente der Auflistung **SelectedItems** aus.

Dabei stellen wir in **strFilenames** eine durch Semikola getrennte Liste von Elementen zusammen und geben diese als Funktionsergebnis zurück.

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP

Formulare: Datensatz wird nicht gespeichert

Es gibt seit vielen Jahren einen Bug in Access-Formularen, der möglicherweise Entwickler und Benutzer in den Wahnsinn treibt. Dabei geht es darum, dass neue oder geänderte Datensätze beim Schließen des Formulars nicht automatisch gespeichert werden. Stattdessen findet man neue Datensätze einfach nicht in der entsprechenden Tabelle vor und bei vermeintlich geänderten Datensätzen wurde die Änderung nicht übernommen. Dies geschieht, wenn man das Formular auf eine bestimmte Art schließt und Restriktionen in der zugrunde liegenden Tabelle dafür sorgen, dass der Datensatz nicht gespeichert werden kann. Wird zum Beispiel ein Pflichtfeld nicht gefüllt und das Formular geschlossen, verwirft Access den neuen Datensatz einfach, anstatt eine entsprechende Meldung zu liefern. Dieser Beitrag dokumentiert das Fehlverhalten und zeigt, welche Möglichkeiten wir zum Umgehen dieses Problems haben.

Tabelle mit Restriktionen

Voraussetzung für das beschriebene Verhalten ist, dass das Formular eine Tabelle anzeigt, für die es eine Restriktion gibt, die beim Speichern nicht erfüllt wurde. Schauen wir uns zum Beispiel die Tabelle **tblMitglieder** an, bei der wir für die beiden Felder **Vorname** und **Nachname** die Eigenschaft **Eingabe erforderlich** auf den Wert **Ja** eingestellt haben (siehe Bild 1).

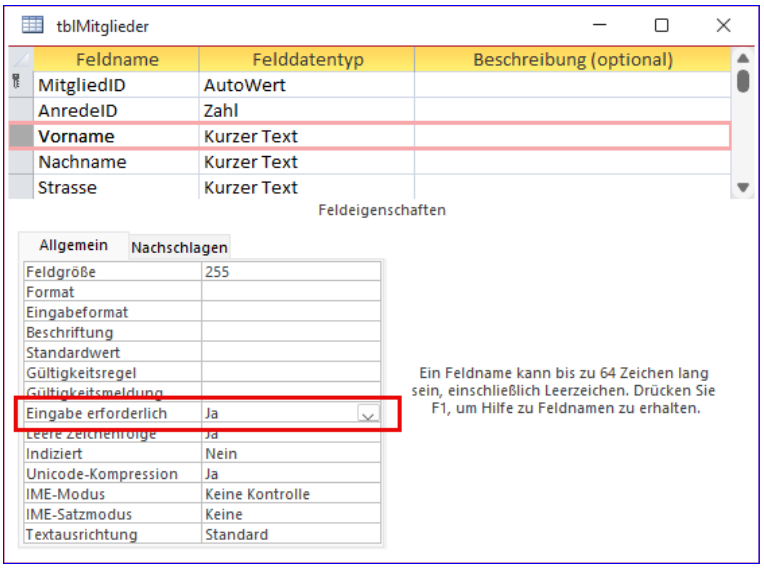


Bild 1: Tabelle mit Restriktion für das Feld **Vorname**

Schon in der Datenblattansicht führt der Versuch, den Datensatz zu speichern, ohne einen Wert in das Feld Vorname einzutragen, zu einer Fehlermeldung (siehe Bild 2). Der Datensatz kann nicht gespeichert werden, ohne dass dieses Feld gefüllt wird.

Tabelle mit Restriktionen im Formular

Schauen wir uns nun das Formular **frmMitgliederDetails** an, das die Daten der Tabelle **tblMitglieder** anzeigt.

Legen wir hier einen neuen Datensatz an, ohne das Feld **Vorname** auszufüllen, und versuchen, das Formular über die **Schließen**-Schaltfläche

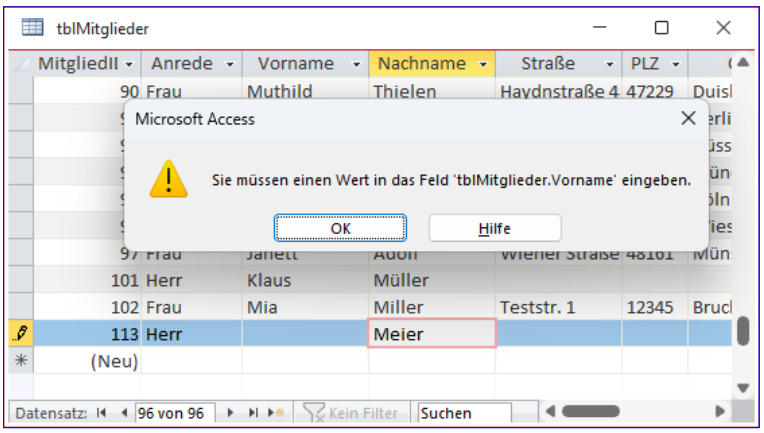


Bild 2: Meldung beim Verletzen der Restriktion

zu schließen, erscheint die gleiche Meldung, die wir bereits aus der Datenblattansicht der Tabelle kennen (siehe Bild 3).

Anschließend zeigt Access noch eine weitere Meldung an, die uns die Möglichkeit bietet, das Formular ohne Speichern der Änderung zu schließen oder die Eingabe fortzusetzen (siehe Bild 4).

Problem beim Schließen mit DoCmd.Close

Was also ist das Problem? Dieses tritt auf, wenn wir eine Schaltfläche wie **cmdOK** dazu nutzen, das Formular mit der folgenden Prozedur zu schließen:

```
Private Sub cmdOK_Click()
    DoCmd.Close acForm, Me.Name
End Sub
```

Das Formular wird geschlossen, ohne das eine Fehlermeldung wegen der Verletzung der Restriktion angezeigt wird – allerdings finden wir anschließend auch keinen neuen Datensatz in der zugrunde liegenden Tabelle. Das Problem wird also durch die Kombination aus **DoCmd.Close** und der Verletzung einer Restriktion ausgelöst.

Auch wenn wir die Methode **DoCmd.Close** von außen aufrufen, wird das Formular ohne Meldung und ohne Anzeige einer Meldung geschlossen – und auch hier wird der Datensatz einfach verworfen:

```
DoCmd.Close acForm,
"frmMitgliederDetails"
```

Immerhin finden wir mittlerweile einen entsprechenden Hinweis auf der Seite zum **DoCmd.Close**-Befehl von Microsoft (siehe Bild 5).

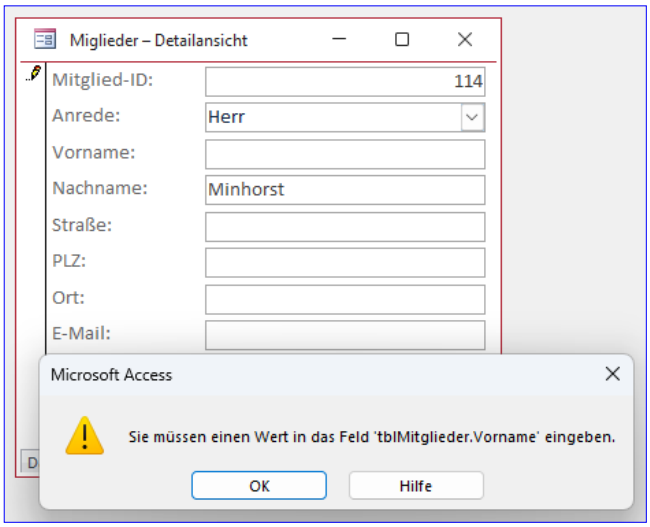


Bild 3: Meldung auch im Formular beim Schließen mit der Taste **X**

Hier wird auch ein Workaround vorgeschlagen. Man soll die Methode **RunCommand** des **Application**-Objekts mit dem Parameter **acCmdSaveRecord** aufrufen, um den Datensatz zu speichern und so den Fehler zu provozieren. Erst danach soll man das Formular mit **DoCmd.Close** schließen:

```
RunCommand acCmdSaveRecord
DoCmd.Close acForm, Me.Name
```

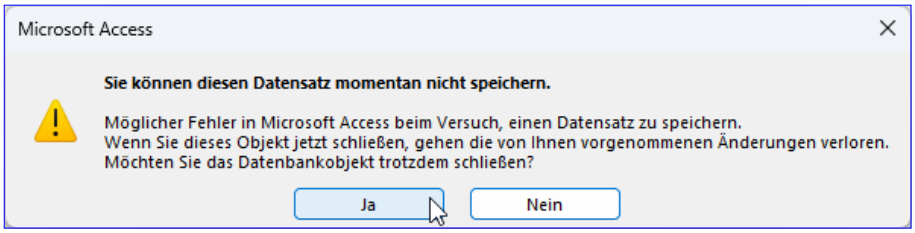


Bild 4: Folgemeldung nach dem Hinweis auf die Verletzung der Restriktion

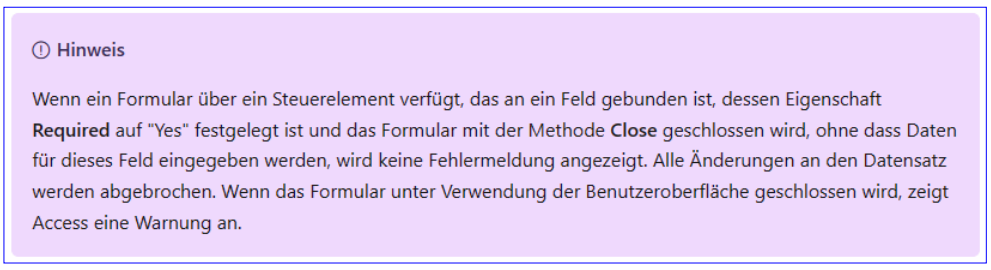


Bild 5: Hinweistext in der Microsoft-Dokumentation

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP

TreeView: Bug durch falsche Einheiten

Wenn man das TreeView-Steuerelement professioneller nutzt, programmiert man unter anderem Ereignisse, die durch das Anklicken von Elementen ausgelöst werden. Damit kann man Ereignisprozeduren auslösen, die beispielsweise Daten zum angeklickten Element in einem Unterformular anzeigen oder man blendet ein Kontextmenü zum jeweils angeklickten Element mit weiteren Optionen ein. Dazu ist es notwendig, zu identifizieren, auf welches Element der Benutzer geklickt hat. Die notwendigen Informationen liefern die Parameter der Prozeduren, das Ermitteln des angeklickten Elements erledigt man mit einer bestimmten Funktion. Seit Kurzem erreichen uns allerdings Meldungen von Lesern, bei denen dies nicht mehr zuverlässig funktioniert: Es werden keine Kontextmenüs mehr angezeigt und auch das Anklicken gelingt nicht mehr wie gewünscht. Interessanterweise tritt das Problem nur bei Verwendung von Office 365 auf. Wir schauen uns in diesem Beitrag an, woher das Problem rührt und wie Sie es lösen können.

Vorbereitung

Für die folgenden Experimente legen wir in einem neuen, leeren Formular einfach ein **TreeView**-Steuerelement namens **ctlTreeView** an.

Dazu wählen wir im Ribbon den Befehl **Formularentwurf|Steuerelemente|ActiveX**-Steuerelemente aus (siehe Bild 1).

Dies öffnet den Dialog **ActiveX-Steuerelement einfügen**, wo wir den Eintrag **Microsoft TreeView Control, version 6.0** auswählen (siehe Bild 2).

Danach sehen wir das neue **TreeView**-Steuerelement im Formular und passen seine Größe entsprechend an (siehe Bild 3). Damit können wir zu den Feinheiten der Referenzierung und Programmierung übergehen.

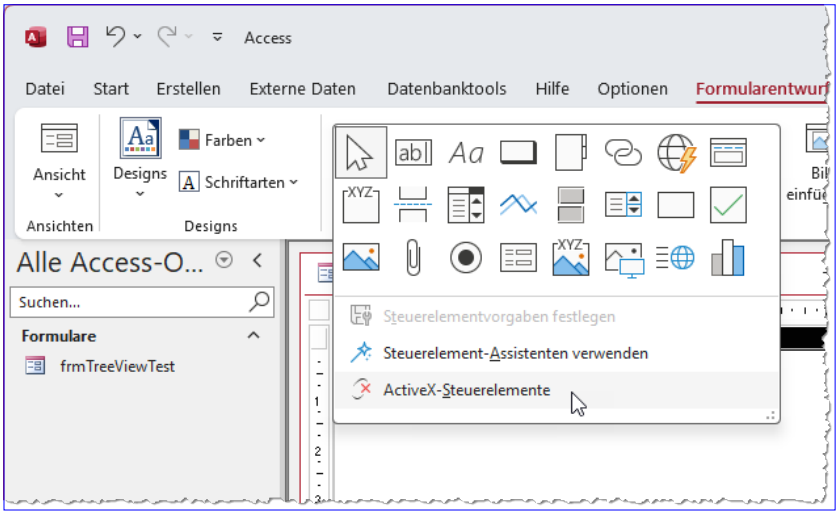


Bild 1: ActiveX-Steuerelement hinzufügen

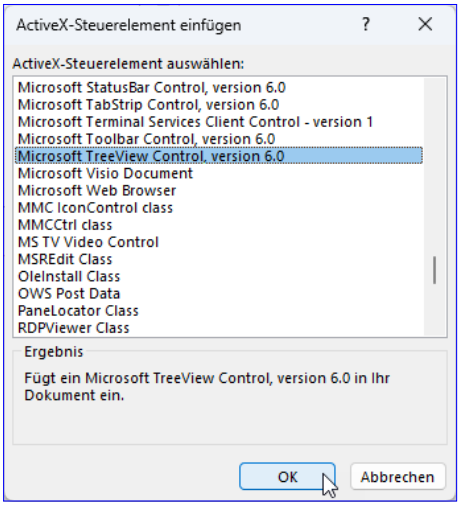


Bild 2: TreeView-Steuerelement selektieren

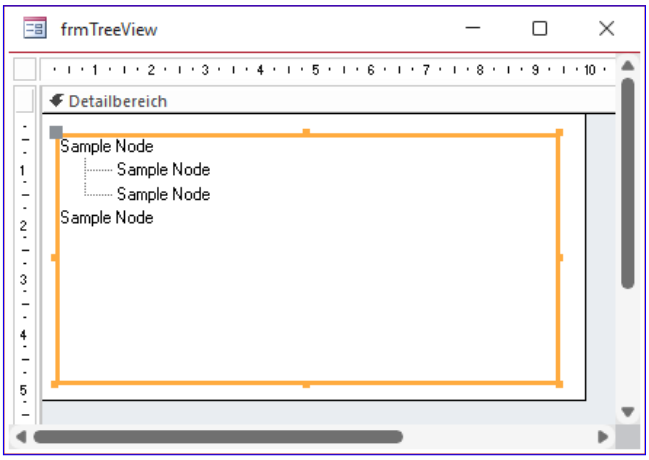


Bild 3: TreeView-Steuerelement nach dem Einbau

Nach dem Einfügen des **TreeView**-Steuerelements passen wir noch seinen Namen auf **ctlTreeView** an.

TreeView als Steuerelement oder als Objektvariable referenzieren

Es gibt zwei Wege, wie man ein **TreeView**-Steuerelement per VBA referenzieren kann. Der erste ist, dieses wie bei den eingebauten Steuerelementen einfach über das Schlüsselwort **Me** in Verbindung mit dem Steuerelementnamen zu referenzieren, hier also **ctlTreeView**. Wenn wir dies im VBA-Editor erledigen und einen weiteren Punkt anfügen, um per IntelliSense auf die Methoden und Eigenschaften des **TreeView**-Steuerelements zuzugreifen, erhalten wir das Ergebnis aus Bild 4.

Hier erkennen wir, dass wir nur die allgemeinen Eigenschaften von Steuerelementen anwählen können, nicht jedoch die speziellen Elemente des TreeView-Steuerelements.

Dazu müssen wir eine Objektvariable anlegen, die wir wie folgt deklarieren:

```
Dim objTreeView As MSCOMCTLLib.TreeView
```

Diese Variable füllen wir wie folgt:

```
Set objTreeView = Me!ctlTreeView.Object
```

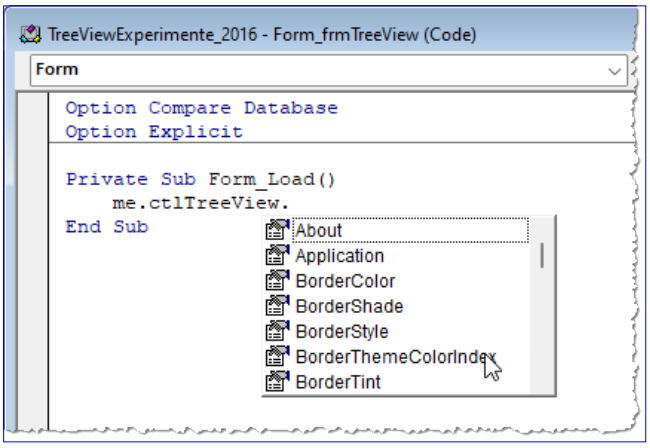


Bild 4: Referenzieren des Steuerelements

Wir können das **TreeView**-Steuerelement nicht direkt über **Me!ctlTreeView** referenzieren, weil wir damit nur die Hülle des Steuerelements referenzieren würden. Diese hat den Datentyp **CustomControl**. Dies können wir belegen, indem wir zuerst den Typ von **ctlTreeView** ausgeben:

```
Debug.Print TypeName(Me!ctlTreeView)
CustomControl
```

Erst über das **Object**-Element erhalten wir den gesuchten Typ:

```
Debug.Print TypeName(Me!ctlTreeView.Object)
TreeView
```

Mit der Referenzierung über die Objektvariable **objTreeView** erhalten wir schließlich auch die **TreeView**-spezifischen Eigenschaften und Methoden per IntelliSense (siehe Bild 5).

Sie sehen: Ein Grund, das **TreeView**-Steuerelement nicht über das als Container verwendete Steuerelement **CustomControl**, sondern über das darin enthaltene **TreeView**-Steuerelement zu referenzieren, erlaubt den Zugriff auf die Eigenschaften und Methoden per IntelliSense.

Verwendung von Ereignissen

Wir wollen exemplarisch das Ereignis **MouseDown** verwenden, weil es das Problem verdeutlicht. Wir können das

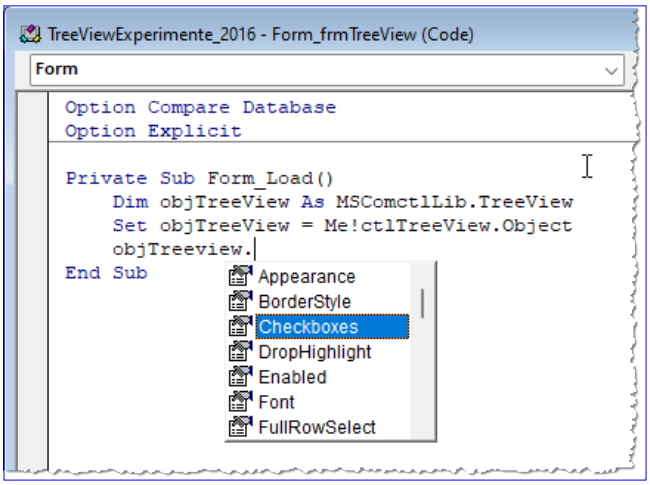


Bild 5: Referenzieren des **TreeView**-Steuerelements per Objektvariable

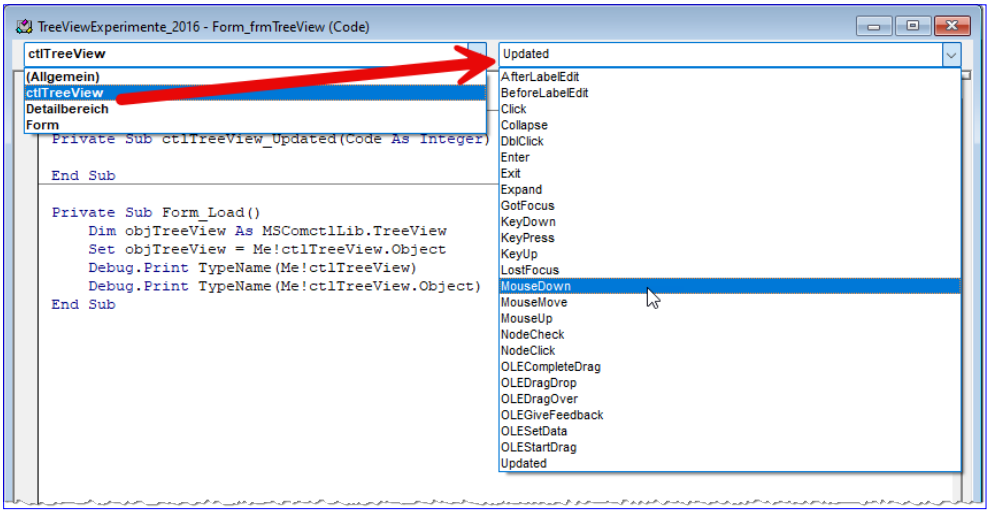


Bild 6: Anlegen des **MouseDown**-Ereignisses für **ctlTreeView**

Ereignis auf zwei Arten in Form einer Ereignisprozedur implementieren. Bei der ersten verwenden wir die Ereignisse des Steuerelements **ctlTreeView**. Um das Ereignis anzulegen, wählen wir im Codefenster des Klassenmoduls des Formulars oben links den Eintrag für das **TreeView**-Steuerelement **ctlTreeView** aus und aus der nun aufgeklappten Liste auf der rechten Seite den Eintrag **MouseDown**. Die automatisch hinzugefügte Ereignisprozedur **ctlTreeView_Updated** können wir wieder löschen (siehe Bild 6).

Die neue Ereignisprozedur hat die folgende Signatur:

```
Private Sub ctlTreeView_MouseDown( _
    ByVal Button As Integer, _
    ByVal Shift As Integer, _
    ByVal x As Long, _
    ByVal y As Long)
```

Button liefert einen Hinweis, ob die linke oder rechte Maustaste gedrückt wurde, **Shift** gibt an, ob eine der Tasten **Alt**, **Strg** oder **Umschalt** beim Betätigen der Maustaste gehalten wurde.

Mit **x** und **y** erhalten wir die Koordinaten, an denen der Mausklick erfolgte. Wie es unter Access üblich ist, werden diese Koordinaten im Format **Twips** geliefert.

Ereignis für die Objektvariable anlegen

Wir können das Ereignis allerdings auch für die oben vorgestellte Objektvariable **objTreeView** anlegen. Dazu müssen wir die Deklaration dieser Variablen nur aus der Prozedur **Form_Load** herausholen und in den allgemeinen Teil des Moduls verschieben (siehe Bild 7). Außerdem fügen wir das Schlüsselwort **WithEvents** hinzu, damit der VBA-Editor weiß, dass wir in diesem Klassenmodul Ereignisse für dieses Element implementieren können wollen.

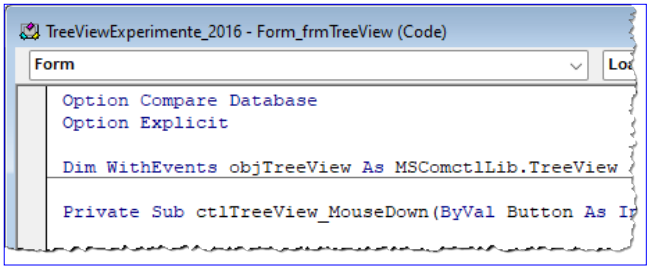


Bild 7: Änderung der Deklaration von **objTreeView**

Damit können wir nun wie zuvor für **ctlTreeView** auch für **objTreeView** die Ereignisse über die Auswahlfelder oben im Codefenster hinzufügen (siehe Bild 8).

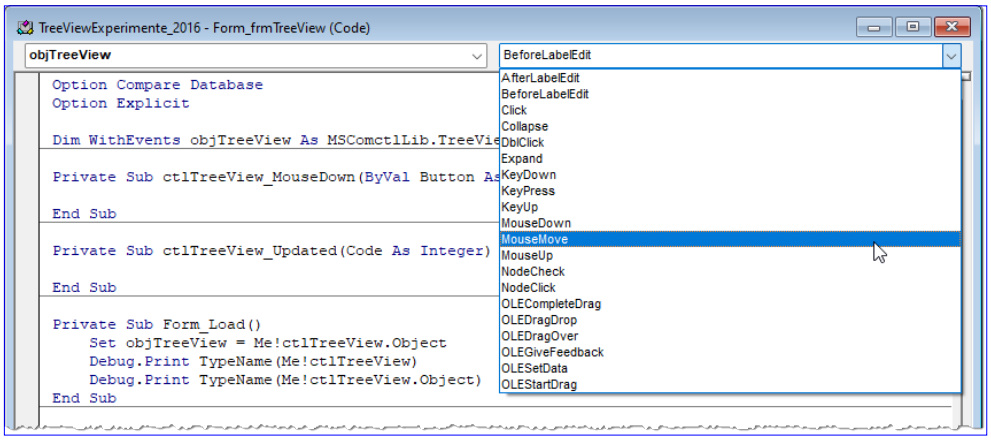


Bild 8: Anlegen des **MouseDown**-Ereignisses für die Objektvariable **objTreeView**

Das erledigen wir wie zuvor und sehen uns dann die Signatur des **MouseDown**-Ereignisses für die Objektvariable **objTreeView** an:

```
Private Sub objTreeView_MouseDown( _
    ByVal Button As Integer, _
    ByVal Shift As Integer, _
    ByVal x As stdole.OLE_XPOS_PIXELS, _
    ByVal y As stdole.OLE_YPOS_PIXELS)
```

Die Signatur ist bezüglich der Anzahl und der Benennung der Elemente identisch, jedoch unterscheidet sich der Datentyp bei den Parametern **x** und **y**. Während wir es zuvor mit **Long**-Werten zu tun hatten, finden wir nun den Datentyp **stdole.OLOE_XPOS_PIXELS** vor. Lassen wir uns innerhalb dieses Ereignisse jedoch einmal den Datentyp mit **Typename** ausgeben, erhalten wir hier ebenfalls **Long** als Datentyp. Allerdings sehen wir, dass der Datentyp die Zeichenkette **PIXELS** enthält, war darauf hindeutet, dass hier nicht Twips wie bei der gleichnamigen Ereignisprozedur für das Steuerelement geliefert werden, sondern Pixel.

Die Frage ist nun: Ist das auch der Fall?

Dazu fügen wir dem Formular vier Textfelder namens **txtCtlX**, **txtCtlY**, **txtObjX** und **txtObjY** hinzu (siehe Bild 9). Die Textfelder füllen wir beim Auslösen des Ereignisses MouseDown in der Ereignisprozedur mit den Werten für die Parameter **x** und **y**:

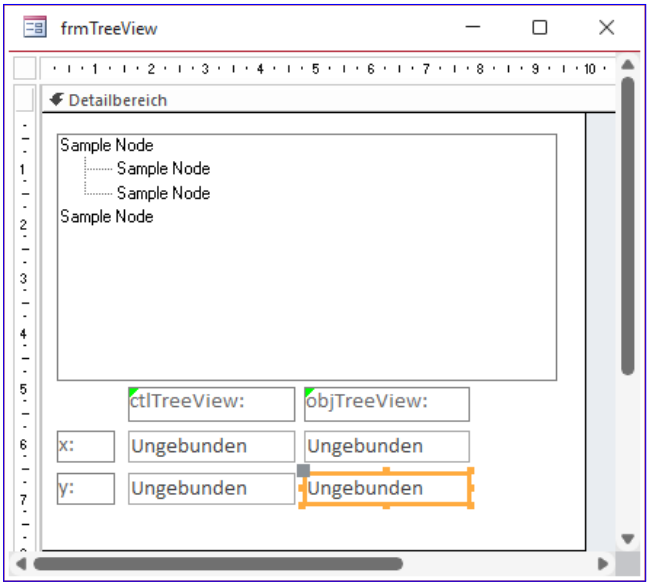


Bild 9: Steuerelemente zum Anzeigen der x- und y-Werte für die beiden Ereignisprozeduren

```
Private Sub ctlTreeView_MouseDown(_
    ByVal Button As Integer, _
        ByVal Shift As Integer, _
        ByVal x As Long, ByVal y As Long)
    Me!txtCtlX = x
    Me!txtCtlY = y
End Sub

Private Sub objTreeView_MouseDown( _
        ByVal Button As Integer, ByVal Shift As Integer, _
        ByVal x As stdole.OLE_XPOS_PIXELS, _
        ByVal y As stdole.OLE_YPOS_PIXELS)
    Me!txtObjX = x
```

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP

Tabellendaten mit Übersicht und Details anzeigen

Eine gute Basis für das Anzeigen und Bearbeiten von Daten ist die folgende Konstellation: Wir verwenden ein Haupt- und Unterformular, um die Daten einer Tabelle in einer Übersichtsansicht anzuzeigen. Dabei können wir die Daten, die im Unterformular angezeigt werden, filtern und sortieren. Außerdem stellen wir Schaltflächen bereit, mit denen wir die Datensätze anlegen, bearbeiten und löschen können. Zum Bearbeiten verwenden wir ein weiteres Formular, das die Daten in einer übersichtlichen Ansicht darstellt. Zusätzlich fügen wir noch den Code hinzu, der für das Interagieren der Formulare untereinander notwendig ist. Diese Konstellation von Formularen können wir für alle möglichen Tabellen verwenden und schaffen so die Basis einer professionellen Anwendung. Anpassungen können schließlich dort vorgenommen werden, wo m:n- oder 1:n-Beziehungen dargestellt werden sollen.

Eine oft verwendete Konstellation aus Formularen zur Darstellung der Daten einer Tabelle sehen wir in Bild 1. Links finden wir ein aus Haupt- und Unterformular bestehendes Formular, das im Unterformular die Daten in der Datenblattansicht anzeigt und im Hauptformular Steuerelemente, um mit den Daten zu arbeiten. Hier können wir nach den Daten im Unterformular suchen, einen neuen Datensatz hinzufügen, einen Datensatz zur Bearbeitung öffnen oder den markierten Datensatz löschen. Ein Doppelklick auf einen der Einträge im Unterformular soll ebenfalls das Bearbeitungsformular öffnen.

Dieses sieht wie das Formular rechts in der Abbildung aus. Es liefert einfach die Felder der zugrunde liegenden Tabelle und bietet zwei Schaltflächen an, mit denen Änderungen übernommen oder verworfen werden können.

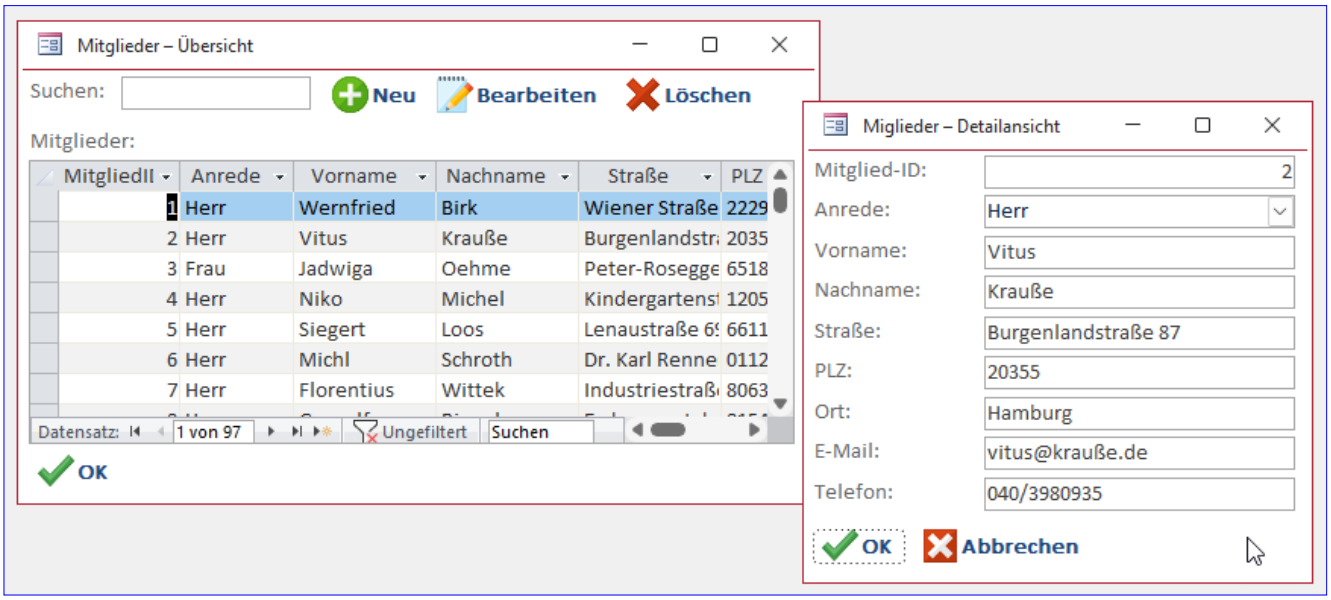


Bild 1: Die gewünschten Formulare

Übersichtsformular programmieren

Die Erstellung dieser Formulare beginnen wir immer mit dem Unterformular des Übersichtsformulars. Warum das? Weil wir dieses, wenn wir es fertiggestellt haben, direkt in das im Anschluss erstellte Hauptformular einbauen können.

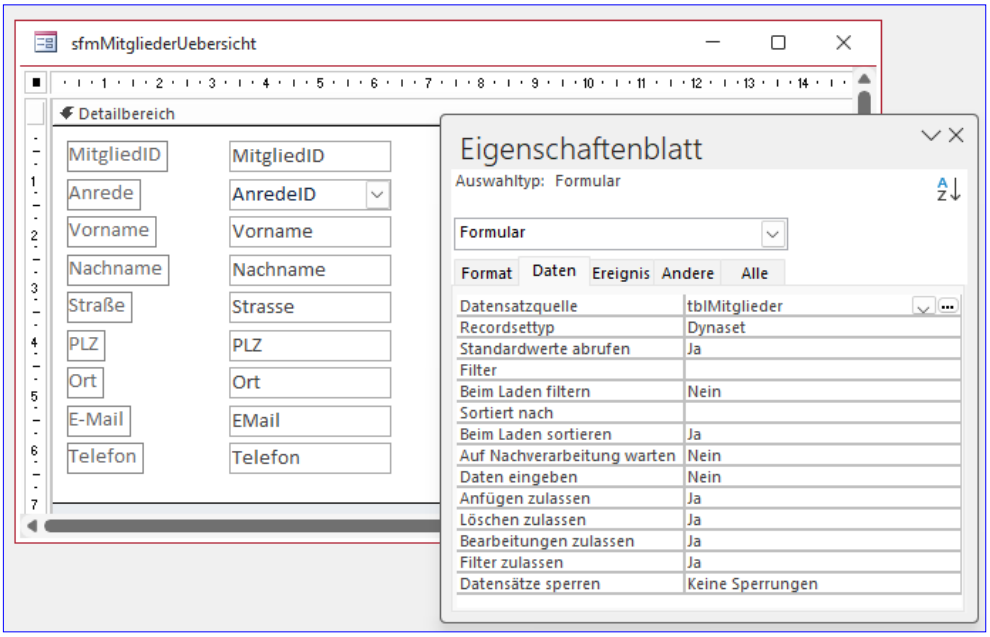


Bild 2: Erstellen des Unterformulars der Übersicht

Das Unterformular erstellen wir als neues, leeres Formular. Diesem weisen wir für die Eigenschaft **Datensatzquelle** die Tabelle mit den anzuzeigenden Daten zu, in diesem Fall **tblMitglieder**.

Dann ziehen wir alle Felder, die das Unterformular anzeigen soll, aus dem Bereich **Feldliste** in den Detailbereich (siehe Bild 2).

Damit das Unterformular seine Daten in der Datenblattansicht präsentiert, stellen wir die Eigenschaft **Standardansicht** auf den Wert **Datenblatt** ein.

Details anzeigen per Doppelklick

Außerdem wollen wir, dass der Benutzer mit einem Doppelklick auf einen der Einträge im Unterformular das Formular mit der Detailansicht öffnen kann. Dazu hinterlegen wir für die Ereigniseigenschaft **Beim Doppelklicken** eines jeden Steuerelements den Wert **[Ereignisprozedur]** und legen durch einen Klick auf die Schaltfläche mit den drei Punkten die entsprechende Ereignisprozedur an (siehe Bild 3). Die hier hinzugefügte Ereignisprozedur füllen wir wie folgt mit einer einzigen Anweisung:

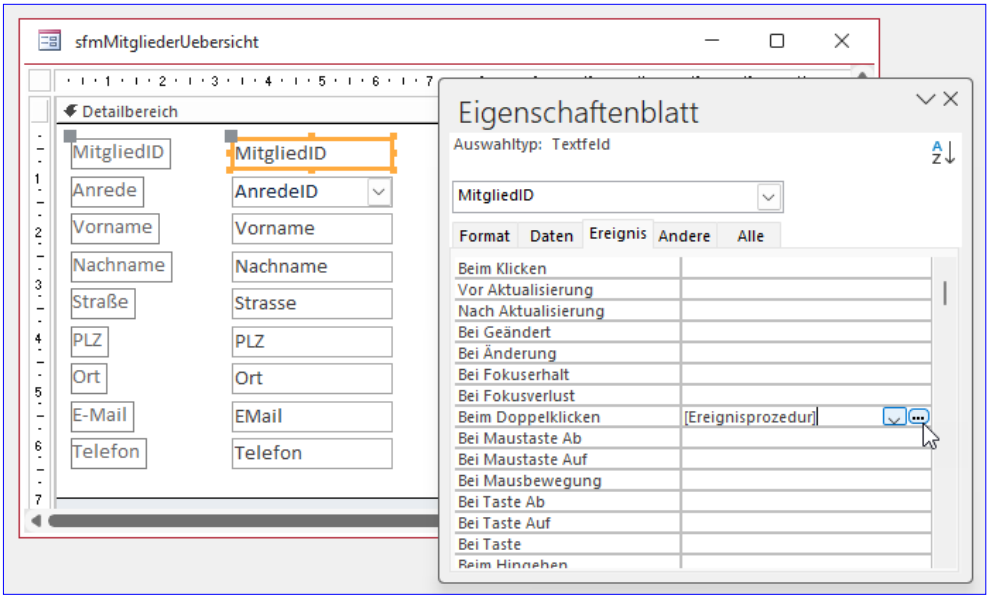


Bild 3: Ereignis **Beim Doppelklicken** implementieren

```
Private Sub EditDetails()
    Dim strForm As String
    strForm = "frmMitgliederDetails"
    DoCmd.OpenForm strForm, WindowMode:=acDialog, WhereCondition:="MitgliedID = " & Me!MitgliedID, DataMode:=acFormEdit
    If IstFormularGeoeffnet(strForm) Then
        Me.Refresh
        DoCmd.Close acForm, Me.Name
    End If
End Sub
```

Listing 1: Öffnen der Detailansicht eines Datensatzes

```
Private Sub MitgliedID_DblClick(Cancel As Integer)
    Call EditDetails
End Sub
```

Auf die gleiche Weise hinterlegen wir eine solche Ereignisprozedur für die übrigen Steuerelemente. Die hier aufgerufene Prozedur **EditDetails** finden wir in Listing 1. Sie öffnet das Detailformular namens **frmMitgliederDetails** mit der **DoCmd.OpenForm**-Methode. Dabei stellt sie den Öffnungsmodus auf einen modalen Dialog ein. Als Datenmodus verwenden wir den Bearbeitungsmodus. Außerdem übergibt die Prozedur eine Bedingung an das Formular, die dafür sorgt, dass es genau den Datensatz anzeigt, der soeben doppelt angeklickt wurde.

Wenn der Benutzer das Formular geschlossen hat, läuft die aufrufende Prozedur weiter und prüft, ob das Formular geschlossen ist. Falls nicht, und das ist der Fall, wenn der Benutzer im Detailformular die **OK**-Schaltfläche gedrückt hat, aktualisiert die Prozedur seine eigene Datenquelle und schließt das Detailformular mit **DoCmd.Close** endgültig.

Hat der Benutzer hingegen die **Abbrechen**-Schaltfläche gedrückt, wodurch die Änderungen verworfen und das Formular geschlossen werden soll, wird das Formular direkt richtig geschlossen. In diesem Fall müssen im aufrufenden Formular keine Daten aktualisiert werden.

Danach können wir das Unterformular schließen und es unter dem Namen **sfmMitgliederUebersicht** speichern.

Hauptformular programmieren

Das Hauptformular öffnen wir ebenfalls als neues, leeres Formular. Dieses benötigte keine Datensatzquelle, da die Daten ja bereits im Unterformular angezeigt werden.

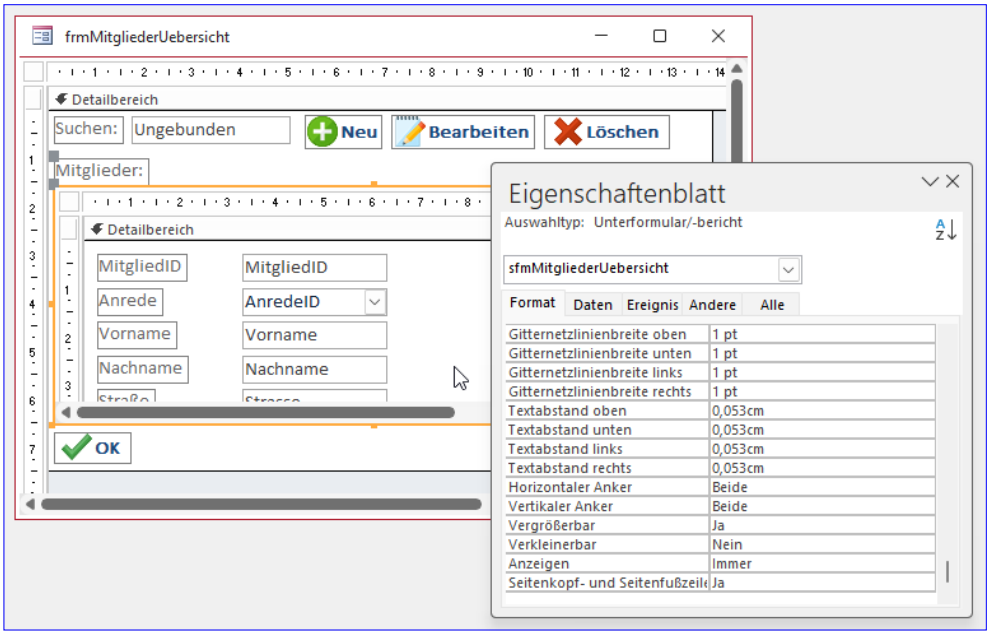


Bild 4: Das Unterformular im Hauptformular

Stattdessen fügen wir vier Schaltflächen namens **cmdAdd**, **cmdEdit**, **cmdDelete** und **cmdOK** hinzu. Außerdem legen wir ein Textfeld namens **txtSearch** an.

Dann ziehen wir das Unterformular **sfmMitgliederUebersicht** aus dem Navigationsbereich in das Hauptformular und positionieren alle Elemente ungefähr wie in Bild 4.

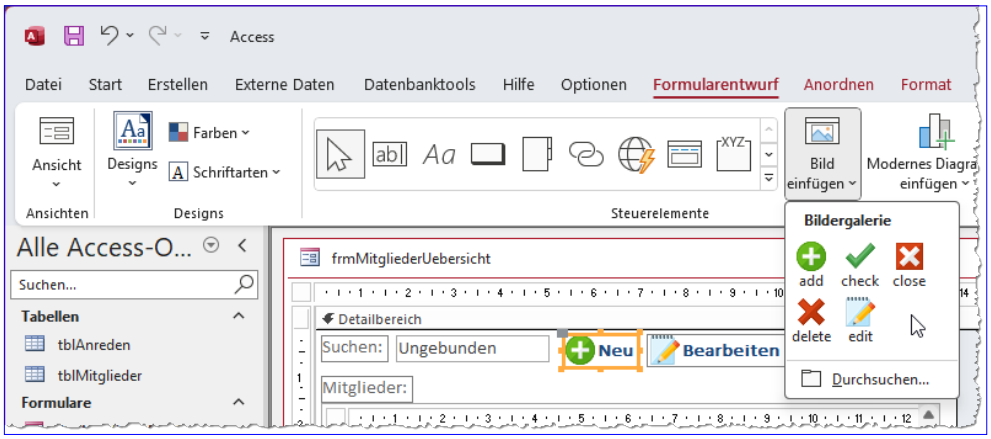


Bild 5: Hinzufügen von Icons zu Schaltflächen

Flexible Größe des Übersichtsformulars

In der aktuellen Ansicht ist das Formular recht klein gehalten. Der Benutzer soll es aber vergrößern können und dabei sollen sich auch bestimmte Elemente vergrößern beziehungsweise verschoben werden. In diesem Fall ist das Unterformular **sfmMitgliederUebersicht** das Element, das sowohl vertikal als auch horizontal angepasst werden soll, wenn der Benutzer das Hauptformular vergrößert oder verkleinert. Deshalb stellen wir die Eigenschaften **Horizontaler Anker** und **Vertikaler Anker** auf den Wert **Beide** ein. Das wirkt sich automatisch auch auf das Bezeichnungsfeld des Unterformular-Steuerelements aus, für das die Eigenschaften nun auf **Rechts** und **Unten** eingestellt sind. Diese legen wir wieder auf **Links** und **Oben** fest.

Da sich das Unterformular-Steuerelement beim Vergrößern der Höhe nach unten ausdehnt, müssen wir die darunter liegenden Steuerelemente ebenfalls nach unten verschieben. Deshalb stellen wir für die Schaltfläche **cmdOK** den Wert der Eigenschaft **Vertikaler Anker** auf **Unten** ein.

Design der Schaltflächen

Die Schaltflächen haben wir ein wenig angepasst, da das Standarddesign etwas altbacken daherkommt. Dazu haben wir zunächst Bilder hinzugefügt. Das gelingt am schnellsten, indem man im Entwurf die jeweilige Schaltfläche anklickt und den Befehl **Bild einfügen** aus dem Ribbon-Tab **Formularentwurf** auswählt. Danach können wir das gewünschte Icon, am besten im PNG-Format, selektieren und es wird direkt der Schaltfläche zugeordnet. Einmal zur Anwendung hinzugefügte Schaltflächen sind danach über die Bildergalerie verfügbar (siehe Bild 5).

Damit der Text rechts vom Icon erscheint, stellen wie die Eigenschaft **Anordnung der Bildbeschriftung** auf **Rechts** ein.

Den Rahmen haben wir durch Einstellen der Eigenschaft **Rahmenart** auf den Wert **Transparent** ausgeblendet. Für die Schrift haben wir eine andere Farbe gewählt und sie zusätzlich fett gedruckt. Außerdem haben wir auch den Hintergrund durch Einstellen der Eigenschaft **Hintergrundart** auf **Transparent** ausgeblendet.

Ereignisprozeduren im Übersichtsformular

Nun fügen wir die Ereignisprozeduren für die Schaltflächen im Hauptformular hinzu. Wie beginnen mit der einfachsten Schaltfläche **cmdOK**. Diese löst die folgende Prozedur aus, die einfach das Formular schließt:

```
Private Sub cmdOK_Click()
    DoCmd.Close acForm, Me.Name
End Sub
```

Suchfunktion programmieren

Damit kommen wir zur Suchfunktion. Das Suchfeld **txtSearch** soll auf die Eingabe eines jeden Zeichens reagieren und die angezeigten Datensätze im Unterformular filtern. Dazu reicht es nicht aus, mit dem Ereignis **Nach Aktualisierung** auf das Betätigen der Eingabetaste oder das Verlassen des Steuerelements zu reagieren. Wir verwenden hier das Ereignis **Bei Änderung**, das bei der Eingabe jedes einzelnen Zeichens und auch beim Löschen von Zeichen ausgelöst wird.

```
Private Sub txtSearch_Change()
    Dim strSearch As String
    Dim strFilter As String
    strSearch = Me!txtSearch.Text
    If Not Len(strSearch) = 0 Then
        strFilter = strFilter & "OR Vorname LIKE '*" & strSearch & "*'"
        strFilter = strFilter & "OR Nachname LIKE '*" & strSearch & "*'"
        strFilter = strFilter & "OR Strasse LIKE '*" & strSearch & "*'"
        strFilter = strFilter & "OR PLZ LIKE '*" & strSearch & "*'"
        strFilter = strFilter & "OR Ort LIKE '*" & strSearch & "*'"
        strFilter = strFilter & "OR EMail LIKE '*" & strSearch & "*'"
        strFilter = strFilter & "OR Telefon LIKE '*" & strSearch & "*'"
        If Not Len(strFilter) = 0 Then
            strFilter = Mid(strFilter, 4)
        End If
        Me!sfmMitgliederUebersicht.Form.Filter = strFilter
        Me!sfmMitgliederUebersicht.Form.FilterOn = True
    Else
        Me!sfmMitgliederUebersicht.Form.FilterOn = False
    End If
End Sub
```

Listing 2: Code für die Suchfunktion

Dafür hinterlegen wir die Prozedur aus Listing 2. Hier lesen wir zuerst den aktuellen Text aus dem Textfeld **txtSearch** ein. Es reicht nicht, einfach mit **Me!txtSearch** oder **Me!txtSearch.Value** den Wert des Textfeldes auszulesen, denn dieser ändert sich erst mit dem Speichern des Inhalts beispielsweise durch Verlassen.

Stattdessen verwenden wir die Eigenschaft **Text**, die den tatsächlich angezeigten Text liefert. Den damit ermittelten Wert tragen wir in die Variable **strSearch** ein.

Erst wenn die Länge dieser Zeichenkette größer als **0** ist, stellen wir einen Filterausdruck zusammen. Bei einer leeren Zeichenketten deaktivieren wir den Filter für das Unterformular einfach mit **FilterOn = False**, sodass dieses alle Datensätze der Datenquelle anzeigt.

Finden wir jedoch einen Text in **strSearch**, beginnen wir mit dem Zusammenstellen des Filterausdrucks, den wir in der Variablen **strFilter** verwalten. In diesem Beispiel wollen wir die Felder **Vorname**, **Nachname**, **Strasse**, **PLZ**, **Ort**, **EMail** und **Telefon** nach dem Suchbegriff durchsuchen und alle Datensätze zurückliefern, in denen der Suchbegriff in irgendeinem der Felder auftritt.

Deshalb verbinden wir den Suchausdruck für das jeweilige Feld, also beispielsweise **Vorname LIKE '*Suchbegriff*'**, mit dem **OR**-Operator. Die Sternchen verwenden wir, damit der Suchbegriff an beliebiger Stelle vorkommen kann.

Am Ende erhalten wir ein Kriterium, das alle gewünschten Felder nach dem angegebenen Suchbegriff durchsucht.

Diesen weisen wir schließlich der Eigenschaft **Filter** des mit der **Form**-Eigenschaft des Unterformular-Steuerelements referenzierten Formulars zu. Mit **FilterOn = True** stellen wir dann den Filter scharf.

Dies führt dazu, dass die Ergebnismenge tatsächlich nach der Eingabe eines jeden Zeichens angepasst wird (siehe Bild 6).

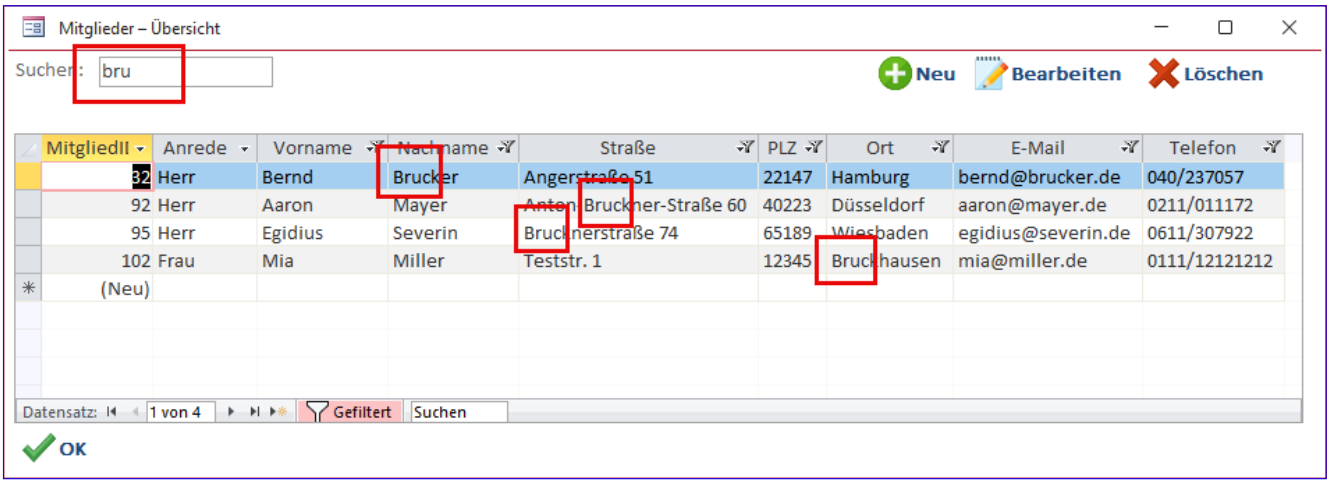


Bild 6: Die Suchfunktion in Aktion

Neuen Datensatz hinzufügen

Zum Hinzufügen eines neuen Datensatzes soll der Benutzer die Schaltfläche **cmdAdd** mit der Beschriftung **Neu** betätigen.

Die dadurch ausgelöste Prozedur (siehe Listing 3) trägt den Namen des Detailformulars in die Variable **strForm** ein. Warum verwenden wir hier eine Variable? Weil der Name des Formulars gleich an mehreren Stellen verwendet wird. Sollten wir diesen Namen einmal ändern oder gar den hier verwendeten Code noch in einem anderen Formular verwenden, brauchen wir den Formularnamen nur an einer Stelle zu ändern.

In diesem Fall nutzen wir die Variable, um das Formular **frmMitgliederDetails** mit der **DoCmd.OpenForm**-Methode zu öffnen. Dies erledigen wir mit einem modalen Dialog (**WindowMode:=acDialog**), damit der aufrufende Code angehalten wird und erst nach dem Schließen oder Ausblenden des geöffneten Formulars weiterläuft.

Außerdem stellen wir den Datenmodus zum Anlegen eines neuen Datensatzes ein (**DataMode:=acFormAdd**).

Dadurch erscheint das Formular **frmMitgliederDetails** wie in Bild 7 mit einem neuen, leeren Datensatz. Nachdem wir dort einen neuen Datensatz eingetragen haben und die **OK**-Schaltfläche betätigen, wird das Formular mit **Me.Visible = False** ausgeblendet und der aufrufende Code wird fortgesetzt.

Dieser stellt mit der Hilfsfunktion **IstFormularGeoeffnet** fest, dass das Formular noch geöffnet ist. Daraufhin liest sie den Primärschlüsselwert des neu angelegten Datensatzes in die Variable **lngID** ein und schließt das Detailfor-

```
Private Sub cmdAdd_Click()
    Dim strForm As String
    Dim lngID As Long
    strForm = "frmMitgliederDetails"
    DoCmd.OpenForm strForm, WindowMode:=acDialog, DataMode:=acFormAdd
    If IstFormularGeoeffnet(strForm) Then
        lngID = Forms(strForm)!MitgliedID
        DoCmd.Close acForm, strForm
        Me!sfmMitgliederUebersicht.Form.Requery
        Me!sfmMitgliederUebersicht.Form.Recordset.FindFirst "MitgliedID = " & lngID
    End If
End Sub
```

Listing 3: Code zum Anlegen eines neuen Datensatzes

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP

SQL Server: Von der Abfrage zur Stored Procedure

Die einfache Migration der Tabellen einer Datenbank von Access zum SQL Server ist meist schnell erledigt. Der SQL Server Migration Assistant (SSMA) leistet gute Arbeit und schnell landen alle Tabellen in der SQL Server-Datenbank – samt Erstellung entsprechender Tabellenverknüpfungen im Access-Frontend. Damit lässt sich erst einmal arbeiten, da die Verknüpfungen Lesen und Schreiben der Daten wie gewohnt zulassen. Früher oder später wird man jedoch auf Abfragen im Access-Frontend stoßen, die schlicht zu langsam sind. Und hier kann der SQL Server sein wahres Potential ausspielen: Zum Beispiel, indem wir dort eine Stored Procedure (Gespeicherte Abfrage) anlegen, welche die Abfrage für uns direkt auf dem SQL Server ausführt, was viel schneller gehen wird. Und genau das ist Thema dieses Beitrags: Wie bekommen wir eine mehr oder weniger komplizierte Abfrage als Stored Procedure zum SQL Server und rufen diese von der Access-Datenbank aus auf?

Der SQL Server hat gegenüber einer Access-Datenbank eine besondere Stärke: Im Vergleich zu der Fileserver-Datenbank von Access kann der SQL Server Abfragen direkt auf dem Server ausführen und nur die Daten ausliefern, die tatsächlich benötigt werden.

Bei Verwendung einer aufgeteilten Access-Datenbank werden immer alle Daten auf den anfragenden Rechner übertragen und dort ausgewertet. Was schneller geht, liegt auf der Hand, vor allem, je mehr Daten von der Abfrage betroffen sind.

Deshalb schauen wir uns in diesem Beitrag im Detail an, wie wir eine reine Access-Abfrage zum SQL Server transferieren und dort eine Stored Procedure daraus erzeugen.

Voraussetzungen

Damit wir in diesem Artikel reibungslos starten können, setzen wir voraus, dass die grundlegenden Migrationsarbeiten schon erledigt sind – also dass die Tabellen der Datenbank bereits auf dem SQL Server liegen und vom Frontend aus per ODBC-Tabellenverknüpfung gelesen und geschrieben werden können.

Vor- und Nachteile von Stored Procedures

Kurz die wichtigsten Gründe für oder gegen gespeicherte Prozeduren: Gespeicherte Prozeduren können genau wie Access-Abfragen mit Parametern und allen anderen dort genutzten Funktionen verwendet werden und sie sind sehr schnell. Der Preis, den man bei Verwendung gespeicherter Prozeduren bezahlt, ist die fehlende Möglichkeit zum Ändern der Daten. Wenn wir eine gespeicherte Prozedur zum Lesen von Daten nutzen, um eine Access-Abfrage zu ersetzen, die auf Basis von per ODBC verknüpften Tabellen funktioniert, können wir die gelieferten Daten nicht direkt ändern beziehungsweise neue Datensätze anlegen oder vorhandene Datensätze löschen.

Das ist aber in vielen Fällen auch nicht nötig – wir können mit gespeicherten Prozeduren erst einmal Daten etwa für Übersichten mit allen Datensätzen einer Tabelle holen, diese in einem Formular anzeigen und in einem weiteren Formular den zu bearbeitenden Datensatz erneut einlesen – diesmal auf Basis der per ODBC verknüpften Tabelle.

Beispielabfrage aus der Praxis

Anlass für diesen Beitrag war eine Abfrage, mit der ich die Kunden ermittle, die in einem bestimmten Zeitraum ein

oder mehrere bestimmte Produkte erworben haben und für die ich noch weitere Kriterien festlegen wollte.

Die Abfrage habe ich bisher in Access auf die ODBC-Tabellenverknüpfungen angewendet. Die Abfrage lautet beispielsweise:

```
SELECT DISTINCT KundeID, AnredeID, Nachname, Vorname,
EMail, Land, Land_Rechnung
FROM tblKunden
WHERE KundeID IN (
    SELECT tblAbonnements.KundeID
    FROM tblAbonnements
    WHERE tblAbonnements.ProduktID IN (146)
    AND Startdatum >= #2024/01/01 00:00:00#
    AND Startdatum <= #2024/02/01 00:00:00#)
AND KundeID NOT IN (
    SELECT tblAbonnements.KundeID
    FROM tblAbonnements
    WHERE tblAbonnements.ProduktID IN (96, 97)
AND tblKunden.NLAbgemeldetAm IS NULL
AND (
    tblKunden.Land = 'Deutschland'
    OR tblKunden.Land_Rechnung = 'Deutschland'
)
```

Ich benötige also alle Kunden, denen über die Tabelle **tblAbonnements** das Abonnement mit der Nummer **146** zugeteilt wurde und deren Startdatum in einem bestimmten Datumsbereich liegt.

Zusätzlich soll der Kunde nicht die Produkte **96** und **97** gebucht haben. Außerdem sollte der Kunde sich noch nicht vom Newsletter abgemeldet haben und entweder die Liefer- oder die Rechnungsadresse sollte in Deutschland liegen.

Diese Abfrage wird zur Laufzeit per VBA zusammengestellt. Die notwendigen Parameter ermittle ich mit dem Formular aus Bild 1. Hier lasse ich mir zuerst alle Produkte anzeigen, die einen bestimmten Namen haben, übertrage dieses Produkt in die Liste oben rechts und füge gegebenenfalls noch Produkte in die Liste der nicht bestellten Produkte hinzu. Darunter gebe ich das Bestelldatum ein und das Land des Kunden. Außerdem kann ich noch angeben, ob Kunden ausgeschlossen werden sollen, die den Newsletter abgemeldet haben.

Diese Parameter unter VBA zu einer funktionierenden Access-Abfrage zusammenzustellen ist eine Fleißarbeit, aber für jeden Programmierer zu bewerkstelligen. Der Code enthält eine Reihe von **If...Then**-Bedingungen, die prüfen, ob eine bestimmte Option einen Wert enthält und fügt die entsprechenden Kriterien zur Abfrage hinzu.

Der Weg zur Stored Procedure

Um diese Abfrage in eine gespeicherte Prozedur umzuwandeln und diese zu nutzen, sind einige Schritte nötig:

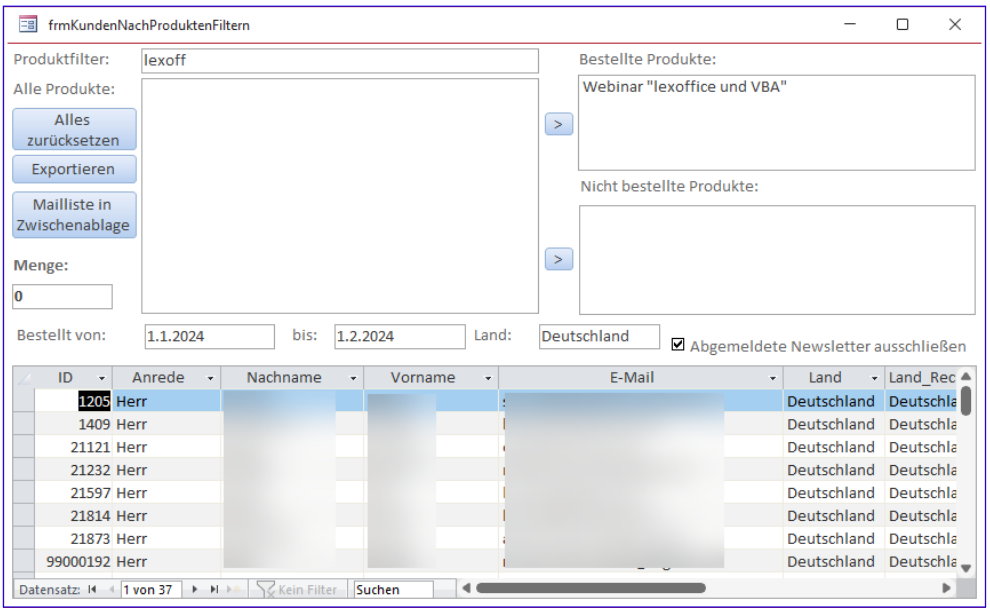


Bild 1: Formular zum Zusammenstellen der Abfrage

- Übertragen einer Version der Abfrage mit möglichst allen Kriterien in die Zwischenablage
- Einfügen des SQL-Codes in eine Stored Procedure
- Beheben der ersten auftretenden Syntaxfehler
- Ausprobieren, ob die Performance tatsächlich besser ist als in der Access-Abfrage
- Schrittweises Einführen von Parametern und Aufruf mit oder ohne die jeweiligen Parameter
- Wenn alle Parameter abgebildet sind, den Aufruf der gespeicherten Prozedur in der Access-Datenbank realisieren.

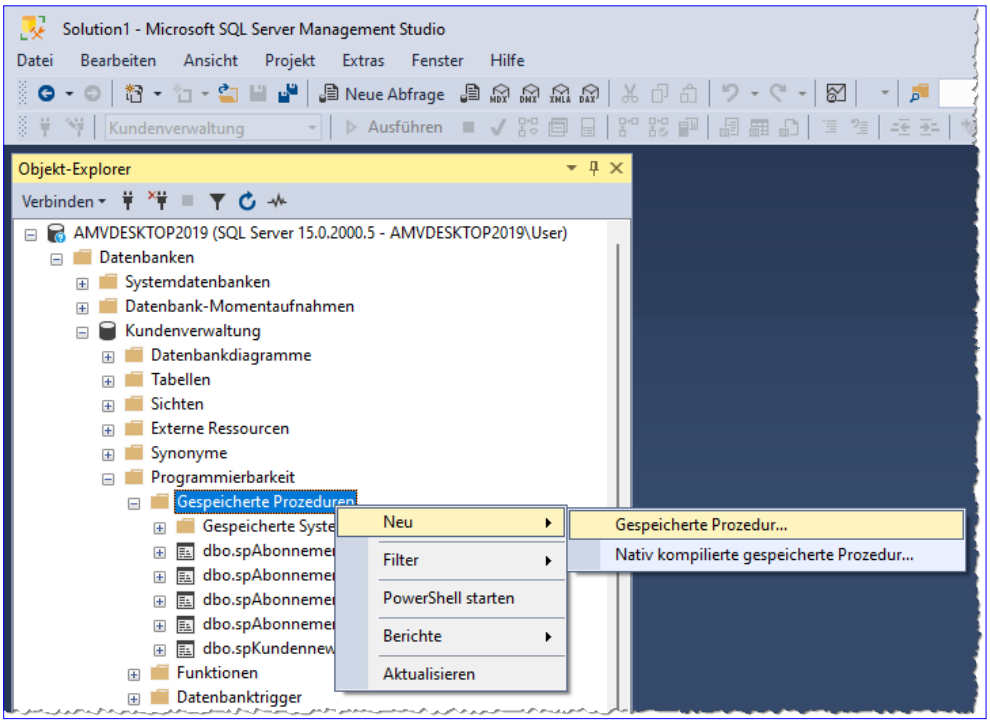


Bild 2: Hinzufügen einer neuen gespeicherten Prozedur

Anlegen der Stored Procedure

Die gespeicherte Prozedur legen wir im SQL Server Management Studio über den Kontextmenü-Befehl **Neu|Gespeicherte Prozedur...** des Eintrags **Programmierbarkeit|Gespeicherte Prozeduren** der jeweiligen Datenbank (siehe Bild 2).

Damit erhalten wir eine Vorlage für eine neue gespeicherte Prozedur (siehe Bild 3, hier mit bereits entfernten Kommentaren). Was wir hier sehen, ist übrigens nicht die gespeicherte Prozedur selbst, sondern der Code, mit dem wir diese anlegen.

Die gespeicherte Prozedur selbst bekommen wir nie zu sehen – wir können uns später nur den Code ausgeben lassen, mit dem wir die gespeicherte Prozedur erneut anlegen oder ändern können.

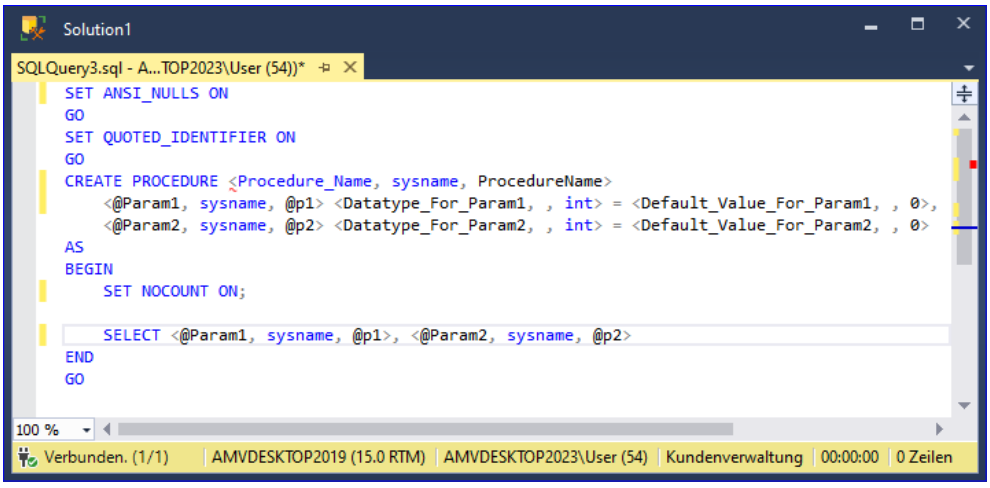


Bild 3: Vorlage für eine neue gespeicherte Prozedur

Haben wir den Code zum Erstellen der gespeicherten Prozedur einmal ausgeführt und wollen diese nun ändern, müssen wir den Befehl **CREATE PROCEDURE** durch **ALTER PROCEDURE** ändern.

Doch eins nach dem anderen. Erst einmal ändern wir den Namen der gespeicherten Prozedur, indem wir **<Procedure_Name, sysname, ProcedureName>** durch unseren Namen ersetzen – zum Beispiel **dbo.spGetCustomersWithSpecificProducts**.

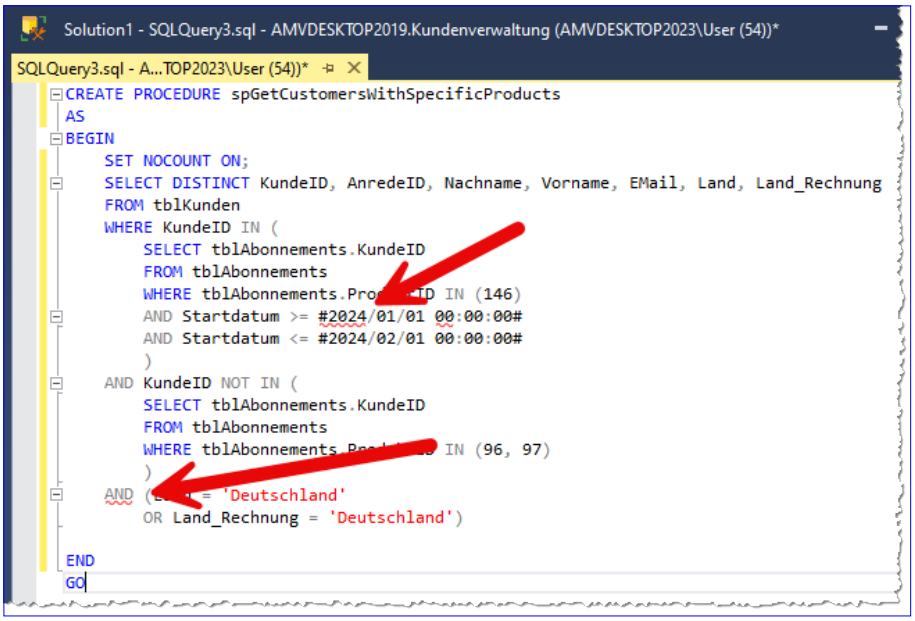


Bild 4: Fehler im ersten Entwurf

Die Parameter löschen wir zuerst einmal, dann fügen wir statt der vorhandenen **SELECT**-Anweisung unsere Abfrage aus der Access-Datenbank ein. Hier erhalten wir gleich markierte Stellen mit Fehlern (siehe Bild 4).

Ich habe mich erst um den zweiten Fehler kümmern wollen, stellte aber dann fest, dass es sich nur um einen Folgefehler des ersten Fehlers handelte. Es empfiehlt sich also, Fehler in T-SQL von oben nach unten zu korrigieren, um solche Probleme zu verhindern.

Das Datum weist offensichtlich ein ungültiges Format auf. Wenn wir es in **'2024-01-01'** und **'2024-02-01'** ändern, verschwinden alle Fehler.

Erstmaliges Erstellen der gespeicherten Prozedur

Nun erstellen wir die Prozedur durch Betätigen der Taste **F5** oder mit dem Menübefehl **Abfrage|Ausführen**.

Dies liefert die Meldung, dass die Befehle erfolgreich ausgeführt wurden und nach einer Aktualisierung des Punktes **Gespeicherte Prozeduren** im Objekt-Explorer finden wir die neue gespeicherte Prozedur dort vor.

Ausführen der gespeicherten Prozedur

Um die gespeicherte Prozedur auszuführen, können wir ein neues Abfragefenster öffnen oder wir geben den Aufruf der Abfrage einfach in das bestehende Abfragefenster ein, in dem wir die gespeicherte Prozedur erstellt haben. Der Aufruf enthält den Befehl **EXEC** und den Namen der auszuführenden gespeicherten Prozedur:

```
EXEC spGetCustomersWithSpecificProducts
```

Wenn wir diesen Aufruf im gleichen Abfragefenster wir die **CREATE PROCEDURE** anlegen, müssen wir diesen markieren und dann die Taste **F5** betätigen. Wenn wir zuvor nicht markieren, werden alle Befehle im Abfragefenster ausgeführt, also auch die **CREATE**-Methode. Dies führt zu einem Fehler, weil wir die gespeicherte Prozedur bereits erstellt haben.

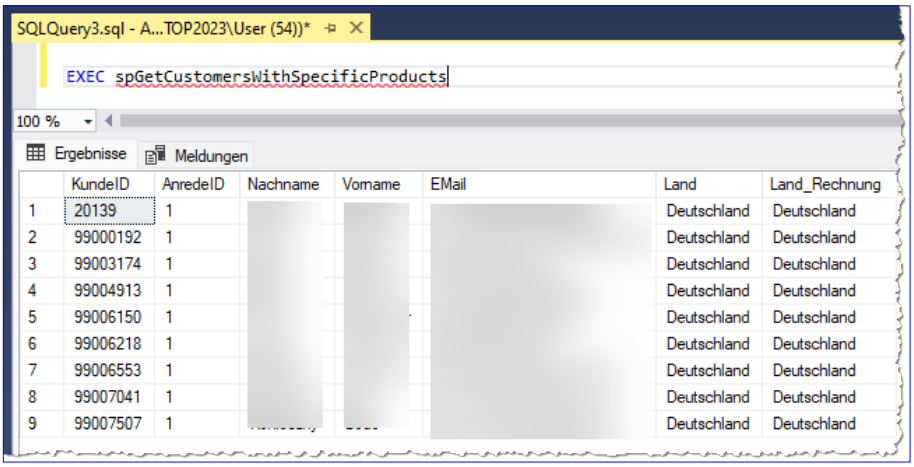
Führen wir die gespeicherte Prozedur jedoch separat aus, erhalten wir die gewünschten Datensätze (siehe Bild 5).

Anpassen der gespeicherten Prozedur

Wenn die die gespeicherte Prozedur nun anpassen wollen, können wir direkt in dem Abfragefenster bleiben, in dem

wir auch den **CREATE PROCEDURE**-Befehl aufgerufen haben. Diesen ersetzen wir einfach durch den **ALTER PROCEDURE**-Befehl. Damit ändern wir die bestehende gespeicherte Prozedur wie in der Anweisung angegeben.

Wenn wir nach dem Ändern immer gleich testen wollen, wie das aktuelle Ergebnis aussieht, können wir die beiden Anweisungen auch direkt untereinander platzieren.



	KundeID	AnredeID	Nachname	Vorname	EMail	Land	Land_Rechnung
1	20139	1				Deutschland	Deutschland
2	99000192	1				Deutschland	Deutschland
3	99003174	1				Deutschland	Deutschland
4	99004913	1				Deutschland	Deutschland
5	99006150	1				Deutschland	Deutschland
6	99006218	1				Deutschland	Deutschland
7	99006553	1				Deutschland	Deutschland
8	99007041	1				Deutschland	Deutschland
9	99007507	1				Deutschland	Deutschland

Bild 5: Abfrageergebnis

Wir erstellen dann zuerst die neue Version der Abfrage und führen diese dann direkt aus.

Tipp: IntelliSense-Cache aktualisieren

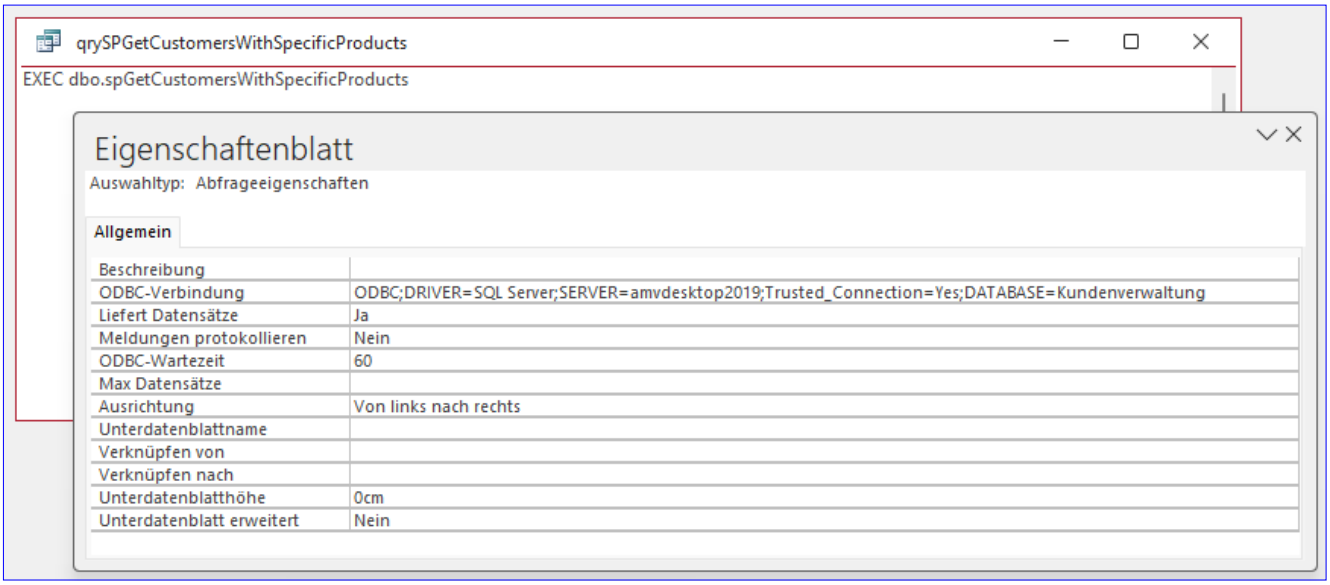
Manchmal markiert der SQL Server den Namen einer bereits angelegten gespeicherten Prozedur oder von anderen Elementen. Dann ist möglicherweise der IntelliSense-Cache noch nicht aktualisiert. Das können wir auf zwei Arten nachholen:

- Tastenkombination **Strg + Umschalt + R** drücken
- Menübefehl **Bearbeiten|IntelliSense|Lokalen Cache aktualisieren** betätigen

Abfrage von Access aus ausführen

Um die Abfrage in der jetzigen Form auszuführen, also ohne Parameter, führen wir die folgenden Schritte aus:

- Erstellen einer neuen Abfrage in der Entwurfsansicht
- Betätigen des Ribbonbefehls **Abfrageentwurf|Abfragetyp|Pass-Through**



Beschreibung	
ODBC-Verbindung	ODBC;DRIVER=SQL Server;SERVER=amvdesktop2019;Trusted_Connection=Yes;DATABASE=Kundenverwaltung
Liefert Datensätze	Ja
Meldungen protokollieren	Nein
ODBC-Wartezeit	60
Max Datensätze	
Ausrichtung	Von links nach rechts
Unterdatenblattname	
Verknüpfen von	
Verknüpfen nach	
Unterdatenblatthöhe	0cm
Unterdatenblatt erweitert	Nein

Bild 6: Abfrageentwurf für den Zugriff auf eine gespeicherte Prozedur

- Einfügen des Aufrufs der Abfrage, also **EXEC dbo.spGetCustomersWithSpecificProducts**
- Eintragen der Verbindungszeichenfolge für die Eigenschaft ODBC-Verbindung.

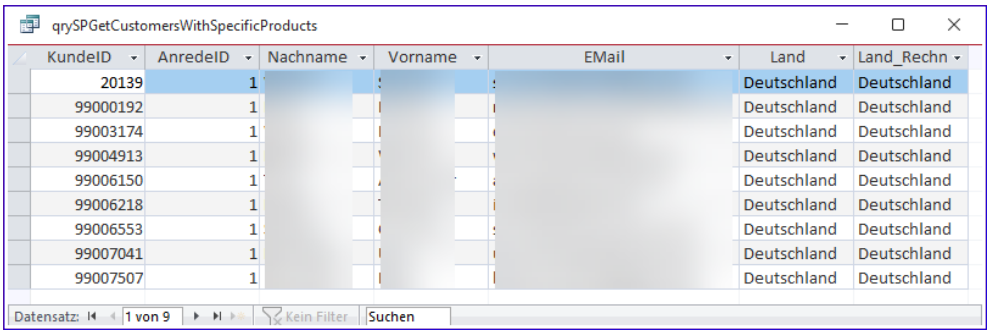


Bild 7: Ergebnis in Access in der Datenblattansicht der Abfrage

Die Verbindungszeichenfolge kann in der Regel einer der per ODBC eingebundenen Tabellen entnommen werden – man braucht nur hinten den Parameter **TABLE** zu entfernen (siehe Bild 6). In unserem Fall lautet die Verbindungszeichenfolge mit dem allgemeinen SQL Server-Treiber so:

```
ODBC;DRIVER=SQL Server;SERVER=amvdesktop2019;Trusted_Connection=Yes;DATABASE=Kundenverwaltung
```

Wechseln wir nun in die Datenblattansicht der Abfrage, finden wir die gewünschten Datensätze in Millisekunden wie in Bild 7 vor.

Parameter hinzufügen

Damit kommen wir zum nächsten Schritt. Bisher liefert die Abfrage die Daten für die Parameter beziehungsweise Kriterien, die wir der Beispielabfrage entnommen haben. Wir wollen die Parameter aber von Access aus ermitteln und an die gespeicherte Prozedur übermitteln, damit diese auf dem SQL Server ausgewertet werden können.

Dazu ersetzen wir zuerst die Vergleichswerte durch entsprechende Parameter. Dabei können wir uns Schritt für Schritt durch die Abfrage arbeiten. Als Erstes wollen wir die Produkt-ID in dieser Zeile durch einen Parameter ersetzen:

```
WHERE tblAbonnements.ProduktID IN (146)
```

Hier stellt sich gleich die Frage, ob wir einen Zahlenwert oder eine Zeichenkette übergeben. Immerhin ermitteln wir die zu untersuchenden Produkt-IDs in der aufrufenden Access-Anwendung und fügen diese zu einer Komma-separierten Liste zusammen – statt **146** könnte hier also auch **146, 147, 148** stehen. Der Einfachheit halber wollen wir hier mit einer Zeichenkette arbeiten. Dazu fügen wir der Abfrage einen Parameter namens **@Produkte** mit dem Datentyp **VARCHAR(255)** hinzu. Außerdem fügen wir diesen Parameter in die Klammern hinter dem **IN**-Schlüsselwort ein:

```
ALTER PROCEDURE spGetCustomersWithSpecificProducts(
  @Produkte VARCHAR(255)
)
AS
BEGIN
        SET NOCOUNT ON;
        SELECT DISTINCT KundeID, AnredeID, Nachname,
          Vorname, EMail, Land, Land_Rechnung
        FROM tblKunden
        WHERE KundeID IN (
                SELECT tblAbonnements.KundeID
                FROM tblAbonnements
                WHERE tblAbonnements.ProduktID
                IN (@Produkte)
```

Dies funktioniert nun mit folgendem Aufruf:

```
EXEC dbo.spGetCustomersWithSpecificProducts "146"
```

Dann versuchen wir es mit dem folgenden Wert für den Parameter:

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP

Detailformular per Mausklick erstellen

Bei der Arbeit mit Microsoft Access gibt es immer wiederkehrende Aufgaben – zum Beispiel das Anlegen von Detailformularen. Diese sollen die Daten aus einfachen Tabellen darstellen und zwei Schaltflächen namens OK und Abbrechen bereitstellen. So kann der Benutzer neue oder geänderte Datensätze übernehmen oder diese verwerfen. Dazu sind immer wieder viele kleine Handgriffe nötig. Damit dies ab jetzt schneller geht, schauen wir uns an, wie wir die meisten der Schritte automatisieren können. Dazu bauen wir ein Formular, mit dem wir alle Konfigurationsschritte erledigen können – von der Auswahl der Datenquelle über die Benennung des Formulars bis hin zur Erstellung des vollständigen Formulars inklusive Code.

Viele Formulare sind eigentlich immer gleich aufgebaut. Für die herkömmliche Bearbeitung von Daten benötigt man ein Detailformular, das die Felder eines Datensatzes anzeigt und zwei Schaltflächen bietet, mit denen der Benutzer die eingegebenen Daten übernehmen oder verwerfen kann.

tblMitglieder

MitgliedII	Anrede	Vorname	Nachname	Straße	PLZ	Ort	E-Mail	Telefon
1	Herr	Wernfried	Birk	Wiener Straße	22297	Hamburg	wernfried@bi	040/707870
2	Herr	Vitus	Krauße	Burgenlandstr	20355	Hamburg	vitus@krauße.	040/3980935
3	Frau	Jadwiga	Oehme	Peter-Rosegge	65187	Wiesbaden	jadwiga@oehr	0611/4266228
4	Herr	Niko	Michel	Kindergartens	12051	Berlin	niko@michel.(	030/9269251
5	Herr	Siegert	Loos	Lenaustraße 6	66115	Saarbrücken	siegert@loos.(	0681/511720
6	Herr	Michl	Schroth	Dr. Karl Renne	01129	Dresden	michl@schrotl	0351/994211
7	Herr	Florentius	Wittek	Industriestraß	80638	München	florentius@wi	089/736215
8	Herr	Gernulf	Riegel	Erzherzog-Joh	81545	München	gernulf@riege	089/2732966
9	Herr	Tristan	Hübsch	Jahnstraße 11	65193	Wiesbaden	tristan@hübsc	0611/6980594
10	Herr	Heinfried	Steinert	Kaplanstraße 5	30159	Hannover	heinfried@ste	0511/4342029
11	Frau	Herma	Aigner	Burgstraße 40	22089	Hamburg	herma@aigne	040/414265
12	Frau	Mina	Dörfler	Schloßstraße 9	38118	Braunschweig	mina@dörfler.	0531/5317475

Datensatz: 1 von 100 | Kein Filter | Suchen

Bild 1: Beispieltabelle **tblMitglieder**

Im Artikel **Tabellendaten mit Übersicht und Details anzeigen** (**www.access-im-unternehmen.de/1488**) zeigen wir, welche Handgriffe alle nötig sind, um ein solches Formular manuell zu erstellen.

Im vorliegenden Beitrag stellen wir eine Lösung vor, mit der wir die Daten aus beliebigen Tabellen schnell in ein Formular umwandeln können.

Ein Beispiel für eine Tabelle, auf deren Basis wir ein Formular erstellen wollen, sehen wir in Bild 1.

Formulare und Tabellen zur Steuerung der Erstellung der Formulare

Wir wären nicht in Access, wenn wir nicht auch für das Erstellen von Formularen per Code die dazu benötigten Daten für die Konfiguration in Tabellen speichern und die Konfiguration in Formularen bearbeiten würden.

Aber welche Daten müssen wir hier überhaupt erfassen?

- Die wichtigste Information ist: Aus welcher Tabelle sollen die Daten überhaupt stammen?
- Wie soll das zu erstellende Formular heißen?

- Sollen Navigationsschaltflächen, Datensatzmarkierer oder Bildlaufleisten angezeigt werden – und soll einer oder mehrere Datensätze angesteuert werden können?

- Welche Felder der Tabelle sollen im Detailformular angezeigt werden? Und in welcher Reihenfolge und Anordnung? Welche Breite sollen die Felder aufweisen?

- Wie sollen die Schaltflächen **OK** und **Abbrechen** aussehen und welche Funktionen sollen sie auslösen? Sollen sie Icons anzeigen?

Allein die Einstellung dieser Eigenschaften erfordert ein technisch anspruchsvolles Formular. Aber zuerst schauen wir uns an, welche Tabellen wir zum Erfassen dieser Daten benötigen.

Vorbereitungen im Datenmodell

Damit wir möglichst nah am Endergebnis arbeiten können, müssen wir bereits im Tabellenentwurf ein wenig Vorarbeit leisten. Wenn ein Feldname nicht dem Text entspricht, der nachher als Beschriftung in der Datenblattansicht und im Detailformular verwendet werden soll, müssen wir den gewünschten Text für die Eigenschaft **Beschriftung** für die jeweiligen Felder im Tabellenentwurf eintragen. Außerdem wollen wir für Fremdschlüsselfelder, auf deren Basis Datensätze aus anderen Tabellen ausgewählt werden sollen, direkt Nachschlagfelder definieren. Die damit

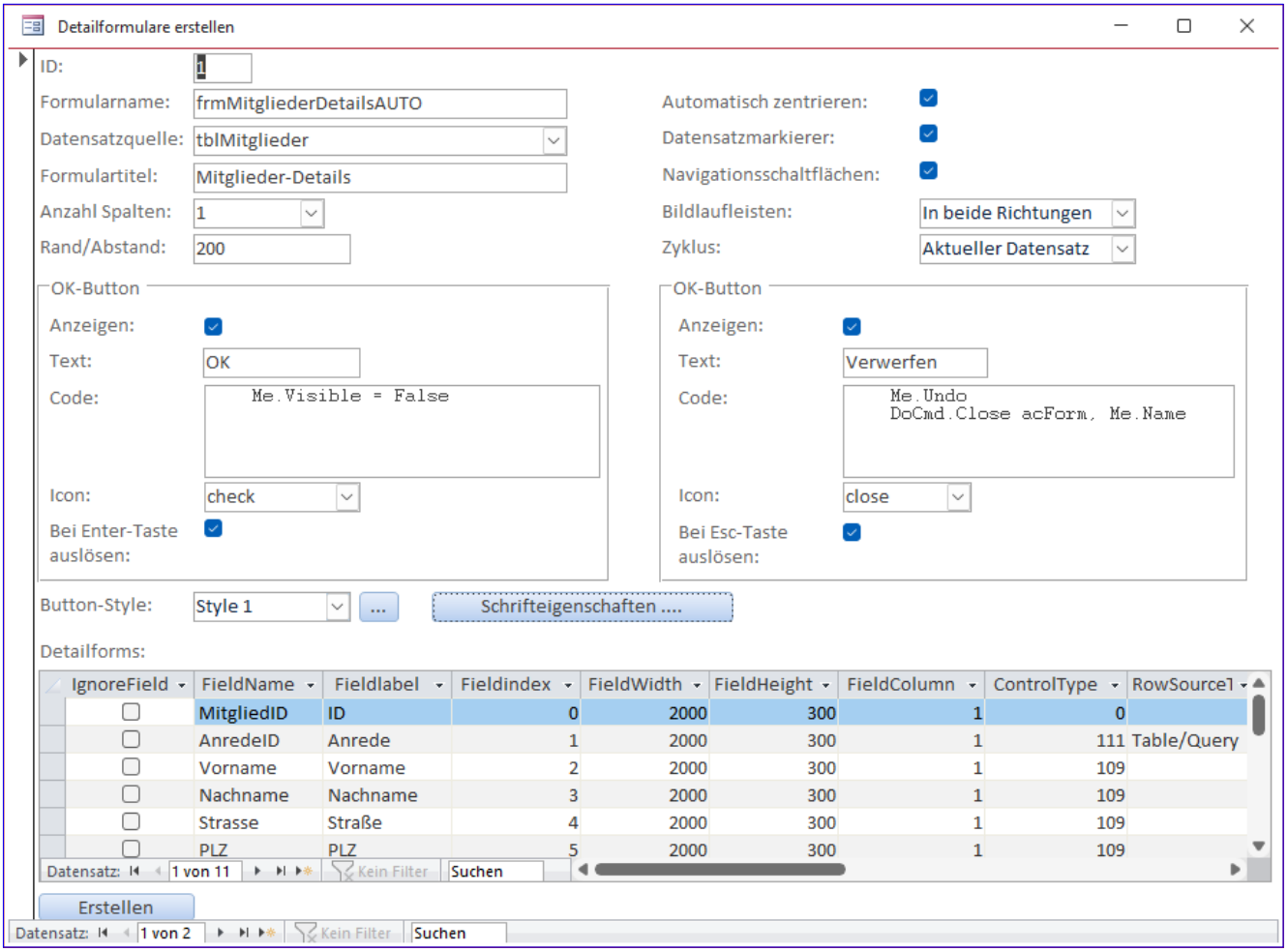


Bild 2: Formular zum Konfigurieren neuer Detailformulare

festgelegten Eigenschaften können wir dann direkt in das zu erstellende Formular übernehmen, genau wie das auch beim Hinzufügen von Feldern aus der Feldliste in manuell erstellten Formularen gelingt.

Das Konfigurationsformular im Überblick

Das von uns zum Erstellen von Detailformularen verwendete Formular sehen Sie in Bild 2. Es bietet die Möglichkeit, verschiedene Konfigurationen zu speichern – wir können also Formulare auf Basis unterschiedlicher Tabellen oder Abfragen definieren und diese immer wieder neu erstellen. Das Formular erlaubt die Eingabe des Namens des zu erstellenden Detailformulars, die Auswahl der Datensatzquelle, also einer Tabelle oder Abfrage, das Festlegen eines Formulartitels und die Angabe eines einheitlichen Abstandes. Dieser wird angewendet für den Abstand aller Steuerelemente von den Formularrändern und für den Abstand zwischen den Steuerelementen. Die Eigenschaft **Anzahl Spalten** haben wir vorerst nur vorbereitet, sie erfüllt noch keinen Zweck.

Rechts sehen wir einige typische Formulareigenschaften wie **Automatisch zentrieren**, **Datensatzmarkierer**, **Navigationsschaltflächen**, **Bildlaufleisten** oder **Zyklus**.

Darunter sehen wir zwei Bereiche, in denen wir die Eigenschaften für die beiden Schaltflächen **OK** und **Abbrechen** einstellen können. Hier geben wir an, ob die jeweilige Schaltfläche überhaupt angelegt werden soll. Dann legen wir fest, welchen Text diese anzeigen und welchen Code sie ausführen sollen. Schließlich können wir auch noch ein Icon auswählen und festlegen, ob die Funktion der **OK**-Schaltfläche auch durch die Eingabetaste ausgelöst werden soll und die Funktion der **Abbrechen**-Schaltfläche durch die **Escape**-Taste.

Darunter sehen wir die Möglichkeit, einen Button-Style auszuwählen. Ein Klick auf die Schaltfläche rechts daneben zeigt den Dialog aus Bild 3 an. Hier können wir einige grundlegende Einstellungen für das Aussehen von Schaltflächen vornehmen.

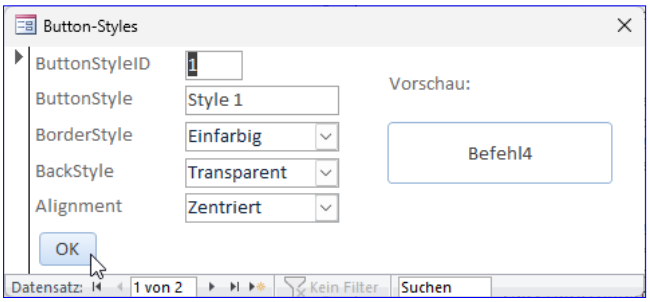


Bild 3: Formular zum Einstellen einiger Schaltflächeneigenschaften

Die Einstellung der Schriftarten haben wir ein wenig rudimentärer gestaltet. Ein Klick auf die Schaltfläche **Schrifteigenschaften ...** öffnet ein Formular in der Entwurfsansicht, das schlicht ein Bezeichnungsfeld anzeigt. Für dieses können wir hier die gewünschten Eigenschaften wie Schriftart, Schriftgröße, Farbe et cetera einstellen. Nach dem Speichern und Schließen holt sich das Formular zum Erstellen neuer Detailformulare die Eigenschaften für die Schrift aus diesem Formular.

Schließlich sehen wir im Unterformular noch die Liste der Felder der zugrunde liegenden Datensatzquelle. Diese wird nach der Auswahl der Tabelle oder Abfrage automatisch aufgrund der Daten der Datensatzquelle gefüllt.

Wir können hier je Feld verschiedene Eigenschaften einstellen. Die erste lautet **IgnoreField** und erlaubt es, ein Feld durch das Setzen eines Hakens aus dem zu erstellenden Formular auszuschließen. **Fieldlabel** nimmt entweder, soweit vorhanden, den Wert der Eigenschaft **Beschriftung** aus dem Tabellenentwurf auf oder den Namen des Feldes. **Fieldindex** gibt die Reihenfolge an, in welcher die Felder angelegt werden sollen. Die Breite können wir individuell einstellen, genauso wie die Höhe. Bei Feldern mit dem Felddatentyp **Langer Text** kann es sinnvoll sein, höhere Felder zu erstellen. **FieldColumn** ist ein vorbereitender Wert, falls wir noch mehrspaltige Detailformulare unterstützen. **ControlType** wird mit dem Steuerelementtyp entsprechend des Feldes aus der Tabelle gefüllt – normalerweise sind dies Textfelder (**109**), gelegentlich aber auch Kombinationsfelder (**111**) oder Kontrollkästchen (**106**). Die übrigen Eigenschaften sind Nachschlagefeldern vorbehal-

ten und werden ebenfalls aus dem Tabellenentwurf entnommen. Wir können diese aber hier auch nachrüsten.

Detailformular erstellen

Ein Klick auf die Schaltfläche **Erstellen** sorgt schließlich dafür, dass ein Formular auf Basis der getätigten Vorgaben erstellt wird (siehe Bild 4). Allein die akkurate Ausrichtung der Steuerelemente würde sonst einige Schritte erfordern, genau wie die manuelle Einstellung der verschiedenen Eigenschaften und des Hinzufügens der Schaltflächen mit dem dahinter liegenden VBA-Code.

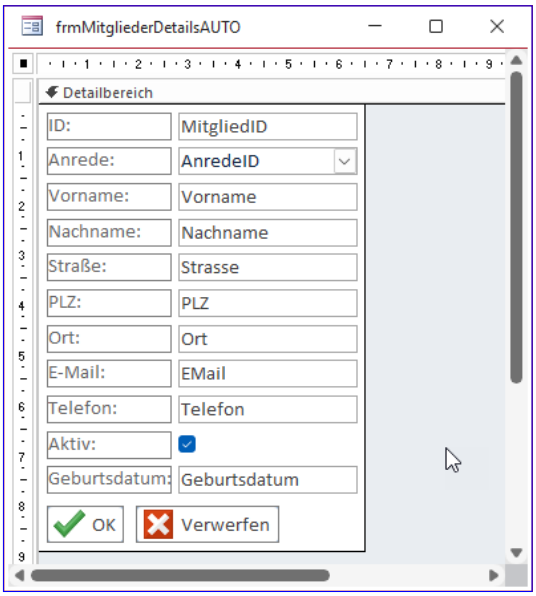


Bild 4: Ein automatisch erstelltes Detailformular

Tabellen der Lösung

Die Lösung verwendet aktuell drei Tabellen zum Speichern von Konfigurationsdaten. Die Tabelle **tblDetailforms** speichert die grundlegenden Daten je Formular. Die Tabelle **tblDetailfields** ist über das Fremdschlüsselfeld **DetailFormID** mit dieser Tabelle verbunden. Schließlich gibt es noch die Tabelle **tblButtonStyles** mit den grundlegenden Designeinstellungen für die Schaltflächen. Der jeweilige Style wird über das Feld **ButtonStyleID** der Tabelle **tblDetailforms** selektiert (siehe Bild 5).

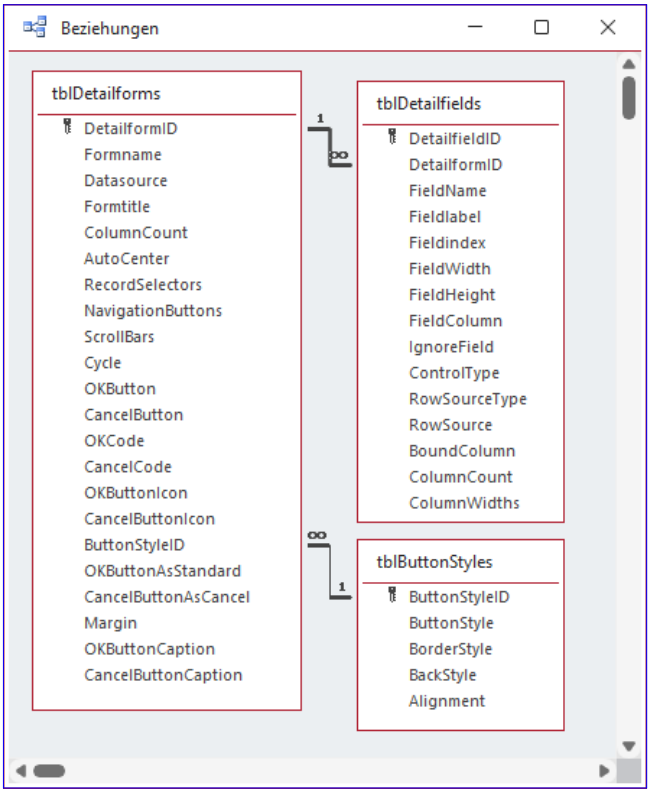


Bild 5: Datenmodell des Add-Ins

Planung der Lösung als Access-Add-In

Wir wollen die Lösung dieses Beitrags später als Access-Add-In bereitstellen, damit wir sie in beliebigen Datenbanken nutzen und damit Detailformulare erstellen können. Daher müssen wir an manchen Stellen etwas aufpassen. Zum Beispiel müssen wir unterschieden, ob wir im Code und in den Steuerelementen auf Daten der Add-In-Datenbank zugreifen oder auf die der Host-Datenbank, also der Datenbank, von der aus wir das Access-Ad-In aufgerufen haben.

Programmierung des Formulars frmDetailforms

Das Formular **frmDetailforms** soll beim Laden zunächst die Daten für die Kombinationsfelder bereitstellen.

Die durch das Ereignis **Beim Laden** ausgelöste Prozedur liest zuerst alle Tabellen und Abfragen der Datenbank ein, von der aus das Access-Add-In gestartet wurde (siehe Listing 1). Dies müssen wir mit dem **Database**-Objekt der Host-Datenbank erledigen, die wir mit **CurrentDb** referenzieren. Hier greifen wir auf die Tabelle **MSysObjects** zu, die alle Objekte der Datenbank enthält. Wir filtern hier nach den Werten **1**, **4**, **5** und **6** im Feld **Type** und erhalten

```
Private Sub Form_Load()
    Dim db As DAO.Database
    Dim dbc As DAO.Database
    Dim rst As DAO.Recordset
    Dim rstIcons As DAO.Recordset
    Set db = CurrentDb
    Set dbc = CodeDb
    Set rst = db.OpenRecordset("SELECT Name FROM MSysObjects WHERE Type IN (1, 4, 5, 6) AND Name NOT LIKE 'F_*' " _
        & "AND Name NOT LIKE 'MSys*' AND Name NOT LIKE 'USys*' AND Name NOT LIKE '~*'", dbOpenDynaset)
    Set Me!cboDatasource.Recordset = rst
    Set rstIcons = dbc.OpenRecordset("SELECT id, [Name] FROM MSysResources WHERE Extension = 'png' ORDER BY [Name]", _
        dbOpenDynaset)
    Set Me!cboOKButtonIcon.Recordset = rstIcons
    Set Me!cboCancelButtonIcon.Recordset = rstIcons
End Sub
```

Listing 1: Beim Laden des Formulars ausgelöste Prozedur

so alle Tabellen und Abfragen. Das so ermittelte Recordset weisen wir dem Kombinationsfeld **cboDatasource** zu.

Die für die Schaltflächen zu verwendenden Icons sollen jedoch aus der Tabelle **MSysResources** der Access-Add-In-Datenbank geholt werden. Diese referenzieren wir nicht mit **CurrentDb**, sondern mit **CodeDb**. Hier holen wir alle Elemente, deren Dateiendung auf png lautet und sortieren diese nach dem Namen. Dieses Recordset weisen wir der entsprechenden Eigenschaft der beiden Kombinationsfelder **cboOKButtonIcon** und **cboCancelButtonIcon** zu.

Felder der gewählten Datensatzquelle einlesen

Wählen wir mit dem Kombinationsfeld **cboDatasource** eine Tabelle oder Abfrage aus, lösen wir die folgende Prozedur aus:

```
Private Sub cboDatasource_AfterUpdate()
    If ReadFieldsFromTable(Me!cboDatasource) = True Then
        Me!sfmDetailforms.Form.Requery
    End If
    If Len(Nz(Me!txtFormname, "")) = 0 Then
        Me!txtFormname = Replace(Replace( _
            Me!cboDatasource, "tbl", "frm"), "qry", "frm")
    End If
    If Len(Nz(Me!txtFormTitle, "")) = 0 Then
        Me!txtFormTitle = Replace(Replace(_
            Me!cboDatasource, "tbl", ""), "qry", "")
    End If

End Sub
```

Diese ruft eine weitere Funktion auf, die wir in Listing 2 sehen. Diese nimmt den Namen der Tabelle oder Abfrage als Parameter entgegen und speichert ihn in der Variablen **strDataSource**. Dann öffnen wir ein Recordset auf Basis der Tabelle **tblDetailfields**, der wir die Feldinformationen zuweisen wollen. Das Recordset holt alle Datensätze, die mit dem aktuell im Hauptformular angezeigten Datensatz der Tabelle **tblDetailforms** verknüpft sind. Die Prozedur bewegt den Datensatzzeiger auf den letzten Datensatz, um anschließend die Anzahl der enthaltenen Datensätze zu ermitteln. Ist diese nicht **0**, stellt die Prozedur dem Benutzer die Frage, ob die Felder neu eingelesen werden sollen. Bejaht der Benutzer dies, löscht die Prozedur zunächst alle für diese Konfiguration bereits vorhandenen Datensätze aus der Tabelle **tblDetailfields**.

Nun referenziert sie mit einem zweiten Recordset die Tabelle, für die wir das Detailformular erstellen wollen, mit der Variablen **rstSource**. Für diese durchlaufen wir alle in der **Fields**-Auflistung enthaltenen **Field**-Elemente

```
Public Function ReadFieldsFromTable(strDataSource As String) As Boolean
    Dim db As DAO.Database, dbc As DAO.Database, rst As DAO.Recordset, rstSource As DAO.Recordset
    Dim bolReadFields As Boolean, intControlType As AcControlType
    Dim varRowSourceType As Variant, varRowSource As Variant, varBoundColumn As Variant
    Dim varCaption As Variant, varColumnCount As Variant, varColumnWidths As Variant
    Dim fld As DAO.Field, i As Integer
    strDataSource = Me!cboDatasource
    Set db = CurrentDb
    Set dbc = CodeDb
    Set rst = dbc.OpenRecordset("SELECT * FROM tblDetailfields WHERE DetailformID = " & Me!DetailformID, dbOpenDynaset)
    If Not rst.EOF Then
        rst.MoveLast
    End If
    bolReadFields = True
    Me.Dirty = False
    If Not rst.RecordCount = 0 Then
        bolReadFields = MsgBox("Sollen die Felder neu eingelesen werden?", vbYesNo + vbExclamation, _
            "Felder neu einlesen?") = vbYes
    End If
    If bolReadFields = True Then
        dbc.Execute "DELETE FROM tblDetailfields WHERE DetailformID = " & Me!DetailformID, dbFailOnError
        Set rstSource = db.OpenRecordset(strDataSource, dbOpenDynaset)
        For Each fld In rstSource.Fields
            Call GetControlProperties(fld, varCaption, intControlType, varBoundColumn, varRowSourceType, _
                varRowSource, varColumnCount, varColumnWidths)
            rst.AddNew
            rst!DetailformID = Me!txtDetailformID
            rst!FieldName = fld.Name
            rst!Fieldindex = i
            rst!FieldWidth = 2000
            rst!Fieldheight = 300
            rst!FieldColumn = 1
            rst!Fieldlabel = Nz(varCaption, fld.Name)
            rst!ControlType = intControlType
            rst!RowSourceType = varRowSourceType
            rst!RowSource = varRowSource
            rst!BoundColumn = varBoundColumn
            rst!ColumnCount = varColumnCount
            rst!ColumnWidths = varColumnWidths
            rst.Update
            i = i + 1
        Next fld
        ReadFieldsFromTable = True
        Me!sfmDetailforms.Form.Requery
    Else
        ReadFieldsFromTable = False
    End If
End Function
```

Listing 2: Einlesen der Feldinformationen in die Tabelle **tblDetailfields**

und referenzieren das aktuelle Element jeweils mit der Variablen **fld**.

Hier rufen wir nun eine weitere Funktion namens **GetControlProperties** auf und übergeben dieser einen Verweis auf das zu untersuchende Feld. Außerdem übergeben wir einige **Variant**-Variablen, die wir mit den Werten der entsprechenden Eigenschaften des Feldes füllen wollen – diese beschreiben wir gleich im Anschluss.

Nachdem wir die verschiedenen **Variant**-Variablen mit den Eigenschaften des Feldes **fld** gefüllt haben, legen wir einen neuen Datensatz im Recordset der Tabelle **tblDetailfields** an und füllen nacheinander die Eigenschaften. Dazu gehören der Verweis auf den aktuellen Eintrag der Tabelle **tblDetailforms**, der Feldname, der Index, Voreinstellungen von 2.000 und 300 Pixeln für die Steuerelementbreite und -höhe und der Wert **1** für die Spalte. Danach folgen die Beschriftung für das Steuerelement, der Steuerelementtyp und dann einige Eigenschaften, die nur für Nachschlagefelder benötigt werden. Danach speichern wir den neuen Datensatz mit der **Update**-Methode und liefern der aufrufenden Prozedur den Wert **True** zurück.

Die aufrufende Prozedur **cboDatasource_AfterUpdate** prüft schließlich noch, ob **txtFormname** und **txtFormTitel**

```
Public Function GetControlProperties(fld As DAO.Field, varCaption As Variant, intControlType As AcControlType, _
        varBoundColumn As Variant, varRowSourceType As Variant, varRowSource As Variant, varColumnCount As Variant, _
        varColumnWidths As Variant)
    intControlType = 0
    varCaption = Null
    varBoundColumn = Null
    varRowSourceType = Null
    varRowSource = Null
    varColumnCount = Null
    varColumnWidths = Null
    On Error Resume Next
    varCaption = fld.Properties("Caption")
    intControlType = fld.Properties("DisplayControl")
    Select Case intControlType
        Case acTextBox
            Debug.Print fld.Name, "acTextBox"
        Case acComboBox
            varBoundColumn = fld.Properties("BoundColumn")
            varRowSourceType = fld.Properties("RowSourceType")
            varRowSource = fld.Properties("RowSource")
            varColumnCount = fld.Properties("ColumnCount")
            varColumnWidths = fld.Properties("ColumnWidths")
        Case acCheckBox
            Debug.Print fld.Name, "acCheckBox"
        Case Else
            Debug.Print "anderer Controltype", intControlType
    End Select
    On Error GoTo 0
End Function
```

Listing 3: Ermitteln verschiedener Feldeigenschaften

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP