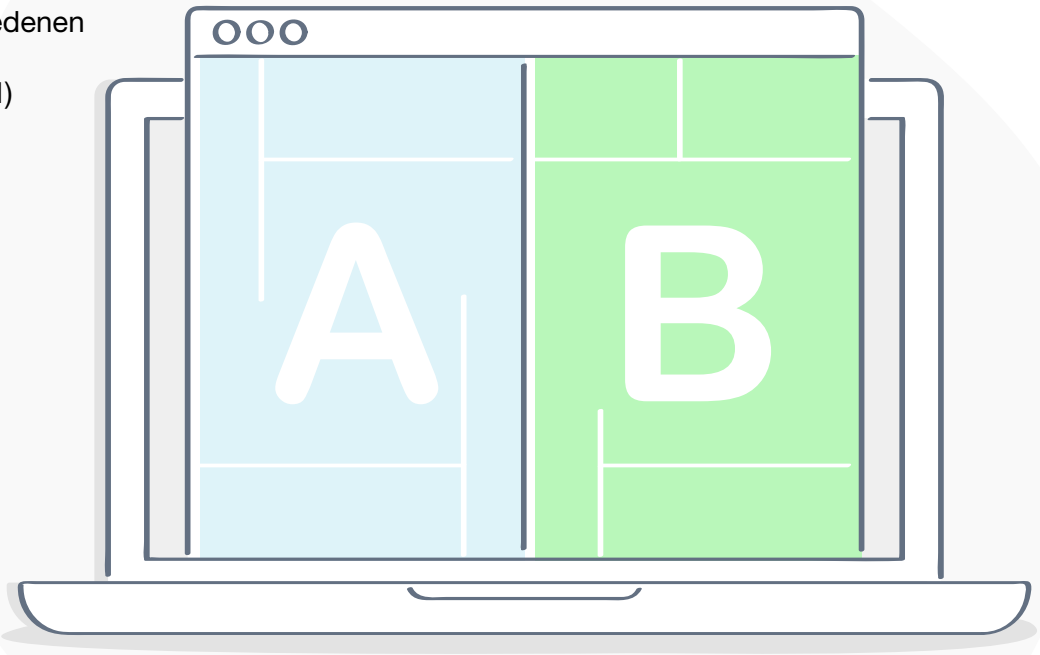
HAUFE.
Ausgabe 04/2020

# ACCESS IM UNTERNEHMEN

## VBA-PROJEKTE VERGLEICHEN

Unterschiede zwischen verschiedenen Versionen eines VBA-Projekts übersichtlich darstellen (ab S. 61)




## In diesem Heft:

# Von Version zu Version

**Man kann mit verschiedenen Tools unter Access die Versionen des Quellcodes verwalten. Wer den Aufwand scheut oder denkt, für kleine Projekt (die dann doch groß werden) lohnt es sich nicht, stellt früher oder später fest, dass er vielleicht gern Änderungen im Code rückgängig machen würde. Aber welche Änderungen genau wurden nochmal seit der letzten Sicherung durchgeführt? Das können Sie auch nachträglich noch feststellen – und zwar mit dem Add-In, das wir in dieser Ausgabe von Access im Unternehmen vorstellen.**

Das Add-In, das wir ab Seite 61 im Beitrag **VBA-Projekte vergleichen** vorstellen, macht Ihnen die das Vergleichen recht einfach: Sie brauchen das Add-In nur zu starten, die Datenbanken mit den beiden zu vergleichenden VBA-Projekten auszuwählen und den Vergleich per Mausklick zu starten. Gleich danach erscheint ein Text-Editor, der Ihnen die Unterschiede zwischen den VBA-Projekten aufzeigt – und anhand derer Sie prüfen können, welche Änderungen Sie gegebenenfalls rückgängig machen wollen.

Im Beitrag **Formularansichten von A-Z** lesen Sie ab Seite 2, welche Formularansichten es gibt und wie Sie diese anzeigen können. Außerdem lernen Sie dort, wie Sie die verschiedenen Ansichten per Tastenkombination erreichen können, was viel schneller geht, als die entsprechenden Schaltflächen erst mit der Maus anzuklicken. Und schließlich zeigen wir Ihnen, wie Sie Formularansichten sperren, um diese für Anwender nicht mehr über die Benutzeroberfläche verfügbar zu machen.

Wer mit TreeViews arbeitet, weiß, dass es sehr praktisch ist, wenn man die für die Verwaltung der Einträge benötigten Befehle schnell und einfach aufrufen kann. Üblicherweise erledigt man das mit Kontextmenü-Befehlen, die man durch einen Rechtsklick auf den entsprechenden Eintrag einblendet. Wie Sie solche Kontextmenüs implementieren, zeigen wir Ihnen ab Seite 7 im Beitrag **TreeView mit Kontextmenü**. Dort lernen Sie, wie Sie die Kontextmenüs zusammenstellen, diese per Rechtsklick aufrufen und wie Sie die Funktionen programmieren müssen, die durch die Kontextmenübefehle ausgeführt werden sollen.

Eine Alternative zu diesen Kontextmenüs bietet das Ribbon. So können Sie, wenn der Benutzer eines der Elemente des TreeView-Steuerelements angeklickt hat, die dazu passenden Befehle im Ribbon einblenden. Das hat gegenüber dem Kontextmenü den Vorteil, dass auch unerfahrenere Benutzer direkt sehen, welche Optionen es für die Verwaltung der angezeigten Daten gibt. Alles zu diesem Thema lesen Sie ab Seite 18 unter dem Titel **TreeView mit Ribbon-Einträgen**.

Unsere in der vorherigen Ausgabe begonnene Beitragsreihe zum Thema SQL Server-Sicherheit setzen wir mit dem zweiten Teil namens **SQL Server-Security – Teil 2: Zugriffsberechtigung** ab Seite 43 fort. Diesmal erfahren Sie, wie Sie die Zugriffsberechtigungen für den SQL Server und die enthaltenen Datenbanken steuern können.

Außerdem lernen Sie im Beitrag **Formulare zur Laufzeit analysieren** ab Seite 29 eine Möglichkeit kennen, um die Eigenschaften von Formularen und Steuerelementen zur Laufzeit über das Kontextmenü einzublenden. Damit sparen Sie sich das aufwendige Abfragen von Eigenschaftswerten über das Direktfenster – insbesondere wenn Sie die Namen der zu untersuchenden Steuerelemente und Eigenschaften gerade nicht zur Hand haben.

Viel Spaß beim Lesen!

Ihr André Minhorst

# Formularansichten von A-Z

**In den aktuelleren Versionen von Access gibt es vier verschiedene Ansichten für Formulare. Während die Entwurfsansicht und die Layoutansicht ihren Nutzen bei der Entwicklung von Formularen unter Beweis stellen, sind die Datenblattansicht und die Formularansicht für die Datenanzeige und -bearbeitung sinnvoll. Aber sind dies alle Formularansichten? Nein, denn es gibt auch noch die Endlosansicht. Wie Sie welche Ansichten einstellen und nutzen, zeigt der vorliegende Beitrag. Und Sie erfahren auch noch, wie Sie Formulare in den unterschiedlichen Ansichten öffnen und wie Sie den Benutzer davon abhalten, zu bestimmten Ansichten zu wechseln.**

Ein neues Formular öffnen Sie in der Regel in der Entwurfsansicht. Von dort aus gelangen Sie dann mit dem Ribboneintrag **Start|Ansichten** und den dort enthaltenen Einträgen **Formularansicht**, **Datenblattansicht**, **Layoutansicht** und **Entwurfsansicht** zu den verschiedenen Ansichten (siehe Bild 1).

## Alternatives Umschalten

Was manche Leser vielleicht nicht wissen (ich selbst bin erst vor Kurzem darauf gestoßen): Es gibt noch eine weitere, schnellere Methode, gezielt eine der anderen Ansichten anzusteuern. Die dazu verwendeten Schaltflächen befinden sich ganz unten rechts in der Statusleiste des Access-Fensters (siehe Pfeil in der Abbildung). Hier brauchen Sie nicht erst zum passenden Ribbon-Tab zu wechseln und dann aus dem Popup-Menü die gewünschte Ansicht auszuwählen, wenn diese nicht zufällig gerade angeboten wird.

Sie klicken einfach auf die gewünschte Ansicht und das aktuelle Formular wird umgehend in dieser Ansicht angezeigt.

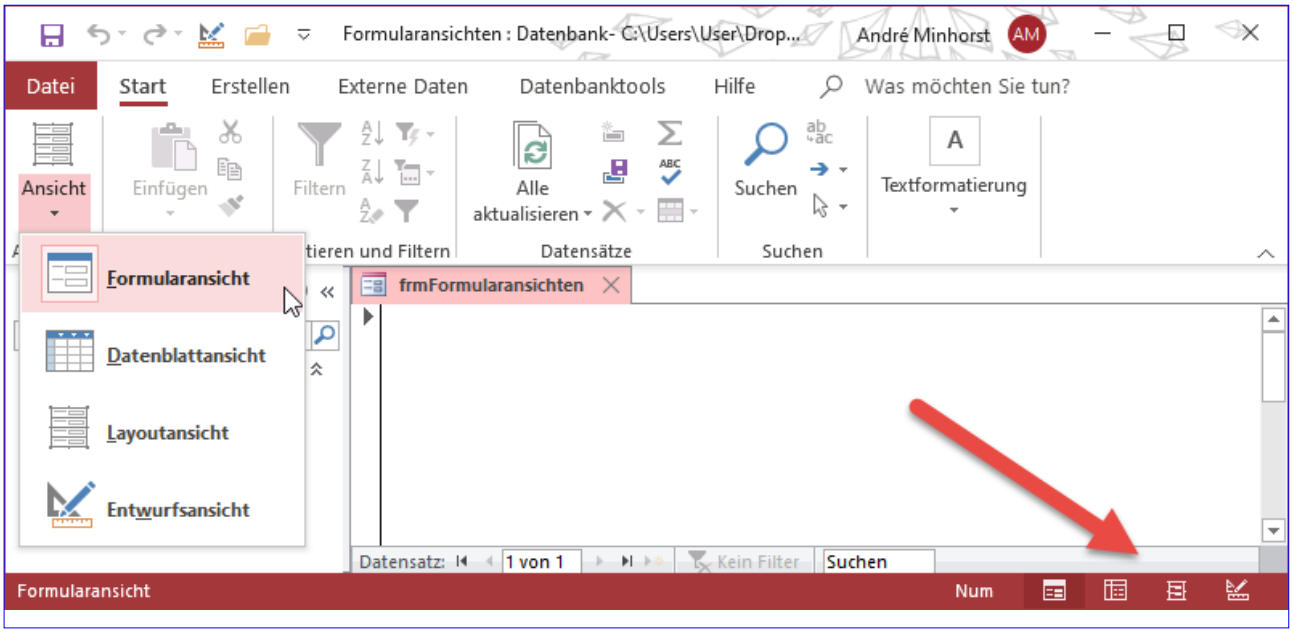



**Bild 1:** Formular und die Möglichkeiten zum Einstellen der Ansichten

### Noch schneller per Tastenkombination

Seit längerer Zeit hingegen arbeite ich jedoch mit den Tastenkombinationen zum Wechseln zwischen den Formularansichten.

Die Tastenkombinationen lauten:

- **Strg + .** (Punkt): Wechselt in der Reihenfolge Entwurfsansicht – Formularansicht – Datenblattansicht zur jeweils nächsten Ansicht. Haben Sie die Datenblattansicht erreicht, benötigen Sie die nachfolgend vorgestellte Tastenkombination, um wieder zu den vorherigen Ansichten zu gelangen.

- **Strg + ,** (Komma): Wechselt in der Reihenfolge Datenblattansicht – Formularansicht – Entwurfsansicht zur jeweils nächsten Ansicht. Sind Sie bei der Entwurfsansicht angelangt, finden Sie über die zuvor beschriebene Tastenkombination wieder zu den anderen Ansichten.

Sie sehen hier, dass Sie die Layout-Ansicht nicht über diese Tastenkombinationen aktivieren können.

### Formularansichten per VBA

Wenn Sie mit VBA arbeiten, möchten Sie Formulare in verschiedenen Ansichten öffnen. In der Regel ist die Ansicht, in der ein Formular angezeigt werden soll, jedoch schon über die Eigenschaft **Standardansicht** festgelegt.

Das heißt, dass wir in der Regel einfach nur das Formular in der Standardansicht öffnen, also etwa so:

```
DoCmd.OpenForm "frmTest", acNormal
```

Diesen Parameter brauchen Sie übrigens gar nicht anzugeben, weil es sich um den Standardwert handelt. Es geht also auch mit:

```
DoCmd.OpenForm "frmTest"
```

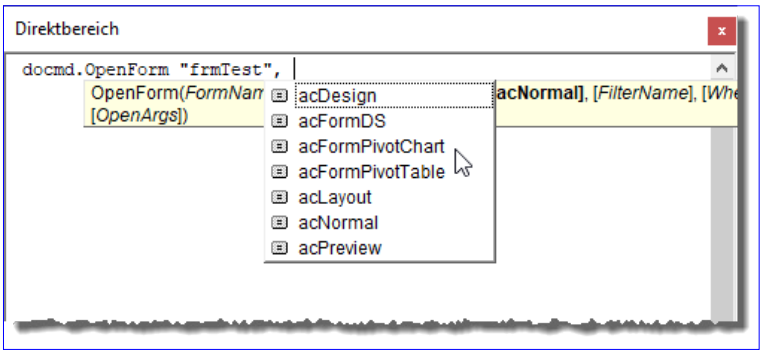



**Bild 2:** Öffnen eines Formulars in den verschiedenen Ansichten per VBA

Wie Bild 2 zeigt, gibt es jedoch auch einige weitere Parameterwerte für die Festlegung der Formularansicht beim Öffnen mit der **DoCmd.OpenForm**-Methode:

- **acDesign**: Entwurfsansicht
- **acFormDS**: Datenblattansicht
- **acFormPivotChart**: PivotChart-Ansicht, veraltet
- **acFormPivotTable**: PivotTable-Ansicht, veraltet
- **acLayout**: Layout-Ansicht
- **acNormal**: Standardmäßig festgelegte Ansicht
- **acPreview**: Vorschauansicht zum Drucken des Inhalts eines Formulars

### Und was ist mit dem Endlosformular?

Wenn Sie die Endlosansicht verwenden wollen, müssen Sie für die Eigenschaft **Standardansicht** den Wert **Endlosformular** einstellen (siehe Bild 3). Öffnen Sie das Formular danach per Doppelklick auf den Eintrag im Navigationsbereich oder mit der Methode **DoCmd.OpenForm**, wird das Formular in der **Endlosansicht** angezeigt.

Bei dieser Gelegenheit entdecken wir noch eine weitere Ansicht, nämlich die Ansicht **Geteiltes Formular**. Diese Ansicht ist ein Hybrid zwischen der Formularansicht

und der Datenblattansicht. Beide Ansichten werden mit synchronen Daten in jeweils einem Bereich des Formulars angezeigt. Auch für diese Ansicht stellen Sie die Eigenschaft **Standardansicht** auf **Geteiltes Formular** ein und öffnen das Formular dann in der Formularansicht.

## Formularansichten zulassen oder sperren

Direkt unter der Eigenschaft **Standardansicht** finden wir noch weitere interessante Eigenschaften:

- **Formularansicht zulassen**: Legt mit **Ja** oder **Nein** fest, ob die Formularansicht zugelassen werden soll.
- **Datenblattansicht zulassen**: Aktiviert oder sperrt die Datenblattansicht.
- **Layoutansicht zulassen**: Aktiviert oder sperrt die Layoutansicht.

Sie können die Ansichten nur in den folgenden Kombinationen sperren:

- Formularansicht und Layoutansicht
- Datenblattansicht und Layoutansicht

Sie können nicht gleichzeitig die Formularansicht und die Datenblattansicht deaktivieren. Interessant ist, wie sich die Sperrung verschiedener Ansichten auf die Anzeige der Schaltflächen zum Auswählen der Formularansicht in der Statusleiste auswirkt. Wenn Sie die Formularansicht deaktivieren, wird nur noch die Schaltfläche für die Datenblattansicht angezeigt – ein Wechsel in die Layoutansicht ist dann auch nicht mehr möglich. Deaktivieren Sie die Datenblattansicht, bleiben die Schaltflächen für die Formularansicht und die Layoutansicht erhalten.

Deaktivieren Sie die Layoutansicht, wird nur diese Schaltfläche ausgeblendet.

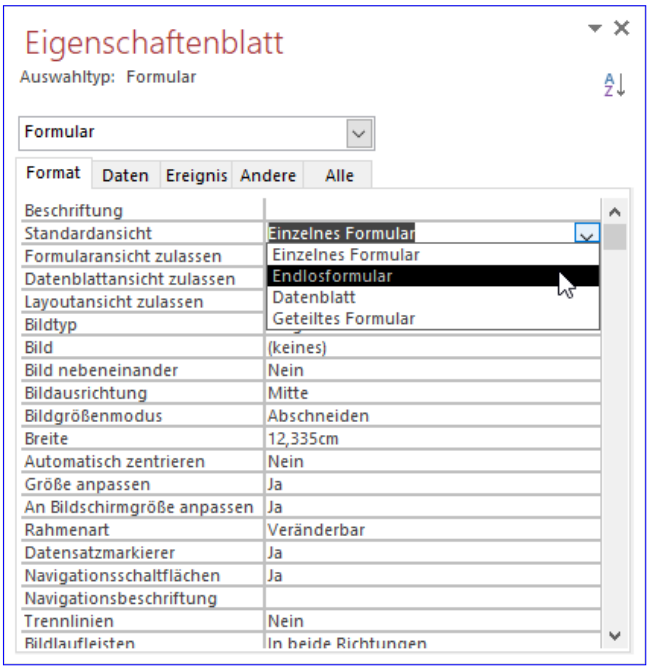



**Bild 3:** Einstellen der Standardansicht eines Formulars

Die Anzeige der Layoutansicht können Sie auch anwendungsweit deaktivieren, indem Sie die entsprechende Option in den Access-Optionen deaktivieren (siehe Bild 4).

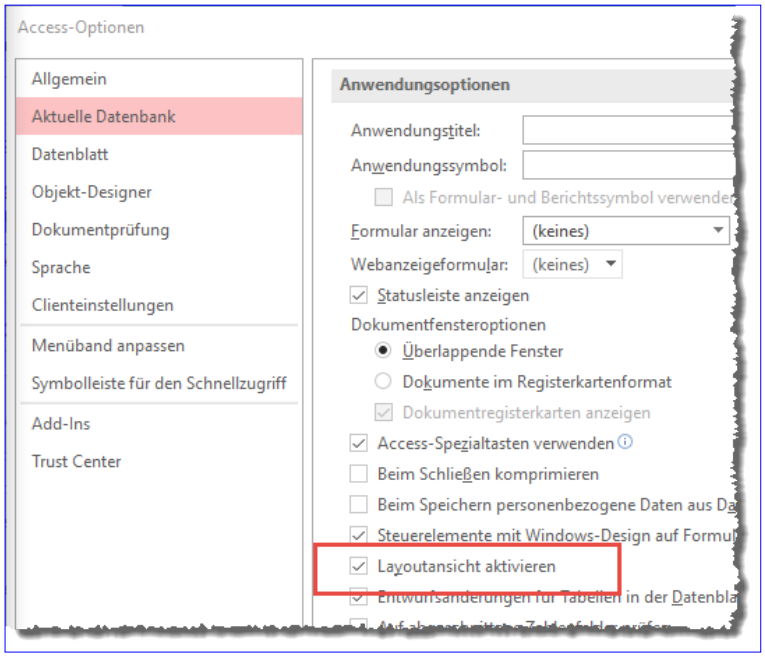



**Bild 4:** Deaktivieren der Layoutansicht

# TreeView mit Kontextmenü

**TreeViews sind sehr flexible Steuerelemente, die sich vor allem zur Darstellung von Daten in hierarchischen Strukturen etabliert haben. Sie bieten viele verschiedene Möglichkeiten zur Visualisierung von Daten. Außerdem lassen sich die gängigen Ereignisse für die Elemente des TreeView-Steuerelements abbilden – zum Beispiel Mausklicks, Drag and Drop und so weiter. Leider bietet das TreeView-Steuerelement keine eingebaute Möglichkeit, wie bei den Standard-Steuerelementen Kontextmenüs anzuzeigen. Daher schauen wir uns in diesem Beitrag an, wie Sie ein TreeView-Steuerelement um Kontextmenü-Funktionen erweitern und welche Strategien dabei wichtig sind.**

## Aktionen zu TreeView-Einträgen

Ein TreeView, das Daten aus verknüpften Tabellen hierarchisch anzeigt, ist in vielen Szenarien sinnvoll: egal, ob Sie nun Kunden, Projekte und Aufgaben, Bauteile mit den einzelnen Elementen oder andere Daten abbilden. Sie können Elemente auf- und zuklappen, Elemente anklicken, um Details dazu in weiteren Steuerelementen anzuzeigen und vieles mehr.

Was aber noch fehlt, ist die Möglichkeit, Befehle für das aktuell angeklickte Element anzuzeigen. Wenn das TreeView-Steuerelement beispielsweise einen Kunden anzeigt, möchten Sie vielleicht verschiedene Aktionen für diesen Kunden ausführen – ein neues Projekt anlegen, den Kunden löschen oder eine Mail an diesen Kunden schicken.

Natürlich könnte man solche Befehle über Schaltflächen auslösen, die in einem Detailformular angezeigt werden, das nach einem Klick auf das Kundenelement im TreeView erscheint.

## Die Lösung: Ein Kontextmenü

Es geht aber auch noch eleganter, und zwar mit einem Kontextmenü, das genau die zu dem angeklickten Element gehörenden Befehle anzeigt.

Ein Kontextmenü hat den Vorteil, dass es keinen Platz benötigt – Sie zeigen es an, indem Sie mit der rechten Maustaste auf das Element klicken, wählen die gewünschte Aktion aus und es verschwindet wieder. Im Detailformular würden die Befehle beziehungsweise die Schaltflächen Platz wegnehmen.

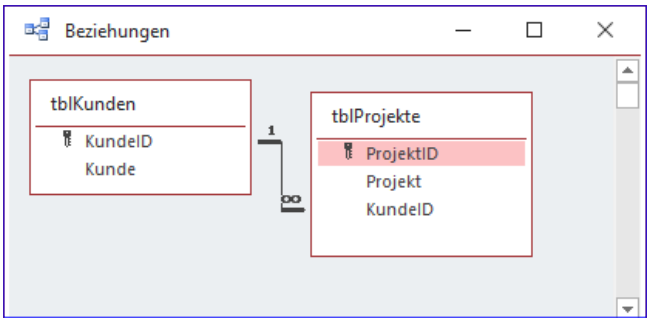



**Bild 1:** Datenquelle des **TreeView**-Steuerelements

Der Nachteil von Kontextmenüs ist, dass der Benutzer erst einmal wissen muss, dass es diese gibt – immerhin gibt es in der Regel keinen Hinweis darauf, dass sich durch einen Rechtsklick auf ein Element weitere Optionen erschließen lassen.

## Beispieldaten

Die Beispieldaten zu diesem Beitrag stammen aus den beiden Tabellen **tblKunden** und **tblProjekte**. Diese sind über das Fremdschlüsselfeld **KundeID** der Tabelle **tblProjekte** verknüpft (siehe Bild 1).

Die beiden Tabellen haben wir mit einigen Beispieldaten gefüllt – siehe Bild 2.

## TreeView-Steuerelement

Das **TreeView**-Steuerelement fügen wir unter dem Namen **ctlTreeView** zu einem neuen Formular namens **frmKundenProjekte** hinzu (siehe Bild 3).

Hier erkennen Sie auch, dass wir bereits ein **ImageList**-Steuerelement hinzugefügt haben.

Dieses nennen wir **ctlImageList** und fügen ihm gleich noch zwei Icons hinzu, die wir mit den Werten **user** und **folder** für die Eigenschaft **Key** versehen (siehe Bild 4).

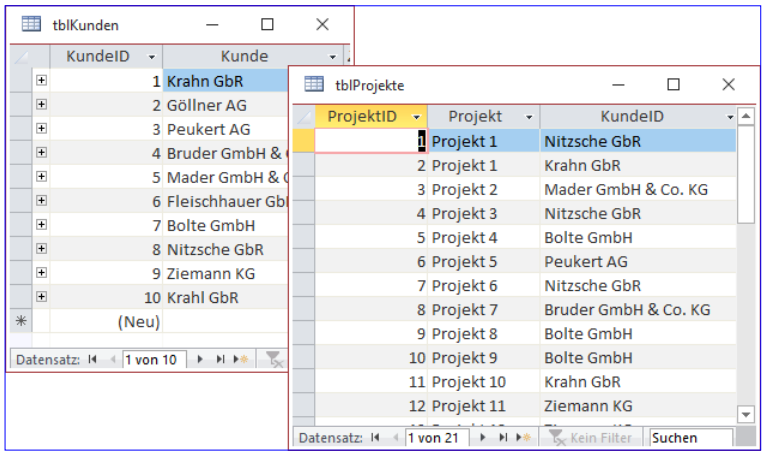



**Bild 2:** Daten der beiden Tabellen **tblKunden** und **tblProjekte**

## Formular vorbereiten

Da das Formular keine Daten anzeigen soll, stellen wir seine Eigenschaften **Datensatzmarkierer**, **Navigationsschaltflächen** und **Bildlaufleisten** auf den Wert **Nein** ein und **Automatisch zentrieren** auf **Ja**.

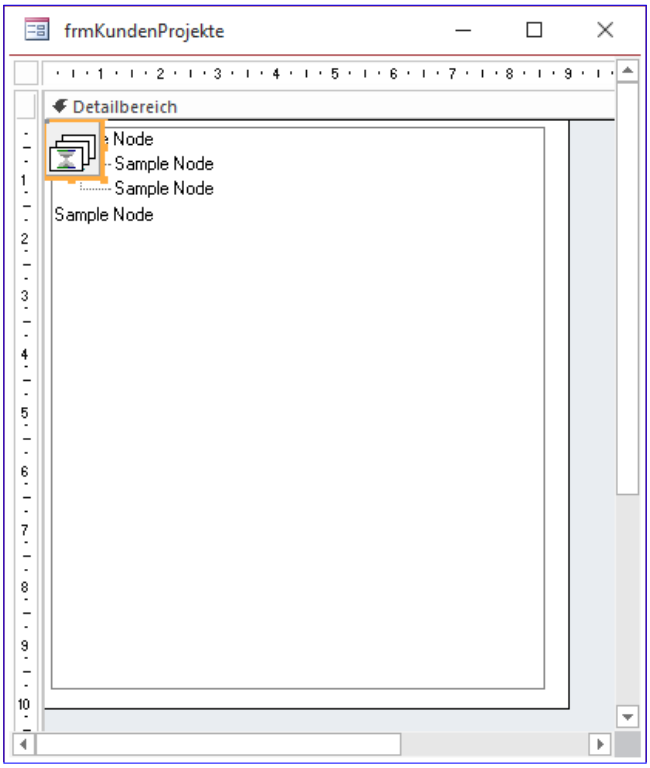



**Bild 3:** TreeView und ImageList in der Entwurfsansicht

## TreeView formatieren

Danach stellen wir bereits die notwendigen Eigenschaften für das **TreeView**-Steuerelement ein, was wir per VBA erledigen statt über den Eigenschaften-Dialog des Steuerelements.

Allein das **ImageList**-Steuerelement weisen wir über den Eigenschaften-Dialog zu, und zwar über den Wert **ctlImageList** für die Eigenschaft **ImageList**.

Im Klassenmodul des Formulars hinterlegen wir eine Objektvariable für das **TreeView**-Steuerelement:

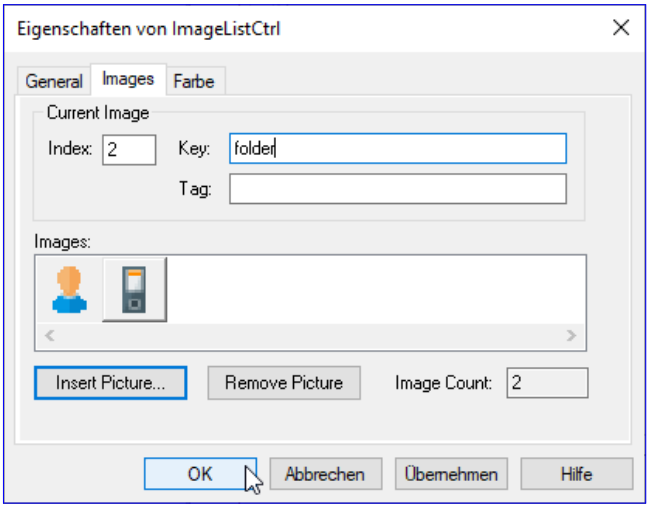



**Bild 4:** Bilder im **ImageList**-Steuerelement

```
Dim WithEvents objTreeView As MSComctlLib.TreeView
```

Dann implementieren wir die Ereignisprozedur, die durch das Ereignis **Beim Laden** des Formulars ausgelöst wird, und stellen darin die wichtigsten Eigenschaften des **TreeView**-Steuerelements ein:

```
Private Sub Form_Load()
    Set objTreeView = Me!ctlTreeView.Object
    With objTreeView
        .Appearance = ccFlat
        .BorderStyle = ccNone
        .LineStyle = tvwRootLines
        .Style = tvwTreelinesPlusMinusPictureText
    End With
    FillTreeView
End Sub
```

Dazu gehören die Eigenschaften **Appearance** (wir wollen keinen 3D-Effekt, daher stellen wir diese Eigenschaft auf den Wert **ccFlat** ein), **BorderStyle**, **LineStyle** und **Style** – diese Eigenschaft erhält den Wert **tvwTreelinesPlus-MinusPictureText**, damit wir ein TreeView wie in Bild 5 erhalten.

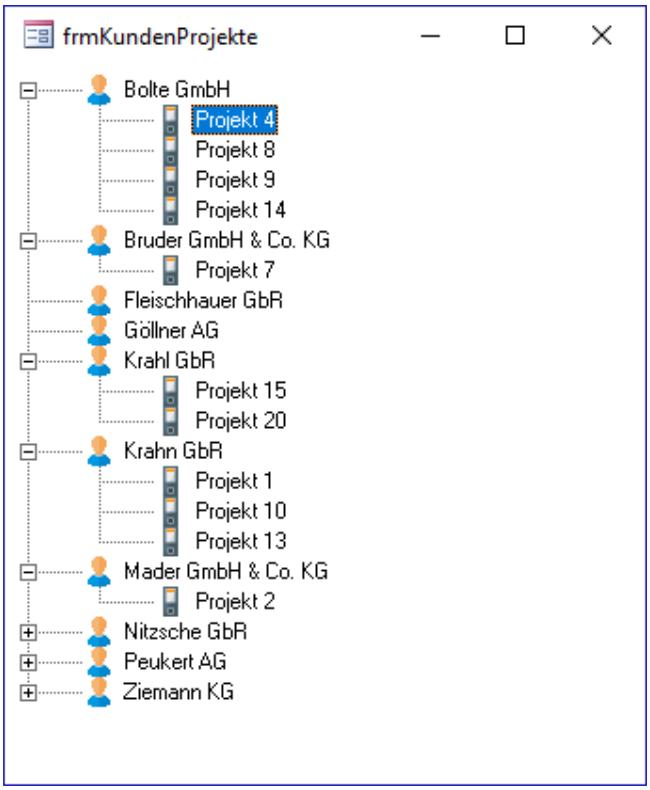



**Bild 5:** Kunden und Projekte im TreeView

```
Private Sub FillTreeView()
    Dim db As DAO.Database
    Dim rstKunden As DAO.Recordset
    Dim rstProjekte As DAO.Recordset
    Set db = CurrentDb
    Set rstKunden = db.OpenRecordset("SELECT * FROM tblKunden ORDER BY Kunde", dbOpenDynaset)
    Do While Not rstKunden.EOF
        objTreeView.Nodes.Add , , "k" & rstKunden!KundeID, rstKunden!Kunde, "user"
        Set rstProjekte = db.OpenRecordset("SELECT * FROM tblProjekte WHERE KundeID = " & rstKunden!KundeID, _
            dbOpenDynaset)
        Do While Not rstProjekte.EOF
            objTreeView.Nodes.Add "k" & rstKunden!KundeID, tvwChild, "p" & rstProjekte!ProjektID, _
                rstProjekte!Projekt, "folder"
            rstProjekte.MoveNext
        Loop
        rstKunden.MoveNext
    Loop
End Sub
```

**Listing 1:** Diese Prozedur füllt das **TreeView**-Steuerelement

Die letzte Anweisung der Prozedur ruft eine weitere Routine namens **FillTreeView** auf, die wie in Listing 1 aussieht. Diese Routine referenziert zunächst das **Database**-Objekt der aktuellen Datenbank und füllt dann ein **Recordset**-Objekt mit den Daten der Tabelle **tblKunden** – und zwar nach dem Feld **Kunde** sortiert.

Die Datensätze dieses Recordsets durchläuft sie dann in einer **Do While**-Schleife. Innerhalb dieser Schleife erstellt sie zunächst jeweils ein **Node**-Element im **TreeView**-Steuerelement. Dabei weist sie mit der **Add**-Methode der **Nodes**-Auflistung ein Element in der ersten Ebene zu – deshalb bleiben die ersten beiden Parameter der Methode leer.

Der dritte Parameter gibt den Key an, also einen eindeutigen Identifizierer für das **Node**-Element im **TreeView**-Steuerelement. Für den Wert gibt es genau eine Voraussetzung: Er muss mit einem Buchstaben beginnen. Wir können also nicht einfach den Primärschlüsselwert des Kunden für die Eigenschaft **Key** angeben.

Das ist aber auch kein Problem, denn wenn wir das tun könnten, würden wir schnell doppelte **Key**-Werte produzieren, denn ein Primärschlüsselwert kann ja durchaus mehrfach in den Tabellen vorkommen, deren Daten im **TreeView**-Steuerelement angezeigt werden sollen.

Also fügen wir einfach einen Buchstaben voran, der die Art des Eintrags kennzeichnet – hier **k** für Kunde –, und fügen dahinter den Primärschlüsselwert ein. Für den Primärschlüsselwert **123** lautet der **Key**-Wert dann also **k123**.

Dahinter folgt der anzuzeigende Text, den wir aus dem Feld **Kunde** des Recordsets entnehmen (**rstKunden!Kunde**). Schließlich geben wir für den Parameter **Image** den Wert **user** an.

Dieser entspricht dem Icon aus dem **ImageList**-Steuerelement, das wir in Zusammenhang mit dem Kunden-Element anzeigen wollen.

Danach erstellt die Prozedur ein weiteres Recordset, dass diesmal alle Datensätze der Tabelle **tblProjekte** enthält, die über das Feld **KundeID** mit dem Kunden verknüpft sind, den wir gerade im Datensatz der übergeordneten **Do While**-Schleife bearbeiten.

Diese Projekt-Datensätze durchlaufen wir ebenfalls in einer **Do While**-Schleife. Innerhalb der Schleife legen wir für jedes Projekt ein neues Element unterhalb des jeweiligen Kunden-Elements an. Für diese Zuordnung nutzen wir die ersten beiden Parameter der **Add**-Methode der **Nodes**-Auflistung.

Der erste nimmt den **Key**-Wert des **Node**-Elements entgegen, unter dem wir das Projekt-Element anlegen wollen.

Der zweite legt fest, in welchem Verhältnis das neue Element zu dem mit dem ersten Parameter referenzierten Element stehen soll – in diesem Fall soll es untergeordnet werden, also verwenden wir den Wert **tvwChild**.

Der dritte, vierte und fünfte Parameter werden analog zum Aufruf der **Add**-Methode für die Kunden-Nodes gefüllt. Der wichtigste Unterschied ist, dass der Wert der **Key**-Eigenschaft mit **p** beginnt (für Projekt). Außerdem verwenden wir das Icon mit der Bezeichnung **folder**.

Auf diese Weise füllen wir mit der Prozedur schnell das TreeView mit allen Datensätzen der beiden Tabellen **tblKunden** und **tblProjekte**.

## Kontextmenü anzeigen

Wenn es keine Eigenschaft gibt, mit der wir ein Kontextmenü zuweisen können und auch kein spezielles Ereignis vorliegt, das dies erledigen würde – wie zeigen wir das Kontextmenü dann an?

Das erledigen wir über einen kleinen Umweg, nämlich über das Ereignis, dass durch das Herunterdrücken der rechten Maustaste ausgelöst wird. Auch dafür gibt es kein spezielles Ereignis, daher nutzen wir das Ereignis,

das beim Herunterdrücken beliebiger Maustasten ausgelöst wird: nämlich **Bei Maustaste ab**. Bei ActiveX-Steuerelementen finden wir dieses Ereignis nicht im Eigenschaftsfenster, daher legen wir es direkt über den VBA-Editor an.

Dazu wählen Sie im linken Kombinationsfeld den Namen des Steuerelements aus, hier **ctlTreeView**, und dann im rechten Kombinationsfeld den Eintrag **MouseDown**.

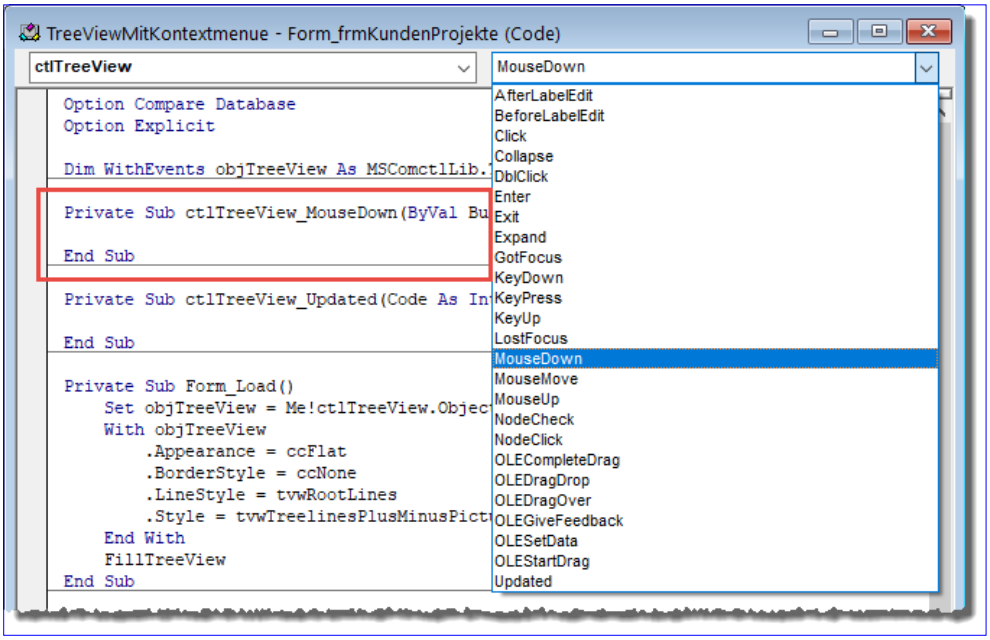



**Bild 6:** Anlegen der Ereignisprozedur für das Ereignis **Bei Maustaste ab**

Dies legt dann die Ereignisprozedur **ctlTreeView_MouseDown** an (siehe Bild 6).

## Auf rechten Mausklick reagieren

Die Ereignisprozedur füllen wir wie in Listing 2. Sie nimmt unter anderem mit X und Y die aktuelle Position des Maus-

```
Private Sub ctlTreeView_MouseDown(ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Long, ByVal Y As Long)
    Dim strTyp As String
    Dim lngID As Long
    Dim strText As String
    Dim objNode As MSComctlLib.Node
    Select Case Button
        Case acRightButton
            Set objNode = objTreeView.HitTest(X, Y)
            If Not objNode Is Nothing Then
                strTyp = Left(objNode.Key, 1)
                lngID = Mid(objNode.Key, 2)
                strText = objNode.Text
                Select Case strTyp
                    Case "k"
                        MsgBox "Rechtsklick auf Kunde '" & strText & "'"
                    Case "p"
                        MsgBox "Rechtsklick auf Projekt '" & strText & "'"
                End Select
            Else
                MsgBox "Rechtsklick auf leeren Bereich"
            End If
    End Select
End Sub
```

**Listing 2:** Diese Prozedur reagiert auf Klicks mit der rechten Maustaste.

# TreeView mit Ribbon-Einträgen

**Die Programmierung des TreeView-Steuerelements haben wir bereits in vielen Beiträgen dokumentiert. Dort haben wir auch gezeigt, wie Sie Befehle für spezielle Elementtypen im TreeView in Kontextmenüs unterbringen, die beim Rechtsklick auf die jeweiligen Elemente angezeigt werden. Es gibt noch eine andere Möglichkeit, solche Befehle abhängig vom Elementtyp abzubilden: als Ribbon-Einträge. Wie das gelingt, zeigt der vorliegende Beitrag.**

## Beispieldatenbank

Im Beitrag **TreeView mit Kontextmenü** (**www.access-im-unternehmen.de/1243**) haben wir bereits Vorarbeiten für diesen Beitrag geleistet und ein funktionsfähiges **TreeView**-Steuerelement mit den Daten aus den beiden Tabellen **tblKunden** und **tblProjekte** gefüllt.

Dort haben wir beschrieben, wie Sie die Daten im TreeView mit Kontextmenü-Einträgen anpassen können. Im vorliegenden Beitrag wollen wir nun zeigen, wie Sie die entsprechenden Befehle in einem Ribbon unterbringen, das nur dann angezeigt wird, wenn der Benutzer auf ein Element eines bestimmten Typs klickt – also beispielsweise auf ein Kunden-Element oder ein Projekt-Element.

Das Ribbon soll dann die gleichen Befehle anzeigen, die auch das Kontextmenü zu dem entsprechenden Eintrag anzeigt und die gleichen Funktionen auslösen (siehe Bild 1). Wir beginnen mit der Beispieldatenbank **TreeViewMitRibbon.accdb**.

## Tabelle für die Ribbon-Definitionen anlegen

Wenn Sie Ribbons anzeigen möchten, benötigen Sie eine Tabelle namens **USysRibbons**, die nach einem bestimmten Schema aufgebaut ist und die Ribbon-Definitionen aufnimmt. Diese Tabelle enthält die folgenden drei Felder:

- **ID**: Primärschlüsselfeld der Tabelle mit Autowert-Funktion
- **Ribbonname**: Name des Ribbons

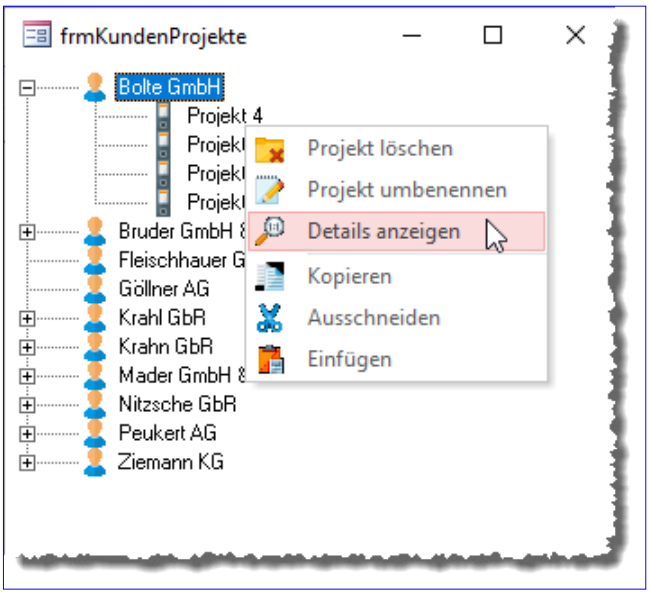



**Bild 1:** Die Befehle dieses Kontextmenüs wollen wir im Ribbon anzeigen, wenn der Benutzer einen der Einträge im TreeView anklickt.

- **RibbonXML**: XML-Definition des Ribbons

Die Tabelle mit dem Namen **USysRibbons** wird aufgrund des Präfixes **USys...** als Systemobjekt erkannt und wird nur angezeigt, wenn Sie in den Navigationsoptionen von Access die Option **Systemobjekte anzeigen** aktivieren.

Diese Tabelle füllen wir dann gleich mit den gewünschten Ribbon-Definitionen (siehe Bild 2). Mit der ersten möchten wir erreichen, dass beim Anklicken eines der **Kunde**-Elemente im **TreeView**-Steuerelement ein kontextabhängiges Ribbon-Tab zum Ribbon hinzugefügt wird.

Deshalb deklarieren wir dieses wie in Listing 1. Das Element **customUI** enthält zwei Callback-Attribute:

- **onLoad**: Dieses Callback wird einmalig beim Laden der Ribbon-Definition ausgeführt. Die Callback-Funktion dient meist dazu, eine Variable des Typs **IRibbonUI** mit einem Verweis auf die Ribbon-Definition zu füllen, was auch hier der Fall ist.

- **loadImage**: Dieses Callback wird ausgelöst, wenn die Ribbon-Definition Elemente enthält, für die das Attribut **image**

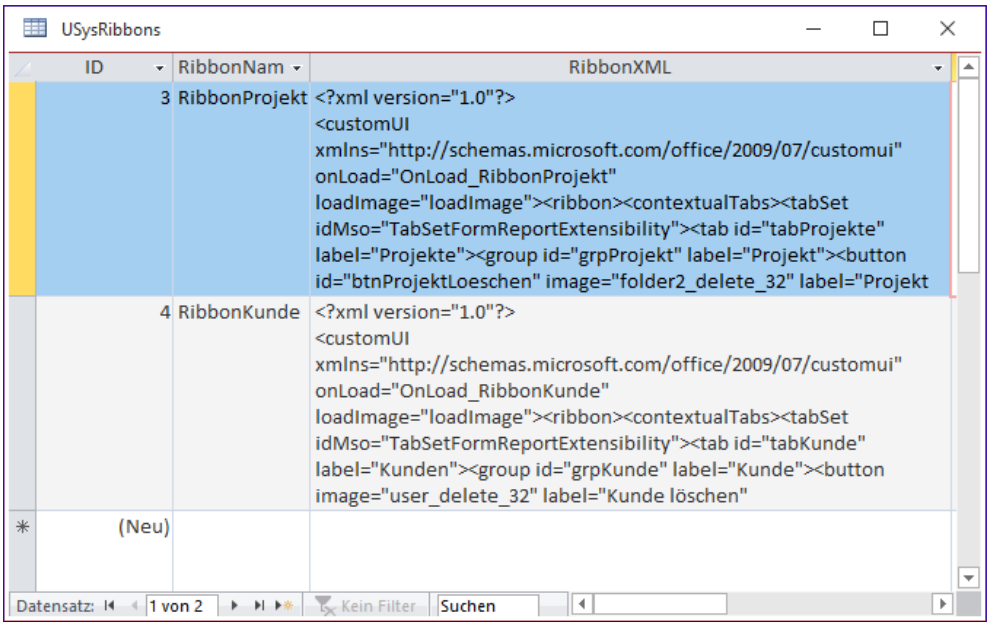



**Bild 2:** Die Tabelle **USysRibbons** mit den Ribbon-Definitionen

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui" onLoad="OnLoad_RibbonKunde"
     loadImage="loadImage">
  <ribbon>
    <contextualTabs>
      <tabSet idMso="TabSetFormReportExtensibility">
        <tab id="tabKunde" label="Kunden">
          <group id="grpKunde" label="Kunde">
            <button image="user_delete_32" label="Kunde löschen" id="btnKundeLoeschen" onAction="onAction"
                size="large"/>
            <button image="folder2_plus_32" label="Neues Projekt" id="btnNeuesProjekt" onAction="onAction"
                size="large"/>
            <button onAction="onAction" size="large" image="edit_32" label="Umbenennen" id="btnKundeUmbenennen"/>
          </group>
          <group id="grpZwischenablage" label="Zwischenablage">
            <button id="btnAusschneiden" image="cut_32" label="Ausschneiden" onAction="onAction" size="large"/>
            <button id="btnKopieren" image="copy_32" label="Kopieren" onAction="onAction" size="large"/>
            <button image="clipboard_paste_32" label="Einfügen" id="btnEinfuegen" onAction="onAction" size="large"/>
          </group>
        </tab>
      </tabSet>
    </contextualTabs>
  </ribbon>
</customUI>
```

**Listing 1:** Definition des ersten Ribbons, das beim Anklicken eines Kunden angezeigt werden soll

auf ein anzuzeigendes Bild eingestellt ist. Sie muss durch den Entwickler gefüllt werden, damit das Ribbon-Element das gewünschte Bild anzeigt.

Das ribbon-Element könnte man mit dem Attribut **startFromScratch="true"** ausstatten, wenn man möchte, dass alle anderen und eingebauten Elemente des Ribbons beim Anzeigen ausgeblendet werden. Wir wollen dem Benutzer aber hier die Möglichkeit geben, dennoch die Befehle in den übrigen Ribbon-Tabs zu verwenden.

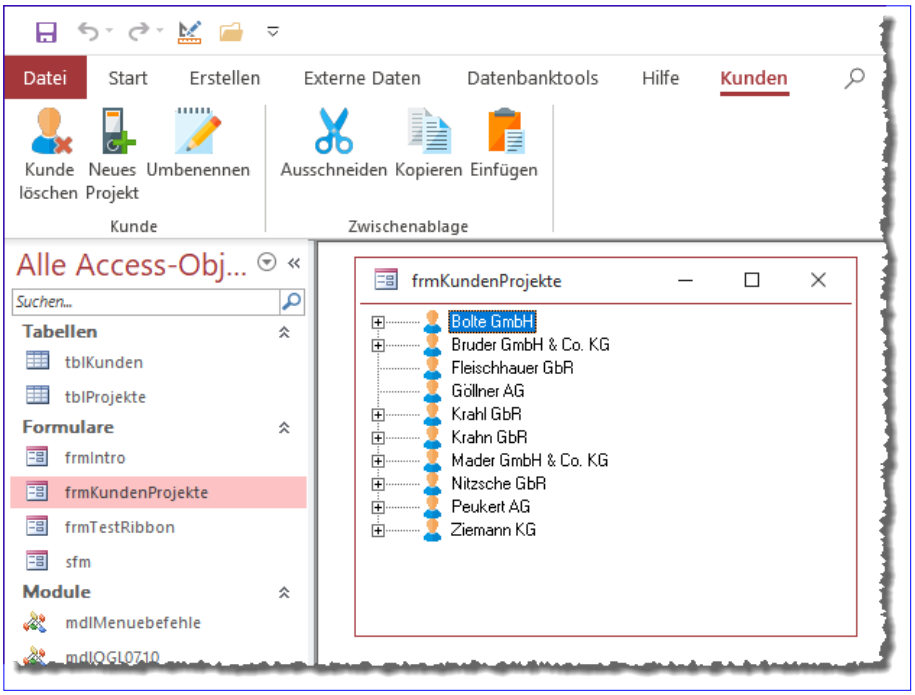



**Bild 3:** Das Ribbon zu den Kundeneinträgen im **TreeView**-Steuerelement

Um Ribbon-Tabs so anzuzeigen, dass sie nur im Kontext mit einem bestimmten Formular oder Element erscheinen, verwenden wir nicht die üblicherweise genutzte **tabs**-Auflistung, sondern das Element **contextualTabs**. Dieses geht meist einher mit dem untergeordneten Element **tabSet**, dem wir über das Attribut **idMso** mit dem Wert **TabSetFormReportExtensibility** die Information mitgeben, dass es gemeinsam mit Formularen oder Berichten angezeigt wird.

Danach folgt ein herkömmliches **tab**-Element mit der Beschriftung **Kunden**, darunter zwei **group**-Elemente. Das erste erhält die Beschriftung **Kunde**, das zweite die Beschriftung **Zwischenablage**. Die erste Gruppe enthält die drei Befehle **Kunde löschen**, **Neues Projekt** und **Umbenennen**. Für alle **button**-Elemente legen wir über das Attribut **image** den Namen des für diese Schaltfläche anzuzeigenden Bildes fest.

Außerdem stellen wir mit **size="large"** ein, dass große Schaltflächen angezeigt werden sollen. Schließlich erhalten alle Schaltflächen das Attribut **onAction** mit dem Namen der Callback-Funktion, die beim Anklicken der Schaltfläche aufgerufen werden soll. Warum erhalten alle Schaltflächen den Aufruf der gleichen Callback-Funktion namens **onAction**? Weil diese als Parameter den Namen des aufrufenden Elements erhält und wir in dieser Prozedur prüfen, von welchem Element der Aufruf kommt und die entsprechenden Befehle ausführen.

Dies soll ein Ribbon wie in Bild 3 ergeben. In den folgenden Abschnitten sehen wir uns an, wie wir das realisieren.

## Ribbon-Verweis speichern

Wie oben erwähnt, soll beim Laden des Ribbons ein Verweis auf die Ribbon-Definition in einer Variablen gespeichert werden. Für diesen und anderen Ribbon-relevanten Code legen wir ein neues Modul namens **mdlRibbons** an. Diesem fügen wir als Erstes eine Variable für unser soeben definiertes Ribbon hinzu:

```
Public objRibbon_RibbonKunde As IRibbonUI
```

Dann können wir auch schon die Callback-Funktion implementieren, die wir für das Attribut **onLoad** des **customUI**-

```
Public Sub loadImage(control, ByRef image)
    Dim lngID As Long
    On Error Resume Next
    lngID = Nz(DLookup("ID", "MSysResources", "Name = '" & control & "'"), 0)
    If Err.Number = 3078 Then
        MsgBox "Die Tabelle 'MSysResources' mit den Images für die Anzeige im Ribbon fehlt." & vbCrLf _
            & "Fügen Sie die Images mit dem Ribbon-Admin hinzu", vbOKOnly + vbExclamation, _
                "Tabelle MSysResources fehlt"
        Exit Sub
    End If
    On Error GoTo 0
    If lngID = 0 Then
        MsgBox "Das Image '" & control & "' ist nicht in der Tabelle MSysResources vorhanden. " & vbCrLf _
            & "Fügen Sie dieses über den Kontextmenüeintrag 'Benutzerdefiniertes Image hinzufügen' " & vbCrLf _
            & "des image-Attributs des entsprechenden Ribbon-Steuerelements hinzu."
    Else
        Set image = PicFromSharedResource_Ribbon(CStr(control))
    End If
End Sub
```

**Listing 2:** Callback-Prozedur zum Laden der Bilder in die Ribbon-Schaltflächen

Elements der Ribbon-Definition angegeben haben. Diese sieht wie folgt aus:

```
Sub onLoad_RibbonKunde(ribbon As IRibbonUI)
    Set objRibbon_RibbonKunde = ribbon
End Sub
```

## Callback zum Zuweisen von Bildern

Danach legen wir die Callback-Prozedur an, die wir für das Callback-Attribut **loadImage** hinterlegt haben. Diese finden Sie in Listing 2. Die Prozedur erhält den für das Attribut image des jeweiligen Steuerelements angegebenen Namen des Bildes.

Sie wird für jedes Steuerelement, das dieses Attribut enthält, einmal aufgerufen. Die Prozedur greift auf Bilddateien zu, die in der Tabelle **MSysResources** hinterlegt sein müssen. Wie Sie diese dort hinterlegen, haben wir bereits im Beitrag **TreeView mit Kontextmenü** (**www.access-im-unternehmen.de/1243**) kurz beschrieben – in diesem Fall verwenden wir allerdings Bilder im Format 32 x 32 statt 16 x 16.

Die Funktion prüft mit einem Aufruf der **DLookup**-Funktion, ob sich in der Tabelle **MSysResources** ein Datensatz befindet, dessen Feld **Name** den Namen des Bildes aufweist und schreibt den gegebenenfalls gefundenen Primärschlüsselwert in die Variable **lngID**. Die Datensätze in der Tabelle **MSysResources** sehen etwa wie in Bild 4 aus.

Hier kann es zu einem Fehler kommen, wenn die Tabelle **MSysResources** gar nicht vorhanden ist. Der Fehler hat die Nummer **3078**. Tritt dieser Fehler auf, zeigt die Prozedur eine entsprechende Fehlermeldung an und die Prozedur wird beendet.

Anderenfalls prüft die Prozedur, ob **lngID** den Wert **0** hat. Das ist der Fall, falls die **DLookup**-Funktion keinen passenden Datensatz gefunden hat.

Auch in diesem Fall erscheint eine Meldung, die tunlichst während der Entwicklung der Anwendung auftreten sollte, damit der Entwickler die fehlenden Bilddateien zur Tabelle **MSysResources** hinzufügen kann.

Ist **lngID** jedoch nicht **0**, dann wurde ein passender Datensatz gefunden. Dann übergeben wir den Namen des Steuerelements an die Funktion **PicFromSharedResource_Ribbon**, welche ein Objekt des Typs **StdPicture** zurückgibt. Dieses wird dann als Ergebnis der Callback-Funktion dem auslösenden Steuerelement zugewiesen.

Die Funktion **PicFromSharedResource_Ribbon** und einige andere nützliche Bildfunktionen finden Sie im Modul **mdlRibbonImages**.

Damit haben wir nun folgende Dinge erledigt:

- Das Ribbon wird beim Aufruf über die Prozedur **onLoad** mit der Variablen **objRibbon_RibbonKunde** referenziert und
- die Steuerelemente werden durch die Funktion **loadImage** mit den zugeordneten Bildern gefüllt.

MSysResources

| | Extension | Id | Name | Type |
|---|---|---|---|---|
| (1) | thmx | 1 | Office Theme | thmx |
| (1) | png | 2 | user_16_delete | img |
| (1) | png | 3 | folder_plus | img |
| (1) | png | 4 | alarm | img |
| (1) | png | 5 | folder_delete | img |
| (1) | png | 6 | user_16_new | img |
| (1) | png | 7 | edit | img |
| (1) | png | 8 | view_1_1 | img |
| (1) | png | 9 | copy | img |
| (1) | png | 10 | cut | img |
| (1) | png | 11 | clipboard_paste | img |
| (1) | png | 36 | user_delete_32 | img |
| (1) | png | 37 | folder2_plus_32 | img |
| (1) | png | 41 | folder2_delete_32 | img |
| (1) | png | 42 | edit_32 | img |
| (1) | png | 43 | clipboard_paste_32 | img |
| (1) | png | 44 | cut_32 | img |
| (1) | png | 45 | copy_32 | img |
| (0) | | (Neu) | | |

Datensatz: 1 von 18 | Kein Filter | Suchen

**Bild 4:** Die Tabelle **MSysResources** mit den Bildern für das Ribbon

## Ribbon anzeigen, wenn Kunde den Fokus im TreeView-Steuerelement erhält

Nun geht es weiter. Wir müssen dafür sorgen, dass dieses Ribbon angezeigt wird, sobald der Benutzer im **TreeView**-Steuerelement des Formulars **frmKundenProjekte** auf eines der **Kunden**-Elemente klickt.

Aber wie wollen wir das erledigen? Immerhin gibt es nur die folgenden beiden Möglichkeiten, überhaupt für die Anzeige von Ribbons zu sorgen, deren Definitionen in der Tabelle **USysRibbons** gespeichert sind:

- Durch Zuweisen des Namens des Ribbons an die Option **Name des Menübands** im Dialog **Access-Optionen** (siehe Bild 5) – dies stellt das Ribbon ein, das direkt nach dem Start von Access angezeigt werden soll –, oder
- durch Zuweisen des Namens des Ribbons an die Eigenschaft **Name des Menübands** eines Formulars oder Berichts. Dies stellt ein, welches Ribbon angezeigt werden soll, wenn das Formular geöffnet wird.

Die erste Option fällt flach, weil man diese zwar zur Laufzeit neu einstellen kann, die Änderung aber erst nach einem Neustart der Access-Anwendung wirksam wird.

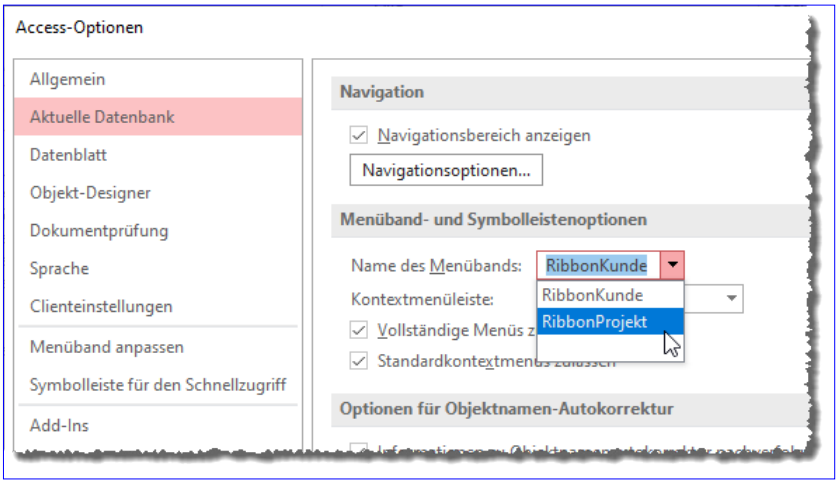



**Bild 5:** Einstellen des Anwendungsribbons

# Formulare zur Laufzeit analysieren

**Wenn Sie selbst eine Anwendung programmieren und ein Formular nicht so funktioniert wie gewünscht, wissen Sie, wie die Steuerelemente heißen, an welche Datenquellen sie gebunden sind und so weiter. Vielleicht ist die Entwicklung der Anwendung auch etwas länger her und Sie müssen sich erst wieder einarbeiten – oder Sie erhalten von einem Kunden eine Anwendung mit der Bitte, dort eine Funktion in einem Formular zu überprüfen. In den beiden letzteren Fällen ist es recht mühsam, zwischen Formular- und Entwurfsansicht zu wechseln, um Steuerelementnamen, Datenherkünfte und andere Informationen herauszufinden, die zur Lösung des Problems beitragen könnten. Der vorliegende Beitrag zeigt, wie Sie ein Formular schnell mithilfe eines zur Laufzeit hinzugefügten Kontextmenüs analysieren können, um schneller Lösungen zu finden.**

Die üblichen Mittel, die einem zur Verfügung stehen, um Informationen über das Formular, die enthaltenen Steuerelemente oder ihre Eigenschaften zu ermitteln, sind die Elemente der **Screen**-Klasse. Dieses bietet Eigenschaften wie **ActiveForm** für das Ermitteln eines Verweises auf das aktuelle Formular oder **ActiveControl** für das Ermitteln des aktuellen Steuerelements.

Den Namen des aktuellen Formulars können Sie so beispielsweise über das Direktfenster ermitteln, ohne in die Entwurfsansicht wechseln zu müssen. Dazu geben Sie den folgenden Befehl ein:

```
Debug.Print Screen.ActiveForm.Name
```

Vielleicht wollen Sie auch auf die Eigenschaften des aktuellen Steuerelements zugreifen. Dann verwenden Sie **ActiveControl**:

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

Wenn Sie dann auf eine spezielle Eigenschaft des Formulars oder Steuerelements zugreifen wollen, die Sie nicht genau kennen, weil Sie diese sonst immer per IntelliSense eingeben, wird es allerdings schwierig, weil zumindest **Screen.ActiveControl** nur einige allgemeine Eigenschaften liefert.

## Geplante Lösung

Was wir nun machen wollen, ist die Bereitstellung aller Eigenschaften eines Formulars oder Steuerelements über das Kontextmenü des jeweiligen Elements. Wenn der Entwickler also mit der rechten Maustaste auf ein Formular oder Steuerelement klickt, soll ein Kontextmenü erscheinen, das zusätzlich zu den eingebauten Befehlen noch weitere Einträge enthält – einen, der direkt den Namen des Elements anzeigt und einen, der in einem Untermenü alle Eigenschaften des Elements samt Werten liefert.

Das soll etwa wie in Bild 1 aussehen. Diese Kontextmenü-Einträge wollen wir für alle Elemente des Formulars liefern – also sowohl für das Formular selbst, für die Steuerelemente als auch für die Elemente in Unterformularen, unabhängig davon, ob es sich um die Datenblatt- oder die Formularansicht handelt.

## Realisierung

Im aufwendigsten Fall fügt man für jedes Formular und für jedes Steuerelement eine Ereignisprozedur hinzu, die durch das Ereignis **Bei Maustaste ab** ausgelöst wird.

In dieser Prozedur prüft man dann, ob der Benutzer die linke oder die rechte Taste gedrückt hat. Im Falle der rechten Maustaste soll dann das nun anzuzeigende Kon-

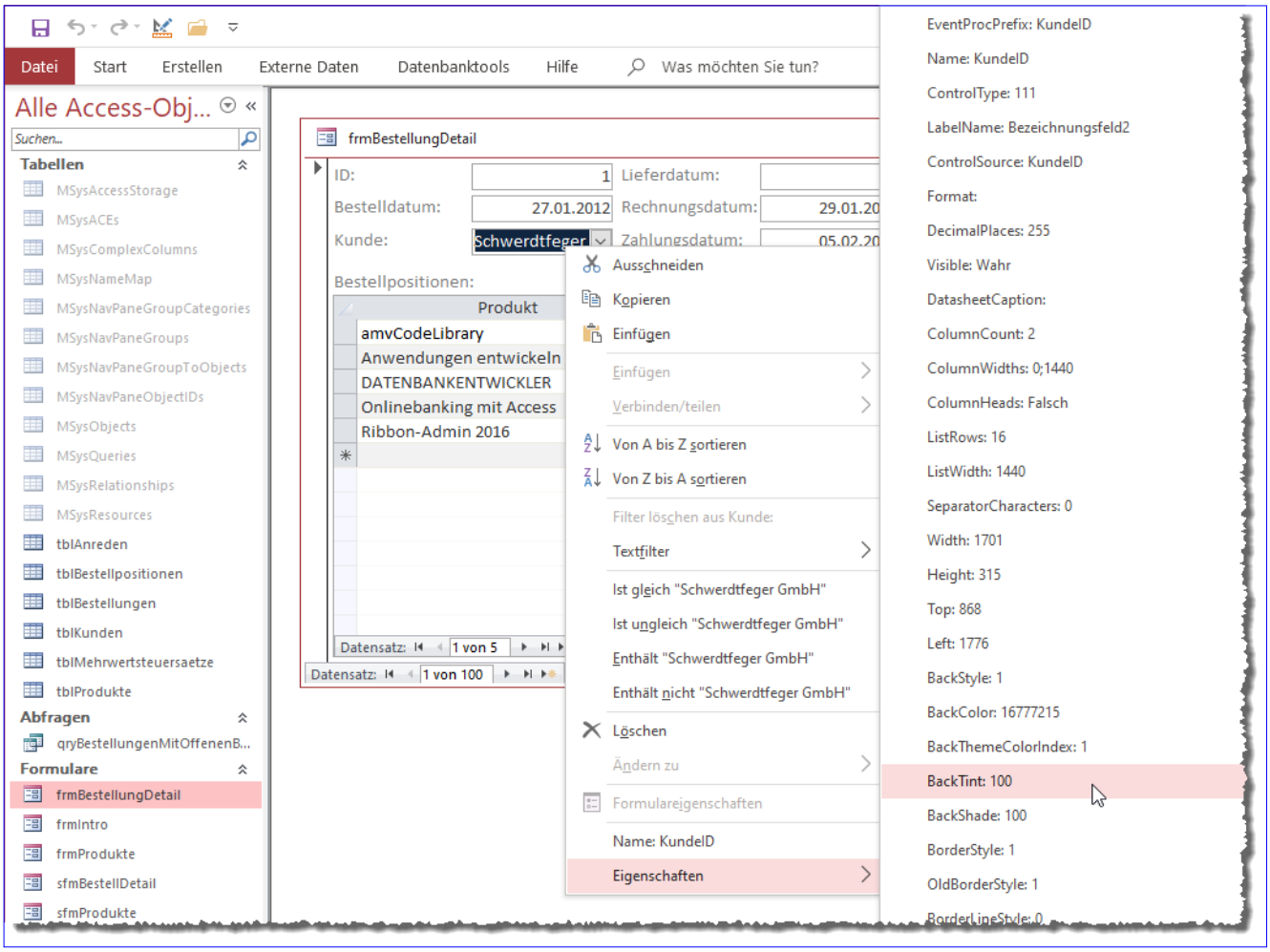



**Bild 1:** Anzeige der Eigenschaften per Kontextmenü

textmenü erweitert werden, und zwar um ein Element mit dem Namen des Formulars oder Steuerelements und mit einem Untermenü mit den Eigenschaften des Elements.

Allerdings müsste man das dann für alle Formulare und Steuerelemente machen, die man untersuchen möchte – und das wollen wir nicht. Stattdessen arbeiten wir, wie schon des Öfteren, mit zwei Klassenmodulen, denen wir die Formulare und Steuerelemente zuweisen und welche die notwendigen Ereignisprozeduren bereits enthalten.

Diese schauen wir uns nun an – zuerst das Klassenmodul, mit dem wir das Formular referenzieren und das die Kontextmenü-Befehle für dieses bereitstellt.

## Klassenmodul für die Kontextmenüs in Formularen

Dieses Klassenmodul nennen wir **clsForm**.

Das Formular verwendet die folgenden Variablen:

```
Private WithEvents m_frm As Form
Private WithEvents m_Detail As Section
Public colControls As Collection
Private m_IsSubform As Boolean
```

Die beiden Variablen **m_frm** und **m_Detail** werden mit dem Schlüsselwort **WithEvents** ausgestattet, wodurch wir in dem Klassenmodul, in dem die Objekte deklariert

sind, Ereignisprozeduren für diese Objekte implementieren können.

Die Klasse stellt eine öffentliche Eigenschaft namens **Form** zur Verfügung, die das mit den Kontextmenüs auszustattende Formular entgegennimmt und die wie folgt aussieht:

```
Public Property Set Form(frm As Form)
    Dim objControl As clsControl
    Dim ctl As Control
    Set m_frm = frm
    With m_frm
        .OnMouseDown = "[Event Procedure]"
    End With
    Set m_Detail = Form.Section(0)
    With m_Detail
        .OnMouseDown = "[Event Procedure]"
    End With
    Set colControls = New Collection
    For Each ctl In m_frm.Controls
        Set objControl = New clsControl
        With objControl
            .IsSubform = m_IsSubform
            Set .Control = ctl
            colControls.Add objControl
        End With
    Next ctl
End Property
```

Diese **Property Set**-Prozedur speichert zunächst den mit **frm** übergebenen Verweis auf das betreffende Formular in der Variablen **m_frm**. Dann stellt es für die Ereigniseigenschaft **Bei Maustaste ab** (**OnMouseDown**) des Formulars den Wert **Ereignisprozedur** ein (**[Event Procedure]**). Danach ermittelt die Prozedur den Detailbereich des Formulars und referenziert diesen mit der Variablen **m_Detail**. Auch dafür stellt sie den Wert **[Event Procedure]** für die Eigenschaft **OnMouseDown** ein. Danach durchläuft die Methode die Steuerelemente im Formular und legt für jedes Steuerelement eine neue Instanz der Klasse **clsControl**, die wir weiter unten erläutern, an und weist der Eigenschaft **Control** dieses Objekts das aktuelle Steuerelement zu. Gegebenenfalls befinden wir uns gerade in einem Unterformular, dann wird noch der Wert **True** für die Eigenschaft **IsSubform** übergeben. Schließlich landen alle Objekte auf Basis der Klasse **clsControl** in der Collection **colControls**.

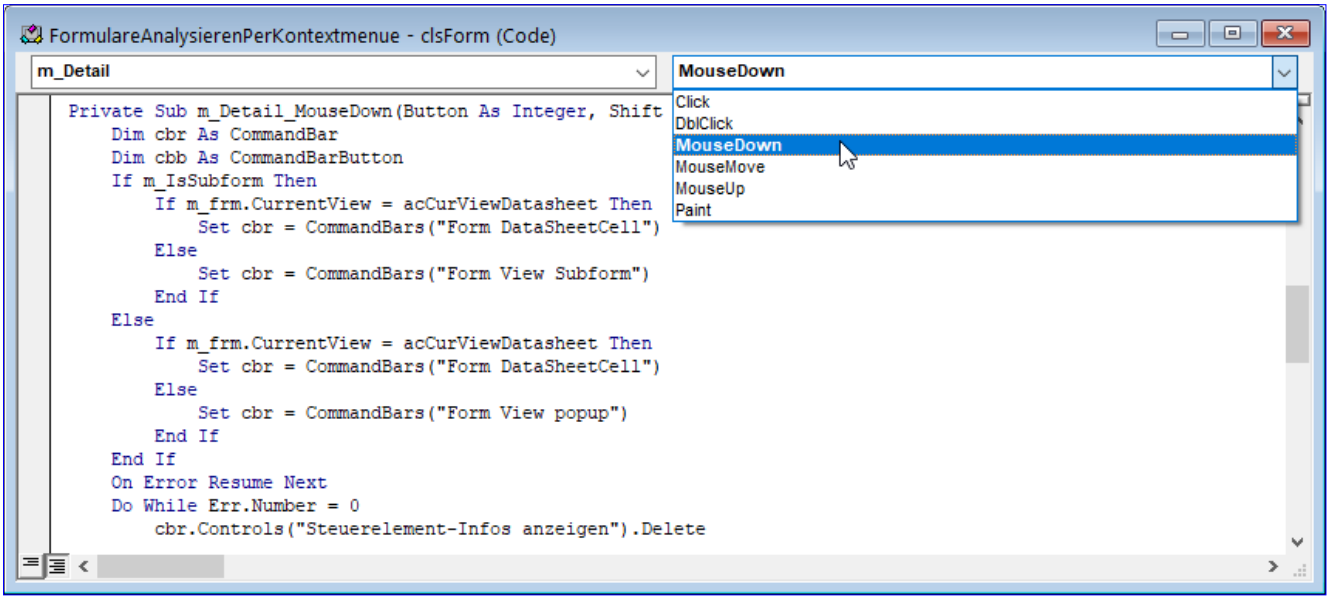



**Bild 2:** Anlegen einer Ereignisprozedur für eine Objektvariable

### Rechter Mausklick auf Formular oder Detailbereich

Wenn der Benutzer mit der rechten Maustaste auf das Formular oder den Detailbereich klickt, löst er die Prozedur **m_Detail_MouseDown** aus. Diese legen Sie an, indem Sie im Codefenster der Klasse **clsForm** im linken Kombinationsfeld den Eintrag **m_Detail** und im rechten den Eintrag **MouseDown** auswählen (siehe Bild 2).

Dadurch wird eine Ereignisprozedur angelegt, die wir wie in Listing 1 ergänzen. Hier deklarieren wir zunächst zwei Variablen, um eine Menüleiste und eine Menüschaltfläche zu referenzieren.

### Kontextmenüs identifizieren

Nun folgen wichtige Schritte, in denen wir festlegen, welches Kontextmenü überhaupt mit unseren zusätzlichen Einträgen versehen werden soll. Access zeigt nämlich immer andere Kontextmenüs an, wenn Sie unterschiedliche Elemente mit der rechten Maustaste anklicken – und deren Namen wir durch den Aufruf einer bestimmten Prozedur ermitteln können. Diese sieht wie folgt aus:

```
Private Sub m_Detail_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
    Dim cbr As CommandBar
    Dim cbb As CommandBarButton
    If m_frm.CurrentView = acCurViewDatasheet Then
        Set cbr = CommandBars("Form DataSheetCell")
    Else
        If m_IsSubform Then
            Set cbr = CommandBars("Form View Subform")
        Else
            Set cbr = CommandBars("Form View popup")
        End If
    End If
    On Error Resume Next
    Do While Err.Number = 0
        cbr.Controls("Steuerelement-Infos anzeigen").Delete
    Loop
    On Error Resume Next
    Do While Err.Number = 0
        cbr.Controls("Eigenschaften").Delete
    Loop
    Set cbb = cbr.Controls.Add(msoControlButton, , , , True)
    With cbb
        If m_IsSubform Then
            .Caption = "Unterformular: " & m_frm.Name
        Else
            .Caption = "Formular: " & m_frm.Name
        End If
    End With
    PropertiesHinzufuegen cbr
End Sub
```

**Listing 1:** Prozedur beim Mausklick auf ein Formular

```
Public Sub ButtonMitNameZuKontextmenuesHinzufuegen()
    Dim cbr As CommandBar
    Dim cbb As CommandBarButton
    For Each cbr In CommandBars
        On Error Resume Next
        Set cbb = cbr.Controls.Add(7
                            msoControlButton, , , , True)
        cbb.Caption = cbr.Name
        If Not Err.Number = 0 Then
            Debug.Print cbr.Name
        End If
        On Error GoTo 0
    Next
End Sub
```

Die Prozedur durchläuft alle Einträge der **CommandBars**-Auflistung von Access und fügt jedem ein Element des Typs **msoControlButton** hinzu, also eine einfache Schaltfläche.

Die Beschriftung des jeweiligen Elements füllen wir mit dem Namen des aktuellen Kontextmenüs. Dadurch zeigt jedes Kontextmenü von nun an als letzten Eintrag seinen Namen an – wie am Beispiel eines Datenblatts zu erkennen (siehe Bild 3).

## Kontextmenü im Kontext des jeweiligen Objekts anpassen

Dann prüfen wir, ob das aktuelle Formular in der Datenblattansicht angezeigt wird, was der Fall ist, wenn die Eigenschaft **CurrentView** den Wert **acCurViewDatasheet** liefert. In diesem Fall erstellen wir ein **CommandBar**-Objekt auf Basis des Menüs **Form DataSheetCell**.

Anderenfalls, also wenn das Formular nicht in der Datenblattansicht angezeigt wird, prüfen wir den Inhalt der Variablen **m_IsSubform**. Ist dieser **True**, befinden wir uns gerade in einem Unterformular. Dann zeigt der rechte Mausklick das Menü **Form View Subform** an, welches wir der Variablen **cbr** zuweisen. Ist es kein Unterformular, verwenden wir das Kontextmenü namens **Form View Popup**.

Danach löschen wir alle eventuell bereits aus vorherigen Anwendungen vorhandenen Schaltflächen mit den Beschriftungen **Steuerelement-Infos anzeigen** und **Eigenschaften** aus dem Kontextmenü. Das erledigen wir bei deaktivierter Fehlerbehandlung in jeweils einer **Do While**-Schleife, die verlassen wird, wenn ein Fehler aufgetaucht ist – und das geschieht, wenn das Löschen der genannten Elemente nicht mehr erfolgreich ist, weil kein passendes Element mehr gefunden werden konnte.

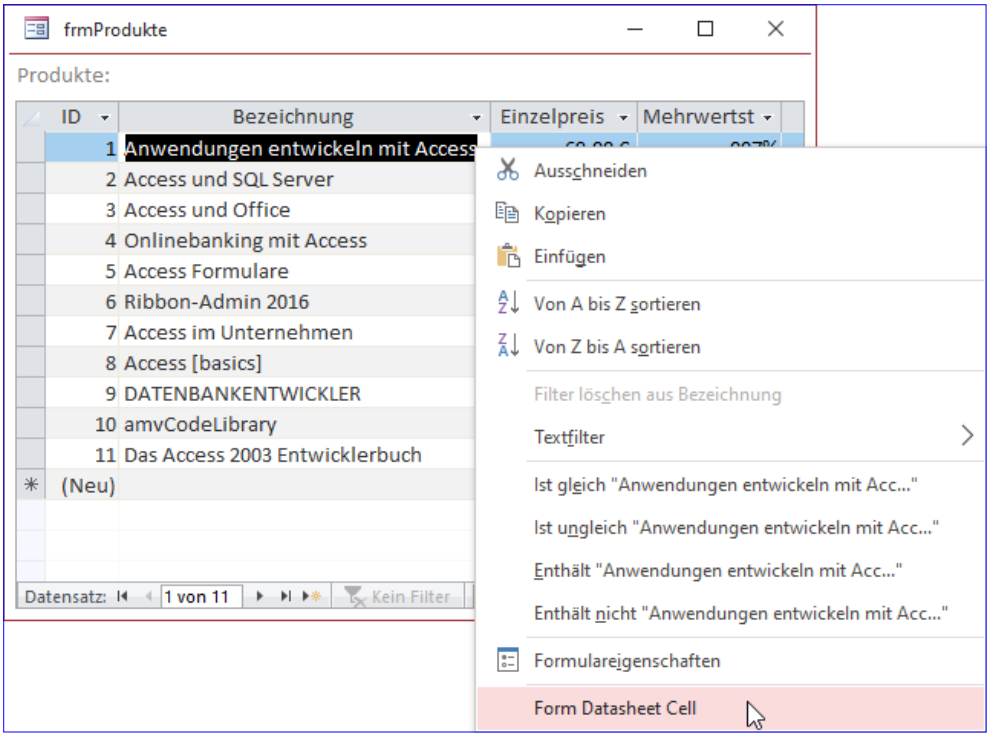



**Bild 3:** Anzeige des Kontextmenü-Namens im Kontextmenü

```
Private Sub PropertiesHinzufuegen(cbr As CommandBar)
    Dim cbp As CommandBarPopup
    Dim prp As DAO.Property
    Dim cbb As CommandBarButton
    Set cbp = cbr.Controls.Add(msoControlPopup, , , , True)
    On Error Resume Next
    Do While Err.Number = 0
        cbr.Controls("Eigenschaften").Delete
    Loop
    On Error GoTo 0
    cbp.Caption = "Eigenschaften"
    For Each prp In m_frm.Properties
        Set cbb = cbp.Controls.Add(msoControlButton, , , , True)
        On Error Resume Next
        cbb.Caption = prp.Name & ": " & prp.Value
        On Error GoTo 0
    Next prp
End Sub
```

**Listing 2:** Prozedur zum Hinzufügen der Eigenschaften samt ihrer Werte

Dann fügen wir mit der **Add**-Methode der **Controls**-Auflistung der Menüleiste aus **cbr** ein neues Element zur **Menüleiste** hinzu und referenzieren dieses mit der Variablen **cbb**.

Für diese legen wir dann die Beschriftung fest, wobei diese wieder vom Wert der Variablen **m_IsSubform** abhängt. Weist diese den Wert **True** auf, haben wir ein Unterformular und geben den Text **Unterformular** gefolgt von einem Doppelpunkt und den aus **m_frm.Name** ermittelten Formularnamen an. Anderenfalls lautet der Text **Formular:** plus dem Formularnamen.

Schließlich rufen wir die Prozedur **PropertiesHinzufuegen** auf und übergeben dieser die Variable mit dem Verweis auf das **CommandBar**-Objekt.

## Eigenschaften in Kontextmenü eintragen

Die Prozedur **PropertiesHinzufuegen** trägt die Eigenschaften des Formulars als Elemente des noch hinzuzufügenden Untermenüs **Eigenschaften** ein (siehe Listing 2). Sie erwartet eine Verweis auf das **CommandBar**-Objekt, dem das Untermenü hinzugefügt werden soll und deklariert die Variablen **cbp** (**CommandBarPopup**, für das Untermenü), **prp** (**Property**, für das Durchlaufen der Eigenschaften des Formulars) und **cbb** (**CommandBarButton**, für die anzulegenden Elemente im Untermenü).

Das **CommandBarPopup**-Element **cbp** legen wir wieder mit der **Add**-Methode der **Controls**-Auflistung des **CommandBar**-Objekts an, diesmal mit dem Typ **msoControlPopup** als Wert des ersten Parameters.

Dann entfernen wir wieder eventuell vorhandene Elemente namens **Eigenschaften** und weisen dem neu erstellten Element genau diesen Namen zu. Schließlich durchlaufen wir alle Eigenschaften der **Properties**-Auflistung des Formulars aus **m_frm** in einer **For Each**-Schleife. Dabei legen wir für jede Eigenschaft ein neues Element im Untermenü an und weisen diesem als Beschriftung den Namen der Eigenschaft (**prp.Name**) und den Wert der Eigenschaft zu (**prp.Value**).

Damit ist das Kontextmenü für das Abfragen der Eigenschaften des Formulars so erweitert, dass es die gewünschten Informationen beim Rechtsklick anzeigt.

### Die Klasse clsControls für die Kontextmenüs der Steuerelemente

In der Klasse, die für jede einzelne Steuerelement des Formulars angelegt wird – und auch für Unterformulare und die darin enthaltenen Steuerelemente –, deklarieren wir zunächst einige Variablen. Als Erstes benötigen wir für jeden gängigen Steuerelementtyp eine Variable, mit der wir das Steuerelement referenzieren können. Diese deklarieren wir mit dem Schlüsselwort **WithEvents**, damit wir wieder das Ereignis **Bei Maustaste ab** im gleichen Klassenmodul implementieren können. Wir deklarieren deshalb für jeden Steuerelementtyp eine eigene Variable, weil diese zwar eine Menge gemeinsamer Eigenschaften aufweisen, jeder Steuerelementtyp aber auch einige individuelle Eigenschaften aufweist:

```
Private WithEvents txt As TextBox
Private WithEvents chk As CheckBox
Private WithEvents cbo As ComboBox
Private WithEvents cmd As CommandButton
Private WithEvents lst As ListBox
Private WithEvents lbl As Label
Private WithEvents opt As OptionButton
Private WithEvents ogr As OptionGroup
Private WithEvents tgl As ToggleButton
Private WithEvents sfm As SubForm
```

Wir haben hier die wichtigsten Steuerelementtypen berücksichtigt. Sie können noch weitere Steuerelementtypen hinzufügen, wenn Sie Bedarf haben.

Außerdem benötigen wie eine Variable, die wir vorübergehend mit dem Verweis auf das Steuerelement füllen:

```
Private m_ctl As Control
```

Mit **objForm** wollen wir die Instanz der Klasse **clsForm** referenzieren, welche die Steuerelement-Klasse **clsControl** angelegt hat:

```
Private objForm As clsForm
```

Und in **m_IsSubform** speichern wir wiederum die Information, ob das Steuerelement sich in einem Unterformular befindet:

```
Private m_IsSubform As Boolean
```

Zum Einstellen der Eigenschaft **m_IsSubform** verwenden wir die folgende **Property Let**-Prozedur:

```
Public Property Let IsSubform(bol As Boolean)
    m_IsSubform = bol
End Property
```

Die **Property Set**-Prozedur **Control** erlaubt das Hinzufügen des Steuerelements zur Eigenschaft **m_ctl**. Sie wird von der übergeordneten Klasse **clsForm** gefüllt, wenn diese die Steuerelemente des Formulars durchläuft und für jedes Steuerelement ein Objekt des Typs **clsControl** anlegt (siehe Listing 3).

Den mit **ctl** übergebenen Verweis auf das Steuerelement speichert die Prozedur in der Variablen **m_ctl**. Dann prüft sie den Typ des Steuerelements über die Eigenschaft **ControlType**. Abhängig davon stellt sie eine der weiter oben vorgestellten Variablen auf das Steuerelement aus **m_ctl** ein. Wenn es sich beispielsweise um ein Textfeld handelt, wird die Variable **txt** mit dem Verweis aus **m_ctl** gefüllt, wenn es sich um ein Kontrollkästchen handelt, landet der Verweis aus **m_ctl** in der Variablen **chk** und so weiter.

Eine etwas andere Behandlung erfährt das Steuerelement, wenn es sich dabei um ein Unterformular handelt, das den Typ **acSubform** aufweist. In diesem Fall stellen wir zwar wiederum die Variable **sfm** auf den Wert von **m_ctl** ein. Danach prüfen wir aber noch anhand der Eigenschaft **HasModule**, ob das Formular ein Klassenmodul hat. Falls nicht, geben wir eine Meldung aus, denn in diesem Fall können wir zwar das Kontextmenü erweitern, aber die dafür implementierte Ereignisprozedur kann nur ausgelöst werden, wenn das als Unterformular verwendete Formular auch ein Klassenmodul enthält.

# SQL Server-Security – Teil 2: Zugriffsberechtigung

**SQL Server bietet eine Vielzahl an Möglichkeiten und Funktionen im Bereich Security – von Zugriffsberechtigungen auf Objekte und Daten über verschiedene Verschlüsselungsmethoden bis hin zur Protokollierung von Datenzugriffen und Datenänderungen. Viele der Funktionen ähneln sich und haben zum Teil die gleichen Auswirkungen. Da ist es nicht einfach, einen Überblick zu erhalten. Also von Anfang an. Als erstes lernen Sie die Sicherheitsarchitektur von SQL Server und die Anmeldung »sa« kennen.**

Der SQL Server arbeitet mit einer mehrstufigen Sicherheitsarchitektur.

Die erste Stufe ist die Authentifizierung des Anwenders am SQL Server selbst. Dies erfolgt mit der sogenannten Anmeldung. Die Anmeldung alleine erlaubt dem Anwender noch keinen Zugriff auf die Daten.

Das wäre auch zu unsicher, schließlich kann eine SQL Server-Instanz mehrere Datenbanken verwalten und nicht jeder Anwender darf alle Daten lesen und ändern.

Der Zugang zu einer Datenbank ist die zweite Stufe der Sicherheitsarchitektur. Dem Anwender wird der Zugang zu einer Datenbank nur gewährt, wenn seine Anmeldung der Datenbank zugeordnet ist. Die Anmeldung ist dann in der Datenbank als Benutzer zu sehen.

Der Benutzer ist die Basis für die dritte Stufe der Sicherheitsarchitektur. Er besitzt die Zugriffsrechte an den Datenbankobjekten, wie Tabellen, Sichten und Gespeicherten Prozeduren, und somit letztendlich die Lese- und Schreibrechte für die Daten.

## Die SQL Server-Sicherheitsarchitektur

Ein Vergleich mit dem echten Leben soll dies verdeutlichen. Angenommen, Sie haben einen Termin in einem größeren Unternehmen. Sie betreten das Firmengebäude und gehen zur Anmeldung. Dort stellen Sie sich vor und teilen der Empfangsdame mit, dass Sie einen Termin mit Herrn Müller von der Abteilung **Einkauf** haben. Sie erhalten ein Namensschild mit der Aufschrift **Besucher** und werden gebeten, es sich in der Sitzgruppe gemütlich zu machen, während Sie auf Herrn Müller warten.

In diesem Vergleich steht das Firmengebäude für die SQL Server-Instanz und Ihr Namensschild mitsamt der Genehmigung, sich in der Lobby aufzuhalten, entspricht der SQL Server-Anmeldung **Besucher**.

Herr Müller lässt nicht lange auf sich warten. Er begrüßt Sie höflich und nimmt Sie mit in die Abteilung **Einkauf**. Dort sollen Sie in seinem Auftrag Daten lesen, einfügen, ändern und löschen.

Der Zutritt zur Abteilung **Einkauf** ist in der SQL Server-Instanz vergleichbar mit der Zuordnung der Anmeldung **Besucher** zur Datenbank **Einkauf**. In der Datenbank existiert jetzt der Benutzer **Besucher**, dem Lese- und Schreibrechte für alle Daten erteilt wurden

Beim Mittagessen treffen Sie Herrn Schmidt. Er braucht noch weitere Auswertungen und Sie sollen doch bitte um 15 Uhr zu ihm in die Personalabteilung kommen. Gegen 15 Uhr gehen Sie zu Herrn Schmidt und erstellen ihm die gewünschten Auswertungen.

Dieser Abteilungswechsel entspricht der Zuordnung der Anmeldung **Besucher** zu einer weiteren Datenbank der SQL Server-Instanz – der Datenbank **Personal**. Dort gibt

es nun auch einen Benutzer namens **Besucher**. Allerdings besitzt dieser nur Leserechte.

Zusammengefasst zeigt das Beispiel eine SQL Server-Instanz mit der Anmeldung **Besucher**. Diese ist den Datenbanken **Einkauf** und **Personal** zugeordnet.

Die Datenbank **Einkauf** enthält den Benutzer **Besucher** mit Lese- und Schreibrechten. In der Datenbank **Personal** gibt es ebenfalls einen Benutzer namens **Besucher**, dieser besitzt jedoch nur Leserechte.

Beide Benutzer existieren nur innerhalb ihrer Datenbanken. Auch wenn beide den gleichen Namen verwenden, haben sie bis auf eine Ausnahme nichts miteinander zu tun. Sie verweisen beide auf die Anmeldung **Besucher** und somit letztendlich auf den Anwender, der diese Anmeldung verwendet.

Das ist das Prinzip der mehrstufigen Sicherheitsarchitektur. Über die Anmeldung erhält ein Anwender Zugang zur SQL Server-Instanz und kann dort in mehreren Datenbanken mit unterschiedlichen Rechten arbeiten.

### SQL Server-Instanzen

Möglicherweise fragen Sie sich, was genau eine SQL Server-Instanz ist. Die Antwort ist recht einfach. Es ist das Ergebnis Ihrer SQL Server-Installation.

Mit der einfachen Installation des SQL Servers erstellen Sie die Standardinstanz. Diese lässt sich dann mit dem Namen des Rechners ansprechen. Lautet der Rechnername **PC01**, verbinden Sie sich mit der SQL Server-Instanz **PC01**.

Sie dürfen die Installation auf dem gleichen Rechner nochmals ausführen. Jedoch müssen Sie jetzt einen Instanznamen angeben. Nennen Sie die Instanz zum Beispiel **Entwicklung**, steht nach der Installation eine weitere eigenständige SQL Server-Instanz unter dem Namen **PC01\Entwicklung** zur Verfügung.

Eine sogenannte benannte Instanz kennen Sie vielleicht von der SQL Server Express Edition, wird diese doch immer mit einem Instanznamen installiert. Der Name einer SQL Server Express Edition auf dem Rechner **PC01** wäre **PC01\SQLEXPRESS**.

Insgesamt können Sie mit einer Lizenz 50 Instanzen auf einem Rechner installieren, eine Standardinstanz und 49 benannte Instanzen. Dabei gibt es zwischen einer Standardinstanz und den benannten Instanzen keine Unterschiede. Jede SQL Server-Instanz bietet den gleichen Funktionsumfang. Sie können dort mehrere Datenbanken verwalten und natürlich auch eigene Anmeldungen festlegen.

Eine Anmeldung namens **Müller** in der SQL Server-Instanz **PC01** lässt sich nur für den Zugang zu dieser Instanz verwenden. Eine Verbindung zur Instanz **PC01\Entwicklung** ist mit dieser Anmeldung nicht möglich.

Existiert in der SQL Server-Instanz **PC01\Entwicklung** ebenfalls eine Anmeldung namens Müller, ermöglicht diese zwar den Zugang zur Instanz **PC01\Entwicklung**, aber eben nicht zur Instanz **PC01**. Die beiden Anmeldungen haben schlicht und ergreifend nichts miteinander zu tun.

Durch die eigenen Anmeldungen lassen sich SQL Server-Instanzen sehr gut zur Trennung von Daten unterschiedlicher Sicherheitsstufen einsetzen. So legen Sie zum Beispiel die Datenbanken mit den Daten der Produktion auf die Standardinstanz und auf eine weitere SQL Server-Instanz namens **PC01\Forschung** die Datenbanken mit den sensiblen Daten der Forschungsabteilung.

Im oben verwendeten Vergleich mit dem Besucher im Unternehmen stellt die Standardinstanz die Zentrale des Unternehmens dar und die Instanz **PC01\Forschung** deren Forschungszentrum. Das Forschungszentrum ist in einem eigenen Gebäude. Wechselt der Besucher von der Zentrale zum Forschungszentrum, muss er sich dort

beim Empfang erneut anmelden.

## Begrifflichkeiten

Die im SQL Server verwendeten Begriffe **Anmeldung** und **Benutzer** sind nicht besonders geeignet für eine Artikelserie zum Thema **Security**. Beide Begriffe haben mehrere Bedeutungen.

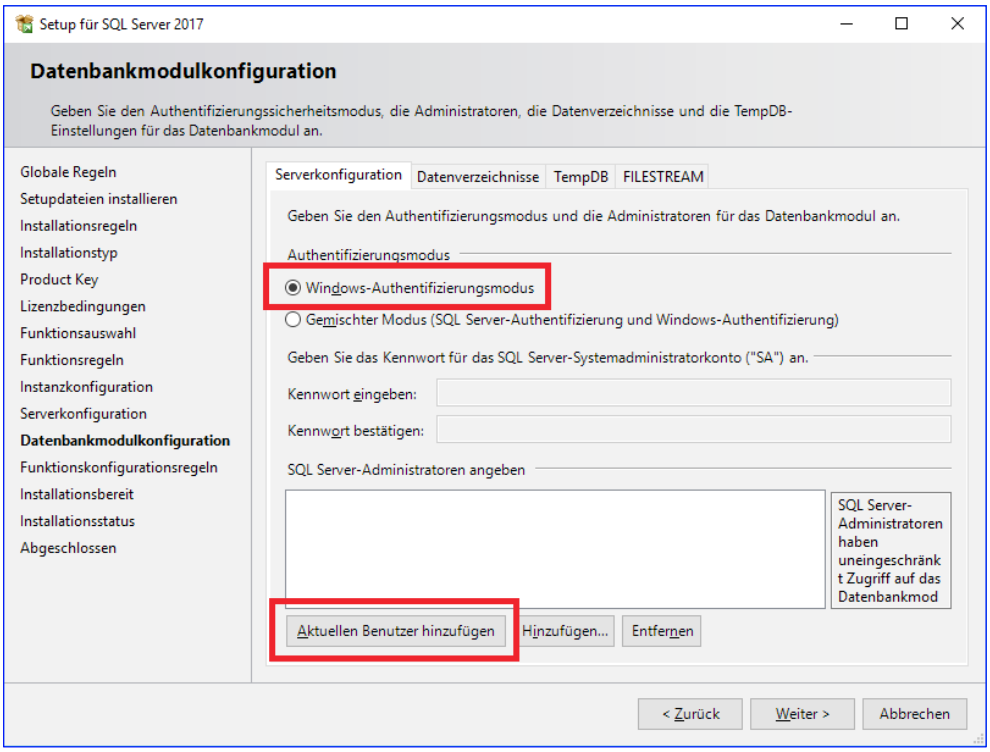



**Bild 1:** Administrator bei der Windows-Authentifizierung

Bei einer Anmeldung kann es sich um eine SQL Server-Anmeldung oder auch um die Anmeldung am Clientrechner handeln. Der Begriff **Benutzer** steht für die Person am Rechner wie für die Zuordnung einer SQL Server-Anmeldung zu einer Datenbank.

Um Missverständnisse zu vermeiden, beschreibt von nun an der Begriff **Anwender** die Person am Rechner und **Benutzerkonto** die von der Person verwendete Anmeldung am Betriebssystem.

Zudem wird der Einfachheit halber ab jetzt der Begriff SQL Server anstatt SQL Server-Instanz verwendet.

## Authentifizierungsmethoden

SQL Server bietet zwei Authentifizierungsmethoden – die Windows-Authentifizierung und die SQL Server-Authentifizierung. Bereits bei der Installation dürfen Sie entscheiden, ob Sie nur die Windows-Authentifizierung oder eine Kombination der beiden nutzen möchten.

Dabei ist die Windows-Authentifizierung als Standard vorausgewählt. Nicht ohne Grund, denn diese Authentifizierungsmethode wird von Microsoft empfohlen. Sie erlaubt als SQL Server-Anmeldungen ausschließlich Benutzerkonten von Windows-Benutzern und Windows-Gruppen.

Für den Anwender ist dies sogar von Vorteil. Er muss sich am SQL Server nicht erneut anmelden. Da sein Benutzerkonto mit einer SQL Server-Anmeldung verbunden ist, erhält er den direkten Zugang zum SQL Server.

Wenn Sie sich bei der Installation für die Windows-Authentifizierung entscheiden, sollten Sie mit der Schaltfläche **Hinzufügen** ein Benutzerkonto als Administrator aufnehmen (siehe Bild 1). Nur mit diesem Benutzerkonto haben Sie nach der Installation entsprechende Administrationsrechte im SQL Server.

Reicht Ihnen die Windows-Authentifizierung nicht aus, können Sie zusätzlich noch die SQL Server-Authentifizierung aktivieren.

Diese Kombination wird **Gemischter Modus** genannt.

Der gemischte Modus erlaubt Ihnen die Definition eigener Anmeldungen mit Benutzernamen und Kennwort. Diese Art der Authentifizierung ist vor allem in Umgebungen notwendig, die keine Windows-Benutzerkonten unterstützen, wie ein SQL Server auf einem Linux-System ohne Anbindung an ein Active Directory oder ein SQL Server als Backend eines Webshops.

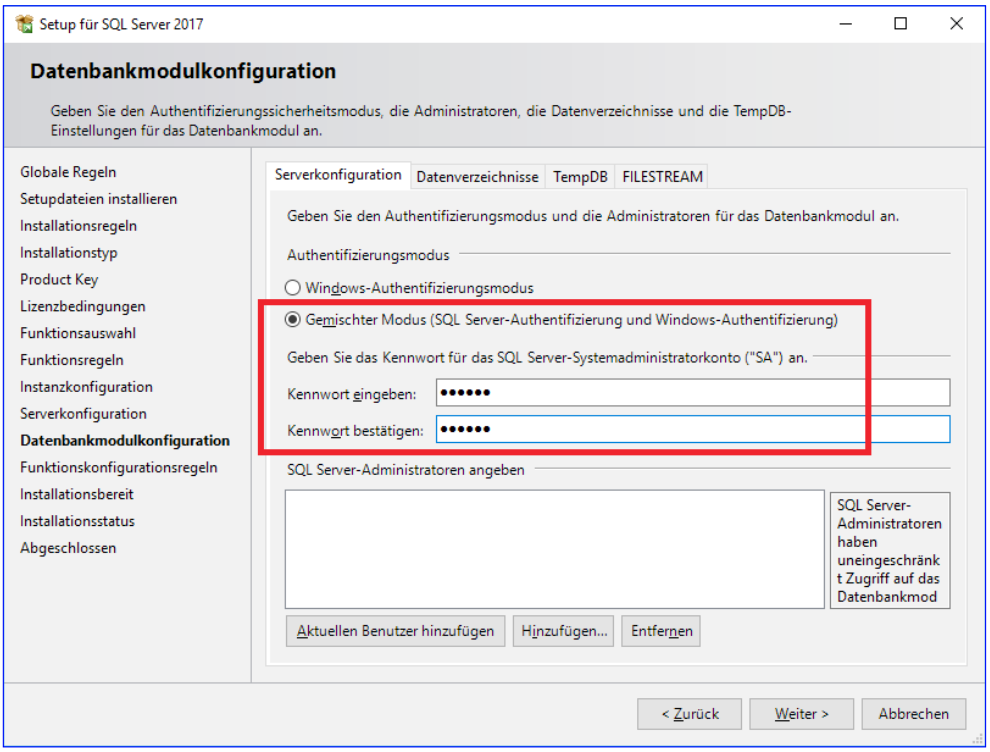



**Bild 2:** **sa**-Konto bei der SQL Server-Authentifizierung

Entscheiden Sie sich für den gemischten Modus, müssen Sie der SQL Server-Anmeldung **sa** ein Kennwort vergeben (siehe Bild 2). **sa** steht für **System Administrator** und beinhaltet alle administrativen Rechte für den SQL Server.

Die Anmeldung **sa** ist jedoch nicht nur den Administratoren vorbehalten. Jeder Anwender kann sie nutzen. Er muss nur das korrekte Kennwort eingeben und schon hat er alle Rechte im SQL Server. Es gibt für ihn dort keinerlei Einschränkungen. Aus diesem Grund sollten Sie unbedingt ein komplexes und starkes Kennwort verwenden.

Jetzt haben Sie vielleicht Ihren SQL Server schon installiert und würden gerne im Nachhinein die Authentifizierungsmethode wechseln. Das ist kein Problem, denn Sie können dies im SQL Server Management Studio jederzeit ändern.

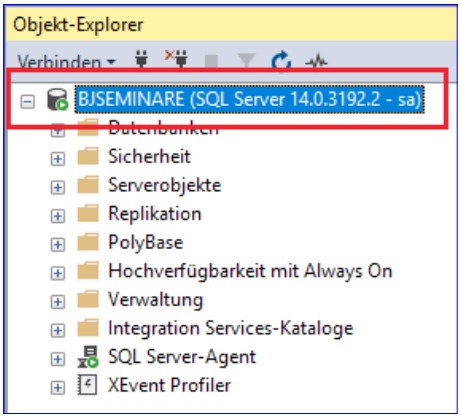



**Bild 3:** Die SQL Server-Instanz im Objekt-Explorer

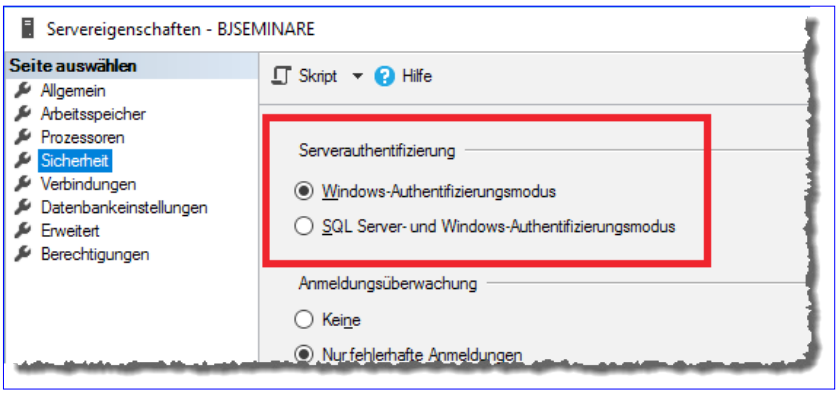



**Bild 4:** Wechsel der Authentifizierungsmethode

Dazu starten Sie das SQL Server Management Studio und melden sich mit der Ihnen bekannten Anmeldung am SQL Server an. Nach der Anmeldung sehen Sie auf der linken Seite den Objekt-Explorer. Dort klicken Sie mit der rechten Maustaste auf den ersten Eintrag, den mit dem Namen Ihres SQL Servers (siehe Bild 3).

Im Kontextmenü wählen Sie dann den Eintrag **Eigenschaften** und wechseln im folgenden Dialog zur Seite **Sicherheit**. Unter **Serverauthentifizierung** können Sie nun zwischen den einzelnen Modi wechseln (siehe Bild 4).

Nach einem Klick auf **OK** werden Sie darauf hingewiesen, dass die Änderungen erst nach einem Neustart des SQL Server-Diensts wirksam sind. Dazu öffnen Sie im SQL Server Management Studio das gleiche Kontextmenü wie eben und wählen dort den Eintrag **Neu starten**.

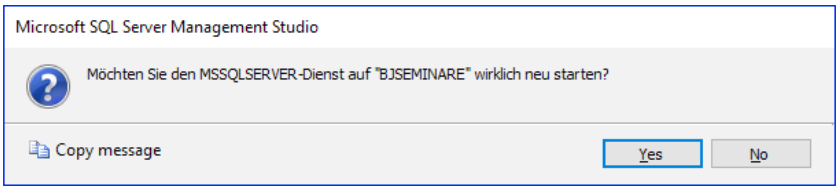



**Bild 5:** Neustart der SQL Server-Instanz

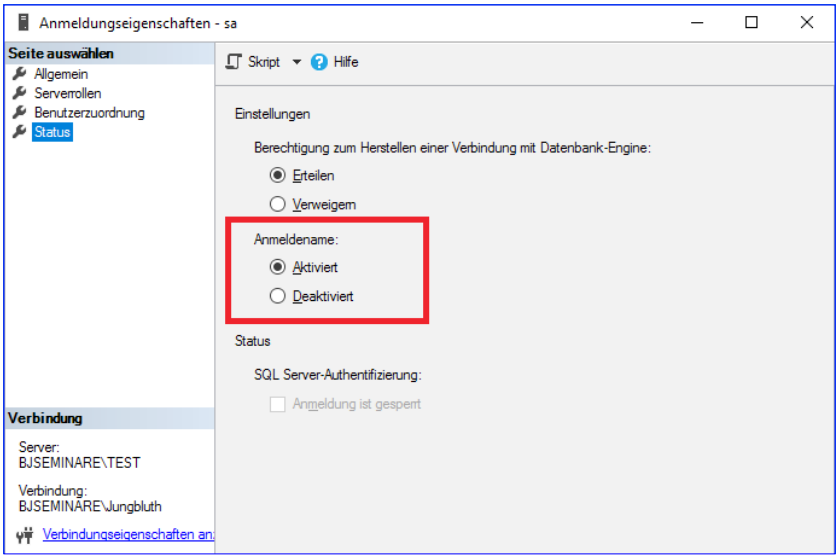



**Bild 6:** Aktivieren der Anmeldung **sa**

Es folgt die Sicherheitsabfrage aus Bild 5. Aus gutem Grund, denn während des Neustarts steht der SQL Server nicht zur Verfügung. Angemeldete Anwender werden in dieser Zeit entsprechende Fehlermeldungen erhalten. Sie sollten also den SQL Server nur dann neu starten, wenn Sie sich sicher sind, dass außer Ihnen niemand sonst angemeldet ist.

Für die Beispiele in diesem Artikel benötigen Sie den gemischten Modus. Verwenden Sie aktuell die reine Windows-Authentifizierung, ändern Sie die Einstellung wie oben beschrieben auf SQL Server- und Windows-Authentifizierung.

Nach dem Wechsel der Authentifizierungsmethode müssen Sie die Anmeldung **sa** noch aktivieren und ihr ein Kennwort vergeben. Dazu navigieren Sie zum Ordner **Sicherheit**, erweitern den Unterordner **Anmeldungen** und öffnen per Doppelklick auf den Eintrag **sa** den Dialog **Anmeldungseigenschaften**.

Hier wechseln Sie zur Seite **Status** und aktivieren unter **Anmeldename** die Option **Aktiviert** (siehe Bild 6). Danach geben Sie der Anmeldung in der Seite **Allgemein** ein Kennwort und speichern die Änderungen mit einem Klick auf **OK**. Jetzt können Sie sich am SQL Server mit der Authentifizierungsmethode **SQL Server-Authentifizierung**, dem Anmeldename **sa** und dem eben erstellten Kennwort anmelden (siehe Bild 7).

## Die SQL Server-Anmeldung »sa«

Entgegen der Empfehlung von Microsoft wird gerade für die Verbindung vom Access-Frontend zum SQL Server-Backend selten die Windows-Authentifizierung verwendet. In vielen Fällen findet dies über die SQL Server-Au-

thentifizierung statt – und dabei oft mit der SQL Server-Anmeldung **sa**.

Es bietet sich ja auch an. Die Anmeldung **sa** ist fester Bestandteil des SQL Servers. Sie ist allen SQL Servern verfügbar und man kann sie nach der Installation direkt verwenden. Zudem gibt es mit dieser Anmeldung im SQL Server keinerlei Einschränkungen.

Bei der Migration von Access nach SQL Server ist dies von Vorteil. Schließlich gibt es genügend Punkte zu beachten und Neues zu lernen. Da möchte man sich nicht mit den Widrigkeiten fehlender Rechte aufhalten.

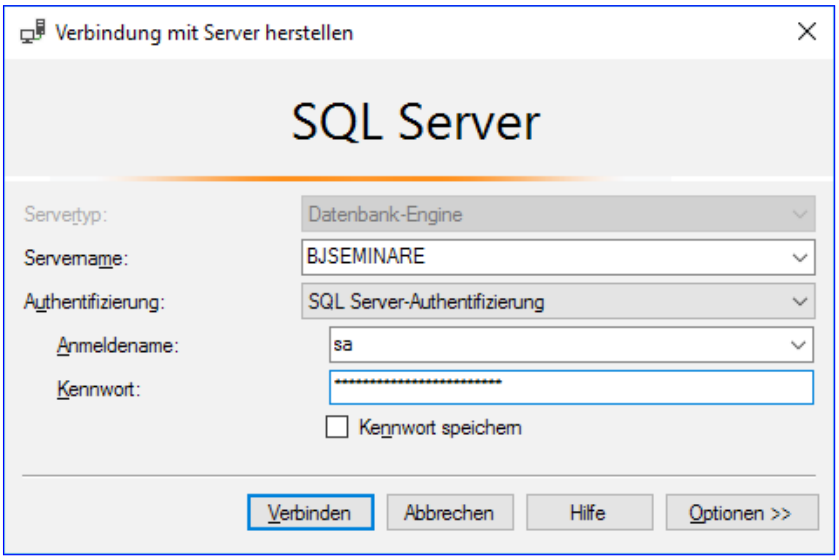



**Bild 7:** Authentifizierung mit der Anmeldung **sa**

Leider bleibt es in vielen Fällen dann bei dieser Authentifizierung. Die **sa**-Anmeldung ist nicht nur in Entwicklungs- und Testumgebungen zu finden, sondern auch in vielen Produktivumgebungen.

Falls Sie in Ihrer Access und SQL Server-Applikation bereits eine andere Anmeldung verwenden, haben Sie schon einen höheren Zugriffsschutz erreicht. Nutzen Sie allerdings die **sa**-Anmeldung, sollten Ihnen die Möglichkeiten klar sein, die Sie Ihren Anwendern damit in die Hand geben. Genau diese werden Sie nun kennenlernen.

Es gibt in Access mehrere Methoden die Daten einer SQL Server-Datenbank zu verarbeiten. In den meisten Fällen findet dies wohl mit eingebundenen Tabellen statt, gefolgt von Pass Through-Abfragen per DAO in VBA und mittels Direktzugriff über ADO. ADO benötigt für den Datenzugriff einen OLE DB-Treiber, für eingebundene Tabellen und Pass Through-Abfragen hingegen ist ein ODBC-Treiber erforderlich.

Beiden Treibern muss für den Datenzugriff neben dem Benutzernamen auch das Kennwort der SQL Server-Anmeldung übergeben werden. Doch wie übergibt man das Kennwort?

Die einfachste Variante verlangt die Kennworteingabe vom Anwender. Dies ist mit Abstand aber auch die unsicherste Variante. Schließlich kann sich der Anwender mit dieser Anmeldung direkt am SQL Server anmelden. Dazu benötigt er nur ein Frontend, wie eine eigene Access-Datenbank oder das kostenlose SQL Server Management Studio.

Mit der Anmeldung **sa** erhält der Anwender im SQL Server die Rechte eines Administrators. Denkt man über die hiermit verbundenen Möglichkeiten nach, fallen einem in erster Linie administrative Tätigkeiten ein, wie Berechtigungen ändern, Datenbanken löschen et cetera.

Apropos Datenbanken löschen. Das Löschen von Datenbanken ist zwar ärgerlich, es ist aber auch lösbar. Sie werden recht schnell mitbekommen, wenn die Datenbank nicht mehr existiert.

Alles halb so wild, ist die Datenbank doch mit der letzten Datensicherung schnell wiederhergestellt – sofern es eine aktuelle und fehlerfreie Datensicherung gibt. An

dieser Stelle sei die Frage erlaubt, wann Sie Ihre Datensicherung das letzte Mal getestet haben?

Unrechtmäßige administrative Vorgänge lassen sich in der Regel beheben, wenn man sie denn erkannt hat. Viel riskanter aber ist das heimliche Lesen oder gar Ändern von Daten.

Mit der Anmeldung **sa** hat der Anwender nicht nur administrative Rechte, er besitzt auch vollen Zugang zu allen Daten des SQL Servers. Es hindert ihn niemand daran, die Daten zu lesen und zu seinen Gunsten zu ändern.

Folgendes Beispiel soll dies verdeutlichen. Ein Anwender legt in der entsprechenden Datenbank einen Lieferanten an und erfasst danach in unregelmäßigen Abständen zugehörige Eingangsrechnungen unterhalb der Budgetfreigabegrenze. Da er dabei eine ihm bekannte Bankverbindung angegeben hat, erhält er von der nichtsahnenden Buchhaltung einen kleinen Nebenverdienst.

Sie haben in Ihrer Datenbank keine Lieferantendaten? Aber vielleicht liegen diese in einer anderen Datenbank auf dem gleichen SQL Server. Auch wenn es bereits erwähnt wurde: Für die Anmeldung **sa** gibt es im SQL Server keine Grenzen.

Der Anwender kann die Daten aller Datenbanken einsehen und ändern! Dem Benutzer das Kennwort für die Anmeldung **sa** mitzuteilen, mag eine einfache Lösung sein.

Sie ist aber, wie so oft bei einfachen Lösungen, die unsicherste Methode, die Sie verwenden können.

## Access und das Kennwort

Eine weitaus bessere Variante bietet Ihnen Access. Beim Einbinden der Tabellen wie auch beim Erstellen der Pass Through-Abfragen können Sie das Kennwort der SQL Server-Anmeldung speichern. Danach ist die Kennworteingabe nicht einmal mehr beim Start der Access-Applikation erforderlich.

Hierzu brauchen Sie eine ODBC-Verbindung. Diese erstellen Sie am besten vorab als Systemdatenquelle über den ODBC-Dialog des Betriebssystems. Auf diese Weise steht die ODBC-Datenquelle jedem Anwender zur Verfügung, der sich an dem Rechner anmeldet. Anhand der ODBC-Datenquelle wird die Verbindung von Ihrem Access-Frontend zur Datenbank Ihres SQL Servers hergestellt. Somit ist das Einrichten dieser ODBC-Datenquelle auf jedem Rechner erforderlich, auf dem auch Ihr Access-Frontend installiert ist.

Die Konfiguration der ODBC-Datenquelle können Sie anhand der Beispielapplikation **WaWi** nachvollziehen. Dazu installieren Sie zunächst die Datenbank **WaWi_SQL** in Ihrem SQL Server.

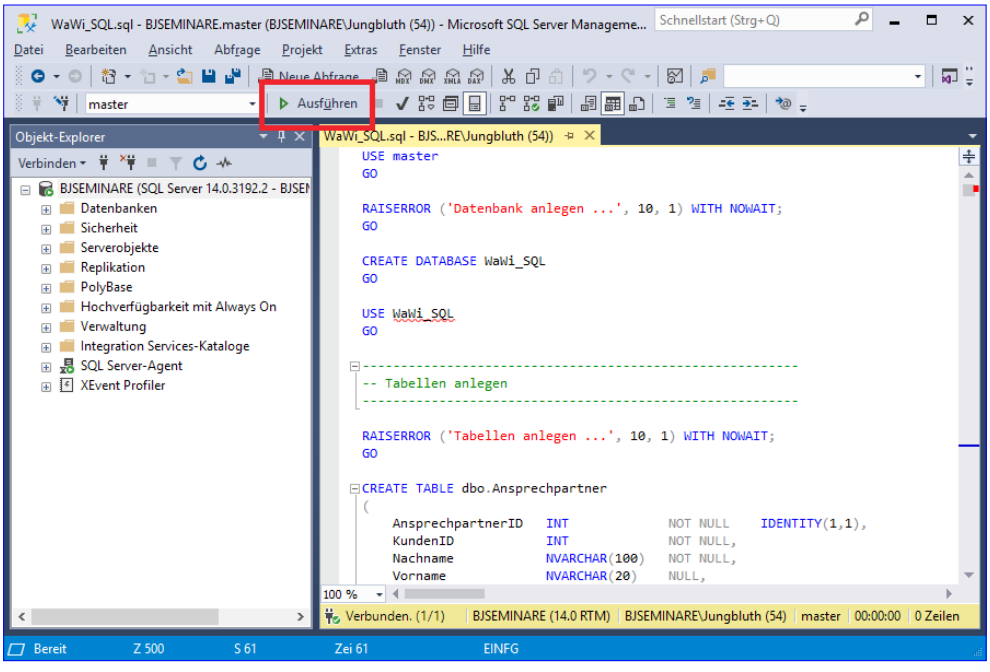



**Bild 8:** Installation der Beispieldatenbank

Öffnen Sie im SQL Server Management Studio die Datei **WaWi_SQL.sql** und starten Sie das Skript mit einem Klick auf **Ausführen** (siehe Bild 8). Nachdem die Datenbank angelegt ist, sehen Sie in der Registerkarte Meldungen die Ausgabe **Fertig!.**

Nun drücken Sie die Windows-Taste, geben als Suchbegriff **ODBC** ein und wählen den Eintrag **ODBC-Datenquellen (32-Bit)**. Im Dialog **ODBC-Datenquellen-Administrator** aktivieren Sie die Registerkarte **System-DSN** und klicken dort auf die Schaltfläche **Hinzufügen** (siehe Bild 9).

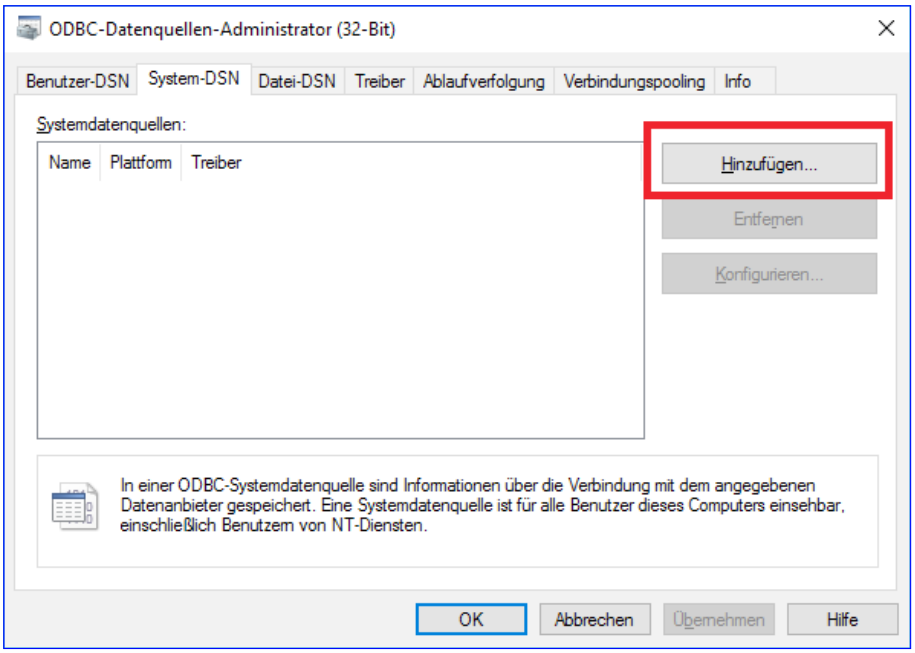



**Bild 9:** Anlegen der ODBC-Datenquelle

Es folgt die Auswahl des ODBC-Treibers (siehe Bild 10). Hier haben Sie die Qual der Wahl. Für den Zugriff auf SQL Server gibt es mehrere ODBC-Treiber. Abhängig von der Version bietet er neben einer besseren Performance auch die Unterstützung neuer SQL Server-Funktionen, insbesondere im Bereich Security.

Aus diesem Grund sollten Sie die neueste Version verwenden. Aktuell ist dies **ODBC Driver 17 for SQL Server**. Falls Sie diesen Treiber in Ihrer Auswahl nicht sehen, müssen Sie ihn zunächst installieren. Sie finden den Treiber im Microsoft **Downloadbereich** unter dem Suchbegriff **ODBC Driver 17 for SQL Server**.

Bitte verwenden Sie nicht die ODBC-Treiber SQL Server und SQL Server Native Client. Die Treiber werden von Microsoft weder weiterentwickelt noch gibt es entsprechende Fehlerbehebungen oder gar Sicherheitsupdates.

Somit stellen diese beiden Treiber eine Sicherheitslücke dar. Mit einem veralteten Treiber ist die sichere Kommunikation zwischen Access-Frontend und SQL Server-Backend nicht gewährleistet. Unbefugte könnten heimlich Mithören oder gar Mitreden, im Sinne von Daten lesen, ändern und löschen.

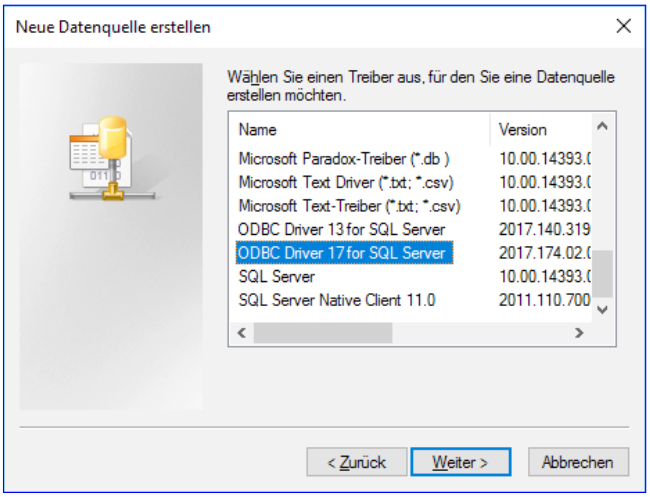



**Bild 10:** Auswahl des ODBC-Treibers

Leider werden genau diese beiden Treiber in vielen Access und SQL Server-Applikationen eingesetzt. Der Grund hierfür ist recht einfach. Die Treiber sind bereits vorinstalliert und müssen nicht extra auf den Rechnern der Anwender verteilt werden.

# Zwischenablage per VBA, 64-Bit-Version

**Den Code für die Ablage von Inhalten per VBA in die Zwischenablage benötigt man immer wieder. Dieser Artikel zeigt die notwendigen Änderungen für die 64-Bit-Version.**

Seit Microsoft das Office-Paket standardmäßig in der 64-Bit-Version installiert, kracht es regelmäßig bei Datenbanken, die mit der 32-Bit-Version programmiert wurden und die API-Funktionen verwenden.

Das gilt auch für die Funktion **InZwischenablage**, die wir schon in zahlreichen Beispielen verwendet haben. Wir haben das Modul mit dieser Funktion erweitert, sodass es sowohl unter 64-Bit-Office als auch unter 32-Bit-Office

```
'Original aus http://support.microsoft.com/?kbid=210216

#If Win64 Then
Declare PtrSafe Function GlobalUnlock Lib "kernel32" (ByVal hMem As LongPtr) As Long
Declare PtrSafe Function GlobalLock Lib "kernel32" (ByVal hMem As LongPtr) As LongPtr
Declare PtrSafe Function GlobalAlloc Lib "kernel32" (ByVal wFlags As Long, ByVal dwBytes As Long) As LongPtr
Declare PtrSafe Function CloseClipboard Lib "user32" () As Long
Declare PtrSafe Function OpenClipboard Lib "user32" (ByVal hWnd As LongPtr) As Long
Declare PtrSafe Function EmptyClipboard Lib "user32" () As Long
Declare PtrSafe Function lstrcpy Lib "kernel32" (ByVal lpString1 As Any, ByVal lpString2 As Any) As LongPtr
Declare PtrSafe Function SetClipboardData Lib "user32" (ByVal wFormat As Long, ByVal hMem As LongPtr) As LongPtr

Public Const GHND = &H42
Public Const CF_TEXT = 1
Public Const MAXSIZE = 4096

Function Inzwischenablage(MyString As String)
  Dim hGlobalMemory As LongPtr, lpGlobalMemory As LongPtr
  Dim hClipMemory As LongPtr, X As Long
  hGlobalMemory = GlobalAlloc(GHND, Len(MyString) + 1)
  lpGlobalMemory = GlobalLock(hGlobalMemory)
  lpGlobalMemory = lstrcpy(lpGlobalMemory, MyString)
  If GlobalUnlock(hGlobalMemory) <> 0 Then
     MsgBox "Could not unlock memory location. Copy aborted."
     GoTo OutOfHere2
  End If
  If OpenClipboard(0&) = 0 Then
     MsgBox "Could not open the Clipboard. Copy aborted."
     Exit Function
  End If
  X = EmptyClipboard()
  hClipMemory = SetClipboardData(CF_TEXT, hGlobalMemory)
OutOfHere2:
  If CloseClipboard() = 0 Then
     MsgBox "Could not close Clipboard."
  End If
End Function
```

**Listing 1:** 64-Bit-Teil des Codes, Teil 1

# VBA-Projekte vergleichen

**Bevor man umfangreichere Änderungen an bestehenden Anwendungen vornimmt, sollte man zumindest eine Sicherungskopie der Datenbankdatei machen. Außen vor sind hier Entwickler, die den Code ihrer Anwendungen mit einer Quellcodeverwaltung tracken – hier kann man den Verlauf der Änderungen leicht nachvollziehen. Wer aber den Aufwand nicht betreiben will, kann auch mit einfacheren Mitteln prüfen, welche Änderungen seit dem letzten Speichern durchgeführt wurden – und zwar mit der in diesem Beitrag vorgestellten Lösung. Hier bauen wir auf einer einfachen Funktion auf, die den kompletten Inhalt aller VBA-Module exportiert und schauen uns dann die Unterschiede mit einem Texteditor an.**

## VBE zugreifbar machen

Wenn wir auf den Inhalt des VBA-Projekts einer Anwendung zugreifen wollen, benötigen wir die Bibliothek mit den Befehlen für den Zugriff auf das VBE-Objekt. Dieses bietet alle notwendigen Objekte, Eigenschaften und Methoden, um auf die Module und ihre Inhalte zuzugreifen. Die Bibliothek machen wir verfügbar, indem wir einen Verweis darauf zum aktuellen Projekt hinzufügen.

Dazu öffnen Sie den VBA-Editor (von Access beispielsweise mit **Alt + F11**) und starten dann den **Verweise**-Dialog mit dem Menübefehl **Extras|Verweise**. Hier aktivieren Sie den Eintrag **Microsoft Visual Basic for Applications Extensibility 5.3 Object Library** (siehe Bild 1).

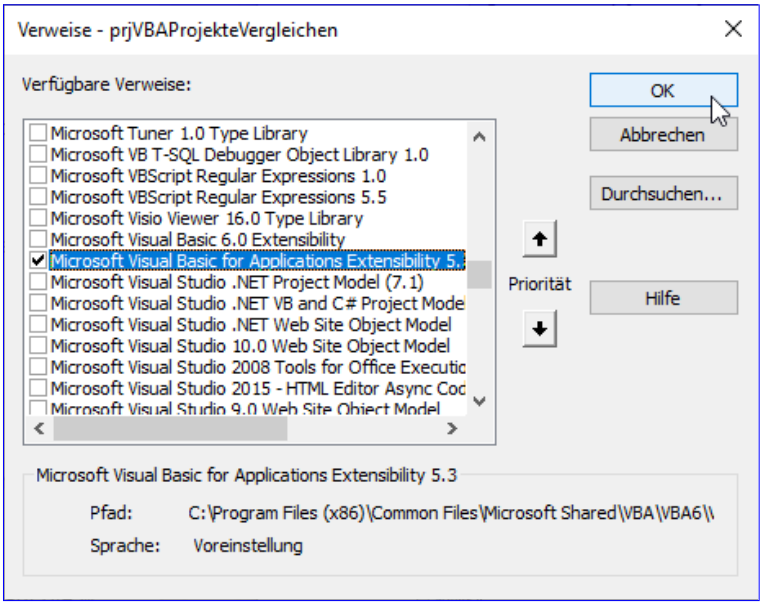



**Bild 1:** Verweis auf die Bibliothek **Microsoft Visual Basic for Applications Extensibility 5.3 Object Library**

Damit können wir mit einer recht einfachen Prozedur alle VBA-Module durchlaufen, diese in einer String-Variablen zusammenfassen und sie dann in einer Textdatei speichern, um sie mit einem Export aus einer anderen Datenbank zu vergleichen.

## VBA-Code komplett exportieren

Die folgende Prozedur exportiert den kompletten Code eines VBA-Projekts in eine Textdatei. Sie deklariert Variablen zum Referenzieren des VBA-Editors (**objVBE**), des aktuellen VBA-Projekts (**objVBProject**), einer VBA-Komponente (**objVBComponent**) und des darin enthaltenen Code-Moduls (**objCodeModule**).

Außerdem benötigen wir noch Variablen zum Zusammenfügen des Codes der Elemente und für den Dateinamen:

```
Public Sub CodeKomplettExportieren()
    Dim objVBE As VBIDE.VBE
    Dim objVBProject As VBIDE.VBProject
```

```
    Dim objVBComponent As VBIDE.VBComponent
    Dim objCodemodule As VBIDE.CodeModule
    Dim strVBA As String
    Dim strDateiname As String
```

Damit referenziert die Prozedur den VBA-Editor und ermittelt mit der Eigenschaft **ActiveVBProject** das aktuell geöffnete VBA-Projekt:

```
    Set objVBE = VBE
    Set objVBProject = objVBE.ActiveVBProject
```

Dann durchläuft sie alle **VBComponent**-Elemente, also alle Standardmodule, Klassenmodule und Formular- und Berichtsmodule aus der Auflistung **VBComponents** in einer **For Each**-Schleife und referenziert diese mit der Variablen **objVBComponent**:

```
    For Each objVBComponent In objVBProject.VBComponents
```

In der Schleife referenziert die Prozedur das im **VBComponent**-Element enthaltene Code-Modul mit der Variablen **objCodeModule** und fügt dieses mit einer führenden und einer nachfolgenden Zeile zur optischen Hervorhebung zusammen:

```
        Set objCodemodule = objVBComponent.CodeModule
        strVBA = strVBA & "============================" _
            & vbCrLf
        strVBA = strVBA & objVBComponent.Name & vbCrLf
        strVBA = strVBA & "============================" _
            & vbCrLf
```

Danach folgt der eigentliche Inhalt des VBA-Moduls, der mit der **Lines**-Eigenschaft für alle Zeilen von der ersten bis zu der mit **CountOfLines** ermittelten letzten Zeile ermittelt wird:

```
        strVBA = strVBA & objCodemodule.Lines(1, _
            objCodemodule.CountOfLines) & vbCrLf
    Next objVBComponent
```

Nachdem auf diese Weise alle Module in der Variablen **strVBA** zusammengestellt wurden, schreibt die Prozedur deren Inhalt in eine Textdatei, die den Namen der Datenbank plus die Endung **.vba.txt** enthält:

```
    strDateiname = CurrentDb.Name & ".vba.txt"
    Open strDateiname For Output As #1
    Print #1, strVBA
    Close #1
End Sub
```

Diesen Vorgang führen Sie nun mit der aktuellen und mit der zu vergleichenden Version des VBA-Projekts durch und speichern auch diese Datei.

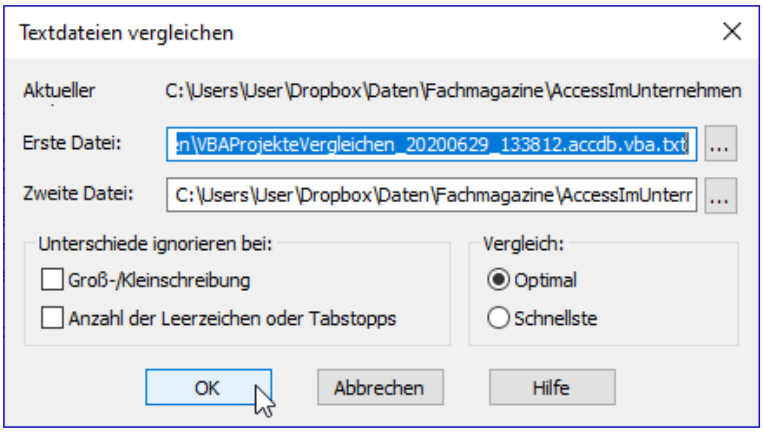



**Bild 2:** Optionen für das Vergleichen von Textdateien

## Vergleichen der beiden Dateien

Zum Vergleichen gibt es zahlreiche Anwendungen. Wir arbeiten seit Ewigkeiten mit TextPad, welches eine Funktion zum Vergleichen des Inhalts zweier Dateien anbietet.

Die umständliche Vorgehensweise hier lautet:

- TextPad starten
- Textdatei mit der aktuellen Version des VBA-Projekts öffnen

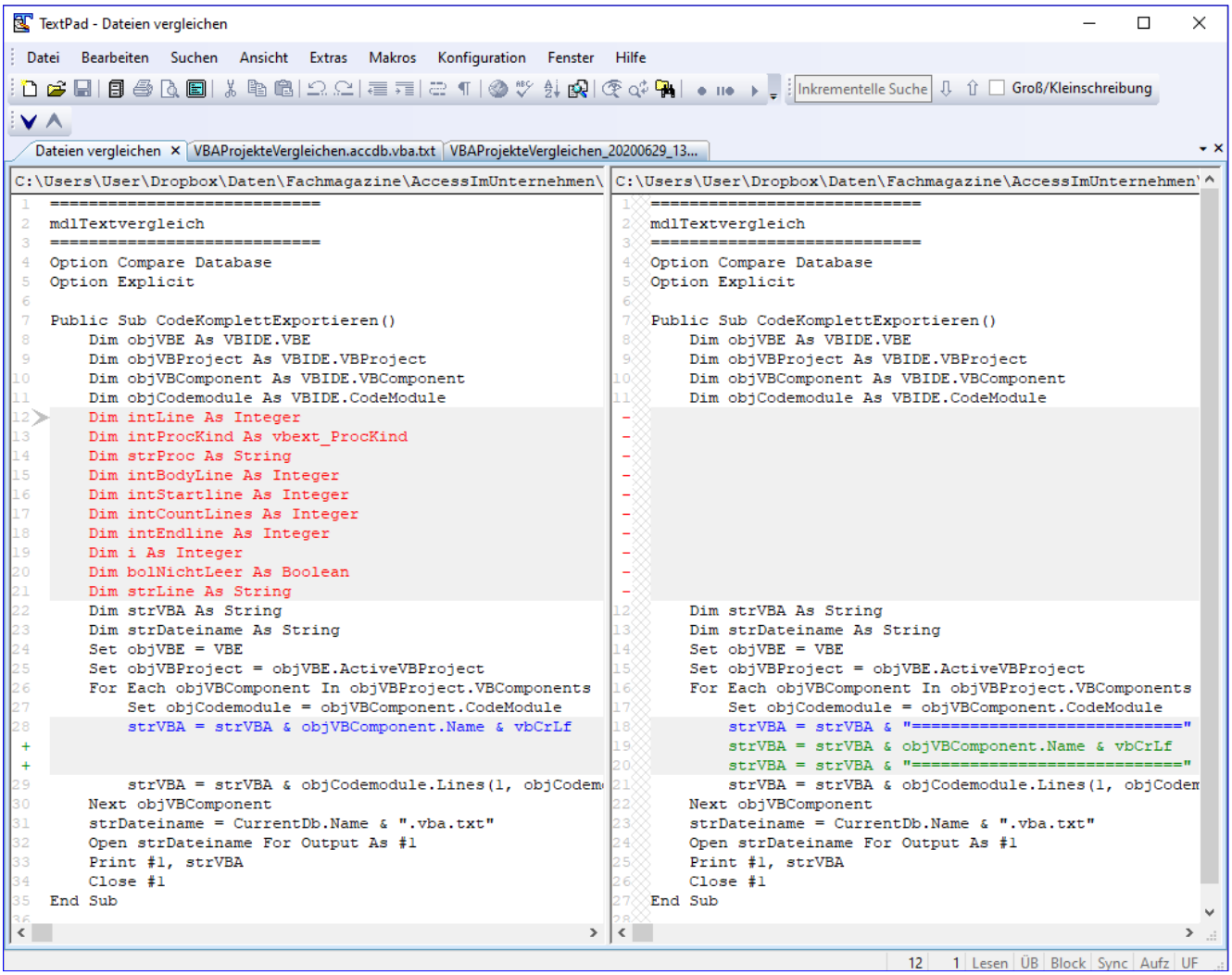



**Bild 3:** Vergleich des Inhalts zweier VBA-Projekte im Text-Editor TextPad

- Textdatei mit der zu vergleichenden Version des VBA-Projekts öffnen

- In Textpad den Menübefehl **Extras|Dateien vergleichen...** aufrufen

TextPad zeigt dann den Dialog **Textdateien vergleichen** an, der die beiden aktuell geöffneten Dateien als erste und zweite zu vergleichende Datei anbietet (siehe Bild 2) – Sie brauchen also nur noch auf **OK** zu klicken.

Danach zeigt Textpad die Unterschiede entsprechend hervorgehoben an (siehe Bild 3).

## TextPad direkt mit Dateien öffnen

TextPad bietet einige Startparameter an, mit denen wir unter anderem direkt eine oder mehrere Dateien angeben können, die beim Start geöffnet werden sollen. Dazu nutzen wir unter VBA die **Shell**-Funktion und geben hinter dem Pfad zur **TextPad.exe** die Namen der zu öffnenden Dateien an – zur Sicherheit in Anführungszeichen eingeschlossen, falls die Dateinamen Leerzeichen enthalten:

```
Public Sub TextPadMitDateienOeffnen()
    Shell "C:\Program Files\TextPad 8\TextPad.exe """ _
        & c:\...\Datei1.vba.txt"" ""c:\...\Datei2.vba.txt"""
End Sub
```

Danach brauchen Sie nur noch den Menüeintrag **Extras|Dateien vergleichen...** aufzurufen.

## Add-In-Unterstützung

Da die Vorgehensweise immer noch ein wenig kompliziert ist – immerhin müssen wir die Prozedur **CodeKomplettExportieren** in jede betroffene Datenbank kopieren, immer beide Datenbanken öffnen, um den VBA-Code zu exportieren und diese dann wieder schließen –, behelfen wir uns mit einem kleinen, aber effizienten Access-Add-In.

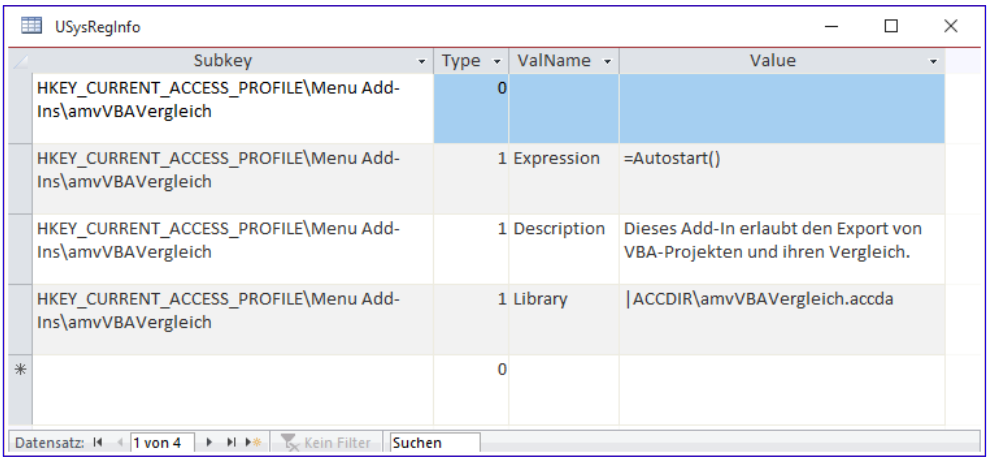



**Bild 4:** Add-In-Einstellungen in der Tabelle **USysRegInfo**

Dieses legen wir in Form einer neuen, leeren Access-Datenbank an, deren Name wir noch vor der ersten Änderung in **amvVBAVergleich.accda** umbenennen (**.accda** ist die Dateiendung, unter der Access-Add-Ins erkannt werden).

## USysRegInfo-Tabelle

Damit die Datenbank als Add-In registriert werden kann, fügen wir ihr eine Tabelle namens **USysRegInfo** hinzu, welche die Informatoinen enthält, die beim Installieren des Add-Ins in der Registry gespeichert werden. Von dort liest Access diese Informationen beim Start aus und zeigt das Add-In an der entsprechenden Stelle an – meist im Ribbon-Menü **Datenbanktools|Add-Ins|Add-Ins**.

Diese Tabelle sieht wie in Bild 4 aus und muss genau die dort abgebildeten Informationen enthalten – zumindest jedoch muss der Wert in **Subkey** übereinstimmen und der im vierten Datensatz angegebene Dateiname im Feld **Value** muss mit dem Dateinamen der Add-In-Datenbank übereinstimmen.

## Datenbankinformationen

Damit das Add-In später im Add-In-Manager ordentlich angezeigt wird, stellen wir außerdem noch Werte im

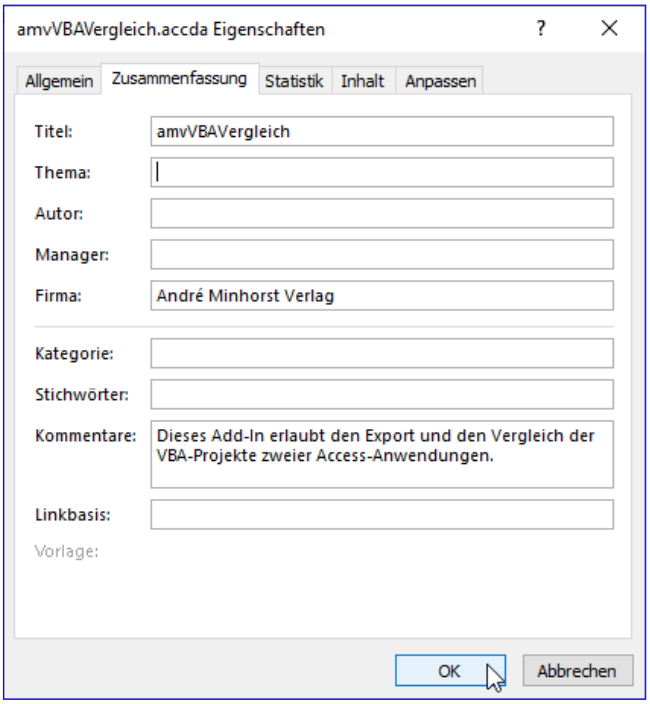



**Bild 5:** Datenbank-Eigenschaften

Eigenschaften-Fenster von Access ein. Dieses öffnen Sie in aktuellen Versionen von Access unter **Datei|Informationen** mit dem Link **Datenbankeigenschaften anzeigen und bearbeiten**. Hier legen Sie die Werte wie in Bild 5 fest.

## Autostart-Funktion

Damit beim Aufrufen des Add-Ins etwas geschieht, fügen Sie einem Standardmodul der Add-In-Datenbank eine öf-

fentliche Funktion namens **Autostart** hinzu (dass diese Funktion so heißt, haben wir in der Tabelle **USysRegInfo** festgelegt). Diese Funktion soll ein Formular namens **frmVBAVergleich** öffnen.

```
Public Function Autostart()
    DoCmd.OpenForm "frmVBAVergleich",
WindowMode:=acDialog
End Function
```

## Das Formular frmVBAVergleich

Dieses Formular legen wir als Nächstes an. Da dieses Formular keine Möglichkeit zum Navigieren in Datensätzen anzeigen soll, stellen wir die Eigenschaften **Datensatzmarkierer**, **Navigationsschaltflächen** und **Bildlaufleisten** auf **Nein** ein und die Eigenschaft **Automatisch zentrieren** auf **Ja**.

Es soll aber dennoch an eine Datenherkunft gebunden werden, nämlich an die Tabelle **tblOptionen**. Diese soll den Namen des aufzurufenden Tools für den Vergleich der VBA-Projekte speichern sowie die Konstellation der Platzhalter für die Namen der zu vergleichenden Dateien, falls diese per Startparameter übergeben wer-

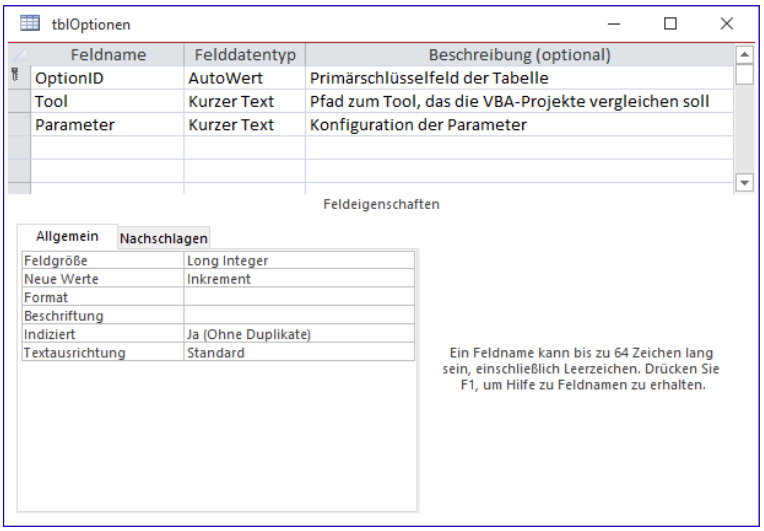



**Bild 6:** Optionen-Tabelle der Anwendung

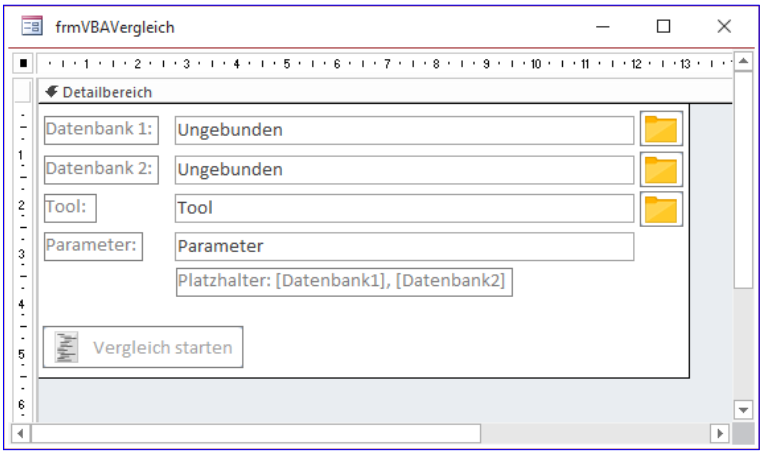



**Bild 7:** Das Formular zum Starten des Vergleichs zweier VBA-Projekte

```
Private Sub cmdDatenbank1Auswaehlen_Click()
    Dim strResult As String
    Dim strFiles() As String
    Dim i As Integer
    strResult = OpenFileNameMultiple(CurrentProject.Path, "Zu vergleichende Dateien auswählen", _
        "Access-Datenbank(*.mdb;*.accdb)")
    strFiles = Split(strResult, vbTab)
    If UBound(strFiles) - LBound(strFiles) > 0 Then
        MsgBox "Sie haben zwei oder mehr Dateien ausgewählt. Die ersten beiden gewählten Dateien werden übernommen.", _
            vbOKOnly + vbInformation, "Mehrere Dateien ausgewählt"
    End If
    For i = LBound(strFiles) To UBound(strFiles)
        If i = 2 Then Exit For
        Me("txtDatenbank" & i + 1) = strFiles(i)
    Next i
    SchaltflaecheAktivieren Nz(Me!txtDatenbank1), Nz(Me!txtDatenbank2), Nz(Me!txtTool)
End Sub
```

**Listing 1:** Prozedur zum Öffnen der ersten/zweiten Datei

# Textvergleich mit WinMerge

**Im Beitrag »VBA-Projekte vergleichen« (www.access-im-unternehmen.de/1248) haben wir gezeigt, wie Sie mit einem Access-Add-In die VBA-Projekte zweier Datenbanken auswählen und vergleichen können. Dort haben wir TextPad genutzt, um den Vergleich der beiden zuvor exportierten Dateien durchzuführen. Unser Lektor Carsten Gromberg hat noch ein besseres Tool für diesen Zweck empfohlen: WinMerge. Damit können Sie den Textvergleich im Gegensatz zum Einsatz von TextPad direkt durchführen, ohne noch weitere Schritte in dieser Anwendung durchführen zu müssen.**

Mit TextPad haben wir bereits die Möglichkeit, die beiden zu vergleichenden Dateien per Öffnungsargument zu übergeben. Allerdings müssen wir dort noch den Menübefehl **Extras|Dateien vergleichen...** aufrufen, um den Vergleich durchzuführen.

Mit dem Tool **WinMerge**, das im Gegensatz zu TextPad kostenlos ist, können Sie sogar noch den Vergleich der beiden Dateien per Parameter initiieren. Doch eins nach dem anderen!

## WinMerge herunterladen

**WinMerge** finden Sie unter dem folgenden Link:

```
https://winmerge.org/
```

Nach einem Klick auf **Jetzt herunterladen!** landen Sie auf der SourceForge-Seite des Projekts, wo Sie die direkt über die Schaltfläche **Download** auf das Tool zugreifen (siehe Bild 1).

## WinMerge installieren

Anschließend können Sie das Tool direkt installieren.

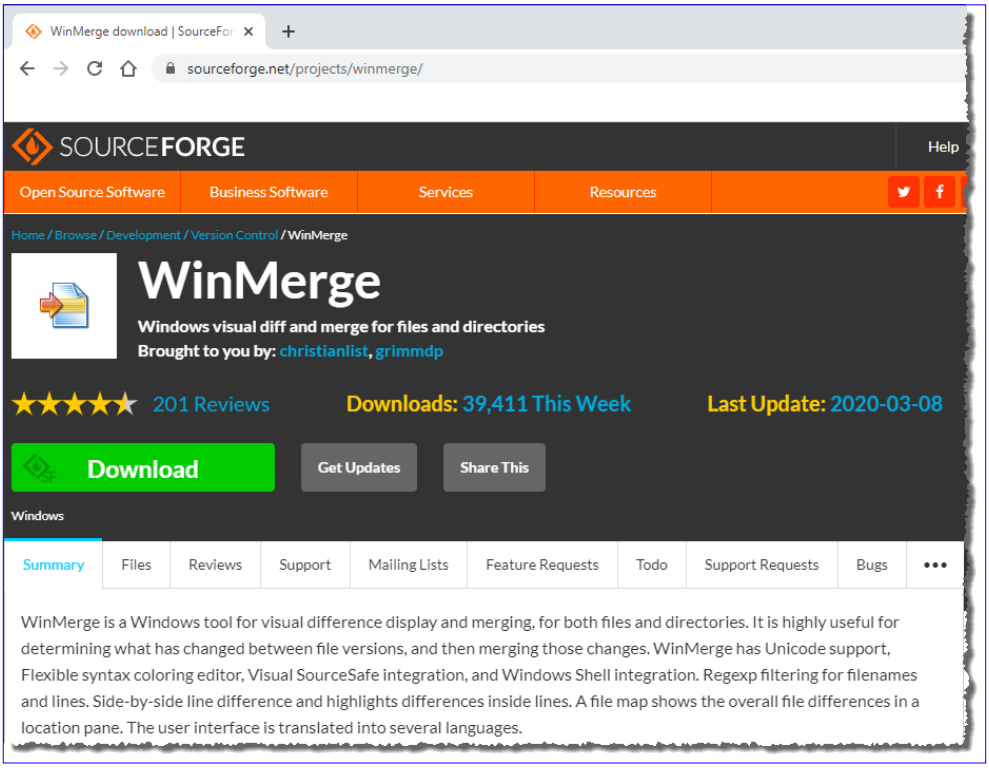



**Bild 1:** Download von WinMerge

Nach dem Bestätigen der Lizenzbedingungen legen Sie noch das Installationsziel und die zu installierenden Komponenten fest. Dabei selektieren wir auch gleich noch die Option **Explorer Kontextmenü-Integration aktivieren**.

## Befehlszeilen-Parameter

Nach dem Start sieht WinMerge nicht viel anders aus als andere Text-Editoren. Wir wollen uns allerdings direkt mit den Befehlszeilenparametern beschäftigen, damit wir das

Tool für den direkten Textvergleich nutzen können.

Alle dafür notwendigen Optionen werden auf der Seite **Command line** der Hilfe-Funktion von **WinMerge** erläutert (siehe Bild 2).

Hier finden wir schnell alle Parameter, die wir für unseren Anwendungsfall nutzen können. Bevor wir diese in das Formular des Access-Add-Ins eintragen, probieren wir uns noch mit der Eingabeaufforderung von Windows aus.

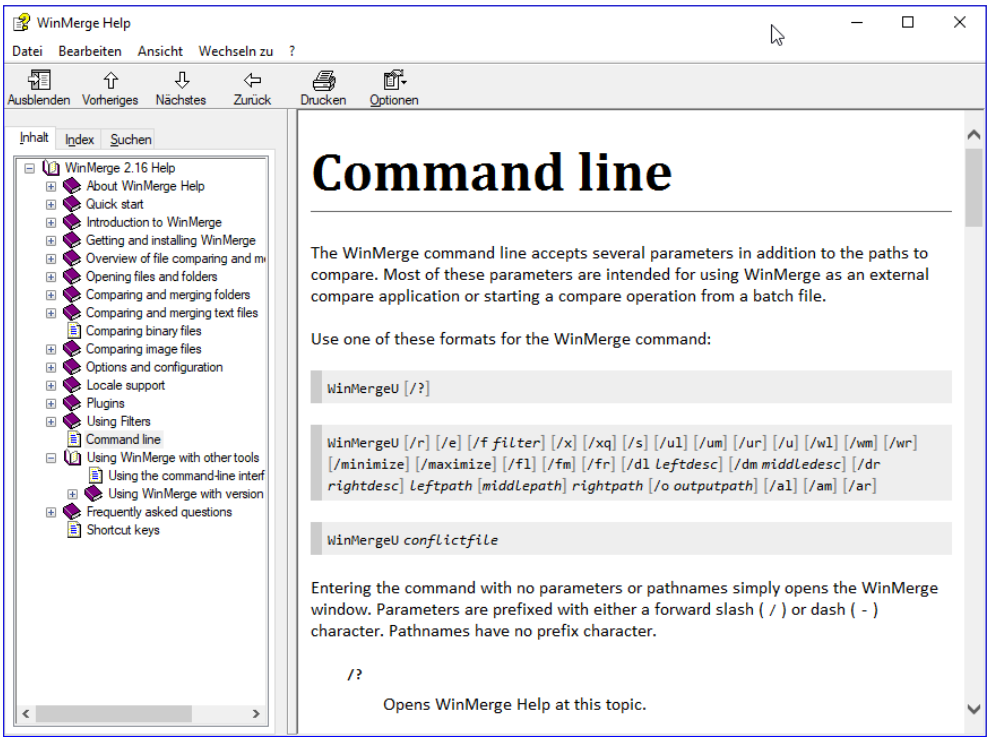



**Bild 2:** Befehlszeilenparameter von **WinMerge**

## Umgebungsvariable für WinMerge einstellen

Damit wir von überall direkt auf den Dateinamen **WinMergeU.exe** zugreifen können und nicht immer noch den kompletten Pfad angeben müssen, fügen wir den Pfad zur Datei **WinMergU.exe** zur **Path**-Variablen von Windows hinzu.

Dazu öffnen Sie Systemsteuerung und klicken dort auf das Element **System**. Auf der linken Seite finden Sie dann den Eintrag **Erweiterte Systemeigenschaften**. Dieser öffnet den Dialog **Systemeigenschaften**, wo Sie unten auf die Schaltfläche **Umgebungsvariablen** klicken.

Im nun erscheinenden Dialog **Umgebungsvariablen** klicken Sie unten auf den Eintrag **Path** und dann auf **Bearbeiten...**, was den Dialog **Umgebungsvariable bearbeiten** öffnet. Hier klicken

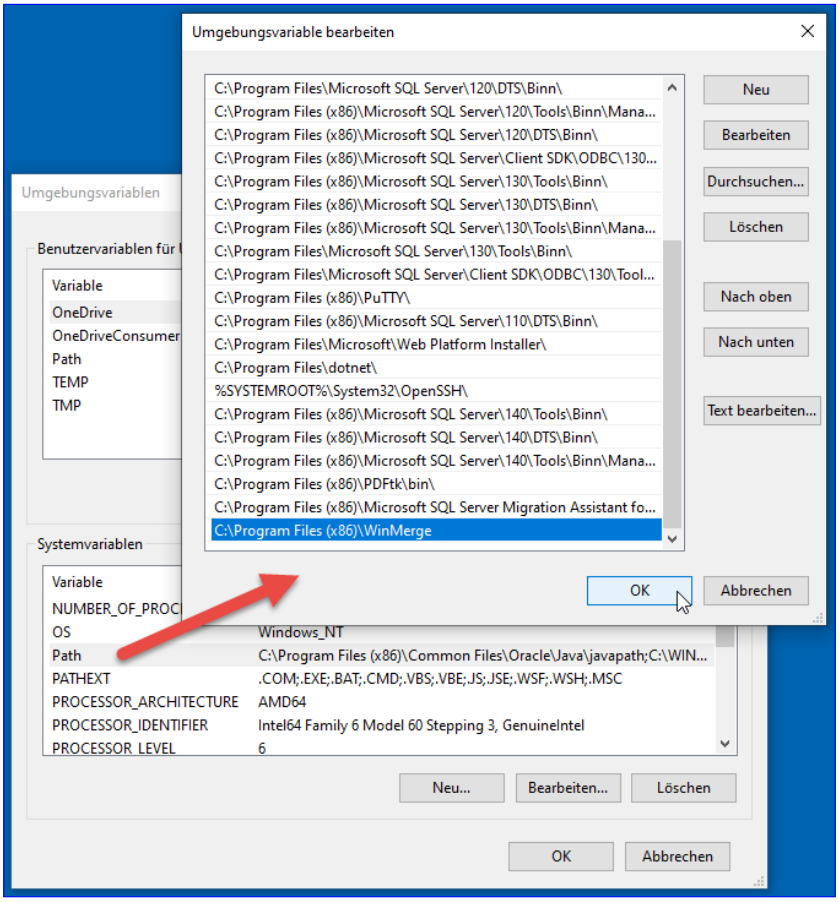



**Bild 3:** WinMerge zur **Path**-Variablen hinzufügen