HAUFE.
Ausgabe 03/2019

ACCESS IM UNTERNEHMEN

DATENSICHERHEIT

Sichern Sie Ihre Daten mit Microsoft Access unter Verwendung einer selbst programmierten Benutzerverwaltung (ab S. 21)

In diesem Heft:

HTML-TABELLEN MIT FESTER KOPFZEILE

Zeigen Sie Daten in einer HTML-Tabelle mit scrollbaren Inhalten an – bei fixierter Kopfzeile.

SEITE 2

KENNWÖRTER GENERIEREN

Generieren Sie sichere Kennwörter mit einer VBA-Funktion.

SEITE 65

BERECHTIGUNGEN PER DATENMAKRO

Definieren Sie Datenmakros zur Anwendung von Schreib- und Leserechten an einer Tabelle.

SEITE 57

Sicherheit in Access-Datenbanken

Wenn es um das Thema Datensicherheit geht, winken Entwickler beim Thema Access-Datenbanken ab. Spätestens seit Microsoft das Sicherheitssystem abgeschafft hat, ist hier nichts mehr zu holen. »Wechseln Sie zum SQL Server als Backend«, hört man dann. Dabei bietet Access durchaus Möglichkeiten, den Zugriff auf sensible Daten zu erschweren. Natürlich bietet der SQL Server eigens dafür vorgesehene Features, aber wenn man will, kann man seine Daten auch unter Access zumindest vor den meisten Langfingern schützen.

Deshalb starten wir in dieser Ausgabe mit der Programmierung eines eigenen kleinen Sicherheitssystems. Dabei beginnen wir mit einer eigenen Verwaltung für Benutzer und Berechtigungen an den verschiedenen Tabellen der Datenbankanwendung. Wie Sie dieses einrichten, erfahren Sie im Beitrag **Benutzerverwaltung mit verschlüsselten Kennwörtern** ab Seite 21. Hier zeigen wir neben dem Datenmodell mit den Tabellen für die Benutzer, Benutzergruppen, Berechtigungen und den mit Berechtigungen zu versehenen Tabellen auch, wie Sie verschlüsselte Kennwörter für die einzelnen Benutzer hinterlegen. In diesem Beitrag stellen wir auch das erste Formular der Anwendung vor. Mit diesem können Sie neue Benutzer anlegen und diese den Benutzergruppen der Anwendung hinzufügen.

Der zweite Beitrag zum Thema Sicherheit heißt **Berechtigungen per HTML verwalten** (ab Seite 29). Dieser fügt der Beispielanwendung aus dem vorherigen Beitrag ein weiteres Formular hinzu, mit dem Sie die Berechtigungen der einzelnen Benutzergruppen an den Tabellen verwalten. Sie legen damit fest, ob die Benutzergruppen Daten in diesen Tabellen lesen, anlegen, ändern oder löschen darf – oder ob sie gar keine Berechtigungen für den Zugriff auf diese Daten hat. Der Clou bei diesem Formular ist, dass wir die Zuordnung in einer übersichtlichen, per HTML erstellten Tabelle durchführen.

Im dritten Beitrag zu diesem Thema namens **Benutzer und Berechtigungen ermitteln** erfahren Sie ab Seite 45, wie Sie die Tabellen des Berechtigungssystems auslesen, um die Berechtigungen eines bestimmten Benutzers an den Tabellen über die ihm zugeordneten Benutzergruppen ermitteln. Damit können Sie dann etwa in Formularen dafür sorgen, dass Daten nur für den lesenden oder schreibenden Zugriff angezeigt werden. Außerdem stellen wir in diesem Beitrag das Formular vor, mit dem sich die Benutzer an der Datenbank anmelden können.

Richtig spannend wird es im Beitrag **Zugriffsrechte mit Datenmakros** ab Seite 57: Hier schauen wir uns an, wie Sie ohne Einsatz von VBA sicherstellen können, dass der angemeldete Benutzer nur die für ihn vorgesehenen Änderungen wie Anlegen, Ändern oder Löschen an den Daten der Tabellen vornimmt. Die dafür notwendigen Mechanismen legen wir direkt in den Tabellen an. Somit brauchen Sie sich keine Sorgen zu machen, dass Benutzer die Sicherheitsmechanismen der Benutzeroberfläche umgehen und Änderungen direkt in den Tabellen durchführen.

Schließlich liefern wir noch eine Lösung, mit der Sie sichere Kennwörter für die Benutzerverwaltung automatisch generieren lassen können. Diese finden Sie ab Seite 65 unter dem Titel **Kennwörter generieren**.

In der folgenden Ausgabe gehen wir noch einen Schritt weiter und liefern das Know-how, mit dem Sie die Tabellen der Anwendung unsichtbar für den Benutzer machen – und die Daten der Anwendung zumindest vor herkömmlichen Anwendern ohne kriminelle Energie sichern.

Und nun: Viel Erfolg beim Sichern Ihrer Daten!

Ihr André Minhorst

HTML-Tabellen mit fester Kopfzeile

In den vorherigen Ausgaben von Access im Unternehmen und in der aktuellen Ausgabe arbeiten wir in einigen Beiträgen mit dem Webbrowser-Steuerelement und stellen Daten, die wir mit den Bordmitteln von Access nicht adäquat darstellen können, mit HTML im Webbrowser-Steuerelement dar. Damit erhalten wir wesentlich flexiblere Möglichkeiten, aber auch der Aufwand steigt. Eine Anforderung, die bisher nicht erfüllt wurde, ist das Fixieren der obersten Zeile in einer Ansicht, in der wir zur Anzeige aller Daten nach unten scrollen mussten. Wie wir den Daten einer Tabelle eine fixe Kopfzeile hinzufügen, zeigen wir in diesem Beitrag.

Problem

Wenn wir in einem Webbrowser-Steuerelement eine HTML-Seite mit einer größeren Menge Daten in einer HTML-Tabelle laden, wird diese wie gewünscht angezeigt – auch inklusive der Überschriften im Tabellenkopf.

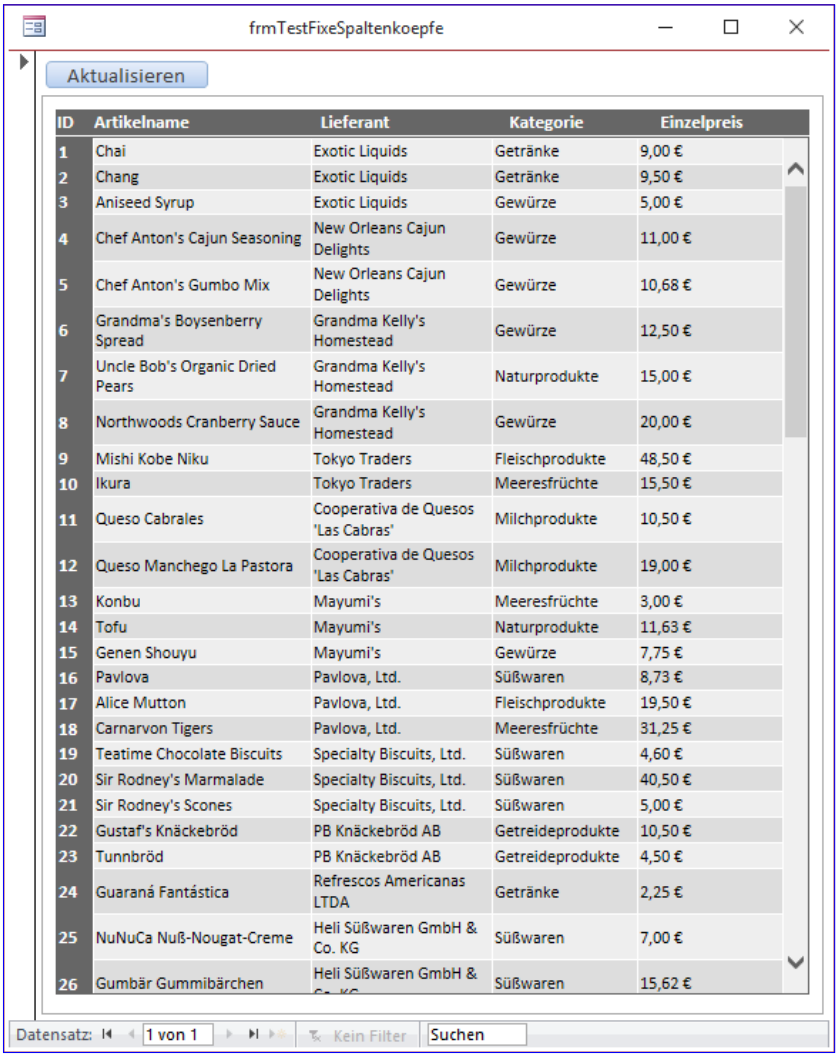
Sobald wir jedoch nach unten scrollen, um die aktuell nicht sichtbaren Daten anzusehen, verschwinden nicht nur die obersten Datensätze, sondern auch die Kopfzeile mit den Spaltenüberschriften wird nach oben aus dem sichtbaren Bereich geschoben. Das wollen wir ändern, indem wir die Kopfzeile irgendwie oben fixieren.

Im Web werden viele Lösungen für dieses Problem vorgeschlagen. Viele davon nutzen Techniken wie Javascript. Hier ist nicht sichergestellt, dass alle Browser dies wie gewünscht darstellen.

Da das Webbrowser-Steuerelement den aktuell im System als Standardbrowser eingestellten Browser verwendet, sollten Sie hier eine Technik verwenden, die möglichst von allen aktuellen Browsern unterstützt wird. Unser Ergebnis soll wie in Bild 1 aussehen.

Lösung

In diesem Fall wählen wir eine Lösung, die nur auf CSS basiert. In diesem Fall wollen wir nicht mehr, dass der



ID	Artikelname	Lieferant	Kategorie	Einzelpreis
1	Chai	Exotic Liquids	Getränke	9,00 €
2	Chang	Exotic Liquids	Getränke	9,50 €
3	Aniseed Syrup	Exotic Liquids	Gewürze	5,00 €
4	Chef Anton's Cajun Seasoning	New Orleans Cajun Delights	Gewürze	11,00 €
5	Chef Anton's Gumbo Mix	New Orleans Cajun Delights	Gewürze	10,68 €
6	Grandma's Boysenberry Spread	Grandma Kelly's Homestead	Gewürze	12,50 €
7	Uncle Bob's Organic Dried Pears	Grandma Kelly's Homestead	Naturprodukte	15,00 €
8	Northwoods Cranberry Sauce	Grandma Kelly's Homestead	Gewürze	20,00 €
9	Mishi Kobe Niku	Tokyo Traders	Fleischprodukte	48,50 €
10	Ikura	Tokyo Traders	Meeresfrüchte	15,50 €
11	Queso Cabrales	Cooperativa de Quesos 'Las Cabras'	Milchprodukte	10,50 €
12	Queso Manchego La Pastora	Cooperativa de Quesos 'Las Cabras'	Milchprodukte	19,00 €
13	Konbu	Mayumi's	Meeresfrüchte	3,00 €
14	Tofu	Mayumi's	Naturprodukte	11,63 €
15	Genen Shouyu	Mayumi's	Gewürze	7,75 €
16	Pavlova	Pavlova, Ltd.	Süßwaren	8,73 €
17	Alice Mutton	Pavlova, Ltd.	Fleischprodukte	19,50 €
18	Carnarvon Tigers	Pavlova, Ltd.	Meeresfrüchte	31,25 €
19	Teatime Chocolate Biscuits	Specialty Biscuits, Ltd.	Süßwaren	4,60 €
20	Sir Rodney's Marmalade	Specialty Biscuits, Ltd.	Süßwaren	40,50 €
21	Sir Rodney's Scones	Specialty Biscuits, Ltd.	Süßwaren	5,00 €
22	Gustaf's Knäckebröd	PB Knäckebröd AB	Getreideprodukte	10,50 €
23	Tunnbröd	PB Knäckebröd AB	Getreideprodukte	4,50 €
24	Guaraná Fantástica	Refrescos Americanas LTDA	Getränke	2,25 €
25	NuNuCa Nuß-Nougat-Creme	Heli Süßwaren GmbH & Co. KG	Süßwaren	7,00 €
26	Gumbär Gummibärchen	Heli Süßwaren GmbH &	Süßwaren	15,62 €

Bild 1: HTML-Tabelle mit Scrollbalken

Benutzer den kompletten Inhalt des Webbrowser-Steuerelements scrollen muss, sondern nur noch den Teil der Tabelle mit den Daten.

Der Trick dabei ist, dass wir zwei Tabellen nutzen: Die erste zeigt nur eine Zeile mit den Spaltenüberschriften an, die zweite die eigentlichen Daten. Damit der Benutzer in der zweiten Tabelle durch die Datensätze scrollen kann, benötigen wir ein spezielles Attribut. Außerdem fügen wir die zweite Tabelle in die erste Tabelle ein.

Füllen des Webbrowser-Steuerelements

Bevor wir uns die Funktionen zum Erstellen des HTML-Codes und des CSS-Codes ansehen, wollen wir noch die Prozeduren zum Formular hinzufügen, welche das Webbrowser-Steuerelement mit den Daten aus diesen Funktionen füllen. Im Klassenmodul des Formulars fügen wir im allgemeinen Teil zwei Objektvariablen ein:

```
Dim WithEvents objWebbrowser As WebBrowser
Dim WithEvents objDocument As HTMLDocument
```

Die erste wird in der Prozedur gefüllt, die durch das Ereignis **Beim Öffnen** des Formulars ausgelöst wird:

```
Private Sub Form_Open(Cancel As Integer)
    Set objWebbrowser = Me!ctlWebbrowser.Object
    Me.TimerInterval = 50
End Sub
```

Hier füllen wir das Webbrowser-Steuerelement allerdings noch nicht. Dies erledigen wir nach einer kurzen, aus technischen Gründen notwendigen Verzögerung. Dazu stellen wir das Zeitgeberintervall auf 50ms ein und hinterlegen eine Prozedur für das Ereignis **Bei Zeitgeber**, die nach diesem Zeitraum ausgelöst wird:

```
Private Sub Form_Timer()
    objWebbrowser.Navigate "about:blank"
    Set objDocument = objWebbrowser.Document
    Me.TimerInterval = 0
    DoEvents
    objDocument.body.innerHTML = HTMLCode
    Inzwischenablage objDocument.body.innerHTML
End Sub
```

Die Prozedur leert das Webbrowser-Steuerelement, weist der Objektvariablen **objDocument** das im Webbrowser-Steuerelement enthaltene Dokument zu, setzt **TimerInterval** auf **0**, damit diese Ereignisprozedur nicht nochmals ausgelöst wird, und weist dann der Eigenschaft **innerHTML** das Ergebnis der später vorgestellten Funktion **HTMLCode** zu. Diese liefert den vollständigen HTML-Code.

Damit Sie bei Änderungen des HTML-Codes die Änderungen direkt einsehen können, haben wir die Schaltfläche **cmdAktualisieren** hinzugefügt. Diese liest den HTML-Code neu ein:

```
Private Sub cmdAktualisieren_Click()
    objDocument.body.innerHTML = HTMLCode
    Inzwischenablage objDocument.body.innerHTML
End Sub
```

Außerdem rufen beide Prozeduren die Routine **InZwischenablage** auf, um den Inhalt des Webbrowser-Steuerelements in die Zwischenablage zu kopieren. Sie können diesen dann etwa in einen Texteditor einfügen, um sich den resultierenden HTML-Code anzusehen. Wenn Sie diese Funktion nicht benötigen, können Sie die Aufrufe von **InZwischenablage** auch einfach entfernen.

HTML-Code in zwei Prozeduren

Wir haben die Anweisungen für das Zusammenstellen des HTML-Codes zunächst in zwei Prozeduren aufgeteilt. Die erste übernimmt die Hauptaufgabe und stellt den eigentlichen HTML-Code zusammen, die zweite steuert das **Style**-Element mit den CSS-Eigenschaften bei.

Schauen wir uns zunächst die Prozedur **HTMLCode** an, die Sie in Listing 1 finden. Hier starten wir gleich mit der Deklaration einer Database- und einer Recordset-Varia-

blen sowie einer Textvariablen namens **strHTML**, in welcher wir den HTML-Code zusammenstellen.

Die Prozedur referenziert das aktuelle **Database**-Objekt mit der Variablen **db** und alle Datensätze der Abfrage **qryArtikelMitKategorieUndLieferant** mit der Recordset-Variablen **rst**.

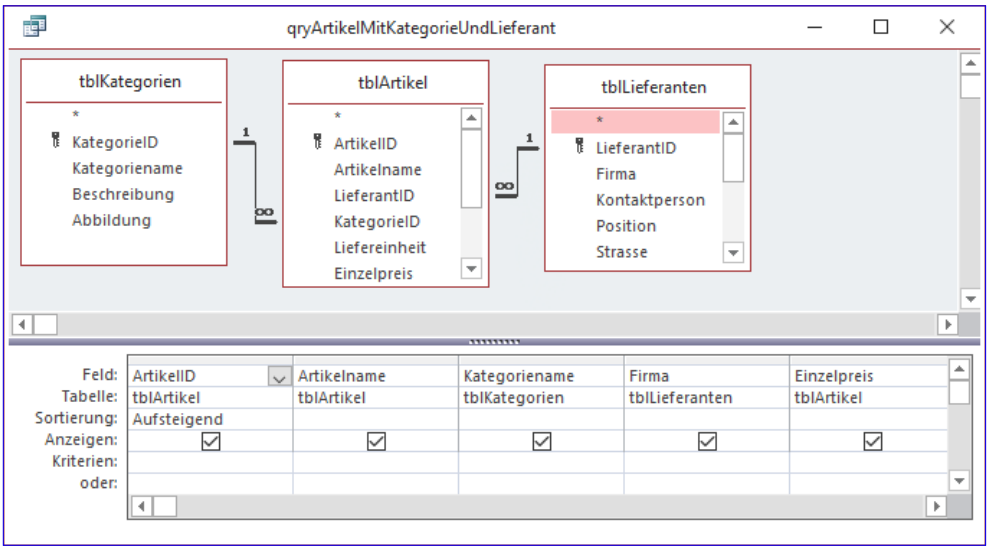


Bild 2: Datenquelle für die HTML-Tabelle

Die Abfrage **qryArtikelMitKategorieUndLieferant** stellt die Daten der Tabellen **tblArtikel**, **tblKategorien** und **tblLieferanten** wie in Bild 2 zusammen.

Im Anschluss beginnt die Prozedur mit dem Zusammenstellen des HTML-Codes in der Variablen **strHTML**. Direkt zu Beginn greift sie dabei auf die Funktion **HTMLStyle** zu, welche die CSS-Definition liefert – mehr dazu weiter unten. Der Teil nach der CSS-Definition sieht wie folgt aus. Dabei startet der HTML-Code mit einem **table**-Element mit der Klasse **tableheaders**:

```
<table class="tableheaders">
```

Danach folgen im **thead**-Element die Spaltenüberschriften, denen wir als Klasse **th1** bis **th6** zuweisen:

```
<thead>
  <tr>
    <th class="th1" scope="col">ID</th>
    <th class="th2" scope="col">Artikelname</th>
    <th class="th3" scope="col">Lieferant</th>
    <th class="th4" scope="col">Kategorie</th>
    <th class="th5" scope="col">Einzelpreis</th>
  </tr>
</thead>
```

Das **tbody**-Element nimmt zunächst ein **tr**- und ein **td**-Element auf, wobei das **td**-Element fünf Spalten umfassen soll (**colspan="5"**). Somit ist es genauso breit wie die fünf Überschriftenspalten:

```
<tbody>
  <tr>
    <td colspan="5">
```

Darunter beginnen wir ein **div**-Element, dass die Klasse **scrolling** abbildet:

```
      <div class="scrolling">
```

Darin finden wir schließlich die Tabelle mit den eigentlichen Daten, die in jeweils einem **tr**-Element je Zeile und fünf **th**- beziehungsweise **td**-Elementen für jede Spalte abgebildet werden. Hier haben wir beispielhaft die ersten beiden Elemente dargestellt, damit Sie einen kleinen, aber feinen Unterschied erkennen, der für jedes zweite **tr**-Element auftaucht – nämlich die Zuweisung einer Klasse für die alternative Hintergrundfarbe mit **class="dk"**:

```
        <table class="tablecontent">
          <tbody>
            <tr>
```

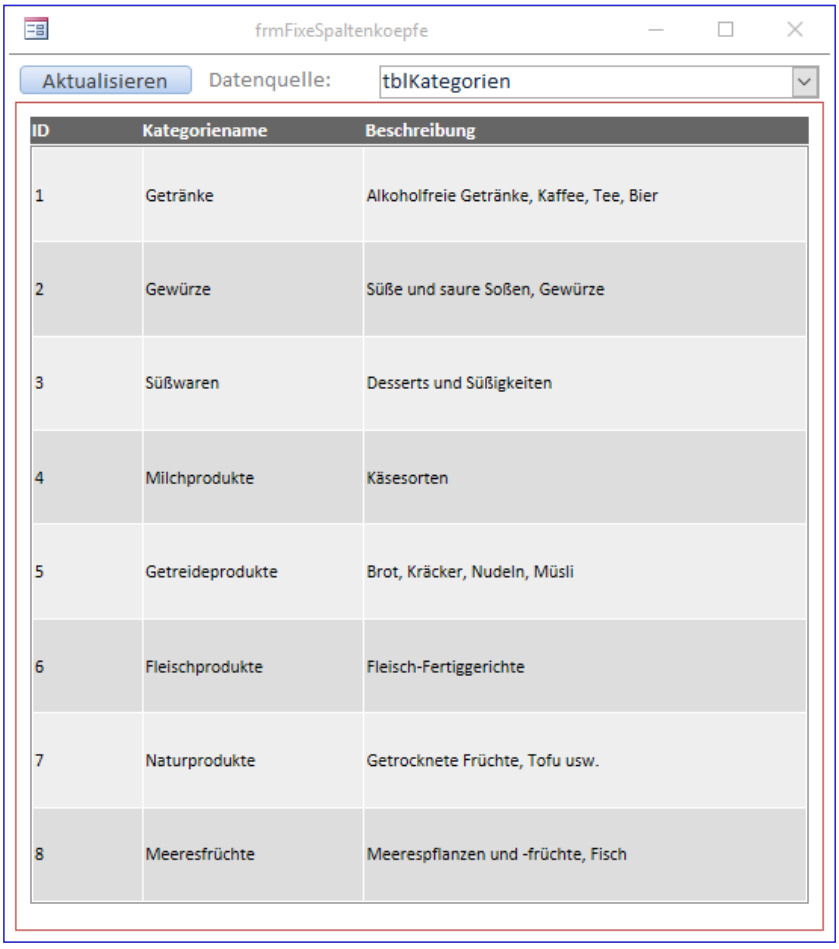
```
Private Function HTMLCode() As String
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim strHTML As String
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM qryArtikelMitKategorieUndLieferant", dbOpenDynaset)
    strHTML = strHTML & HTMLStyle
    strHTML = strHTML & "<table class=""tableheaders"">" & vbCrLf
    strHTML = strHTML & "  <thead>" & vbCrLf
    strHTML = strHTML & "    <tr>" & vbCrLf
    strHTML = strHTML & "      <th class=""th1"" scope=""col"">ID</th> " & vbCrLf
    strHTML = strHTML & "      <th class=""th2"" scope=""col"">Artikelname</th> " & vbCrLf
    strHTML = strHTML & "      <th class=""th3"" scope=""col"">Lieferant</th> " & vbCrLf
    strHTML = strHTML & "      <th class=""th4"" scope=""col"">Kategorie</th> " & vbCrLf
    strHTML = strHTML & "      <th class=""th5"" scope=""col"">Einzelpreis</th> " & vbCrLf
    strHTML = strHTML & "    </tr>" & vbCrLf
    strHTML = strHTML & "  </thead>" & vbCrLf
    strHTML = strHTML & "  <tbody>" & vbCrLf
    strHTML = strHTML & "    <tr>" & vbCrLf
    strHTML = strHTML & "      <td colspan=""5"">" & vbCrLf
    strHTML = strHTML & "        <div class=""scrolling"">" & vbCrLf
    strHTML = strHTML & "          <table class=""tablecontent"">" & vbCrLf
    strHTML = strHTML & "            <tbody>" & vbCrLf
    Do While Not rst.EOF
        If rst.AbsolutePosition Mod 2 = 0 Then
            strHTML = strHTML & "              <tr>" & vbCrLf
        Else
            strHTML = strHTML & "              <tr class=""dk"">" & vbCrLf
        End If
        strHTML = strHTML & "                <th class=""td1"" scope=""row"">" & rst!ArtikelID & "</th>" & vbCrLf
        strHTML = strHTML & "                <td class=""td2"">" & rst!Artikelname & "</td>" & vbCrLf
        strHTML = strHTML & "                <td class=""td3"">" & rst!Firma & "</td>" & vbCrLf
        strHTML = strHTML & "                <td class=""td4"">" & rst!Kategoriename & "</td>" & vbCrLf
        strHTML = strHTML & "                <td class=""td5"">" & Format(rst!Einzelpreis, "0.00 €") & "</td>" _
            & vbCrLf
        strHTML = strHTML & "              </tr>" & vbCrLf
        rst.MoveNext
    Loop
    strHTML = strHTML & "            </tbody>" & vbCrLf
    strHTML = strHTML & "          </table>" & vbCrLf
    strHTML = strHTML & "        </div>" & vbCrLf
    strHTML = strHTML & "      </td>" & vbCrLf
    strHTML = strHTML & "    </tr>" & vbCrLf
    strHTML = strHTML & "  </tbody>" & vbCrLf
    strHTML = strHTML & "</table>" & vbCrLf
    HTMLCode = strHTML
End Function
```

Listing 1: Listing zum Zusammenstellen des HTML-Codes

Flexible HTML-Tabellen mit fester Kopfzeile

Im Beitrag »HTML-Tabellen mit fester Kopfzeile« haben wir gezeigt, wie Sie Daten aus einer bestimmten Abfrage in einem Webbrowser-Steuerelement so anzeigen, dass die Spaltenköpfe oben fixiert bleiben, während der Benutzer den Inhalt der Tabelle, also die eigentlichen Datensätze, nach unten scrollt. Wir wollen das Beispiel aus diesem Beitrag nun so erweitern, dass Sie die scrollbare HTML-Tabelle für beliebige Tabellen oder Abfragen als Datenquelle nutzen können. Diese sollen per Parameter beim Öffnen des Formulars festgelegt werden. Das Layout der Tabellen sowie die anzuzeigenden Daten sollen dann automatisch ermittelt werden – samt den enthaltenen Feldnamen oder -bezeichnungen als Spaltenköpfen.

Das Ziel dieses Beitrags ist, die Lösung aus dem Artikel **HTML-Tabellen mit fester Kopfzeile** (**www.access-im-unternehmen.de/1188**) so anzupassen, dass Sie damit Daten aus beliebigen Tabellen oder Abfragen in einer scrollbaren HTML-Tabelle anzeigen können.



ID	Kategoriename	Beschreibung
1	Getränke	Alkoholfreie Getränke, Kaffee, Tee, Bier
2	Gewürze	Süße und saure Soßen, Gewürze
3	Süßwaren	Desserts und Süßigkeiten
4	Milchprodukte	Käsesorten
5	Getreideprodukte	Brot, Kräcker, Nudeln, Müsli
6	Fleischprodukte	Fleisch-Fertiggerichte
7	Naturprodukte	Getrocknete Früchte, Tofu usw.
8	Meeresfrüchte	Meerespflanzen und -früchte, Fisch

Bild 1: HTML-Tabelle mit Scrollbalken

Das Ergebnis soll dann etwa so aussehen, dass Sie wie in Bild 1 einfach die anzuzeigende Tabelle oder Abfrage aus einem Kombinationsfeld auswählen können, und das Webbrowser-Steuerelement eine Tabelle mit den Daten der gewählten Tabelle oder Abfrage anzeigt.

Auswahl der Datenquelle

Dazu benötigen wir zunächst einmal ein Kombinationsfeld, mit dem Sie die Datenquelle auswählen können.

Dieses fügen wir dem Kopf des Formulars wie in Bild 2 hinzu und stellen die Eigenschaft **Name** auf **cboDatenquelle** ein.

Als Datensatzherkunft des Kombinationsfeldes verwenden wir die folgende Abfrage:

```
SELECT Name FROM MSysObjects WHERE
Type In (1,5) And Not Left(Name,1)='~'
And Not Left(Name,4)='MSys' And Not
Left(Name,2)='f_' ORDER BY Name;
```

Diese Abfrage ermittelt alle Tabellen und Abfragen aus der Systemtabelle **MSysObjects** und sortiert diese nach dem Namen. Das Kombinationsfeld zeigt die Einträge dann etwa wie in Bild 3 an.

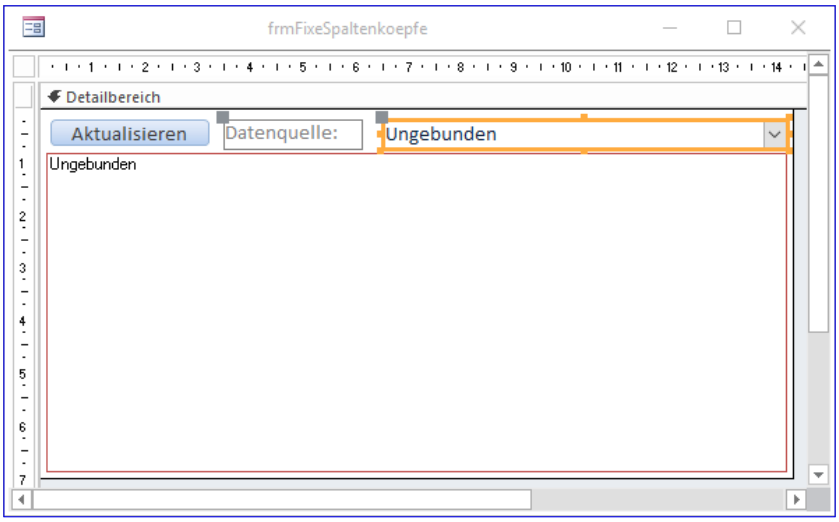


Bild 2: Hinzufügen des Kombinationsfeldes zur Auswahl der Datenquelle

Daten beim Öffnen des Formulars anzeigen

Beim Öffnen des Formulars soll das Webbrowser-Steuerelement direkt die Daten der ersten Datenquelle des Kombinationsfeldes anzeigen.

Dazu sorgen wir dafür, dass direkt der erste Eintrag im Kombinationsfeld vorausgewählt wird, und zwar durch die zweite Zeile der Ereignisprozedur **Form_Open**:

```
Private Sub Form_Open(Cancel As Integer)
    Set objWebbrowser = Me!ctlWebbrowser.Object
    Me!cboDatenquelle = Me!cboDatenquelle.ItemData(0)
    Me.TimerInterval = 50
End Sub
```

Durch Einstellen der Eigenschaft **TimerInterval** auf den Wert **50** wird 50 Millisekunden nach dem Öffnen des Formulars die folgende **Form_Timer**-Prozedur ausgelöst:

```
Private Sub Form_Timer()
    objWebbrowser.Navigate "about:blank"
    Set objDocument = objWebbrowser.Document
    Me.TimerInterval = 0
    DoEvents
    objDocument.body.innerHTML = HTMLCode(Me!cboDatenquelle)
    Inzwischenablage objDocument.body.innerHTML
End Sub
```

Diese unterscheidet sich in einem Detail von der entsprechenden Prozedur, die wir im oben genannten Beitrag vorgestellt haben. Sie übergibt beim Aufruf der Funktion **HTMLCode**, welche den HTML-Code für die Darstellung der Tabelle zusammensetzt, für den Parameter **strDatenquelle** den Namen der Datenquelle.

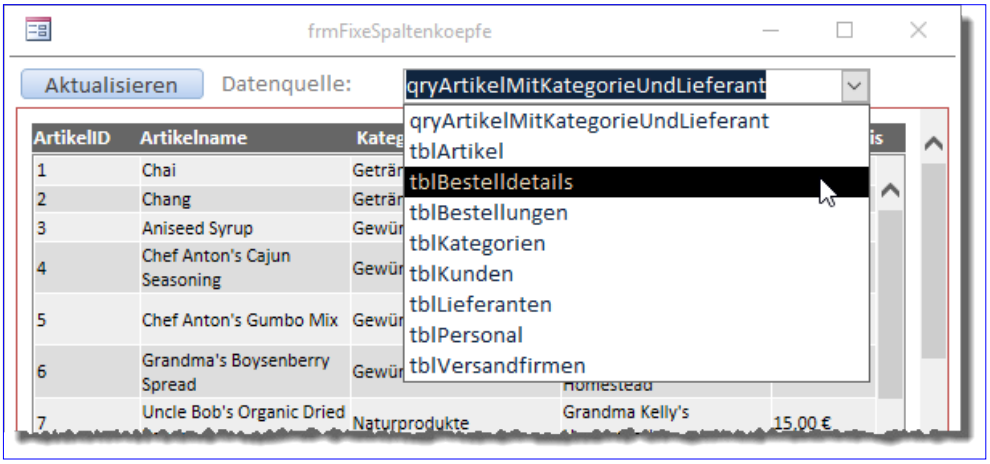


Bild 3: Auswahl der Datenquelle für die HTML-Tabelle

Das Gleiche geschieht auch in der Prozedur, die der Benutzer durch einen Mausklick auf die Schaltfläche **cmdAktualisieren** auslöst. Die Ereignisprozedur ruft ebenfalls die Funktion **HTMLCode** mit dem Wert des Kombinationsfeldes

cboDatenquelle als Parameter auf. Diese Prozedur wird dazu wie folgt geändert:

```
Private Sub cmdAktualisieren_Click()
    objDocument.body.innerHTML = HTMLCode(Me!cboDatenquelle)
    Inzwischenablage objDocument.body.innerHTML
End Sub
```

Schließlich möchten wir auch, dass der Inhalt des Webbrowser-Steuerelements auch beim Aktualisieren der ausgewählten Datenquelle über das Kombinationsfeld **cboDatenquelle** aktualisiert wird, was wir mit der folgenden Ereignisprozedur realisieren:

```
Private Sub cboDatenquelle_AfterUpdate()
    If Not Len(Me!cboDatenquelle) = 0 Then
        objDocument.body.innerHTML = _
                            HTMLCode(Me!cboDatenquelle)
    End If
End Sub
```

HTML-Code datenquellenabhängig zusammenstellen

Damit kommen wir zum interessanten Teil, nämlich dem Code, mit dem wir den HTML-Code abhängig von der gewählten Datenquelle zusammenstellen. Den ersten Teil dieser Funktion finden Sie in Listing 1.

Die Funktion erwartet den Namen der mit dem Kombinationsfeld ausgewählten Tabelle oder Abfrage als Parameter. Sie erstellt ein **Database**-Objekt namens **db** sowie ein **Recordset**-Objekt namens **rst**, welches die ersten 100 Datensätze der gewählten Datenquelle auswählt. Die Erstellung des HTML-Codes dauert für größere Datenmengen relativ lange, daher haben wir diese Beschränkung für die Beispieldatei eingefügt. Sie können **TOP 100** in der folgenden Anweisung entfernen, wenn Sie immer alle Datensätze laden wollen:

```
    Set rst = db.OpenRecordset("SELECT TOP 100 * FROM " _
                        & strDatenquelle, dbOpenDynaset)
```

Danach ruft die Funktion eine weitere Funktion namens **HTMLStyle** auf und übergibt dieser das Recordset als Parameter. Die Beschreibung dieser Funktion, die den CSS-Code in Abhängigkeit der Datenquelle zusammenstellt, finden Sie weiter unten.

Danach fügt die Funktion die ersten HTML-Elemente für den Kopfbereich der Tabelle beziehungsweise die umfassende Tabelle an. Nach dem Anlegen der ersten drei Elemente **table**, **thead** und **tr** folgenden die Spaltenüberschriften für die Felder der Datenquelle.

Um dabei auf die jeweiligen Felder der gewählten Datenquelle zuzugreifen, durchlaufen wir die Felder des **Recordset**-Objekts **rst** in einer **For Each**-Schleife und weisen das jeweilige **Field**-Objekt der Variablen **fld** zu. Innerhalb dieser Schleife prüfen wir in einer **Select Case**-Bedingung den Typ des jeweiligen Feldes. Wir wollen nicht alle Felddatentypen in der Tabelle aufführen – so schließen wir beispielsweise OLE- und Anlagefelder aus. Die erste **Case**-Bedingung behandelt somit die Datentypen **Integer**, **Long**, **Text**, **Memo**, **Currency**, **Date**, **Single**, **Double** und **Boolean**. Weitere Datentypen können Sie nach Bedarf hinzufügen. Sollten Sie die Funktion in einer eigenen Anwendung einsetzen und die dort gewählte Datenquelle enthält ein Feld mit einem Datentyp, der hier nicht behandelt wird, gibt die Prozedur die Nummer des Datentyps im Direktbereich des VBA-Editors aus – dies geschieht im **Else**-Zweig der **Select Case**-Bedingung.

Handelt es sich bei **fld** um ein Feld mit einem unterstützten Datentyp, ermittelt die Funktion entweder den Feldnamen des Feldes oder, falls vorhanden, den Wert der Eigenschaft **Beschriftung**, den Sie in den Feldeigenschaften im Tabellenentwurf einstellen können.

Wir stellen die Variable **strFeldname**, die den Feldnamen aufnimmt, zunächst auf eine leere Zeichenkette ein. Dann schalten wir die Fehlerbehandlung temporär aus, und zwar für den Versuch, den Wert der Eigenschaft **Caption** über die **Properties**-Auflistung des Feldes zu ermitteln.

```
Private Function HTMLCode(strDatenquelle As String) As String
    Dim db As DAO.Database, rst As DAO.Recordset, fld As DAO.Field
    Dim strHTML As String, strFeldname As String
    Dim i As Integer
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT TOP 100 * FROM " & strDatenquelle, dbOpenDynaset)
    strHTML = strHTML & HTMLStyle(rst)
    strHTML = strHTML & "<table class=""tableheaders"">" & vbCrLf
    strHTML = strHTML & "  <thead>" & vbCrLf
    strHTML = strHTML & "    <tr>" & vbCrLf
    For Each fld In rst.Fields
        Select Case fld.Type
            Case dbInteger, dbLong, dbText, dbMemo, dbCurrency, dbDate, dbSingle, dbDouble, dbBoolean
                strFeldname = ""
                On Error Resume Next
                strFeldname = fld.Properties("Caption")
                On Error GoTo 0
                If Len(strFeldname) = 0 Then
                    strFeldname = fld.Name
                End If
                i = i + 1
                strHTML = strHTML & "      <th class=""th" & i & """ scope=""col"">" & strFeldname & "</th> " & vbCrLf
            Case Else
        End Select
    Next fld
    strHTML = strHTML & "    </tr>" & vbCrLf
    strHTML = strHTML & "  </thead>" & vbCrLf
    ...
End Function
```

Listing 1: Listing zum Zusammenstellen des HTML-Codes, Teil 1

Gelingt dies, enthält **strFeldname** danach die Beschriftung dieses Feldes. Dies prüfen wir in der folgenden **If...Then**-Bedingung. Hat **strFeldname** dort eine Länge ungleich **0**, tragen wir den Feldnamen aus der Eigenschaft **Name** in die Variable **strFeldname** ein und verwenden diesen als Spaltenüberschrift. Nun erhöhen wir eine Zählervariable namens **i** um den Wert **1** und stellen die HTML-Zeile für die aktuelle Spaltenüberschrift zusammen. Diese sieht dann etwa wie folgt aus:

```
<th class="th1" scope="col">ArtikelID</th>
```

Danach durchlaufen wir die übrigen Felder des Recordsets und stellen so die Spaltenüberschriften zusammen. Es folgen die schließenden Elemente wie **</tr>** und **</thead>**, bevor sich der folgende Teil um die Darstellung der Daten in der HTML-Tabelle kümmert.

Daten in die HTML-Tabelle schreiben

Damit kommen wir zum zweiten Teil der Prozedur, den Sie in Listing 2 finden.

Hier legen wir zunächst die umgebenden Elemente wie **<tbody>**, **<tr>** und **<td>** zusammen, wobei wir den Wert für das Attribut **colspan** aus der Anzahl der Felder aus der Variablen **i** ermitteln, welche für jedes berücksichtigte Feld um den Wert **1** erhöht wurde und nicht die Anzahl der tatsächlichen Felder wiedergibt, sondern

```
Private Function HTMLCode(strDatenquelle As String) As String
    ...
    strHTML = strHTML & "  <tbody>" & vbCrLf
    strHTML = strHTML & "    <tr>" & vbCrLf
    strHTML = strHTML & "      <td colspan=""" & rst.Fields.Count & """>" & vbCrLf
    strHTML = strHTML & "        <div class=""scrolling"">" & vbCrLf
    strHTML = strHTML & "          <table class=""tablecontent"">" & vbCrLf
    strHTML = strHTML & "            <tbody>" & vbCrLf
    Do While Not rst.EOF
        If rst.AbsolutePosition Mod 2 = 0 Then
            strHTML = strHTML & "              <tr>" & vbCrLf
        Else
            strHTML = strHTML & "              <tr class=""dk"">" & vbCrLf
        End If
        i = 0
        For Each fld In rst.Fields
            i = i + 1
            Select Case fld.Type
                Case dbInteger, dbLong, dbText, dbMemo, dbDate
                    strHTML = strHTML & "                <td class=""td" & i & """>" & fld.Value & "</td>" _
                        & vbCrLf
                Case dbCurrency
                    strHTML = strHTML & "                <td class=""td" & i & """>" & Format(fld.Value, "0.00 €") _
                        & "</td>" & vbCrLf
                Case dbSingle, dbDouble
                    strHTML = strHTML & "                <td class=""td" & i & """>" & Format(fld.Value, "0.00") _
                        & "</td>" & vbCrLf
                Case dbBoolean
                    strHTML = strHTML & "                <td class=""td" & i & """>" & IIf(fld.Value = -1, _
                        "Ja", "Nein") & "</td>" & vbCrLf
                Case Else
                    Debug.Print fld.Type
            End Select
        Next fld
        strHTML = strHTML & "              </tr>" & vbCrLf
        rst.MoveNext
    Loop
    strHTML = strHTML & "            </tbody>" & vbCrLf
    strHTML = strHTML & "          </table>" & vbCrLf
    strHTML = strHTML & "        </div>" & vbCrLf
    strHTML = strHTML & "      </td>" & vbCrLf
    strHTML = strHTML & "    </tr>" & vbCrLf
    strHTML = strHTML & "  </tbody>" & vbCrLf
    strHTML = strHTML & "</table>" & vbCrLf
    HTMLCode = strHTML
End Function
```

Listing 2: Listing zum Zusammenstellen des HTML-Codes, Teil 2

Benutzerverwaltung mit verschlüsselten Kennwörtern

Wenn Sie in einer Access-Anwendung Benutzer verwalten wollen, die sich per Benutzername und Kennwort an die Anwendung anmelden, sollten Sie sehr sensibel mit den in der Anwendung gespeicherten Kennwörtern umgehen. Das geht am einfachsten, wenn Sie die Kennwörter verschlüsselt speichern. Damit verringern Sie die Wahrscheinlichkeit, dass jemand die Kennwörter entschlüsselt und für seine Zwecke nutzen kann, erheblich. Dieser Artikel stellt eine kleine Benutzerverwaltung vor, mit der Sie Benutzer und Benutzergruppen verwalten und auch Berechtigungen für verschiedene Tabellen vergeben können.

Die Benutzerverwaltung

Zur Benutzerverwaltung gehören einige Tabellen, mit denen wir die Benutzer, die Benutzergruppen, die Zuordnung der Benutzer zu den Benutzergruppen, die Tabellen, für die wir Berechtigungen vergeben wollen, sowie die Berechtigungen selbst verwalten wollen.

Außerdem gehört dazu ein Formular für die Anmeldung des jeweiligen Benutzers. Dieses nimmt den Benutzernamen und das Kennwort entgegen und prüft, ob ein passendes Paar dieser beiden Daten in der Tabelle der Benutzer gefunden werden kann. In diesem Fall soll eine temporäre Variable mit dem Namen des Benutzers gefüllt werden.

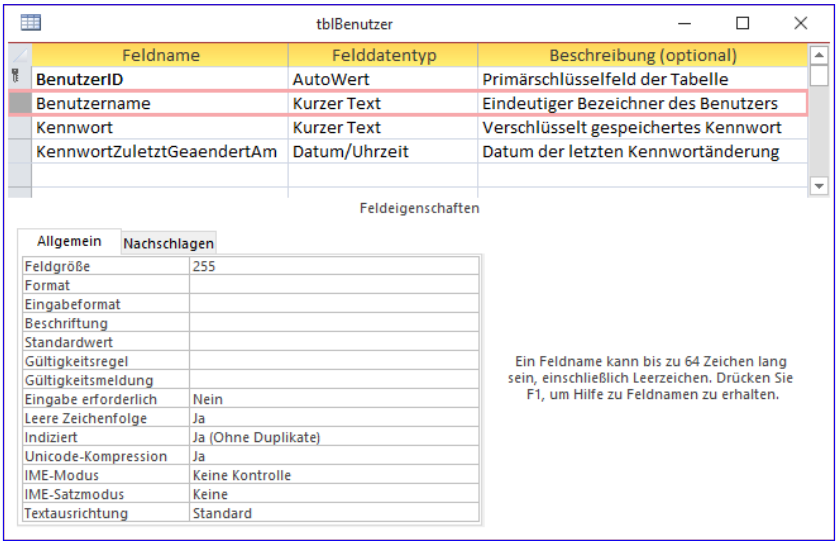


Bild 1: Tabelle zur Verwaltung der Benutzer

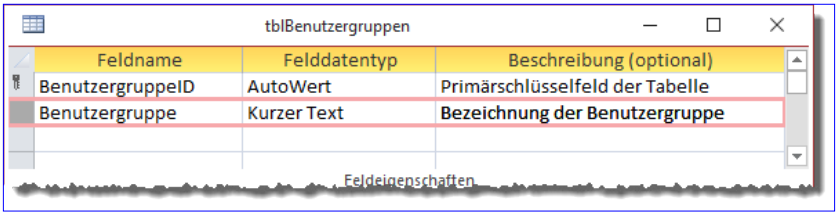


Bild 2: Tabelle zur Verwaltung der Benutzergruppen

Weitere Mechanismen, welche die Prüfung der Berechtigungen dieses Benutzers beim Zugriff auf die Daten durchführen, nutzen den Inhalt dieser Variablen. Diese Mechanismen stellen wir beispielsweise im Beitrag **Zugriffsrechte mit Datenmakros** (**www.access-im-unternehmen.de/1193**) vor.

Tabellen der Benutzerverwaltung

Die erste Tabelle der Benutzerverwaltung heißt **tblBenutzer** und dient zum Speichern der Benutzerdaten, in diesem Fall des Benutzernamens und des Kennworts. Der Entwurf dieser Tabelle sieht wie in Bild 1 aus. Für das Feld **Benutzername** legen wir einen eindeutigen Index fest, damit kein Benutzername zwei Mal vergeben werden kann. Außerdem fügen wir noch ein Feld namens **KennwortZuletztGeaendertAm** mit dem Datentyp

Datum/Uhrzeit hinzu, das speichert, wann das Kennwort des Benutzers zuletzt geändert wurde.

Die Tabelle **tblBenutzergruppen** speichert die Namen der Benutzergruppen (siehe Bild 2). Das einzige Feld neben dem Primärschlüsselfeld heißt **Benutzergruppe** und besitzt ebenfalls einen eindeutigen Index, damit jede Benutzergruppe nur einmal angegeben werden kann.

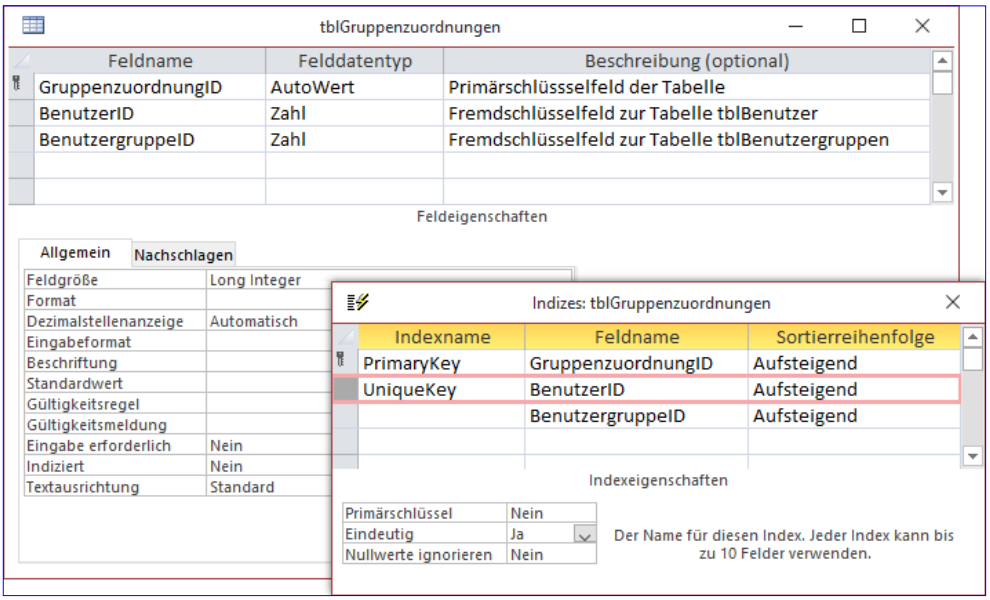


Bild 3: Tabelle zur Zuordnung von Benutzern zu Benutzergruppen

Um die Benutzer den Benutzergruppen zuzuordnen, erstellen wir eine weitere Tabelle namens **tblGruppenzuordnungen**. Diese enthält neben dem Primärschlüsselfeld **GruppenzuordnungID** noch zwei Fremdschlüsselfelder namens **BenutzerID** und **BenutzergruppeID**. Beide Beziehungen haben wir mit dem Nachschlageassistenten eingerichtet. Damit jeder Benutzer nur einmal zu jeder Benutzergruppe zugeordnet werden kann, fügen wir der Tabelle einen zusammengesetzten, eindeutigen Index über die beiden Felder **BenutzerID** und **BenutzergruppeID** hinzu (siehe Bild 3).

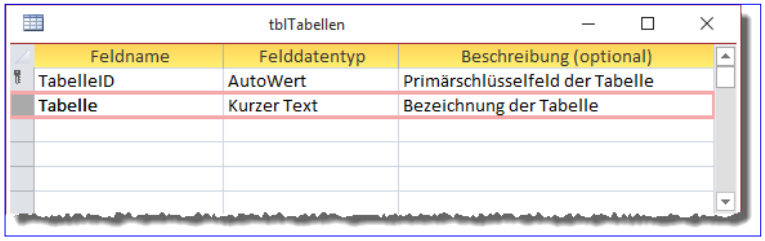


Bild 4: Tabelle zum Speichern der Tabellennamen

Tabelle zum Speichern der Tabellen

Wir wollen für jede Benutzergruppe die Berechtigungen für den Zugriff auf die Tabellen der Datenbank festlegen. Dazu benötigen wir erst einmal eine Tabelle mit den Tabellen der Datenbank. Diese heißt **tblTabellen** und sieht im Entwurf wie in Bild 4 aus.

Füllen der Tabelle mit Tabellennamen

Damit das Eintragen der Tabellen nicht zu aufwendig wird, haben wir eine kleine Prozedur geschrieben, mit der Sie die Bezeichnungen der Tabellen schnell in die Tabelle **tblTabellen** eintragen können. Diese finden Sie in Listing 1. Die Prozedur durchläuft alle Elemente der Auflistung **TableDefs** des **Database**-Objekts für die aktuelle Datenbank. Dabei legt es jeweils einen neuen Datensatz in der Tabelle **tblTabellen** an und trägt die Bezeichnung der Tabelle ein. Dabei untersucht die Prozedur noch die Bezeichnung der Tabelle und trägt nur solche Tabellenbezeichnungen ein, die nicht mit **MSys** (Systemtabellen) oder **~** beginnen (temporäre Tabellen).

Tabelle der Berechtigungen

Die Tabelle **tblBerechtigungen** soll festlegen, welche Berechtigungen für die Kombinationen aus Benutzergruppen und Tabellen festgelegt werden können. Dabei haben wir für diese Tabelle ausnahmsweise einmal kein Primärschlüsselfeld mit dem Datentyp **Autowert** definiert, sondern eines mit dem Datentyp **Zahl**. Auf diese Weise

```
Public Sub TabellenEintragen()
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    Set db = CurrentDb
    For Each tdf In db.TableDefs
        If Not tdf.Name Like "MSys*" And Not tdf.Name Like "~*" Then
            On Error Resume Next
            db.Execute "INSERT INTO tblTabellen(Tabelle) VALUES('" & tdf.Name & "')", dbFailOnError
            On Error GoTo 0
        End If
    Next tdf
End Sub
```

Listing 1: Prozedur zum Eintragen der Tabellennamen

können wir die Werte für das Primärschlüsselfeld selbst vergeben. Das zweite Feld heißt **Berechtigung** und soll die Bezeichnung der Berechtigung enthalten (siehe Bild 5). Auch für dieses Feld haben wir wieder einen eindeutigen Index festgelegt.

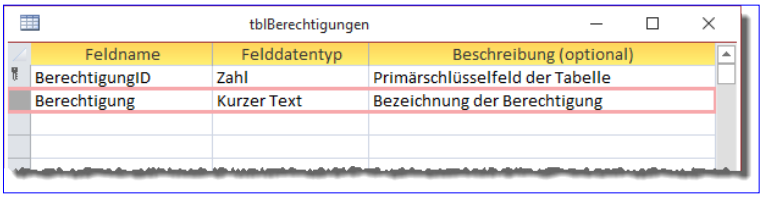


Bild 5: Tabelle zum Speichern der Berechtigungen

Werte der Tabelle tblBerechtigungen

Warum aber wollen wir für das Primärschlüsselfeld dieser Tabelle nicht die Autowert-Funktion verwenden? Weil wir die Werte für das Primärschlüsselfeld selbst vergeben wollen. Diese sollen nämlich wie in Bild 6 aussehen. Die Berechtigung **Keine** erhält beispielsweise den Wert **0**. **Lesen** erhält den Wert **1**, **Anlegen** den Wert **2**, **Ändern** den Wert **4** und **Löschen** den Wert **8**.

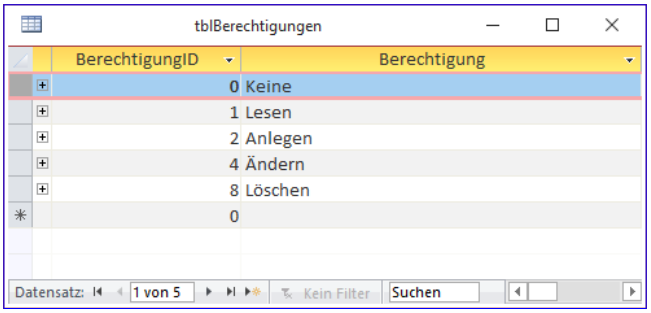


Bild 6: Werte der Berechtigungen

Das Muster ist leicht zu erkennen: Die Werte entsprechen allesamt Zweierpotenzen. Warum das? Weil die Berechtigungen kombinierbar sein sollen und wir die Kombinationen in einem einzigen Wert erfassen wollen. Wenn wir nur Zweierpotenzen für diese Werte verwenden, können wir durch Summieren der Werte einzelne Berechtigungen zusammenfassen.

Der Wert **1** würde bedeuten, dass der Benutzer nur lesende Rechte an der Tabelle hat. Der Wert **5** (aus **1 + 4**) heißt, der Benutzer hat Leserechte und darf die Datensätze ändern. Der Wert **15** (aus **1 + 2 + 4 + 8**) bedeutet, dass der Benutzer alle Rechte an den Datensätzen der Tabelle hat. Wir schauen uns später im Detail an, wie wir diese Werte nutzen können.

Alternative: Alle Kombinationen?

Die Alternative hierzu wäre, gleich alle Kombinationen in der Tabelle zu speichern, also etwa noch Einträge hinzuzufügen wie **Lesen und Anlegen**, **Lesen und Ändern**, **Lesen und Löschen** bis hinzu zu **Lesen, Anlegen, Bearbeiten und Löschen**. Dann wäre nur die Auswahl eines Eintrags für die Ermittlung aller Berechtigungen nötig. Allerdings müsste man dann auch immer, wenn man nur

wissen will, ob der Benutzer einer Benutzergruppe etwa Löschrechte besitzt, den Zahlenwert interpretieren. Also fahren wir mit der Variante, bei der wir die einzelnen Rechte explizit und einzeln setzen müssen.

Tabellen zum Zuordnen der Berechtigungen

Fehlt noch eine Tabelle, mit der wir die verschiedenen Berechtigungen für die Kombination aus Benutzergruppe und Tabelle festlegen.

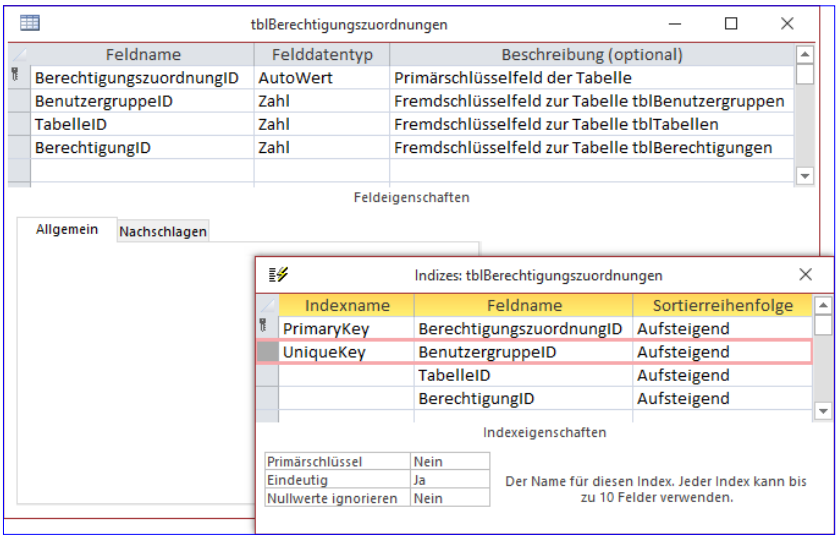


Bild 7: Tabelle zur Zuordnung der Berechtigungen

Die Tabelle heißt **tblBerechtigungszuordnungen** und sieht in der Entwurfsansicht wie in Bild 7 aus. Diese Tabelle hat gleich drei Fremdschlüsselfelder, mit denen die Datensätze der Tabellen **tblBenutzergruppen**, **tblTabellen** und **tblBerechtigungen** kombiniert werden. Damit auch hier jede Kombination nur einmal vorkommt, haben wir einen zusammengesetzten, eindeutigen Index für die drei Felder **BenutzergruppeID**, **TabelleID** und **BerechtigungID** angelegt.

Benutzer verwalten

Für die Verwaltung der Benutzer und die Zuordnung zu den Benutzergruppen legen wir ein Formular an, dass an die Tabelle **tblBenutzer** gebunden ist. Zwei Listenfelder namens **lstZugewiesen** und **lstNichtZugewiesen** dienen der Zuweisung der Benutzergruppen zu den Benutzern (siehe Bild 8).

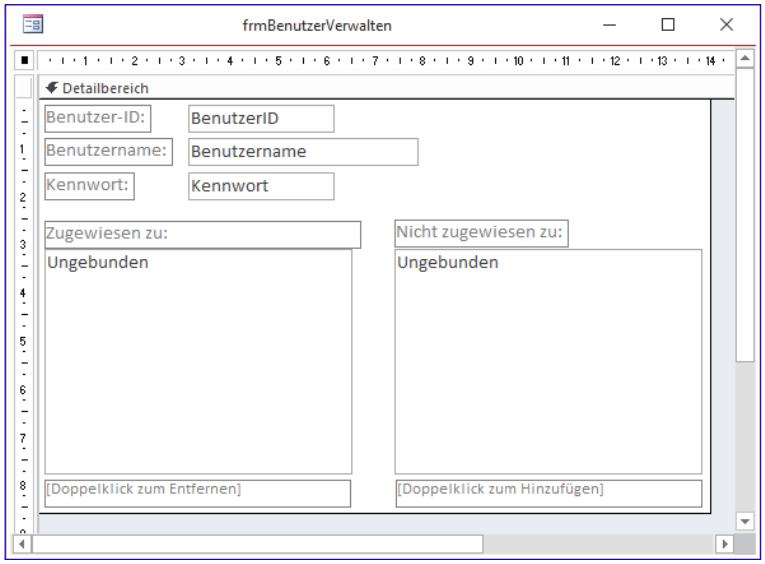


Bild 8: Formular zum Verwalten der Benutzer

Das erste Listenfeld zeigt alle Einträge der Tabelle **tblBenutzergruppen** an, die mit dem aktuell im Formular angezeigten Benutzer verknüpft sind. Dazu legen wir für das Listenfeld die Abfrage mit

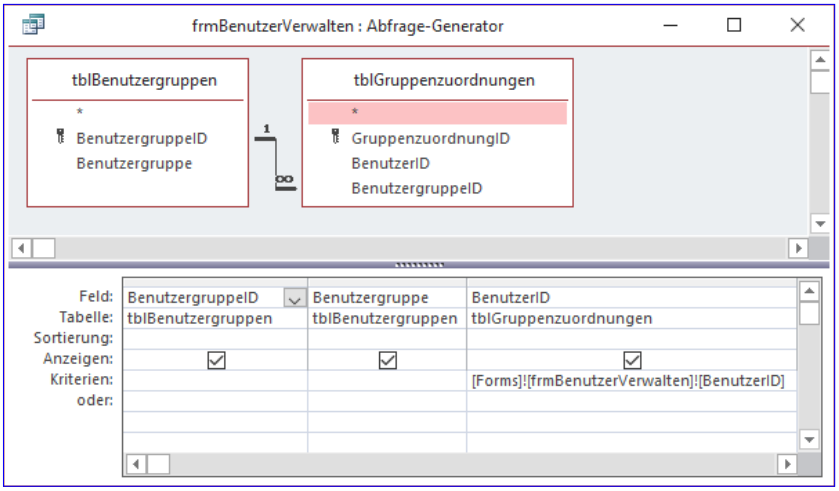


Bild 9: Datensatzherkunft des Listenfeldes **lstZugewiesen**

Berechtigungen per HTML verwalten

Im Beitrag »Benutzerverwaltung mit verschlüsselten Kennwörtern« stellen wir eine Lösung vor, in der wir die Berechtigungen von Benutzergruppen an Datenbankobjekten definieren. Dort benötigen wir ein Formular, mit dem wir die Berechtigungen für die einzelnen Objekte für die jeweilige Benutzergruppe definieren wollen. Dazu wollen wir eine Matrix anzeigen, welche die Berechtigungen und die Objekte als Spalten- und Zeilenüberschriften anzeigt und die Möglichkeit bietet, in den Zellen anzugeben, ob die Berechtigung gesetzt ist oder nicht. Dies erledigen wir mit einem Webbrowser-Steuerelement, indem wir die notwendigen Elemente per HTML darstellen.

Der erste Entwurf des Formulars sieht wie in Bild 1 aus. Wir haben dem Formular ein Kombinationsfeld namens **cboBenutzergruppeID** hinzugefügt sowie ein Webbrowser-Steuerelement namens **objWebbrowser**.

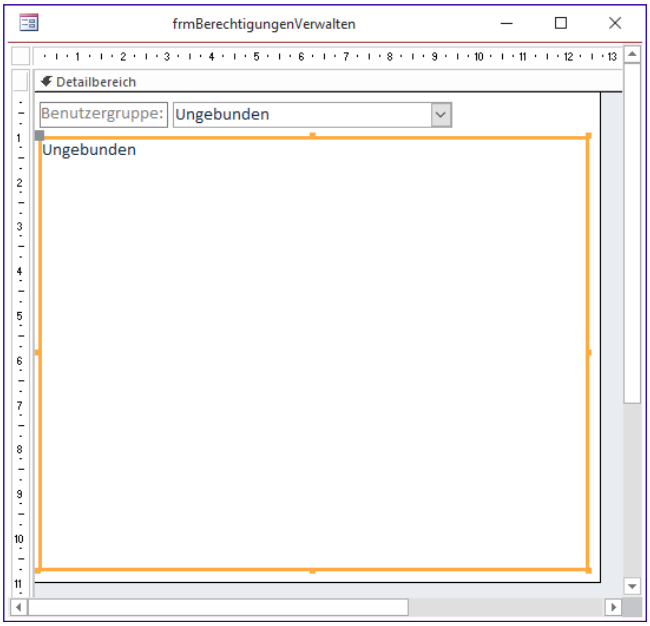


Bild 1: Erster Entwurf des Formulars

Das Kombinationsfeld soll die Benutzergruppen aus der Tabelle **tblBenutzergruppen** anzeigen, und zwar so, dass das Feld **BenutzergruppeID** als gebundene Spalte ausgeblendet wird und nur die Bezeichnung der Benutzergruppe erscheint. Dazu weisen wir der Eigenschaft **Datensatzherkunft** die folgende Abfrage zu:

```
SELECT BenutzergruppeID, Benutzergruppe
FROM tblBenutzergruppen
ORDER BYBenutzergruppe;
```

Damit nur die Benutzergruppe angezeigt wird, stellen wir die Eigenschaft **Spaltenanzahl** auf **2** und die Eigenschaft **Spaltenbreiten** auf **0cm** ein.

Für das Zusammenstellen des HTML-Code benötigen wir noch die Objektbibliothek **Microsoft HTML Object Library**, die wir wie in Bild 2 über den **Verweise**-Dialog des VBA-Editors hinzufügen (Menüeintrag **Extras\|Verweise**).

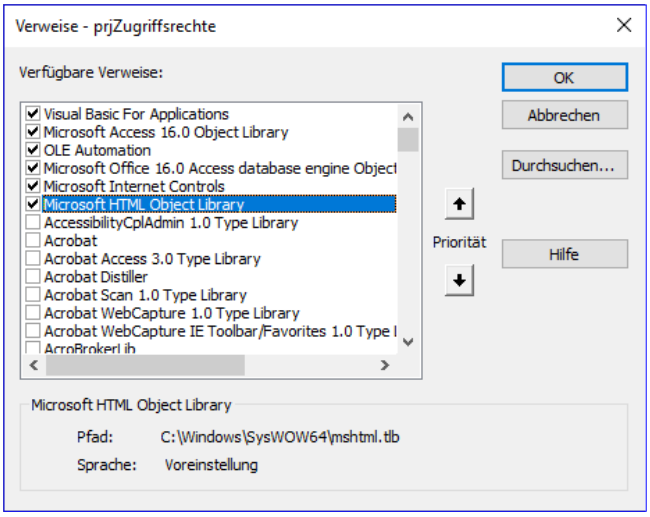


Bild 2: Verweise des VBA-Projekts

Gewünschte Darstellung

Wir möchten nicht einfach nur eine Darstellung mit Kontrollkästchen oder ähnlichen Steuerelementen, sondern einmal etwas Besonderes programmieren: nämlich eine

Darstellung, bei der die gesetzten Rechte in Grün und die nicht gesetzten Rechte in Rot dargestellt werden. Das sieht dann etwa für die Benutzergruppe **Administratoren** wie in Bild 3 aus.

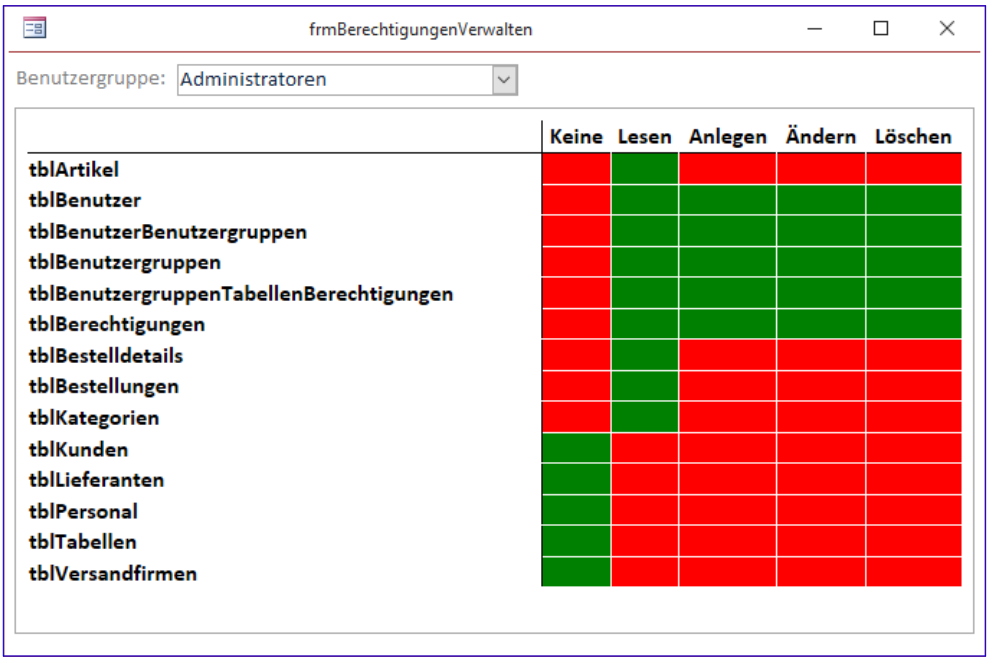


Bild 3: Setzen der Berechtigungen über verschiedene Farben

Funktionen des Formulars

Das Formular soll, wie oben erwähnt, die Auswahl einer der Benutzergruppen über das Kombinationsfeld oben im Formular ermöglichen. Nach der Auswahl sollen die Berechtigungen für die gewählte Benutzergruppe angezeigt werden.

Außerdem wollen wir die folgenden Funktionen durch Anklicken der verschiedenen Bereiche der HTML-Seite im Webbrowser-Steuerelement realisieren:

- Wenn der Benutzer auf die rot markierte Berechtigung **Keine** klickt und diese so auf Grün einstellt, sollen automatisch die übrigen Berechtigungen für die Tabelle für diesen Benutzer auf Rot eingestellt werden.

- Wenn der Benutzer die grün markierte Berechtigung **Keine** anklickt und diese so auf Rot einstellt, sollen die übrigen Berechtigungen für diese Tabelle auf Grün eingestellt werden.

- Wenn der Benutzer eine der Berechtigungen **Lesen**, **Anlegen**, **Ändern** oder **Löschen** anklickt, während diese Rot ist, soll diese auf Grün eingestellt werden und umgekehrt.

- Dabei gilt die folgende Sonderregel: Wenn er die letzte grün markierte Berechtigung (also **Lesen**, **Anlegen**, **Ändern** oder **Löschen**) anklickt und diese so auf Rot einstellt, soll automatisch die Berechtigung **Keine** auf Grün eingestellt werden.

- Klickt der Benutzer auf einen der Spaltenköpfe **Keine**, **Lesen**, **Anlegen**, **Ändern** oder **Löschen**, werden alle Einträge dieser Spalte auf **Grün** eingestellt. Handelt es sich um den Spaltenkopf **Keine**, werden alle anderen Spalten komplett auf Rot eingestellt. Handelt es sich um eine der anderen Spalten, wird diese komplett auf Grün eingestellt und die Spalte **Keine** logischerweise komplett auf Rot.

Wenn **Keine** grün ist, müssen also alle anderen Spalten für diese Tabelle rot sein, wenn eine der anderen Spalten grün ist, muss **Keine** rot sein.

Initialisieren des Formulars

Im Klassenmodul des Formulars deklarieren wir die folgenden Variablen im allgemeinen Teil, das heißt, die Variablen sind modulweit zugreifbar:

```
Dim WithEvents objWebbrowser As WebBrowser
```

```
Dim WithEvents objDocument As HTMLDocument
Public colCells As Collection
Public colCellWrapper As Collection
```

Die ersten beiden Variablen referenzieren später das Webbrowser-Steuerelement sowie das darin angezeigte Dokument. Die übrigen beiden, **colCells** und **colCellWrapper**, sind **Collection**-Objekte, welche Objekte aufnehmen, die wir auf Basis zweier weiter unten beschrieben Klassen erstellen – und zwar für jedes Element der Tabelle aus der Kopfzeile und den farbig zu markierenden Tabellenzellen.

Beim Öffnen des Formulars weisen wir der Variablen **objWebbrowser** das Steuerelement **ctlWebbrowser.Object** zu. Außerdem stellen wir die Eigenschaft **TimerInterval** auf **50** ein, was dazu führt, dass die Ereignisprozedur **Bei Zeitgeber** 50 Millisekunden nach dem Einstellen dieser Eigenschaft aufgerufen wird:

```
Private Sub Form_Open(Cancel As Integer)
    Set objWebbrowser = Me!ctlWebbrowser.Object
    Me.TimerInterval = 50
End Sub
```

Die durch das Ereignis **Bei Zeitgeber** ausgelöste Prozedur zeigt zunächst eine leere Seite im **Webbrowser**-Steuerelement an und stellt die Variable **objDocument** auf die Eigenschaft **Document** des **Webbrowser**-Steuerelements ein. Dann stellt sie **TimerInterval** auf **0**, damit das Ereignis **Bei Zeitgeber** nicht nochmals ausgelöst wird. Schließlich stellt sie das Kombinationsfeld **cboBenutzergruppeID** auf den ersten Eintrag der Datensatzherkunft ein und ruft die Prozedur **TabelleErstellen** auf. Dieser übergibt sie die aktuell im Kombinationsfeld ausgewählte Benutzergruppe als Parameter, damit die Prozedur die dazu gehörenden Berechtigungen im **Webbrowser**-Steuerelement anzeigt:

```
Private Sub Form_Timer()
    objWebbrowser.Navigate "about:blank"
    Set objDocument = objWebbrowser.Document
    Me.TimerInterval = 0
    DoEvents
    Me!cboBenutzergruppeID = 7
        Me!cboBenutzergruppeID.ItemData(0)
    TabelleErstellen Me!cboBenutzergruppeID
End Sub
```

Die Prozedur **TabelleErstellen** wird außerdem aufgerufen, wenn der Benutzer im Kombinationsfeld **cboBenutzergruppe** einen anderen Eintrag auswählt. Dazu legen wir die folgende Prozedur an, die durch das Ereignis **Nach Aktualisierung** des Kombinationsfeldes aufgerufen wird:

```
Private Sub cboBenutzergruppeID_AfterUpdate()
    TabelleErstellen Me!cboBenutzergruppeID
End Sub
```

CSS-Definition

In der nachfolgend mit der Prozedur **TabelleErstellen** zusammengestellten HTML-Tabelle verwenden wir auch einige Zeilen CSS-Code. Diese stellen wir mit der folgenden Funktion zusammen. Die ersten CSS-Anweisungen legen die Schriftart, Schriftgröße, Ausrichtung und Rahmeneigenschaften für die Tabellen fest:

```
Public Function CSS()
    Dim strCSS As String
    strCSS = strCSS & "table {"
    strCSS = strCSS & "  font-family: Calibri;"
    strCSS = strCSS & "  text-align: center;"
    strCSS = strCSS & "  width: 100%;"
    strCSS = strCSS & "  border-collapse: collapse;"
    strCSS = strCSS & "}"
    strCSS = strCSS & "td.columnHeader {"
    strCSS = strCSS & "  font-weight: bold;"
    strCSS = strCSS & "  border-bottom: 1px solid black;"
    strCSS = strCSS & "}"
    strCSS = strCSS & "td.rowHeader {"
    strCSS = strCSS & "  font-weight: bold;"
    strCSS = strCSS & "  border-right: 1px solid black;"
    strCSS = strCSS & "  text-align: left;"
    strCSS = strCSS & "}"
```

Dann folgen zwei Definitionen für CSS-Klassen, welche die Hintergrundfarben für die roten und grünen Zellen festlegen:

```
    strCSS = strCSS & "td.redcell{"
    strCSS = strCSS & "  border-color: white;"
    strCSS = strCSS & "  border-width: 1px;"
    strCSS = strCSS & "  border-style: solid;"
    strCSS = strCSS & "  background: red;"
    strCSS = strCSS & "}"
    strCSS = strCSS & "td.greencell{"
    strCSS = strCSS & "  border-color: white;"
    strCSS = strCSS & "  border-width: 1px;"
    strCSS = strCSS & "  border-style: solid;"
    strCSS = strCSS & "  background: green;"
    strCSS = strCSS & "}"
```

Dann benötigen wir noch zwei Klassen für die Breite der ersten Spalte (50%) und die der folgenden fünf Spalten (jeweils 10%):

```
    strCSS = strCSS & ".th1 {"
    strCSS = strCSS & "  width:50%;"
    strCSS = strCSS & "  text-align:left;"
    strCSS = strCSS & "}"
    strCSS = strCSS & ".th2 {"
    strCSS = strCSS & "  width:10%;"
    strCSS = strCSS & "  text-align:left;"
    strCSS = strCSS & "}"
```

Schließlich fehlt noch eine CSS-Klasse namens **scrolling**, welche die Höhe des zu scrollenden Bereiches festlegt und dass dieser gescrollt werden kann:

```
    strCSS = strCSS & ".scrolling {"
    strCSS = strCSS & "  height:300px;"
    strCSS = strCSS & "  overflow:auto;"
    strCSS = strCSS & "}"
    CSS = strCSS
End Function
```

Tabelle zusammenstellen

Nach dieser Vorbereitung kommen wir zur Prozedur **TabelleErstellen**, deren ersten Teil Sie in Listing 1 finden. Diese Prozedur deklarieren wir als öffentliche Prozedur, damit wir später von Klassenobjekten, die wir von dieser Prozedur aus erstellt haben, erneut auf diese Prozedur zugreifen können, um die Tabelle neu zu erstellen.

Die Prozedur erwartet die ID der Benutzergruppe, deren Berechtigungen an den verschiedenen Tabellen der Datenbank in der HTML-Tabelle dargestellt werden sollen. Sie deklariert eine ganze Reihe von Variablen. Die drei Variablen **rstSpaltenkoepfe**, **rstZeilen** und **rstWerte** nehmen die Recordsets auf, deren Daten wir in der HTML-Tabelle abbilden wollen. Wie wir diese genau füllen, erfahren Sie weiter unten.

Dazu benötigen wir einige Objekte, mit denen wir die Objekte der HTML-Seite definieren und hierarchisch anordnen. Wir wollen in diesem Beispiel nicht wie etwa in dem Beispiel zum Beitrag **HTML-Tabellen mit fester Kopfzeile** (**www.access-im-unternehmen.de/1188**) den HTML-Code in einer **String**-Variablen zusammensetzen, sondern diesmal das Objektmodell für HTML verwenden (DOM, Document Object Model).

Die ersten Anweisungen der Prozedur erzeugen jeweils neue **Collection**-Objekte für die beiden Variablen **colCells** und **colCellWrapper**, die wie im Kopf des Klassenmoduls deklariert haben. Dann erstellen wir die Klasse, welche den Code der Funktion **CSS** als Wert der Eigenschaft **cssText** aufnimmt. Außerdem benötigen wir ein Database-Objekt für den Zugriff auf die Objekte der aktuellen Datenbank.

Für das mit der Variablen **objDocument** referenzierten Objektvariablen für das Dokument im **Webbrowser**-Steuerelement stellen wir die Eigenschaft **body.innerHTML** auf eine leere Zeichenkette ein. Dann erstellen wir für **objDocument** mit der **createElement**-Methode ein neues Element des Typs **Table** und hängen es mit der **append-**

Child-methode an das **body**-Objekt von **objDocument** an. Als Nächstes fügen wir unterhalb des **Table**-Objekts aus **objTable** mit der **insertRow**-Methode ein neues **Row**-Element ein und referenzieren es mit **objRow**. Darin fügen wir nun ein erstes **Cell**-Objekt ein, welches der linken, oberen Zelle entspricht, die keinen Text aufnehmen soll. Dazu verwenden wir die **insertCell**-Methode von **objRow** und speichern das Ergebnis in **objCell**. **objCell** ist das erste Objekt, dem wir CSS-Klassen zuweisen. Das erledigen wir über die Eigenschaft **className**, der

```
Public Sub TabelleErstellen(lngBenutzergruppeID As Long)
    Dim db As DAO.Database
    Dim rstSpaltenkoepfe As DAO.Recordset
    Dim rstZeilen As DAO.Recordset
    Dim rstWerte As DAO.Recordset
    Dim objTable As MSHTML.HTMLTable
    Dim objRow As MSHTML.HTMLTableRow
    Dim objCell As MSHTML.HTMLTableCell
    Dim objCSS As Object
    Dim objCellHeaderWrapper As clsCellHeaderWrapper
    Dim objInnerTable As MSHTML.HTMLTable
    Dim objDiv As MSHTML.HTMLDivElement
    Set colCells = New Collection
    Set colCellWrapper = New Collection
    Set objCSS = objDocument.createStyleSheet("")
    objCSS.cssText = CSS
    Set db = CurrentDb
    objDocument.body.innerHTML = ""
    Set objTable = objDocument.createElement("Table")
    objDocument.body.appendChild objTable
    Set objRow = objTable.insertRow
    Set objCell = objRow.insertCell
    objCell.className = "columnHeader rowHeader th1"
    Set rstSpaltenkoepfe = db.OpenRecordset("SELECT BerechtigungID, Berechtigung FROM tblBerechtigungen", dbOpenDynaset)
    Do While Not rstSpaltenkoepfe.EOF
        Set objCell = objRow.insertCell
        objCell.className = "columnHeader th2"
        objCell.innerText = rstSpaltenkoepfe!Berechtigung
        Set objCellHeaderWrapper = New clsCellHeaderWrapper
        With objCellHeaderWrapper
            Set .cell = objCell
            .BerechtigungID = rstSpaltenkoepfe!BerechtigungID
            .BenutzergruppeID = Me!cboBenutzergruppeID
            Set .Form = Me
        End With
        colCellWrapper.Add objCellHeaderWrapper
        rstSpaltenkoepfe.MoveNext
    Loop
```

Listing 1: Prozedur **TabelleErstellen**, Teil 1

wir den Text **columnHeader rowHeader th1** zuweisen. Dadurch nimmt das Element die in den drei CSS-Klassen **td.columnHeader**, **td.rowHeader** und **th1** angegebenen Attribute an.

Danach füllen wir die erste **Recordset**-Variable namens **rstSpaltenkoepfe**, und zwar mit den Feldern **BerechtigungID** und **Berechtigung** der Datensätze der Tabelle **tblBerechtigungen**. Diese durchlaufen wir anschließend in einer **Do While**-Schleife über alle Einträge des Recordsets. In der **Do While**-Schleife erstellen wir jeweils ein neues **Cell**-Objekt und referenzieren diese wieder mit **objCell**. Wir weisen über die Eigenschaft **className** die beiden CSS-Klassen **columnHeader** und **th2** zu. Außerdem wollen wir den in den Zellen anzuzeigenden Text definieren. Diesen entnehmen wir dem Feld **Berechtigung** des aktuellen Datensatzes des Recordsets **rstSpaltenkoepfe**.

Ereignisse für Spaltenköpfe

Für diese Spaltenköpfe, jeweils einer für jeden Berechtigungstyp, wollen wir nun jeweils eine Ereignisprozedur definieren, die beim Anklicken des Spaltenkopfes ausgelöst wird. Das können wir nicht so einfach wie bei einer Schaltfläche in einem Access-Formular erledigen, denn es handelt sich ja bei den Tabellenzellen nicht um Steuerelemente, die wir dem Formularentwurf hinzufügen und deren Ereigniseigenschaften wir dann definieren und mit Ereignisprozeduren hinterlegen können. Es ist etwas komplizierter. Wir benötigen für jedes mit Ereignissen zu versehende Element eine eigene Instanz einer Klasse, die dann wiederum auf das darin deklarierte Element verweist und die gewünschten Ereignisse des Elements implementiert.

Wir schauen uns die Sache erst einmal der Seite unserer Prozedur an, mit der wir die HTML-Tabelle erstellen. Dort legen wir für jedes **Cell**-Element für eine Berechtigung ein neues Objekt auf Basis der Klasse **clsCellHeaderWrapper** an und speichern es in der Variablen **objCellHeaderWrapper**. Diesem weisen wir dann über die Eigenschaft **cell** die Objektvariable **objCell** zu. Die Eigenschaft **BerechtigungID** erhält den Wert des Feldes **BerechtigungID** des aktuellen Datensatzes des Recordsets **rstSpaltenkoepfe**. Die Eigenschaft **BenutzergruppeID** erhält den Wert des Kombinationsfeldes **cboBenutzergruppeID** des Formulars. Außerdem stellen wir über die Eigenschaft **Form** noch einen Verweis auf das Formular ein (**Me**).

Da wir **objCellHeaderWrapper** schon im nächsten Schleifendurchlauf mit den Eigenschaften des nächsten **Cell**-Objekts füllen, speichern wir die aktuelle Instanz noch schnell mit der **Add**-Methode in der Collection **colCellWrapper**. Danach bewegen wir den Datensatzzeiger zu nächsten Datensatz und beginnen die Schleife von vorn, bis alle fünf Berechtigungen durchlaufen sind. Damit steht schon einmal die erste Zeile der Tabelle.

Tabellenkörper hinzufügen

Den zweiten Teil der Prozedur **TabelleErstellen** finden Sie in Listing 2. Hier füllen wir das Recordset **rstZeilen** mit allen Datensätzen der Tabelle **tblTabellen**. Bevor wir diese durchlaufen, legen wir noch einige weitere Elemente an. Dem Objekt **objTable** fügen wir mit **insertRow** eine neue Zeile hinzu, die wir in **objRow** speichern. Dieser fügen wir diesmal nur eine einzige Zelle hinzu, die sich aber durch den Wert **6** für die Eigenschaft **colSpan** über die kompletten sechs Spalten der ersten Zeile der Tabelle erstreckt.

Wir erstellen in **objDiv** ein neues **Div**-Objekt für **objDocument** und fügen es mit der **appendChild**-Methode in das Objekt **objCell** ein. Diesem **Div**-Element weisen wir die Klasse **scrolling** zu. Das **Div**-Element umfasst also den scrollbaren Bereich der Tabelle – wir wollen ja, dass nur der untere Teil der Tabelle gescrollt wird und die Spaltenköpfe am oberen Rand der HTML-Tabelle fixiert werden.

Danach erstellen wir ein weiteres **Table**-Element, das wir dem **Div**-Element unterordnen. Dieses **Table**-Element füllen wir dann in einer **Do While**-Schleife über alle Datensätze des Recordsets **rstZeilen** mit der entsprechenden Anzahl Zeilen beziehungsweise **Row**-Elementen.

In der **Do While**-Schleife kommen wir nun auf das Recordset **rstSpaltenkoepfe** zurück, dessen Datensatzzeiger wir mit der **MoveFirst**-Methode wieder an die erste Position verschieben. Bevor wir dieses Recordset nutzen, folgen noch einige neue Elemente: Zunächst ein erstes **Cell**-Element, das wir dem aktuellen **objRow**-Element, also der neuen Zeile der HTML-Tabelle, unterordnen. Diesem ersten Element weisen wir als Text den Namen der Tabelle aus dem Feld **Tabelle** des Recordsets **rstZeilen** zu. Außerdem stellen wir die CSS-Klassen **rowHeader** und **th1** für die Eigenschaft **className** ein.

Damit haben wir die erste Zelle der aktuellen Zeile der HTML-Tabelle mit dem Namen der Tabelle gefüllt, deren Berechtigungen nun in Form von roten oder grünen Kästchen folgen. Dazu füllen wir das Recordset **rstWerte** mit den Daten der Tabelle **tblBerechtigungszuordnungen**, deren Benutzergruppe der im Kombinationsfeld ausge-

```
    Set rstZeilen = db.OpenRecordset("SELECT * FROM tblTabellen", dbOpenDynaset)
    Set objRow = objTable.insertRow
    Set objCell = objRow.insertCell
    objCell.colSpan = 6
    Set objDiv = objDocument.createElement("Div")
    objCell.appendChild objDiv
    objDiv.className = "scrolling"
    Set objInnerTable = objDocument.createElement("Table")
    objDiv.appendChild objInnerTable
    Do While Not rstZeilen.EOF
        Set objRow = objInnerTable.insertRow
        rstSpaltenkoepfe.MoveFirst
        Set objCell = objRow.insertCell
        objCell.innerText = rstZeilen!Tabelle
        objCell.className = "rowHeader th1"
        Set rstWerte = db.OpenRecordset("SELECT BerechtigungszuordnungID, BenutzergruppeID, TabelleID, " _
            "BerechtigungID FROM tblBerechtigungszuordnungen WHERE BenutzergruppeID = " & lngBenutzergruppeID _
            & " AND TabelleID = " & rstZeilen!TabelleID, dbOpenDynaset)
        Do While Not rstSpaltenkoepfe.EOF
            rstWerte.FindFirst "BerechtigungID = " & rstSpaltenkoepfe!BerechtigungID
            If Not rstWerte.NoMatch Then
                colCells.Add ZelleHinzufuegen(objRow, "greencell", rstWerte!BerechtigungszuordnungID, _
                    rstSpaltenkoepfe!BerechtigungID, rstZeilen!TabelleID, lngBenutzergruppeID)
            Else
                colCells.Add ZelleHinzufuegen(objRow, "redcell", 0, rstSpaltenkoepfe!BerechtigungID, _
                    rstZeilen!TabelleID, lngBenutzergruppeID)
            End If
            rstSpaltenkoepfe.MoveNext
        Loop
        rstZeilen.MoveNext
    Loop
    Inzwischenablage Replace(Replace(objDocument.documentElement.innerHTML, "></", ">" & vbCrLf & "</"), "><", ">" _
        & vbCrLf & "<")
End Sub
```

Listing 2: Prozedur **TabelleErstellen**, Teil 2

wählten Benutzergruppe und deren Tabelle der Tabelle für die aktuelle Zeile entspricht.

Die Datensätze des Recordsets **rstSpaltenkoepfe** durchlaufen wir nun in einer untergeordneten **Do While**-Schleife. Nun suchen wir im Recordset **rstWerte** nach dem Datensatz, dessen Feld **BerechtigungID** dem Wert des Feldes **BerechtigungID** des aktuellen Datensatzes des Recordsets **Spaltenkoepfe** entspricht.

Ist ein solcher Datensatz vorhanden, wurde diese Berechtigung für die Kombination aus Benutzergruppe und Tabelle in der Tabelle **tblBerechtigungszuordnungen** zugeordnet. Dies prüft die folgende **If...Then**-Bedingung, welche den Wert von **rstWerte.NoMatch** untersucht.

Ist die Berechtigung für die Kombination aus Tabelle und Benutzergruppe gesetzt, fügen wir der Collection **colCells** ein neues Objekt hinzu, das wir mit der Funktion **ZelleHinzufuegen** ermitteln. Dieser übergeben wir als Parameter das **Row**-Objekt aus **objRow**, den Namen der zuzuweisenden Klasse (im Falle einer Berechtigungszuweisung **greencell**), die Werte der Tabellen **tblBerechtigungszuordnungen**, **tblBerechtigungen** und **tblTabellen** aus den jeweiligen Recordsets sowie die ID der Benutzergruppe.

Wenn kein solcher Datensatz vorhanden ist, wird ebenfalls die Funktion **ZelleHinzufuegen** aufgerufen. Diesmal wird allerdings der Wert **0** für den Parameter mit der Berechtigungszuordnung übergeben sowie der Klassenname **redcell**.

Auf diese Weise durchläuft die Prozedur alle Spalten für die aktuelle Tabelle und dann die Spalten in den Zeilen für die übrigen Tabellen. Die letzte Anweisung fügt zu Kontrollzwecken den so erstellen HTML-Code in die Zwischenablage ein – Sie können diesen dann etwa in einen Texteditor kopieren, um den Code zu kontrollieren.

Hinzufügen einer Zelle mit der Funktion ZelleHinzufuegen

Die Anweisungen der Funktion **ZelleHinzufuegen** haben wir in eine eigene Funktion ausgegliedert, weil wir in der zuletzt beschriebenen **If...Then**-Bedingung in der Prozedur **TabelleErstellen** sonst annähernd den gleichen Quellcode zwei Mal abgebildet hätten.

```
Private Function ZelleHinzufuegen(objRow As MSHTML.HTMLTableRow, strClassname As String, lngBerechtigungszuordnungID _
        As Long, lngBerechtigungID As Long, lngTabelleID As Long, lngBenutzergruppeID As Long) As clsCellWrapper
    Dim objCell As MSHTML.HTMLTableCell
    Dim objCellWrapper As clsCellWrapper
    Set objCell = objRow.insertCell
    objCell.ID = lngBerechtigungszuordnungID
    objCell.className = strClassname
    Set objCellWrapper = New clsCellWrapper
    With objCellWrapper
        .BerechtigungzuordnungID = lngBerechtigungszuordnungID
        .BerechtigungID = lngBerechtigungID
        .TabelleID = lngTabelleID
        .BenutzergruppeID = lngBenutzergruppeID
        Set .cell = objCell
    End With
    Set ZelleHinzufuegen = objCellWrapper
End Function
```

Listing 3: Die Funktion **ZelleHinzufuegen**

Benutzer und Berechtigungen ermitteln

In den Beiträgen »Benutzerverwaltung mit verschlüsselten Kennwörtern« und »Berechtigungen per HTML verwalten« haben wir die Voraussetzungen für eine Benutzerverwaltung geschaffen. Im vorliegenden Beitrag bauen wir darauf auf und legen ein Anmeldeformular an, mit dem der Benutzer sich an der Anwendung anmelden kann. Außerdem stellen wir die Funktion vor, mit der wir aus den Tabellen zur Verwaltung der Berechtigungen aufgrund des Benutzernamens die Berechtigung an einer bestimmten Tabelle ermitteln.

In den Beiträgen **Benutzerverwaltung mit verschlüsselten Kennwörtern** (**www.access-im-unternehmen.de/1190**) und **Berechtigungen per HTML verwalten** (**www.access-im-unternehmen.de/1191**) haben wir das Datenmodell und die Formulare zur Verwaltung unseres Berechtigungssystems vorbereitet.

Daraus ist das Datenmodell aus Bild 1 entstanden, auf dem dieser Beitrag aufbaut. Für die Anmeldung an der Anwendung benötigen wir lediglich die Tabelle **tblBenutzer**. Wir erstellen dazu ein Formular, mit dem der Benutzer seinen Benutzernamen sowie das Kennwort eingeben kann und das dann prüft, ob die Daten mit den gespeicherten Daten übereinstimmen. Falls nicht, kann der Benutzer sich bei Bedarf ein neues Kennwort zusenden lassen beziehungsweise dieses ändern. Im zweiten Teil verwenden Sie die Daten der aktuellen Anmeldung dann dazu, um die Berechtigung für diesen Benutzer an einer bestimmten Tabelle zu ermitteln. Dazu definieren wir eine passende VBA-Funktion. Hier kommen dann alle Tabellen der Berechtigungsverwaltung zum Einsatz.

Anmeldeformular programmieren

Im ersten Teil programmieren wir das Anmeldeformular. Das sollte normalerweise schnell zu erledigen sein, aber in der Tat kann man ein solches Formular beliebig kompliziert gestalten. In unserem Fall wollen wir damit starten, zwei Textfelder zur Eingabe von Benutzername und Kennwort bereitzustellen sowie eine Schaltfläche zum Bestätigen der Eingabe und eine zum Beenden der Anmel-

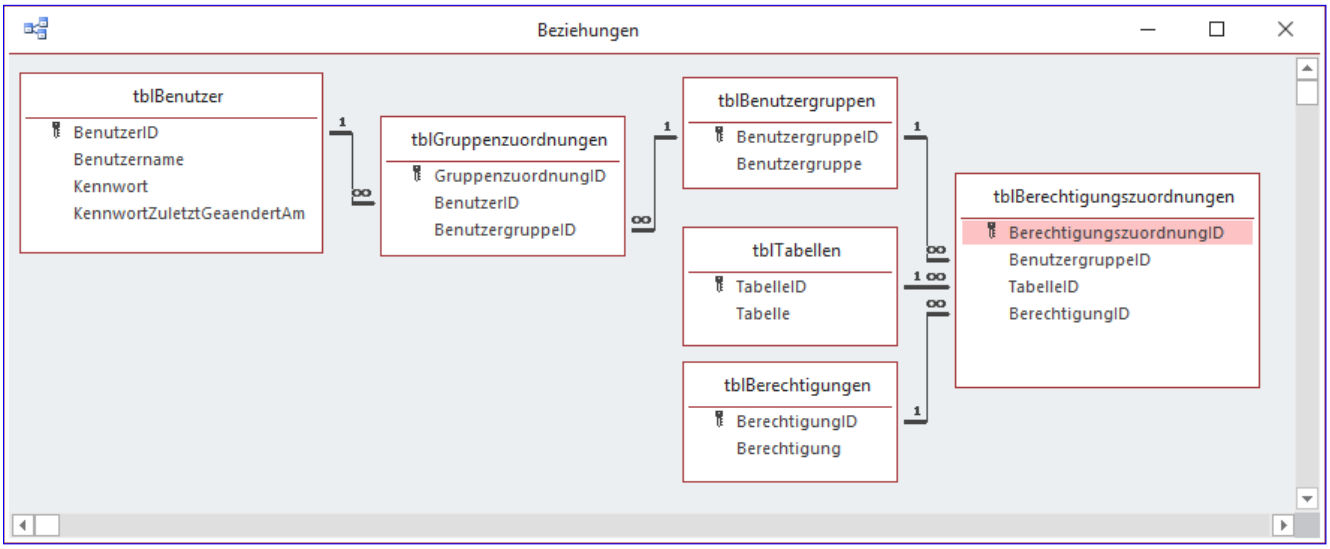


Bild 1: Datenmodell der Berechtigungsverwaltung

dung, die dann auch die Anwendung schließt. Der Entwurf sieht also im ersten Anlauf etwa wie in Bild 2 aus.

Das Formular enthält nun neben dem Bezeichnungsfeld mit dem Text **Anmeldung** die folgenden Steuerelemente:

- **txtBenutzername**: Textfeld zur Eingabe des Benutzernamens
- **txtKennwort**: Textfeld zur Eingabe des Kennworts mit einem Eingabeformat zum Maskieren der Eingabe
- **cmdAnmelden**: Schaltfläche, die eine Prozedur zum Prüfen der Kombination aus Benutzername und Kennwort aufruft
- **cmdAbbrechen**: Schaltfläche, die das Formular und die Anwendung schließt

Für das Textfeld **txtKennwort** legen wir das Eingabeformat **Kennwort** fest. Dazu klicken Sie rechts in der Eigenschaft **Eingabeformat** für dieses Textfeld auf die Schaltfläche mit den drei Punkten (**...**). Es erscheint der Dialog **Eingabeformat-Assistent**, wo Sie das Eingabeformat Kennwort auswählen (siehe Bild 3).

Anschließend wird die Eingabe des Textes im Textfeld **txtKennwort** durch Sternchen maskiert (siehe Bild 4). An dieser Stelle legen wir für die Eigenschaften **Datensatzmarkierer**, **Navigationsschaltflächen**, **Bildlaufleisten** und **Trennlinien** den Wert **Nein** und für **Automatisch zentrieren** den Wert **Ja** fest, damit die entsprechenden Elemente nicht mehr erscheinen beziehungsweise das Anmeldeformular in der Mitte des Access-Fensters angezeigt wird.

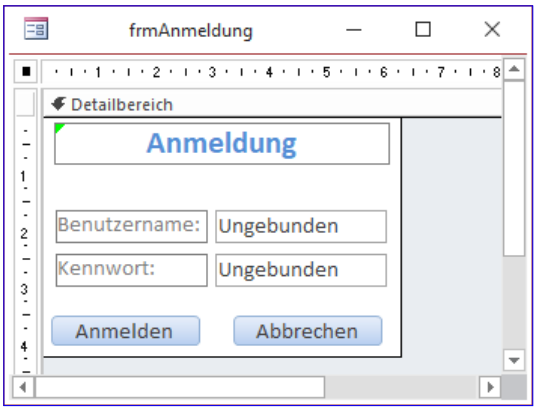


Bild 2: Entwurf des Anmeldeformulars

Prüfen von Benutzername und Kennwort

Wenn der Benutzer auf die Schaltfläche **cmdAnmelden** klickt, soll eine Prozedur prüfen, ob Benutzername und Kennwort zusammengehören. Diese gestalten wir erst einmal so einfach wie möglich. Die Ereignisprozedur soll erst einmal prüfen, ob Benutzername und Kennwort zusammenpassen und in diesem

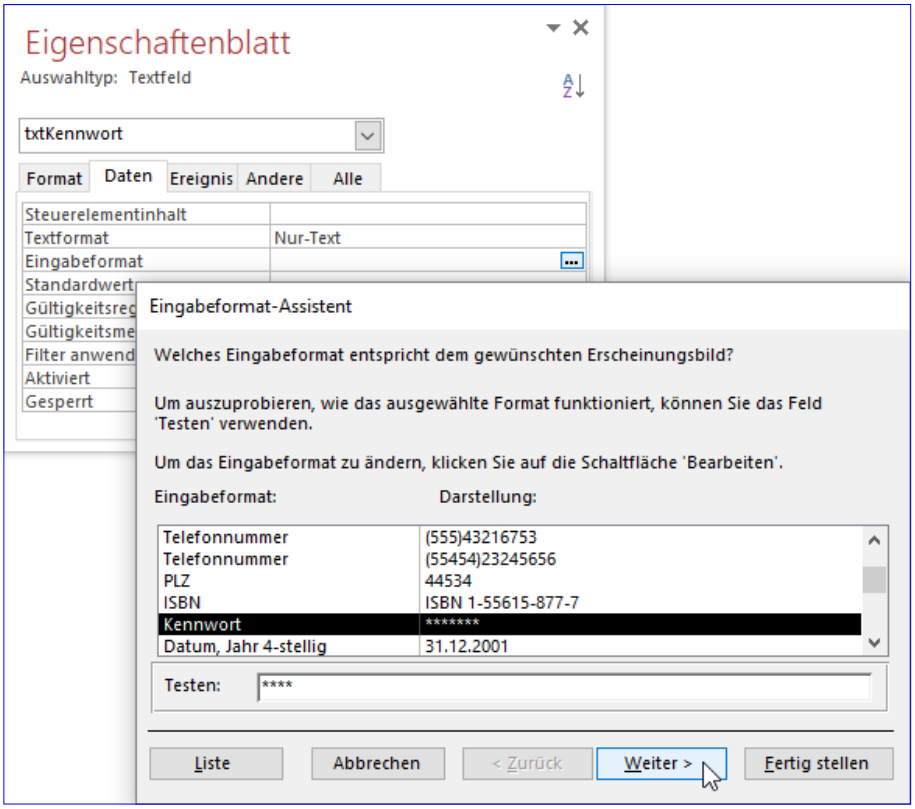


Bild 3: Einstellen des Eingabeformats

Fall ein Meldungsfenster mit dem Ergebnis anzeigen. Das Gleiche soll geschehen, wenn die Kombination aus Benutzername und Kennwort nicht in der Tabelle **tblBenutzer** gespeichert ist:

```
Private Sub cmdAnmelden_Click()
    If AnmeldungPruefen(Me!txtBenutzername, _
            Me!txtKennwort) = True Then
        MsgBox "Anmeldung erfolgreich."
    Else
        MsgBox "Anmeldung nicht erfolgreich."
    End If
End Sub
```

Die Funktion **AnmeldungPruefen** finden Sie in Listing 1. Die Funktion berücksichtigt, dass wir die Kennwörter nicht im Klartext im Feld **Kennwort** der Tabelle **tblBenutzer** speichern, sondern in verschlüsselter Form. Sie nimmt die zu untersuchende Kombination von Benutzername und Kennwort in Form der beiden Parameter **strBenutzername** und **strKennwort** entgegen. Dann erstellt sie eine neue Instanz der Klasse **clsCrypt**, die unter anderem eine Methode zum Verschlüsseln mit dem **SHA1**-Algorithmus bereitstellt. Diese Methode verschlüsselt dann das Kennwort aus **strKennwort**. Das Ergebnis landet in der Variablen **strVerschluesselt**.

Bild 4: Verbergen der Kennworteingabe

Damit können wir dann über den Aufruf einer **DLookup**-Funktion prüfen, ob die Kombination aus Benutzername und **SHA1**-verschlüsseltem Kennwort in der Tabelle **tblBenutzer** vorkommt. Ist dies der Fall, liefert die Funktion **AnmeldungPruefen** den Wert **True** zurück, anderenfalls den Wert **False**.

Nach der Anmeldung

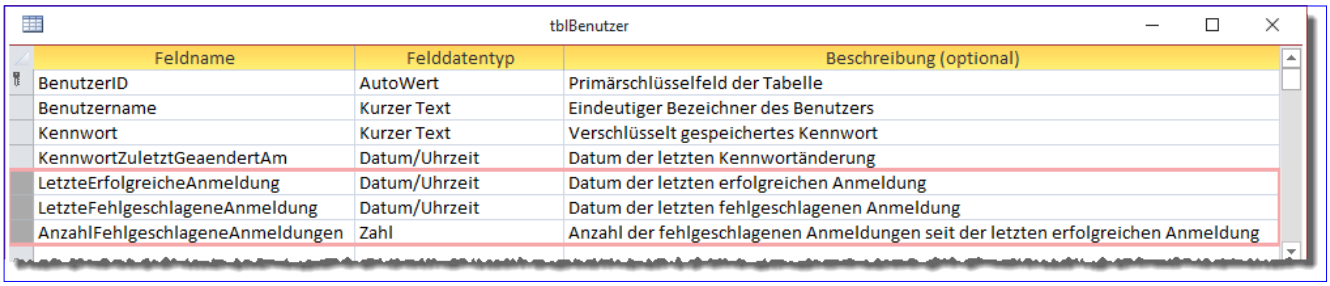
Was soll nach der Anmeldung geschehen? Als Erstes sollte der Anmeldedialog geschlossen werden. Dann speichern wir den Benutzernamen des angemeldeten Benutzers in irgendeiner Form. Aber welche ist die geeignete Form – eine öffentliche Variable? Nein, denn wir wollen später im Beitrag **Zugriffsrechte mit Datenmakros** (**www.access-im-unternehmen.de/1193**) auch auf diese Information zugreifen.

Das ist dort am einfachsten möglich über eine in der **TempVars**-Auflistung gespeicherte Variable.

Also ersetzen wir die **MsgBox**-Anweisung in der Prozedur **cmdAnmelden_Click** durch die folgenden beiden Zeilen:

```
Public Function AnmeldungPruefen(strBenutzername As String, strKennwort As String)
    Dim objCrypt As clsCrypt
    Dim strVerschluesselt As String
    Set objCrypt = New clsCrypt
    strVerschluesselt = objCrypt.GetSHA1(strKennwort)
    If Not IsNull(DLookup("BenutzerID", "tblBenutzer", "Benutzername = '" & strBenutzername & "' AND Kennwort = '" _
            & strVerschluesselt & "'")) Then
        AnmeldungPruefen = True
    End If
End Function
```

Listing 1: Funktion zum Prüfen einer Anmeldung



tblBenutzer

Feldname	Felddatentyp	Beschreibung (optional)
BenutzerID	AutoWert	Primärschlüsselfeld der Tabelle
Benutzername	Kurzer Text	Eindeutiger Bezeichner des Benutzers
Kennwort	Kurzer Text	Verschlüsselt gespeichertes Kennwort
KennwortZuletztGeaendertAm	Datum/Uhrzeit	Datum der letzten Kennwortänderung
LetzteErfolgreicheAnmeldung	Datum/Uhrzeit	Datum der letzten erfolgreichen Anmeldung
LetzteFehlgeschlageneAnmeldung	Datum/Uhrzeit	Datum der letzten fehlgeschlagenen Anmeldung
AnzahlFehlgeschlageneAnmeldungen	Zahl	Anzahl der fehlgeschlagenen Anmeldungen seit der letzten erfolgreichen Anmeldung

Bild 5: Neue Felder in der Tabelle **tblBenutzer**

```
If AnmeldungPruefen(Me!txtBenutzername, Me!txtKennwort) =
                                                    True Then
    TempVars.Add "Benutzername", Me!txtBenutzername.Value
    DoCmd.Close acForm, Me.Name
Else
```

Danach können wir wie folgt auf den Wert der temporären Variablen zugreifen:

```
Debug.Print TempVars!Benutzername
Peter Gross
```

Wenn die Anmeldung fehlschlägt, soll die temporäre Variable übrigens wieder geleert werden:

```
TempVars.Remove "Benutzername"
```

Bei nicht erfolgreicher Anmeldung

Wenn der Benutzer nicht die richtigen Daten eingibt, erscheint aktuell jeweils die Meldung mit dem Text **Anmeldung nicht erfolgreich**. Hier könnten wir nun beispielsweise prüfen, ob ein Benutzer mehrfach versucht, sich mit dem gleichen Benutzernamen anzumelden und dabei immer wieder das falsche Kennwort eingibt.

Ist das der Fall, könnte man verschiedene Aktionen durchführen – beispielsweise die Zeit zwischen den möglichen Anmeldeversuchen schrittweise vergrößern, um zu verhindern, dass jemand automatisiert versucht, das Kennwort für den Benutzernamen zu ermitteln.

Hier wird es schon interessant, denn diese Informationen müssen wir ebenfalls speichern, um beim nächsten Anmeldeversuch wieder darauf zugreifen zu können. Zunächst legen wir fest, wieviel Zeit zwischen zwei Anmeldeversuchen verstreichen soll.

Wir beginnen ganz entspannt und legen fest, dass zwischen den ersten und dem zweiten Versuch eine Sekunde verstreichen soll. Das merkt der Benutzer gar nicht, da die Eingabe des Kennworts ja mindestens eine Sekunde dauern sollte.

Von nun an verdoppeln wir die Zeit mit jedem Versuch. Nach dem zweiten Versuch muss er also zwei Sekunden warten, nach dem dritten Versuch vier Sekunden und so weiter. So wächst die Wartezeit zwischen zwei Anmeldeversuchen schnell an.

Welche Informationen müssen wir dazu speichern? Wir benötigen den Benutzernamen, denn die Anmeldungen sollen nur für Versuche mit dem gleichen Benutzernamen gezählt werden. Dann benötigen wir die Anzahl der Versuche seit der letzten erfolgreichen Anmeldung und den Zeitpunkt des letzten erfolglosen Versuchs.

Diese Informationen können wir am besten direkt in der Benutzertabelle speichern, da diese ja ohnehin bereits für jeden Benutzer einen Datensatz enthält. Also fügen wir die folgenden Felder wie in Bild 5 hinzu:

- **LetzteErfolgreicheAnmeldung**: Datum und Uhrzeit des letzten erfolgreichen Anmeldeversuchs
- **LetzteFehlgeschlageneAnmeldung**: Datum und Uhrzeit des letzten fehlgeschlagenen Anmeldeversuchs

- **AnzahlFehlgeschlageneAnmeldungen**: Anzahl der fehlgeschlagenen Anmeldungen seit der letzten erfolgreichen Anmeldung

Integration der Wartezeit in den Anmeldedialog

Die Daten dieser drei Felder ändern wir bei jedem fehlgeschlagenen und auch bei jedem erfolgreichen Anmeldeversuch. Bei einem erfolgreichen Anmeldeversuch tragen wir Datum und Zeit dieses Versuchs in das Feld **LetzteErfolgreicheAnmeldung** ein und stellen den Wert des Feldes **AnzahlFehlgeschlageneAnmeldungen** auf **0** ein.

Bei einem gescheiterten Anmeldeversuch stellen wir das Feld **LetzteFehlgeschlageneAnmeldung** auf das aktuelle Datum plus Uhrzeit ein und erhöhen den Wert des Feldes **AnzahlFehlgeschlageneAnmeldungen** um **1**.

Auf diese Weise können wir allerdings nur Anmeldeversuche behandeln, für die ein gültiger Benutzername angegeben wurde. Wenn kein Benutzername angegeben wurde, können wir auch keine Informationen über die letzte Anmeldung des Benutzers in der Tabelle **tblBenutzer** speichern. Wie gehen wir dann vor? Ignorieren wir Anmeldeversuche ohne gültigen Benutzernamen, weil der Benutzer – auch wenn er böse Absichten hat – ohne Kenntnis eines gültigen Benutzernamens noch weiter von einer erfolgreichen Anmeldung entfernt ist als mit Benutzername? Oder sollten wir auch hier eine zeitliche Barriere

```
Public Function AnmeldungPruefen(strBenutzername As String, strKennwort As String)
    Dim objCrypt As clsCrypt
    Dim strVerschluesselt As String
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set objCrypt = New clsCrypt
    strVerschluesselt = objCrypt.GetSHA1(strKennwort)
    Set rst = db.OpenRecordset("SELECT * FROM tblBenutzer WHERE Benutzername = '" & strBenutzername & "'", dbOpenDynaset)
    If Not rst.EOF Then
        If strVerschluesselt = rst!Kennwort Then
            rst.Edit
            rst!LetzteErfolgreicheAnmeldung = Now
            rst!AnzahlFehlgeschlageneAnmeldungen = 0
            rst.Update
            AnmeldungPruefen = True
        Else
            rst.Edit
            rst!LetzteFehlgeschlageneAnmeldung = Now
            rst!AnzahlFehlgeschlageneAnmeldungen = Nz(rst!AnzahlFehlgeschlageneAnmeldungen, 0) + 1
            rst.Update
            AnmeldungPruefen = False
        End If
    Else
        AnmeldungPruefen = False
    End If
End Function
```

Listing 2: Funktion zum Prüfen einer Anmeldung, erweiterte Fassung

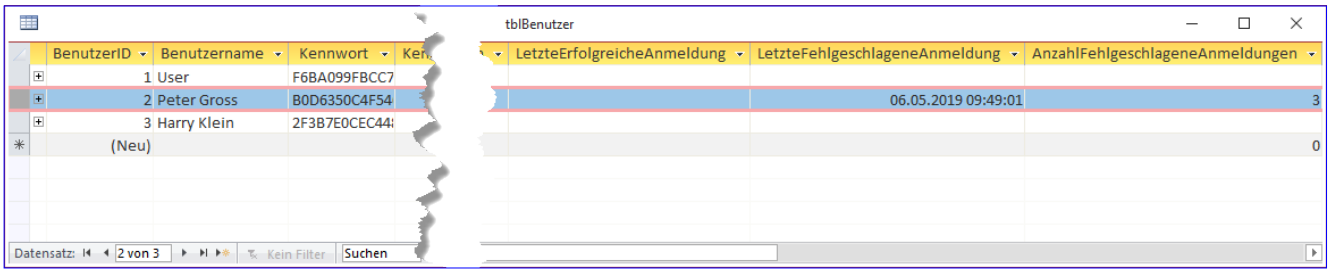


Bild 6: Speichern fehlgeschlagener Anmeldungen

einführen? Wir wollen es einfach halten und ignorieren Anmeldungen ohne gültigen Benutzernamen einfach.

Die Funktion **AnmeldungPruefen** ändern wir wie in Listing 2. Wir haben statt der **DLookup**-Funktion direkt den Zugriff auf den Datensatz der Tabelle **tblBenutzer** mit dem passenden Benutzernamen als Recordset implementiert. Das heißt, dass wir nach dem Ermitteln des verschlüsselten Kennworts ein Recordset öffnen, das den Datensatz der Tabelle **tblBenutzer** mit dem Benutzernamen aus dem Parameter **strBenutzername** enthält. Ist dieses Recordset leer, hat der Benutzer einen ungültigen Benutzernamen eingegeben und die Funktion soll den Wert **False** zurückliefern. Ist das Recordset nicht leer, hat der Benutzer zumindest einen gültigen Benutzernamen angegeben.

Dann prüfen wir in einer **If...Then**-Bedingung, ob das verschlüsselte Kennwort aus **strVerschluesselt** mit der im Feld **Kennwort** gespeicherten, verschlüsselten Version des Kennworts für diesen Benutzer übereinstimmt. In diesem Fall versetzen wir das Recordset mit der **Edit**-Methode in den Verarbeitungsmodus und stellen zwei Felder auf neue Werte ein. Das Feld **LetzteErfolgreicheAnmeldung** erhält mit der Funktion **Now** das aktuelle Datum plus Uhrzeit, das Feld **AnzahlFehlgeschlageneAnmeldungen** stellen wir auf den Wert **0** ein. Danach speichern wir den aktualisierten Datensatz mit der **Update**-Methode und geben den Wert **True** als Funktionswert zurück.

Stimmt das verschlüsselte Kennwort nicht mit dem Wert aus dem Feld **Kennwort** überein, bearbeiten wir den aktuellen Datensatz ebenfalls. Hier legen wir den Wert des Feldes **LetzteFehlgeschlageneAnmeldung** auf das Ergebnis der Funktion **Now** fest und erhöhen den Wert des Feldes **AnzahlFehlgeschlageneAnmeldungen** auf den vorherigen Wert plus **1**. Falls das Feld vorher noch leer war, erhält das Feld den Wert **1**. In diesem Fall geben wir den Wert **False** als Funktionsergebnis zurück.

Der entsprechende Datensatz sieht nach drei fehlgeschlagenen Anmeldungen und ohne erfolgreiche Anmeldung beispielsweise wie in Bild 6 aus.

Zeitverzögerung beim Anmelden

Nun müssen wir noch dafür sorgen, dass das Formular **frmAnmeldung** beim Versuch, sich unter einem Benutzernamen anzumelden, für den es seit der letzten erfolgreichen Anmeldung einen oder mehrere fehlgeschlagene Anmeldungen gab, eine zeitliche Sperre einrichtet und den Benutzer auch darüber informiert.

Was wir dafür zweifelsohne benötigen, ist die Ereignisprozedur **Bei Zeitgeber** sowie ein entsprechendes Intervall für die Eigenschaft **Zeitgeberintervall**, das angibt, nach welchem Zeitraum das Ereignis **Bei Zeitgeber** das nächste Mal ausgelöst werden soll. Die Eigenschaft **Zeitgeberintervall** statten wir mit dem Zeitraum über eine Sekunde aus, was dem Wert **1.000** (in Millisekunden) entspricht.

Wie genau soll die Anzeige der Zeitverzögerung nach mehreren fehlgeschlagenen Anmeldeversuchen mit gültigem Benutzernamen, aber mit falschem Kennwort erfolgen? Wir müssen als Erstes prüfen, ob das Textfeld **txtBenutzername** überhaupt einen gültigen Benutzernamen anzeigt.

Zugriffsrechte mit Datenmakros

Es gibt verschiedene Möglichkeiten, auf Basis des aktuell angemeldeten Benutzers sicherzustellen, dass dieser nur die für ihn vorgesehenen Aktionen mit Daten durchführen darf – beispielsweise durch eine Prüfung in den Ereignisprozeduren, die durch Ereignisse wie »Vor Aktualisierung« ausgelöst werden. Noch praktischer wäre es allerdings, wenn diese Prüfung nicht in jedem Formular programmiert werden müsste, sondern anwendungsweit verfügbar wäre – also auch dann, wenn der Benutzer es schafft, ohne Formulare auf die Tabellen mit den Daten zuzugreifen. Das gelingt mit Datenmakros. Wie genau, zeigt der vorliegende Beitrag.

Grundlage für unsere Beispielanwendung

Wenn wir Datenmakros für Tabellen programmieren wollen, die dafür sorgen, dass Benutzer abhängig von ihrer Benutzergruppe und den für diese vergebenen Berechtigungen auf den verschiedenen Tabellen Daten an den Tabellen ändern können, benötigen wir einige Tabellen, in denen wir die Berechtigungen speichern. In unserem Fall haben wir in den Beiträgen **Benutzerverwaltung mit verschlüsselten Kennwörtern** (**www.access-im-unternehmen.de/1190**) und **Berechtigungen per HTML verwalten** (**www.access-im-unternehmen.de/1191**) bereits einiges an Vorarbeit geleistet. Dort haben wir nämlich sowohl die Tabellen definiert, mit denen wir die Benutzer, Benutzergruppen, Tabellen und Berechtigungen sowie die Zuordnung der Berechtigungen zu den Kombinationen aus Benutzergruppen und Tabellen verwalten.

Ermitteln des aktuellen Benutzers

Um die Berechtigungen für den Zugriff auf eine Tabelle zu ermitteln, benötigen wir die Benutzerdaten. Darüber ermitteln wir die Benutzergruppe und schließlich die Berechtigung an der angegebenen Tabelle. Die Benutzerdaten haben wir im Beitrag **Benutzer und Berechtigungen ermitteln** (**www.access-im-unternehmen.de/1192**) nach der Anmeldung in einer temporären Variablen gespeichert. Das hilft in einem Datenmakro leider nicht weiter, denn von dort aus können wir nicht auf temporäre Variablen zugreifen. Also müssen wir die Prozedur im Formular **frmAnmeldungen**, welche die Benutzerdaten nach der Prüfung der Kombination aus Benutzername und Kennwort in temporären Variablen gespeichert hat, zunächst so ändern, dass die **BenutzerID** und der Benutzername in einer Tabelle landen – die in diesem Fall **tblOptionen** heißen soll. Die Prozedur **cmdAnmelden_Click** ändern wir dazu wie in Listing 1. Die Prozedur trägt zunächst den Benutzernamen aus dem Textfeld **txtBenutzername** in die Variable **strBenutzername** ein. Dann prüft sie mit der Funktion **AnmeldungPruefen**, ob die Anmeldedaten korrekt sind. Ist dies der Fall, ermittelt sie per **DLookup** den Wert des Feldes **BenutzerID** für den angegebenen Benutzernamen und speichert diesen in der Variablen **lngBenutzerID**.

Damit versucht sie, die beiden Felder **Benutzername** und **BenutzerID** in der Tabelle **tblOptionen** durch den Aufruf einer **UPDATE**-Anweisung auf die neuen Werte zu aktualisieren. Ob das gelungen ist, zeigt die Eigenschaft **RecordsAffected** im Anschluss. Liefert sie den Wert **0**, wurde kein Datensatz geändert, was nur bedeuten kann, dass noch kein Datensatz vorhanden ist. In diesem Fall ruft die Prozedur eine **INSERT INTO**-Anweisung auf und fügt einen neuen Datensatz mit den gewünschten Werten zur Tabelle **tblOptionen** hinzu.

Sollte die Funktion **AnmeldungPruefen** den Wert **False** zurückgeliefert haben, füllt die Prozedur auf die gleiche Art das Feld **BenutzerID** mit dem Wert **0** und **Benutzername** mit einer leeren Zeichenkette.

```
Private Sub cmdAnmelden_Click()
    Dim db As DAO.Database
    Dim strBenutzername As String
    Dim lngBenutzerID As Long
    Set db = CurrentDb
    strBenutzername = Nz(Me!txtBenutzername, "")
    If AnmeldungPruefen(strBenutzername, Nz(Me!txtKennwort, "")) = True Then
        lngBenutzerID = DLookup("BenutzerID", "tblBenutzer", "Benutzername = '" & strBenutzername & "'")
        db.Execute "UPDATE tblOptionen SET Benutzername = '" & strBenutzername & "', BenutzerID = " & lngBenutzerID, _
            dbFailOnError
        If db.RecordsAffected = 0 Then
            db.Execute "INSERT INTO tblOptionen(Benutzername, BenutzerID) VALUES('" & strBenutzername & "', " _
                & lngBenutzerID & ")", dbFailOnError
        End If
        DoCmd.Close acForm, Me.Name
    Else
        MsgBox "Anmeldung nicht erfolgreich."
        db.Execute "INSERT INTO tblOptionen(Benutzername, BenutzerID) VALUES("", 0)", dbFailOnError
        If db.RecordsAffected = 0 Then
            db.Execute "UPDATE tblOptionen SET Benutzername = '' AND BenutzerID = 0", dbFailOnError
        End If
    End If
End Sub
```

Listing 1: Prozedur zum Speichern der Daten nach der Anmeldung in der Tabelle **tblOptionen**

Ändern per Datenmakro unterbinden

Wenn wir etwa das Ändern der Daten eines Datensatzes unterbinden wollen, können wir das durch ein ganz einfaches Datenmakro erledigen. Wir schauen uns das am Beispiel der Tabelle **tblKunden** an, für die wir Änderungen zunächst komplett verhindern wollen. Dazu legen wir ein Datenmakro an, indem wir bei in der Datenblattansicht geöffneter Tabelle den Ribbon-Eintrag **Tabellel\Vorabereignisse\lVor Änderung** anklicken (siehe Bild 1).

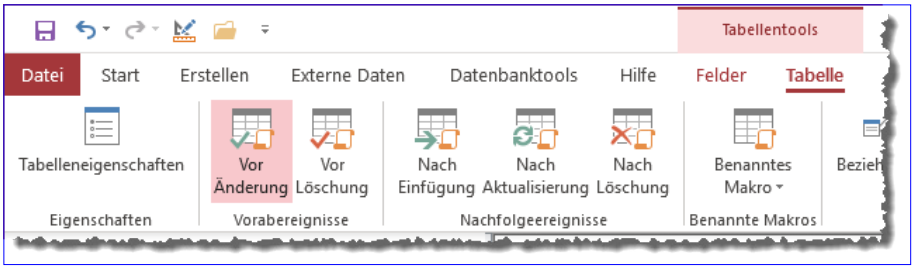


Bild 1: Anlegen eines Datenmakros per Ribbon-Befehl

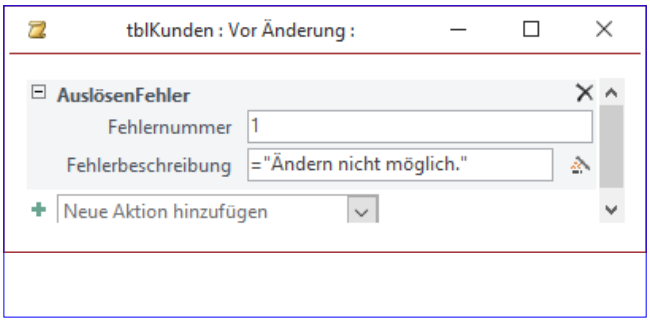


Bild 2: Einfaches Makro zum Unterbinden von Änderungen

Das öffnet den Makro-Editor für dieses Makro, dem wir nur eine einzige Makro-Aktion hinzufügen (siehe Bild 2). Diese enthält den Befehl **AuslösenFehler** mit der Nummer **1** und dem Text **Ändern nicht möglich** als Fehlerbeschreibung.

Wenn Sie nun den Makro-Editor schließen und versuchen, einen der Datensätze der Tabelle **tblKunden** zu ändern, erhalten Sie beim Speichern die Meldung aus Bild 3. Ein ähnliches Datenmakro können Sie für das Ereignis **Vor Löschung** anlegen – in diesem Fall mit einer entsprechenden Fehlermeldung wie **Löschen nicht möglich**. Wenn Sie nun versuchen, einen der Datensätze zu löschen, wird das ebenfalls nicht gelingen.

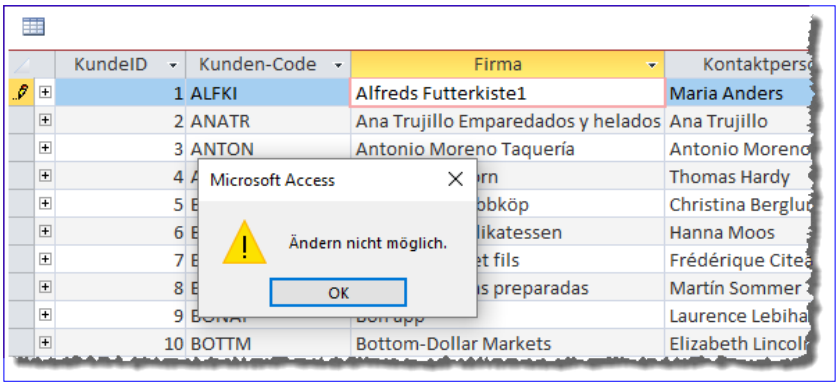


Bild 3: Meldung beim Versuch, einen Datensatz zu ändern

Was aber geschieht, wenn wir einen neuen Datensatz anlegen möchten? Dann erscheint die gleiche Meldung wie beim Versuch, den Datensatz zu ändern. Hier müssen wir also noch differenzieren. Das erledigen wir im Datenmakro für das Ereignis **Vor Änderung**, indem wir eine **Wenn**-Bedingung hinzufügen, für die wir die Bedingung **IstNull([Alt].[KundeID])** festlegen (siehe Bild 4). **Alt** enthält ein Recordset mit den Daten des Datensatzes vor dem Ändern. Bei einem vorhandenen Datensatz liefert **KundeID** den zwangsläufig vorhandenen Primärschlüsselwert, beim Anlegen eines neuen Datensatzes ist **[Alt].[KundeID]** leer, als **NULL**.

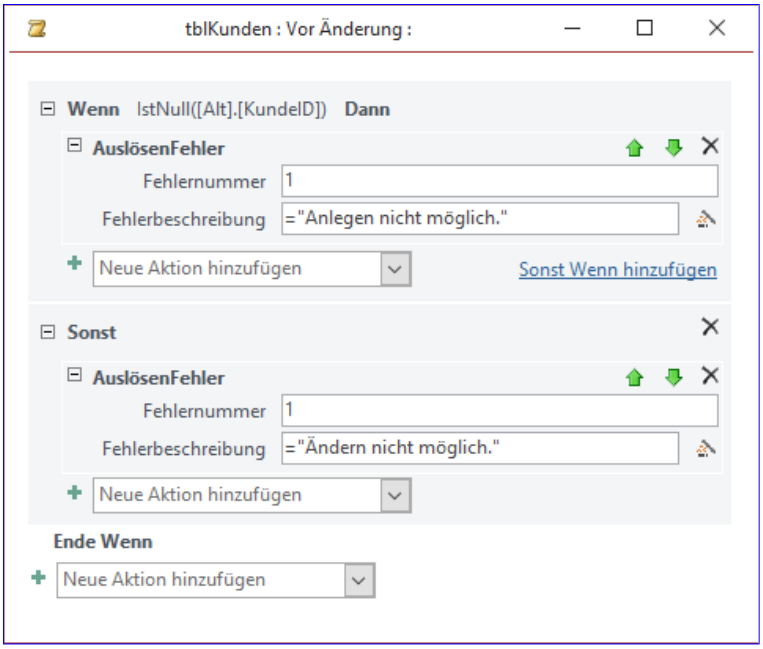


Bild 4: Unterscheidung von neuen und vorhandenen Datensätzen beim Ändern

Wenn dies der Fall ist, soll das Makro die Meldung **Anlegen nicht möglich** ausgeben, sonst **Ändern nicht möglich** (siehe Bild 5).

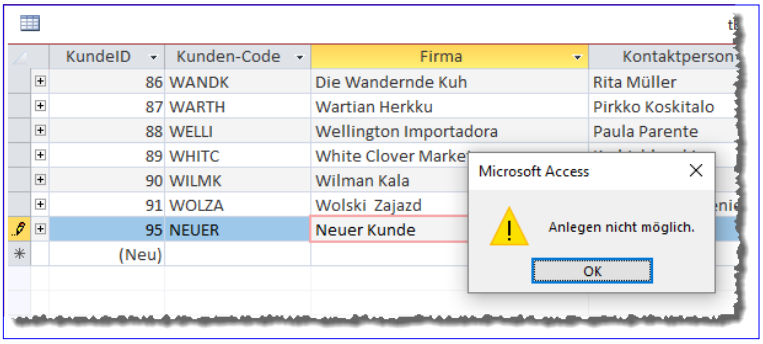


Bild 5: Meldung beim Versuch, einen neuen Datensatz anzulegen

Anmeldung prüfen

Damit kommen wir zu dem Punkt, wo wir die Meldungen nur in Abhängigkeit von den Berechtigungen des Benutzers an der jeweiligen Tabelle anzeigen wollen.

Wenn also etwa ein Benutzer angemeldet ist, der Daten der Tabelle **tblKunden** anlegen und ändern, diese aber nicht löschen darf, soll nur das Löschen von Datensätzen unterbunden werden.

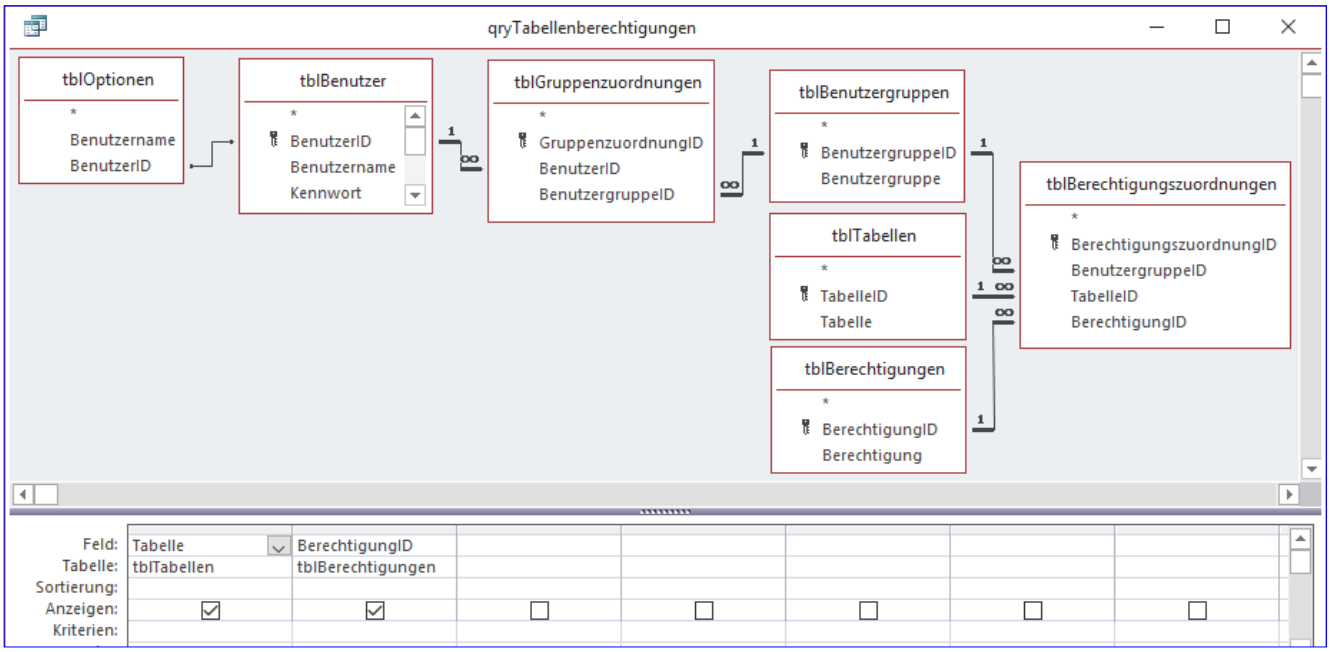


Bild 6: Abfrage zum Ermitteln der Berechtigungen

Abfrage zum Ermitteln der Berechtigung

Da wir nun die Daten des aktuellen Benutzers in einer Tabelle namens **tblOptionen** speichern und diese auch den Wert des Primärschlüsselfeldes des angemeldeten Benutzers enthält, können wir eine Abfrage erstellen, welche diese Daten der Tabelle **tblOptionen** nutzt. Dazu fügen wir dieser Abfrage, die wir unter dem Namen **qryTabellenberechtigungen** speichern wollen, die Tabellen aus Bild 6 hinzu.

Hier sehen Sie auch, dass wir nur zwei Felder zum Entwurfsraster hinzufügen, nämlich **Tabelle** und **BerechtigungID**. Das reicht aus, um zu ermitteln, ob der aktuelle Benutzer eine bestimmte Berechtigung für eine Tabelle hat. Den aktuellen Benutzer ermitteln wir ja bereits über die Tabelle **tblOptionen**, daher müssen wir die Daten der Abfrage nur noch nach dem Namen der zu untersuchenden Tabelle und der zu berücksichtigenden Berechtigung filtern.

Denn wir wollen ja nicht alle Berechtigungen des Benutzers an einer Tabelle ermitteln, sondern nur prüfen, ob der Benutzer über eine bestimmte Berechtigung verfügt. Im Datenmakro **Vor Löschung** etwa wollen wir nur wissen,

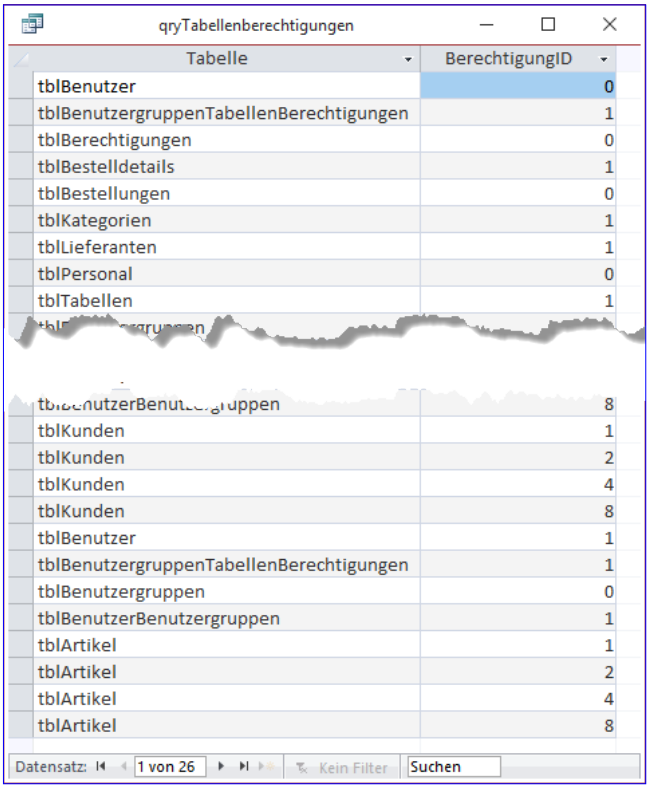


Bild 7: Berechtigungen für den aktuellen Benutzer

ob der aktuell angemeldete Benutzer etwa für die Tabelle **tblKunden** über das **Löschen**-Recht verfügt, also das Recht mit dem Wert **8**. Die Berechtigungen für den aktu-

Kennwörter generieren

Für den einen oder anderen Zweck möchten Sie vielleicht Kennwörter generieren oder in einer Benutzeroberfläche die Möglichkeit zum Generieren von Kennwörtern anbieten. Wenn Sie etwa Benutzer zu einer Benutzerverwaltung hinzufügen, sollten Sie ein initiales Kennwort definieren, das der Benutzer dann bei der ersten Anmeldung ändert. Das Generieren von Kennwörtern wird allerdings umso komplizierter, je mehr Anforderungen es gibt. Einfach nur ein paar Zeichen zufällig auszuwählen, ist noch einfach, aber wenn es Bedingungen gibt, wie sie in sensiblen Umgebungen vorherrschen, soll ein Kennwort beispielsweise mindestens einen Kleinbuchstaben, einen Großbuchstaben, eine Zahl und ein Sonderzeichen enthalten. Diese Anforderungen wollen wir in der hier vorgestellten Funktion natürlich auch unterstützen.

Kennwort mit bestimmter Zeichenanzahl

Die einfachste Variante ermittelt einfach ein Kennwort mit beliebigen Zeichen aus einem bestimmten Zeichenpool, wobei wir mit dem einzigen Parameter der Funktion namens **KennwortGenerierenEinfach** die Anzahl der Zeichen angeben (siehe Listing 1).

Die Funktion definiert eine Konstante, die alle vorgesehenen Zeichen enthält – in diesem Fall alle Großbuchstaben, alle Kleinbuchstaben, alle Zahlen und einige Sonderzeichen:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuv-
wxyz0123456789!""§$%&()[]{}\?**#:;,.-+<>
```

Nach der Definition der beiden übrigen Variablen **strKennwort** zum Speichern des zusammengesetzten Kennworts und **i** als Laufvariable der **For Next**-Schleife initialisiert die Funktion mit der **Randomize**-Funktion den Zufallszahlengenerator.

Die **Rnd**-Funktion, die eine Zufallszahl ermittelt, liefert nämlich sonst nach jedem neuen Start der Access-Anwendung die gleichen Ergebnisse. Sie können das ausprobieren, indem Sie im Direktbereich die **Rnd**-Funktion wie folgt aufrufen:

```
Debug.Print Rnd()
 0,7055475
```

```
Public Function KennwortGenerierenEinfach(intLaenge As Integer)
    Const strZeichen As String = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!""§$%&()[]{}\?**#:;,.-+<>"
    Dim strKennwort As String
    Dim i As Integer
    Randomize
    For i = 1 To intLaenge
        strKennwort = strKennwort & Mid(strZeichen, Fix(Len(strZeichen) * Rnd) + 1, 1)
    Next
    KennwortGenerierenEinfach = strKennwort
End Function
```

Listing 1: Generieren eines einfachen Kennworts

Nach dem Schließen und erneutem Öffnen der Anwendung erhalten Sie mit dem Aufruf der **Rnd()-**Funktion auch das gleiche Ergebnis. Wenn Sie jedoch zuvor einmal die **Randomize**-Anweisung aufrufen, wird die **Rnd()**-Funktion auf Basis der **Timer()**-Funktion initialisiert und liefert somit immer andere Werte.

In der **For...Next**-Schleife durchlaufen wir die Zahlen von **1** bis zu der mit dem Parameter **intLaenge** angegebenen Zeichenzahl. In der einzigen Anweisung innerhalb der Schleife fügen wir der Variablen **strKennwort** jeweils ein zufällig aus der Konstanten **strZeichen** ausgewähltes Zeichen aus. Damit die Funktion **Rnd()** auch genau einen der Werte von **1** bis **87** liefert, was dem ersten und dem letzten Zeichen der Konstanten **strZeichen** entspricht, multiplizieren wir das Ergebnis der **Rnd()**-Funktion, das immer einen Wert zwischen **0** und **1** liefert, mit der Anzahl der Zeichen, runden das Ergebnis und addieren dann den Wert **1** hinzu. Die **Mid**-Funktion ermittelt dann genau den Wert mit der angegebenen Position aus der Zeichenkette **strZeichen**.

Kennwort mit bestimmten Anforderungen

Nun bauen wir eine weitere, etwas leistungsfähigere Variante der Funktion. Diese erhält vier weitere Parameter mit dem Datentyp **Boolean**, die wie folgt heißen:

- **bolGrossbuchstaben**: Gibt an, ob das Kennwort Großbuchstaben enthalten soll.
- **bolKleinbuchstaben**: Gibt an, ob das Kennwort Kleinbuchstaben enthalten soll.
- **bolZahlen**: Gibt an, ob das Kennwort Zahlen enthalten soll.
- **bolSonderzeichen**: Gibt an, ob das Kennwort Sonderzeichen enthalten soll.

Die vier Parameter legen also fest, aus welchen Zeichen das Kennwort bestehen soll.

Abhängig von den Angaben beim Aufruf wird die Ermittlung des Kennwort etwas aufwendiger, denn: Wir wollen nicht etwa die verschiedenen angeforderten Zeichen an einer bestimmten Stelle zurückgeben.

Das wäre einfach: Dann könnten wir zu Beginn einen Großbuchstaben wählen, dann einen Kleinbuchstaben, eine Zahl und schließlich ein Sonderzeichen und den Rest der angeforderten Zeichen für das Kennwort mit beliebigen Zeichen.

Die angeforderten Elemente, also Groß- und Kleinbuchstaben, Zahlen und Sonderzeichen, sollen in beliebiger Reihenfolge im Kennwort auftauchen. Das in einen Algorithmus zu fassen, ist mit etwas Denkarbeit verbunden.

Wir haben uns für den folgenden Ansatz entschieden: Wir starten mit einer Zeichenfolge als Kennwort, die ausschließlich Leerzeichen enthält. Dann prüfen wir die vier **Boolean**-Parameter und fügen an zufällig ausgewählte Stellen die geforderten Zeichen hinzu. Schließlich füllen wir die noch leeren Stellen mit Zeichen aus dem kompletten Pool auf.

Den Ansatz finden Sie in der Funktion **KennwortGenerieren**, deren ersten Teil wir in Listing 2 abgebildet haben. Hier finden Sie zunächst die Parameterliste, die aus der Anzahl der gewünschten Zeichen und den vier bereits beschriebenen Parametern besteht. Alle Parameter sind optional und erhalten den Wert **True**, wenn sie beim Aufruf nicht angegeben werden.

Die Liste der möglichen Zeichen geben wir hier nicht als einzelne Konstante an, sondern wir deklarieren vier verschiedene Konstanten, von denen jede die Zeichen eines bestimmten Typs aufnimmt – also Großbuchstaben, Kleinbuchstaben, Zahlen und Sonderzeichen.

Bevor die Funktion die eigentliche Aufgabe erledigt, prüft sie, ob die Anzahl der Zeichen für das Kennwort mindestens so groß ist wie die Anzahl der gewählten Optionen für

```
Public Function KennwortGenerieren(intLaenge As Integer, Optional bolGrossbuchstaben As Boolean = True, _
        Optional bolKleinbuchstaben As Boolean = True, Optional bolZahlen As Boolean = True, _
        Optional bolSonderzeichen As Boolean = True)
    Const strGrossbuchstaben As String = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    Const strKleinbuchstaben As String = "abcdefghijklmnopqrstuvwxyz"
    Const strZahlen As String = "0123456789"
    Const strSonderzeichen As String = "!""§$%&()[]{}\?**#:;,.-+<>"
    Dim strZeichen As String
    Dim strKennwort As String
    Dim intPositionZufall As Integer
    Dim strZeichenZufall As String
    Dim i As Integer
    Dim intZeichen As Integer
    intZeichen = Abs(bolGrossbuchstaben) + Abs(bolKleinbuchstaben) + Abs(bolZahlen) + Abs(bolSonderzeichen)
    If intLaenge < intZeichen Then
        MsgBox "Die Länge muss mindestens " & intZeichen & " Zeichen betragen."
        Exit Function
    End If
    strKennwort = Space(intLaenge)
    Randomize
    If bolGrossbuchstaben = True Then
        intPositionZufall = Fix(intLaenge * Rnd) + 1
        strZeichenZufall = Mid(strGrossbuchstaben, Fix(Len(strGrossbuchstaben) * Rnd) + 1, 1)
        strKennwort = Left(strKennwort, intPositionZufall - 1) & strZeichenZufall & Mid(strKennwort, intPositionZufall + 1)
        strZeichen = strZeichen & strGrossbuchstaben
    End If
    Do While bolKleinbuchstaben = True
        intPositionZufall = Fix(intLaenge * Rnd) + 1
        If Mid(strKennwort, intPositionZufall, 1) = " " Then
            strZeichenZufall = Mid(strKleinbuchstaben, Fix(Len(strKleinbuchstaben) * Rnd) + 1, 1)
            strKennwort = Left(strKennwort, intPositionZufall - 1) & strZeichenZufall & Mid(strKennwort, _
                intPositionZufall + 1)
            bolKleinbuchstaben = False
            strZeichen = strZeichen & strKleinbuchstaben
        End If
    Loop
    Do While bolZahlen = True
        intPositionZufall = Fix(intLaenge * Rnd) + 1
        If Mid(strKennwort, intPositionZufall, 1) = " " Then
            strZeichenZufall = Mid(strZahlen, Fix(Len(strZahlen) * Rnd) + 1, 1)
            strKennwort = Left(strKennwort, intPositionZufall - 1) & strZeichenZufall & Mid(strKennwort, _
                intPositionZufall + 1)
            bolZahlen = False
            strZeichen = strZeichen & strZahlen
        End If
    Loop
```

Listing 2: Funktion **KennwortGenerieren**, Teil 1

Neuer Datensatz von Frontend zu Backend

Für manche Themen gibt es keine kurze, prägnante Überschrift. In diesem Fall wollen wir zeigen, wie Sie einen neuen Datensatz anlegen, der in einer temporären Tabelle im Frontend gespeichert wird, bis er fertiggestellt ist. Erst danach soll er in einem Rutsch in die entsprechende Tabelle im Backend kopiert werden. Dadurch wollen wir verhindern, dass die Verbindung unnötig lange offen gehalten wird, was beim Zugriff vieler Benutzer gleichzeitig die Performance beeinflussen kann.

Beispiel vorbereiten

Wir benötigen ein Frontend und ein Backend, wobei wir uns die Arbeit vereinfachen wollen. Also erstellen wir zunächst das Frontend namens **NeuerDSFE.accdb** (mit **FE** für Frontend) und fügen dieser eine Tabelle namens **tblVorlagen** hinzu. Die Tabelle enthält die drei Felder aus Bild 1. Danach schließen wir die Datenbank und kopieren diese, wobei die neue Datenbank den Namen **NeuerDSBE.accdb** erhalten soll (mit **BE** für Backend).

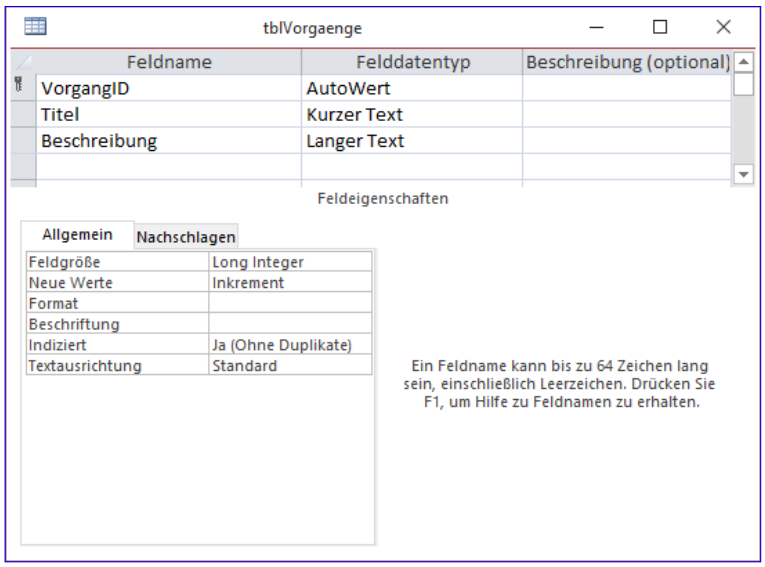


Bild 1: Die Tabelle **tblVorgaenge** in der Entwurfsansicht

Danach ändern wir den Namen der Tabelle **tblVorgaenge** in **tblVorgaengeTemp**. Außerdem erstellen wir eine Verknüpfung zur Tabelle **tblVorgaenge** der neuen Backend-Datenbank. Dazu rufen wir den Ribbon-Befehl **Externe Daten|Importieren und Verknüpfen|Neue Datenquelle|Aus Datenbank|Access** auf (siehe Bild 2). Die folgenden Schritte sind selbsterklärend: Im Dialog **Externe Daten - Access-Datenbank** wählen Sie über die **Durchsuchen**-Schaltfläche die Backend-Datenbank aus und selektieren die Option zum Erstellen einer Verknüpfung.

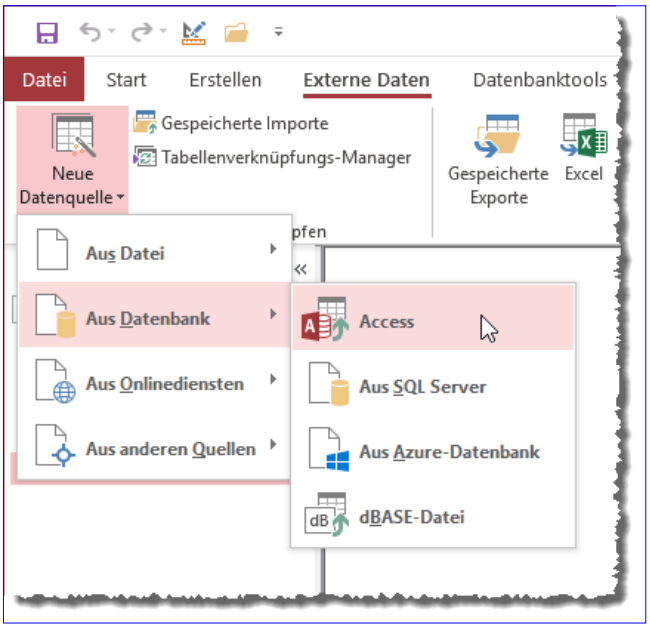


Bild 2: Hinzufügen einer Verknüpfung

Im danach erscheinenden Dialog **Tabellen verknüpfen** wählen Sie die Tabelle **tblVorgaenge** aus und klicken auf **OK**.

Die Tabelle wird nun verknüpft und erscheint wie in Bild 3 im Navigationsbereich der Frontend-Anwendung.

Wenn Sie nun die eingebundene Tabelle **tblVorgaenge** öffnen und den Windows Explorer dabei beobachten, stellen Sie fest, dass die Backend-Datenbank die ganze Zeit als geöffnet markiert wird. Das erkennen Sie daran, dass neben der Datei mit der Endung **.accdb** auch noch eine Datei mit der Endung **.laccdb** erscheint (siehe Bild 4). Damit ist eine Verbindung zu dieser Datei hergestellt, was Vor- und Nachteile haben kann. Wenn das Frontend etwa viel mit dem Backend kommunizieren soll, macht es sogar Sinn, die Verbindung offenzuhalten. Das könnte man beispielsweise erreichen, wenn man beim Starten der Datenbank ein Formular öffnet, das an eine Tabelle des Backends gebunden ist und dieses dann ausblendet. Das Formular ist dann weiterhin geöffnet und hält die Verbindung aufrecht.

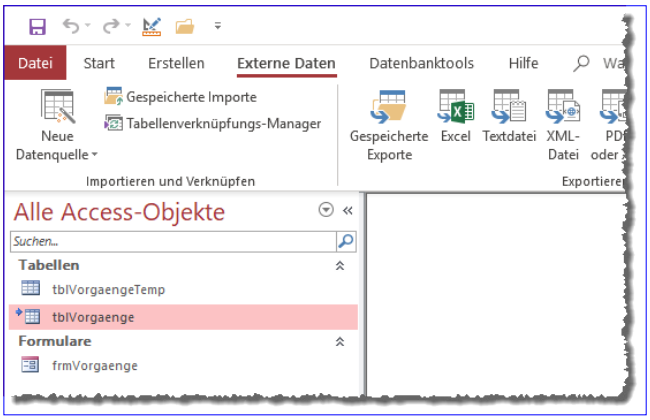


Bild 3: Die temporäre und die verknüpfte Tabelle

Im Gegensatz dazu steht die Konstellation, dass viele Benutzer gleichzeitig mit ihrem Frontend auf die Backend-Datenbank zugreifen wollen. In diesem Fall sollte man die Anzahl der Verbindungen zum Backend eher gering halten.

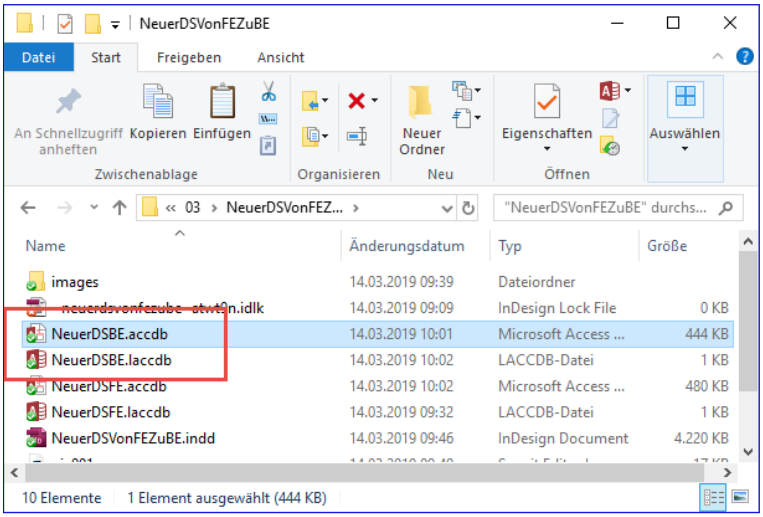


Bild 4: Das geöffnete Backend

Und hier greift die hier vorgestellte Lösung, bei der wir davon ausgehen, dass die Benutzer nicht längere Zeit mit den Daten der Backend-Datenbank arbeiten, sondern nur Datensätze anlegen und diese im Backend speichern wollen. Normalerweise würde man in einem Formular, das an die Tabelle **tblVorgaenge** im Backend gebunden ist, einen neuen Datensatz anlegen, die Daten eingeben und den Datensatz dann speichern. Das Problem ist nur: Auch hier wird die Verbindung vom Zeitpunkt des Öffnens des Formulars bis zum Schließen des Formulars aufrechterhalten. Das heißt, wenn der Benutzer seinen Datensatz angelegt hat und das Formular nicht schließt, bleibt die Verbindung bestehen. Das wollen wir verhindern.

Die Lösung: Temporäre Datensätze

Also legen wir eine temporäre Tabelle im Frontend an, was wir ja eingangs bereits gemacht haben, indem wir die Tabelle **tblVorgaenge** in **tblVorgaengeTemp** umbenannt haben. Der Tabelle **tblVorgaengeTemp** fügen wir noch ein weiteres Feld namens **InBackendUebertragen** hinzu, in dem wir den Zeitpunkt speichern wollen, an dem der Datensatz in die Tabelle **tblVorgaenge** des Backends übertragen wurde (siehe Bild 5).

Wir erstellen ein neues Formular namens **frmVorgaenge**, das wir über die Eigenschaft **Datensatzquelle** an die Tabelle **tblVorgaengeTemp** binden. Das Formular sieht in der Entwurfsansicht wie in Bild 6 aus. Es zeigt alle vier Felder der Tabelle **tblVorgaengeTemp** an, wobei der Benutzer das Feld **InBackendUebertragen** nicht bearbeiten können soll. Daher stellen wir seine Eigenschaft **Aktiviert**

auf **Nein**, **Gesperrt** auf **Ja** und **Rahmen** auf **Transparent** ein.

Außerdem fügen wir dem Formular eine Schaltfläche namens **cmdInBackendUebertragen** hinzu, mit der der Benutzer den Datensatz nach dem Bearbeiten zum Backend kopieren kann.

Damit das Formular direkt beim Start einen neuen, leeren Datensatz anzeigt, hinterlegen wir für die Ereigniseigenschaft **Beim Laden** die folgende Ereignisprozedur:

```
Private Sub Form_Load()
    DoCmd.GoToRecord Record:=acNewRec
End Sub
```

Die einzige Anweisung dieser Prozedur sorgt dafür, dass das Formular einen neuen, leeren Datensatz anzeigt. Wenn das Formular von einem anderen Formular aus geöffnet wird oder über einen Ribbon-Eintrag, können Sie die Anzeige des neuen, leeren Datensatzes auch als Parameter in die **DoCmd.OpenForm**-Methode integrieren:

```
DoCmd.OpenForm "frmVorgaenge", DataMode:=acFormAdd
```

Verwenden Sie letztere Möglichkeit, erscheint das Formular und zeigt nur einen leeren, neuen Datensatz an und bietet nicht die Möglichkeit, die anderen Datensätze einzusehen (siehe Bild 7). Wenn Sie es per Doppelklick auf den Eintrag im Navigationsbereich öffnen, zeigt es alle Datensätze an und der Datensatzzeiger wird auf einen leeren, neuen Datensatz verschoben. Sinnvoller ist es in diesem Fall, das Formular nur zum Anzeigen eines neuen, leeren Datensatzes zu öffnen.

Schaltfläche aktivieren und deaktivieren

Die Schaltfläche **cmdInBackendUebertragen** soll nur für Datensätze aktiviert werden, die noch nicht in das Backend übertragen wurden. Also prüfen wir das in einer Prozedur namens **SchaltflaechenAktivieren**, die wie folgt aussieht:

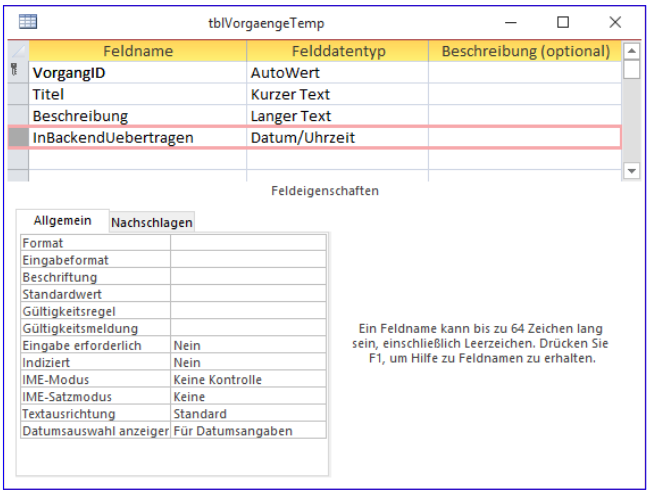


Bild 5: Neues Feld in der Tabelle **tblVorgaengeTemp**

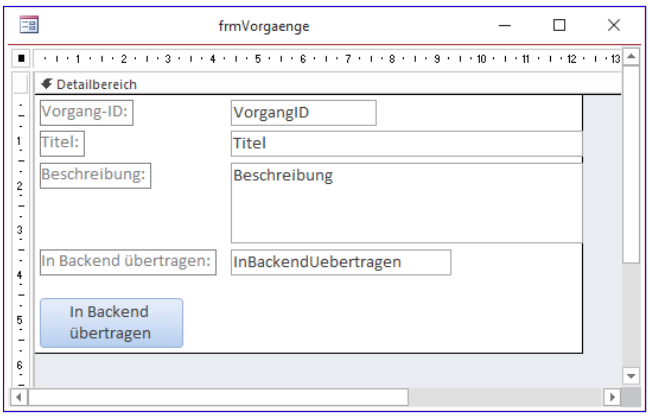


Bild 6: Das Formular **frmVorgaenge** in der Entwurfsansicht

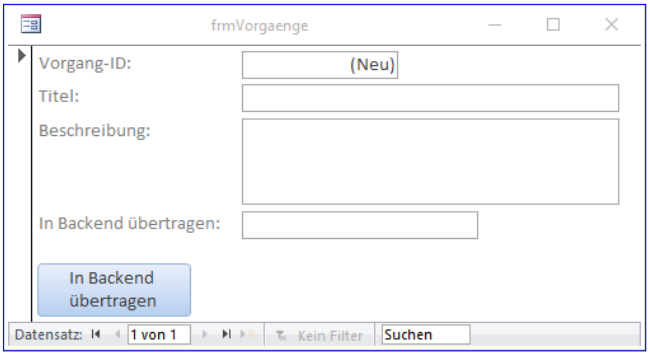


Bild 7: Das Formular **frmVorgaenge** mit einem leeren, neuen Datensatz

```
Private Sub SchaltflaechenAktivieren()
    If IsNull(Me!InBackendUebertragen) Then
        Me!cmdInBackendUebertragen.Enabled = True
    Else
```