

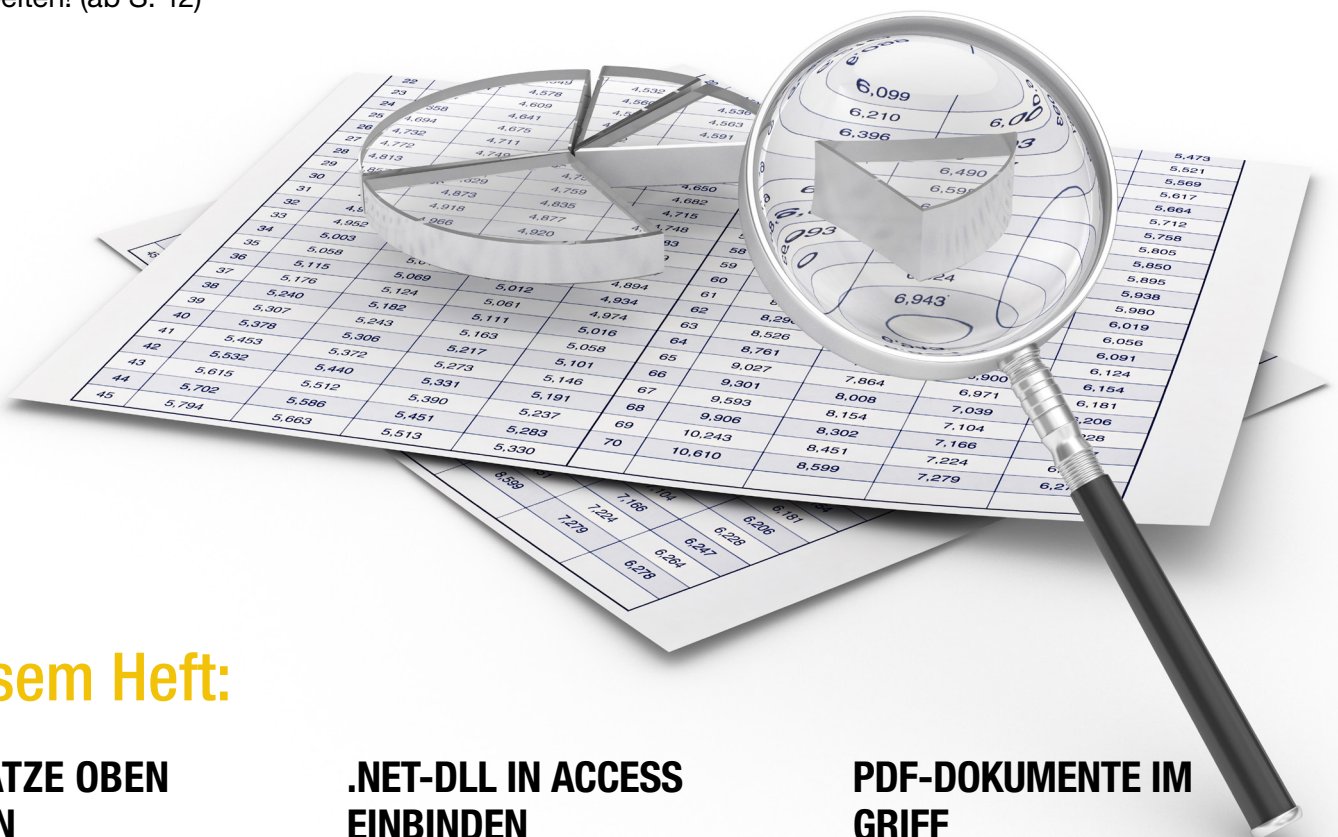
ACCESS

IM UNTERNEHMEN

KREUZTABELLE PER HTML

Kreuztabellen gibt es schon länger – aber nicht per HTML dargestellt und flexibel formatierbar.

Noch besser: Sie können die dargestellten Werte direkt bearbeiten! (ab S. 12)



In diesem Heft:

DATENSÄTZE OBEN EINFÜGEN

Geben Sie Datensätze einmal oben in der Datenblattansicht ein statt unten.

.NET-DLL IN ACCESS EINBINDEN

Nutzen Sie die .NET-Bibliotheken unter Access.

PDF-DOKUMENTE IM GRIFF

Bearbeiten Sie fertige PDF-Dokumente nach Ihren Wünschen mit PDFtk.

SEITE 2

SEITE 65

SEITE 59

Kreuztabellen per HTML

Access im Unternehmen geht gern neue Wege und erfindet Funktionen zu Access hinzu, die es nicht von Haus aus mitbringt. Ein Beispiel sind Kreuztabellen, und zwar ein spezieller Typ: Manche davon zeigen Summen, Mittelwerte et cetera für ihre Daten an. Diese Daten sind logischerweise schreibgeschützt, da sie ja auch mehreren Datensätzen stammen. Wir interessieren uns für die Kreuztabellen, die einfache Werte anzeigen – und liefern Ihnen eine Lösung, wie sie solche Werte bearbeiten können. Und mehr!



Alle paar Monate taucht diese Leserfrage wieder auf: Wie kann man eine Kreuztabelle (englisch: Cross-Table Query) so gestalten, dass man die enthaltenen Daten direkt in dieser Ansicht bearbeiten kann? Die Antwort ist: Mit Bordmitteln funktioniert es nicht. Damit war das Thema auch meist durch. Diesmal haben wir uns allerdings etwas Zeit genommen und einmal geschaut, welche Möglichkeiten es noch gibt. Und sind dabei auf eine interessante Alternative gestoßen: Wie wäre es, wenn man die Daten nicht in der eingebauten Kreuztabellen-Ansicht anzeigt, sondern in einer ganz neuen, vorher noch nicht berücksichtigten Ansicht?

Damit fiel die Wahl schnell auf das Webbrowser-Steuer-element, mit dem wir ja per HTML beliebige Daten in Form von Tabellen anzeigen können. Wir müssen einfach nur etwas mehr Aufwand betreiben. Das sah dann so aus, dass wir erst einmal ein passendes Beispiel brauchten. Das lieferte die Anfrage unseres Lesers direkt mit: Er wollte für verschiedene Abmessungen mit Höhe und Breite feste Preise in einer Tabelle speichern und diese Daten dann in einer Kreuztabelle darstellen, wobei die Höhe in den Spaltenköpfen, die Breite in den Zeilenköpfen und die Preise für die Kombination aus Höhe und Breite in den dazwischen liegenden Zellen abgelegt werden.

Im Beitrag **HTML-Kreuztabelle 1: Basics** zeigen wir ab Seite 12, wie Sie die Kreuztabelle mit HTML aufbauen, um die Daten wie gewünscht darzustellen.

Wie Sie diese Daten dann bearbeiten können, beschreibt der Beitrag **HTML-Kreuztabelle 2: Werte bearbeiten** ab Seite 21. Hier zeigen wir natürlich nicht nur, wie Sie die

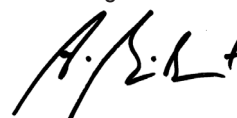
Werte bearbeiten, sondern auch, welcher Mechanismus dafür sorgt, dass die geänderten Daten auch in die Tabelle mit den Preisen geschrieben wird.

In der kommenden Ausgabe geht es dann weiter – hier zeigen wir, wie Sie direkt in der HTML-Tabelle neue Zeilen und Spalten mit Breiten- und Höhenangaben hinzufügen.

Wer öfter mal Berichte als PDF-Dateien speichert, möchte diese vielleicht anschließend noch nachbearbeiten – zum Beispiel, indem er mehrere PDF-Dokumente zusammenführt oder diese mit einem Kennwort schützt. All dies bietet die Anwendung **PDFtk**, die wir unter **PDF-Dokumente im Griff mit PDFtk** ab Seite 59 beschreiben. Wie Sie diese Anwendung von VBA aus aufrufen, zeigen wir im Beitrag **Kommandozeile per DLL** ab Seite 71. Die Voraussetzung dazu, nämlich eine .NET-DLL, liefern wir unter dem Titel **VB.NET-DLL für Access programmieren** ab Seite 65.

Eine praktische Alternative zum Anlegen neuer Datensätze in der untersten Zeile eines Datenblatts, wie von Access standardmäßig vorgesehen, liefert der Beitrag **Neue Datensätze oben anfügen** ab Seite 2. Hier tricksen wir mit einem weiteren Unterformular, das wir über dem eigentlichen Unterformular mit den Daten einfügen und das nur zum Eingeben eines neuen Datensatzes dient.

Viel Erfolg bei der Umsetzung!



Ihr André Minhorst

Neue Datensätze oben anfügen

Es scheint in Stein gemeißelt: Neue Datensätze werden unter Access immer unten, am Ende der Datensatzliste angehängt. Das gilt sowohl für die Datenblattansicht als auch für die Endlosansicht. Für uns kein Anlass, es nicht doch einmal auszuprobieren. Mit ein paar Tricks schaffen wir es schließlich, eine passable Datenblattansicht zu programmieren, die scheinbar das Anfügen eines neuen Datensatzes in der obersten Zeile der Datenblattansicht erlaubt.

Wenn Sie in der Datenblattansicht unter Access einen neuen Datensatz einfügen möchten, haben Sie nur eine Möglichkeit: Sie klicken in der Navigationsleiste auf die Schaltfläche für einen neuen Datensatz und fügen den neuen Datensatz dann unten an (siehe Bild 1). Das gilt aber auch nur, wenn diese Leiste sichtbar ist – anderenfalls gelangen Sie nur durch Scrollen oder entsprechende Tastenkombinationen zum Ende der Liste. Oder Sie nutzen den Ribbon-Eintrag **Start|Datensätze|Neu**, um zu einem neuen, leeren Datensatz zu gelangen.

Nun, wir wollen schauen, ob wir nicht doch einen Weg finden, den neuen, leeren Datensatz oben im Datenblatt anzuzeigen. Das wäre zum Beispiel praktisch, wenn Sie ohnehin immer die neuesten Einträge oben anzeigen und dort auch neue Datensätze anlegen wollen.

Neuer Datensatz dank zweitem Unterformular

Wie aber können wir dies realisieren? Ganz klar: Mit einem einzigen Unterformular in der Datenblatt-Ansicht bekommen wir dies nicht hin. Wir zaubern im oberen Bereich unseres Hauptformulars ein zweites Unterformular mit der gleichen Datenherkunft hin, die auch das eigentliche Unterformular aufweist.

Der Unterschied: Das obere Unterformular soll nur der Eingabe eines neuen Datensatzes dienen und sonst keine Datensätze anzeigen. Das untere Datenblatt soll im Gegenzug keine Möglichkeit mehr bieten, einen neuen Datensatz anzulegen.

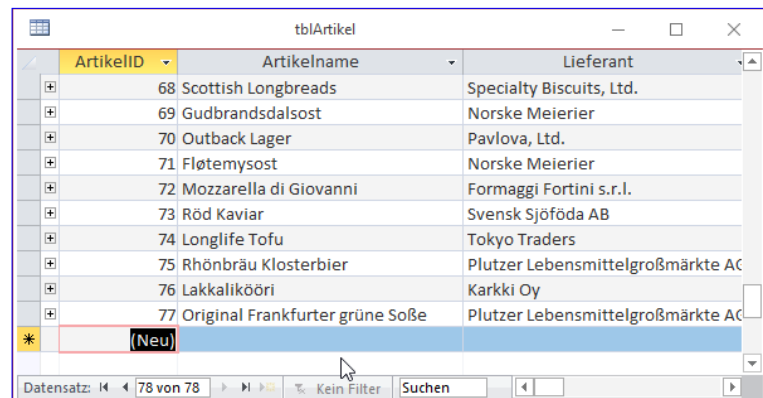


Bild 1: Datensätze werden unter Access unten angefügt.

Soweit, so gut: Damit das obere Formular nur einen neuen, leeren Datensatz anzeigt, brauchen wir diesem lediglich einen Filter hinzuzufügen, der die Anzeige aller anderen Datensätze verhindert – also beispielsweise 1:2.

Damit der neue, leere Datensatz am Ende des zweiten Unterformulars, das ansonsten alle Datensätze anzeigen soll, nicht mehr erscheint, stellen wir einfach die Eigenschaft **Anfügen zulassen** auf den Wert **Nein** ein.

Und dann kommt da natürlich noch ein geschicktes Arrangement der beiden Unterformulare, damit diese wie ein einziges Unterformular aussehen. Doch eins nach dem anderen!

Hauptformular erstellen

Das Hauptformular unserer Anwendung, die wieder auf die Tabellen der Süd Sturm-Datenbank zugreift, soll **frmArtikel** heißen. Legen Sie dieses an und lassen Sie es in der Entwurfsansicht geöffnet. Stellen Sie direkt die Eigenschaften

Datensatzmarkierer und **Navigationsschaltflächen** auf **Nein** und **Automatisch zentrieren** auf **Ja** ein.

Erstes Unterformular

Das erste Unterformular soll **sfmArtikelNeu** heißen. Weisen Sie seiner Eigenschaft **Datenherkunft** die Tabelle **tblArtikel** zu. Stellen Sie außerdem die Eigenschaft **Standardansicht** auf **Datenblatt** ein. Dann speichern und schließen Sie das Formular und ziehen es aus dem Navigationsbereich von Access in den Detailbereich des noch in der Entwurfsansicht geöffneten Hauptformulars (siehe Bild 2).

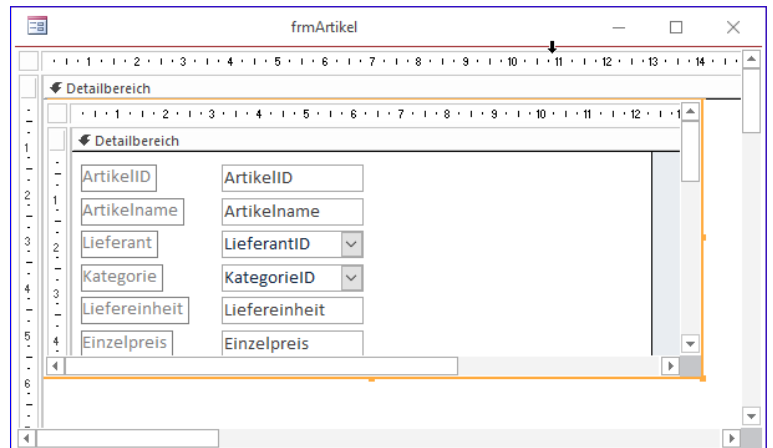


Bild 2: Einfügen des ersten Unterformulars in das Hauptformular

Dieses Formular wollen wir gleich ein wenig breiter gestalten, und zwar so, dass die Felder in der Datenblattansicht nicht die ganze Breite einnehmen und somit keine horizontale Bildlaufleiste angezeigt werden muss. Außerdem können wir die Höhe des Formulars auch gleich soweit reduzieren, dass diese lediglich die Spaltenköpfe und die erste Zeile anzeigt. Das sieht dann etwa wie in Bild 3 aus.

Die Navigationsleiste soll dieses Formular gar nicht anzeigen, daher stellen wir die Eigenschaft **Navigationsschaltflächen** auf den Wert **Nein** ein – ebenso wie die

Eigenschaft **Bildlaufleisten**. Dann können wir in die Formularansicht des Hauptformulars wechseln und die Höhe des Unterformulars kontrollieren. Dieses sollte genau so hoch sein wie in Bild 4 abgebildet.

Damit das Unterformular erst gar keine Datensätze anzeigt, sondern nur den leeren, neuen Datensatz, hinterlegen wir für die Ereigniseigenschaft **Beim Laden** die folgende Ereignisprozedur:

```
Private Sub Form_Load()
    Me!sfmArtikelNeu.Form.Filter = "1=2"
    Me!sfmArtikelNeu.Form.FilterOn = True
End Sub
```

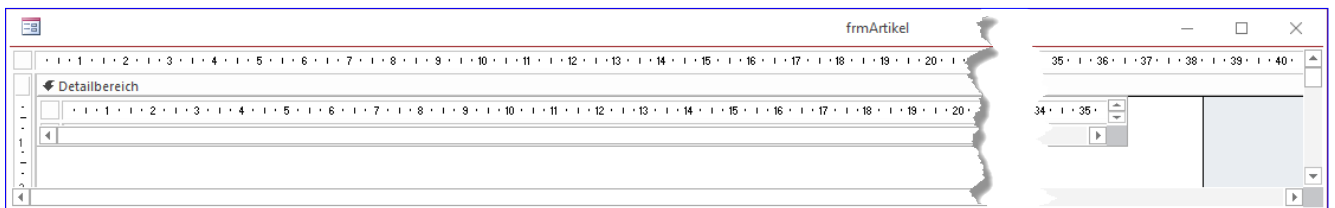


Bild 3: Das Unterformular sollte breit genug sein, damit keine horizontale Bildlaufleiste eingeblendet werden muss.

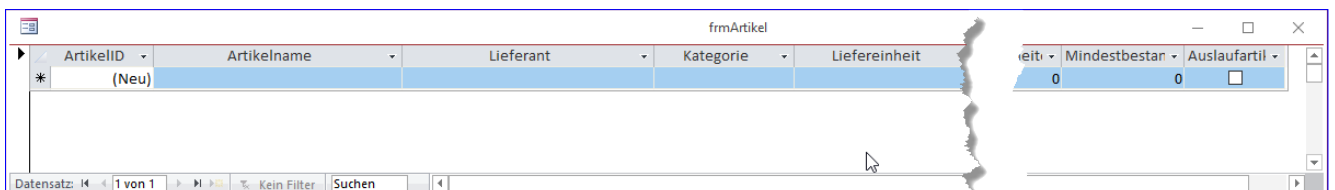


Bild 4: Die Höhe stellen wir so ein, dass soeben die Markierung der Linie unter dem neuen Datensatz sichtbar ist.

Dadurch werden nur die Datensätze angezeigt, für welche die Bedingung **1=2** gilt – also gar keiner.

Zweites Unterformular erstellen und einfügen

Danach begeben wir uns an das zweite Unterformular, das wir **sfmArtikel** nennen. Es soll ebenfalls die Tabelle **tblArtikel** als Datenherkunft verwenden und seine Daten in der Datenblattansicht anzeigen. Darüber hinaus stellen wir seine Eigenschaft **Anfügen zulassen** auf den Wert **Nein** ein. Wie weiter oben bereits beschrieben, soll die Zeile mit dem leeren, neuen Datensatz hier nicht nochmals auftauchen.

Anschließend fügen wir dieses Formular ebenfalls in das Hauptformular **frmArtikel** ein. Dieses bringen wir auf die gleiche Breite wie das Unterformular **sfmArtikelNeu** und eine beliebige Höhe. Im Entwurf sehen die beiden Unterformulare nun wie in Bild 5 aus.

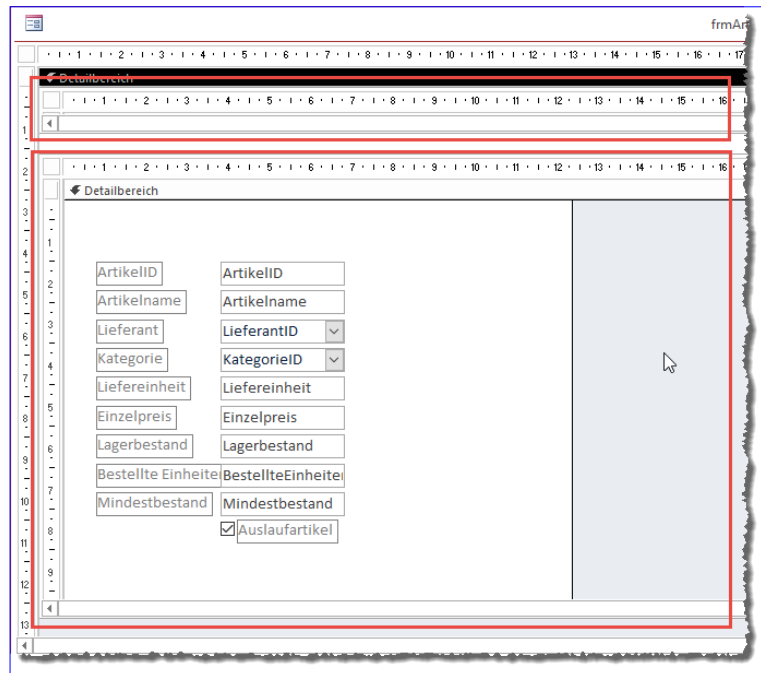


Bild 5: Hauptformular mit den beiden Unterformularen

Unterformulare überlagern

Nun kommt der interessante Teil. Wenn wir uns die Unterformulare in der Formularansicht des Hauptformulars ansehen, erhalten wir die Ansicht aus Bild 6. Wir haben also im oberen Bereich das Unterformular **sfmArtikelNeu**, das nur den leeren, neuen Datensatz anzeigt und keine Navigationsschaltflächen und im unteren Bereich das Unterformular **sfmArtikel**, das alle Datensätze und Navigationsschaltflächen enthält, aber keinen neuen, leeren Datensatz.

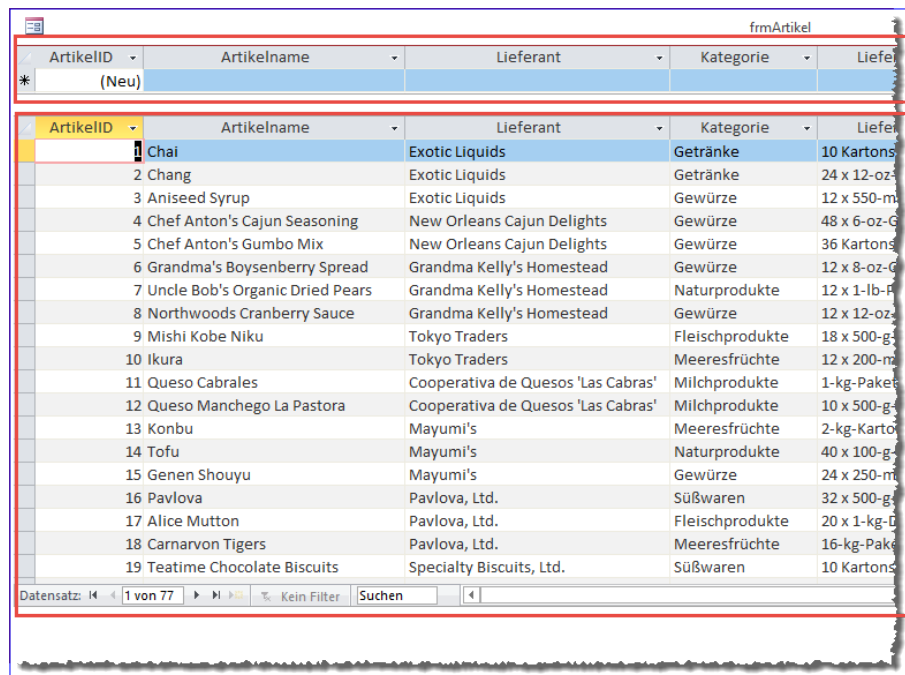


Bild 6: Hauptformular mit den beiden Unterformularen in der Formularansicht

Was stört? Natürlich die Spaltenüberschriften des unteren Unterformulars. Wie ändern wir das? In dem wir einfach das Unterformular **sfmArtikel** unter das Unterformular **sfmArtikelNeu** schieben. Die Sache hat einen kleinen Haken,

denn da wir das Unterformular **sfmArtikelNeu** zuerst angelegt haben, liegt es in der Z-Reihenfolge unter dem Formular **sfmArtikel**. Das können wir aber leicht ändern, indem wir das Formular **sfmArtikelNeu** einmal ausschneiden (**Strg + X**) und es direkt wieder einfügen (**Strg + V**).

Dadurch verschieben wir es in der Z-Reihenfolge über das Formular **sfmArtikel**. Nun schieben wir das Formular **sfmArtikel** soweit nach oben, dass sich das Lineal der Entwurfsansicht unter die Bildlaufleiste von **sfmArtikelNeu** verschiebt (siehe Bild 7). In diesem Bild ist das Unterformular **sfmArtikelNeu** markiert.



Bild 7: Das Formular **sfmArtikelNeu** überlappt nun das Formular **frmArtikel**

Erster Test

Wie sieht dies nun in der Formularansicht aus? So gut wie perfekt, wie Bild 8 zeigt. Ganz oben wird der leere, neue Datensatz angezeigt, darunter befinden sich die vorhandenen Datensätze. Sind wir damit fertig? Nein! Es warten noch eine Menge Aufgaben auf uns:

ArtikelID	Artikelname	Lieferant	Kategorie
(Neu)			
1	Chai	Exotic Liquids	Getränke
2	Chang	Exotic Liquids	Getränke
3	Aniseed Syrup	Exotic Liquids	Gewürze
4	Chef Anton's Cajun Seasoning	New Orleans Cajun Delights	Gewürze
5	Chef Anton's Gumbo Mix	New Orleans Cajun Delights	Gewürze
6	Grandma's Boysenberry Spread	Grandma Kelly's Homestead	Gewürze
7	Uncle Bob's Organic Dried Pears	Grandma Kelly's Homestead	Naturprodukte
8	Northwoods Cranberry Sauce	Grandma Kelly's Homestead	Gewürze
9	Mishi Kobe Niku	Tokyo Traders	Fleischprodukte
10	Ikura	Tokyo Traders	Meeresfrüchte
11	Queso Cabrales	Cooperativa de Quesos 'Las Cabras'	Milchprodukte
12	Queso Manchego La Pastora	Cooperativa de Quesos 'Las Cabras'	Milchprodukte
13	Konbu	Mayumi's	Meeresfrüchte
14	Tofu	Mayumi's	Naturprodukte
15	Genen Shouyu	Mayumi's	Gewürze
16	Pavlova	Pavlova, Ltd.	Süßwaren
17	Alice Mutton	Pavlova, Ltd.	Fleischprodukte
18	Carnarvon Tigers	Pavlova, Ltd.	Meeresfrüchte
19	Teatime Chocolate Biscuits	Specialty Biscuits, Ltd.	Süßwaren

Bild 8: Das Formular, das den neuen Datensatz scheinbar ganz oben anzeigt

- Wenn ein neuer Datensatz angelegt wird, muss das Unterformular **sfmArtikel** aktualisiert werden.
- Wenn der Benutzer die über die Spaltenköpfe des Formulars **sfmArtikelNeu** verfügbaren Sortierungen und Filter nutzt, müssen diese auf die Daten des Unterformulars **sfmArtikel** übertragen werden.
- Und es fallen bei der Bearbeitung dieser beiden Aufgaben bestimmt noch weitere Aufgaben auf ...

Anlegen eines neuen Datensatzes

Wenn wir nun einen neuen Datensatz im Unterformular **sfmArtikelNeu** anlegen, soll dieser nach dem Spei-

chern im Unterformular **sfmArtikel** angezeigt werden. Dazu müssen wir dieses nach dem Anfügen eines neuen Datensatzes aktualisieren. Wir müssen also ein Ereignis im Unterformular **sfmArtikelNeu** abfangen und daraufhin eine Methode des Unterformulars **sfmArtikel** ausführen. Wie erledigen wir das am geschicktesten? Und welches Ereignis wollen wir überhaupt nutzen?

Doch schauen wir uns zunächst einmal an, was geschieht, wenn wir überhaupt einen neuen Datensatz anlegen. Bei der Eingabe des ersten Zeichens oder auch schon vorher kann es geschehen, dass plötzlich die horizontale Bildlaufleiste des Unterformulars auftaucht und die Zeile zur Eingabe verdeckt. In diesem Fall haben Sie offensichtlich die beiden Unterformulare nicht breit genug gestaltet,

sodass die Bildlaufleiste notwendig wird. Also verbreitern Sie nun zunächst die beiden Unterformulare so, dass alle Felder komfortabel Platz finden.

ArtikelID	Artikelname	Lieferant	Kategorie
84	Bananen		
1	Chai	Exotic Liquids	Getränke
2	Chang	Exotic Liquids	Getränke
3	Aniseed Syrup	Exotic Liquids	Gewürze

Bild 9: Hinzufügen eines neuen Datensatzes

Danach beginnen wir, einen neuen Datensatz anzulegen. Das gelingt auch zunächst wie gewohnt – der Datensatzmarkierer erhält das Bearbeiten-Symbol und ein neuer Wert für das Feld **ArtikelID** wird angelegt (siehe Bild 9). Wenn wir die übrigen Felder füllen und dann den Datensatz durch einen Klick auf den Datensatzmarkierer oder durch Verlassen der Zeile speichern, geschieht genau dies – der Datensatz wird gespeichert. Mehr passiert allerdings nicht.

Der neue Datensatz bleibt in der Zeile stehen, die normalerweise zum Anlegen weiterer neuer Datensätze genutzt werden soll. Und der neue Datensatz erscheint auch nicht automatisch im unteren Unterformular **sfmArtikel**.

Davon abgesehen können wir im Unterformular **sfmArtikelNeu** nun nach unten zum nächsten neuen Datensatz scrollen – immerhin.

Wie bekommen wir es nun hin, dass erstens der neue Datensatz aus dem Unterformular **sfmArtikelNeu** verschwindet und dieser zweitens im Unterformular **sfmArtikel** erscheint? Dazu verwenden wir eine Ereignisprozedur, die beim Speichern des neuen Datensatzes ausgelöst wird, nämlich **Nach Aktualisierung**.

Diese könnten wir nun direkt im Unterformular **sfmArtikelNeu** auslösen und dann das Unterformular selbst per **Requery** und das Unterformular **sfmArtikel** über den Bezug **Me!Parent!sfmArtikel.Form** aktualisieren. Das gefällt uns stilistisch aber nicht, sodass wir die Ereignisprozedur im Klassenmodul des Hauptformulars implementieren.

Dort legen wir zunächst zwei Objektvariablen für die beiden Unterformulare **sfmArtikelNeu** und **sfmArtikel** an:

```
Dim WithEvents sfmNeu As Form
Dim WithEvents sfm As Form
```

Diese stattdessen wir mit dem Schlüsselwort **WithEvents** aus, damit wir Ereignisprozeduren für diese beiden Formulare hier in diesem Klassenmodul implementieren können.

Die Ereignisprozedur, die durch das Auslösen des Ereignisses **Beim Laden** ausgelöst wird, nutzen wir, um diese Variablen mit Verweisen auf die beiden Unterformulare zu füllen:

```
Private Sub Form_Load()
    Set sfm = Me!sfmArtikel.Form
    Set sfmNeu = Me!sfmArtikelNeu.Form
    With sfmNeu
        .AfterUpdate = "[Event Procedure]"
        .Filter = "1=2"
        .FilterOn = True
    End With
End Sub
```

Die ersten beiden Anweisungen füllen die Objektvariablen **sfm** und **sfmNeu** mit den Verweisen auf die Unterformulare.

Danach stellen wir Eigenschaften für das Unterformular **sfmArtikelNeu** ein. Die **Filter**-Eigenschaften hatten wir bereits vorher für **Me!sfmArtikelNeu.Form** eingestellt und setzen diese nun für **sfmNeu**. Außerdem legen wir mit der Eigenschaft **AfterUpdate** fest, dass wir in diesem Klassenmodul auf das Auslösen des Ereignisses **Nach Aktualisierung** des Unterformulars **sfmArtikelNeu** lauschen.

HTML-Kreuztabelle 1: Basics

Kreuztabellenabfragen sind eine praktische Sache, wenn es darum geht, Kombinationen aus m:n-Beziehungen abzubilden – zum Beispiel die Preise für verschiedene Bauelemente oder Materialien in Abhängigkeit von der Höhe und der Breite des Materials. Dabei benötigen Sie noch nicht einmal eine m:n-Beziehung, die in einer Kreuztabelle darzustellenden Daten können auch aus einer einzigen Tabelle stammen. Der Haken an Kreuztabellen ist, dass diese in der Regel nicht bearbeitet werden können. Wenn Sie also etwa den Preis für ein Bauelement mit einer Höhe von einem Meter und einer Breite von 50 Zentimetern einstellen wollen, müssen Sie wieder die zugrunde liegende Tabelle oder das darauf aufbauende Formular bemühen. Im vorliegenden Beitrag wollen wir zunächst einmal die Grundlage für die Bearbeitung schaffen – indem wir die Daten der Tabelle per HTML in Kreuztabellenform darstellen.

Basistabelle

Als Basistabelle verwenden wir die Tabelle **tblMaterialpreise**. Diese enthält die Felder **MaterialpreisID**, **Breite**, **Hoehe** und **Preis**. Die Datentypen können Sie dem Tabellenentwurf aus Bild 1 entnehmen.

Hier sehen Sie auch, dass wir für die beiden Felder **Hoehe** und **Breite** einen zusammengesetzten, eindeutigen Index definiert

haben. Damit stellen wir sicher, dass für jede Kombination aus Höhe und Breite nur ein Preis festgelegt werden kann – mehr können wir in der Kreuztabelle nicht darstellen.

Daten in Kreuztabellenform ausgeben

Die Daten sollen nun so ausgegeben werden, dass die Spaltenüberschriften die Höhe, die Zeilenüberschriften die Breite und die Zellen selbst den jeweiligen Preis für die Kombination aus Höhe und Breite anzeigen. Dazu fügen wir der Tabelle erst einmal ein paar Datensätze zum Spielen hinzu. Die gefüllte Tabelle sieht dann wie in Bild 2 aus.

Feldname	Felddatentyp	Beschreibung (optional)
MaterialpreisID	AutoWert	Primärschlüsselfeld der Tabelle
Hoehe	Zahl	Höhe in Millimeter
Breite	Zahl	Breite in Millimeter
Preis	Währung	Preis für die Kombination aus Höhe und Breite

Indexname	Feldname	Sortierreihenfolge
PrimaryKey	MaterialpreisID	Aufsteigend
UniqueKey	Hoehe	Aufsteigend
	Breite	Aufsteigend

Bild 1: Entwurf der Tabelle **tblMaterialpreise**

Materialpre	Hoehe	Breite	Preis	Zum Hinz
1	1000	1000	100,00 €	
2	900	1000	90,00 €	
3	1000	900	90,00 €	
4	900	900	81,00 €	
5	1000	800	80,00 €	
6	900	800	72,00 €	
7	800	800	64,00 €	
8	800	900	72,00 €	
9	800	1000	80,00 €	
* (Neu)	0	0	0,00 €	

Bild 2: Beispieldaten in der Tabelle **tblMaterialpreise**

Danach wollen wir zuerst einmal die Daten in Kreuztabelleform im Direktbereich des VBA-Editors ausgeben. Erst nachdem wir den grundlegenden Ablauf programmiert haben, nehmen wir die Darstellung in einer HTML-Tabelle hinzu.

Die Ausgabe im Direktbereich soll durch die Prozedur **Kreuztabelle** erfolgen (siehe Listing 1). Diese definiert eine Variable für das aktuelle Datenbankobjekt (**db**) sowie drei **Recordset**-Variablen. Das Recordset **rstSpaltenkoepfe** nimmt alle Werte des Feldes **Hoehe** der Tabelle **tblMaterialpreise** auf. Damit jeder Wert nur einmal vorkommt, fügen wir neben **SELECT** als zweites Schlüsselwort noch **DISTINCT** hinzu. Auf diese Weise liefert das Recordset jeden Wert des Feldes **Hoehe** nur einmal, hier **800**, **900** und **1.000**. Außerdem sortieren wir die Wer-

te nach der Größe. Damit können wir schon einmal die Spaltenüberschriften in den Direktbereich schreiben. Hier beginnen wir mit einer leeren Spalte, was wir durch die Ausgabe einer leeren Zeichenkette plus dem Komma-Zeichen (,) stellvertretend für einen Tabulator-Schritt erledigen. Dann durchlaufen wir in einer **Do While**-Schleife alle Datensätze des Recordsets **rstSpaltenkoepfe** und geben nacheinander alle Werte in einer Zeile aus. Damit **Debug.Print** nach der Ausgabe eines Wertes nicht in die nächste Zeile springt, hängen wir auch hier jeweils das Komma an. Dadurch wird auch hier ein Tabulator eingefügt statt des üblichen Zeilenumbruchs. Damit wäre die Zeile mit den Spaltenköpfen bereits geschafft. Das Ergebnis sieht bisher so aus:

800 900 1000

```
Public Sub Kreuztabelle()
    Dim db As DAO.Database
    Dim rstSpaltenkoepfe As DAO.Recordset
    Dim rstZeilen As DAO.Recordset
    Dim rstWerte As DAO.Recordset
    Set db = CurrentDb
    Set rstSpaltenkoepfe = db.OpenRecordset("SELECT DISTINCT Hoehe FROM tblMaterialpreise ORDER BY Hoehe", dbOpenDynaset)
    Debug.Print ,
    Do While Not rstSpaltenkoepfe.EOF
        Debug.Print rstSpaltenkoepfe!Hoehe,
            rstSpaltenkoepfe.MoveNext
    Loop
    Debug.Print
    Set rstZeilen = db.OpenRecordset("SELECT DISTINCT Breite FROM tblMaterialpreise ORDER BY Breite", dbOpenDynaset)
    Do While Not rstZeilen.EOF
        Debug.Print rstZeilen!Breite,
        Set rstWerte = db.OpenRecordset("SELECT Preis FROM tblMaterialpreise WHERE Breite = " & rstZeilen!Breite _
            & " ORDER BY Hoehe", dbOpenDynaset)
        Do While Not rstWerte.EOF
            Debug.Print rstWerte!Preis,
                rstWerte.MoveNext
        Loop
        Debug.Print
        rstZeilen.MoveNext
    Loop
End Sub
```

Listing 1: Ausgabe der Spaltenköpfe, Zeilenköpfe und Preise im Direktbereich

Damit wir danach mit der ersten Zeile mit Daten fortfahren können, folgt nun der Aufruf der **Debug.Print**-Anweisung ohne Parameter. Damit springen wir direkt in die nächste Zeile. Das zweite Recordset namens **rstZeilen** füllen wir mit allen Einträgen des Feldes **Breite** der Tabelle **tblMaterialpreise**, diesmal aufsteigend sortiert nach dem Feld **Breite**. Auch hier verwenden wir das **DISTINCT**-Schlüsselwort, damit jede Breite nur einmal aufgenommen wird. Dieses Recordset enthält nun also jeweils einen Datensatz für jede Breite, hier **800, 900** und **1.000**.

Dieses Recordset durchlaufen wir nun in der nächsten **Do While**-Schleife und wollen in jedem Durchlauf eine komplette Zeile mit dem Zeilenkopf und den Werten für die Kombination aus Spaltenkopf und Zeilenkopf in den Direktbereich schreiben. Den Zeilenkopf geben wir schon einmal per **Debug.Print** aus, wobei wir dieser als Parameter den Wert **rstZeilen!Breite** plus einem Komma für einen Sprung zur nächsten Tabulatorposition statt in die nächste Zeile.

Außerdem brauchen wir für jede Zeile noch ein weiteres Recordset, das die Werte für die aktuelle Zeile und die jeweilige Spalte abrufen. Dieses heißt **rstWerte** und enthält alle Werte des Feldes **Preis** der Tabelle **tblMaterialpreise**, deren Feld **Breite** den Wert des aktuellen Spaltenkopfes enthält – wieder sortiert nach dem Feld **Hoehe**.

Auch diese Werte durchlaufen wir in einer **Do While**-Schleife und geben in dieser jeweils den Inhalt des Feldes Preis aus – gefolgt von einem Komma, damit kein Zeilenumbruch, sondern ein Sprung an die nächste Tabulator-Stelle erfolgt. Das Ergebnis nach dem Durchlaufen sieht dann wie folgt aus:

	800	900	1000
800	64	72	80
900	72	81	90
1000	80	90	100

Materialpre	Hoehe	Breite	Preis	Zum Hinz
1	1000	1000	100,00 €	
2	900	1000	90,00 €	
3	1000	900	90,00 €	
4	900	900	81,00 €	
5	1000	800	80,00 €	
6	900	800	72,00 €	
7	800	800	64,00 €	
8	800	900	72,00 €	
9	800	1000	80,00 €	
10	700	800	56,00 €	
11	700	1000	70,00 €	
(Neu)	0	0	0,00 €	

Bild 3: Erweiterte Beispieldaten in der Tabelle **tblMaterialpreise**

Direktbereich	700	800	900	1000
800	0	64	72	80
900	72	81	90	
1000	0	80	90	100

Bild 4: Ausgabe der neuen Beispieldaten

Gar nicht schlecht für den Start! Allerdings enthält diese Vorgehensweise eine kleine Schwachstelle. Wir ergänzen die Tabelle **tblMaterialpreise** wie in Bild 3. Wie Sie sehen, haben wir mit **700** eine neue Höhe eingeführt, aber nur Werte für die Breiten **800** und **1.000** angegeben – die Kombination aus **700** und **900** fehlt. Das Ergebnis im Direktfenster sieht dann wie in Bild 4 aus. Der Preis für die Kombination **700** und **900** wird einfach weggelassen und die übrigen Kombinationen für die Höhe von 700 mm werden einfach nach links verschoben.

Alle Kombinationen ohne Ausnahme ausgeben

Wie bekommen wir es also hin, dass alle Kombinationen berücksichtigt werden, auch wenn ein Wert nicht angegeben ist – in der Form, dass die entsprechende Stelle einfach frei gelassen wird?

Dazu müssen wir eine kleine Ergänzung vornehmen, und zwar innerhalb der zweiten **Do While**-Schleife (siehe Listing 2). In dieser fügen wir zunächst eine Anweisung

```
Public Sub Kreuztabelle()
    '... wie in der ersten Fassung
    Set rstZeilen = db.OpenRecordset("SELECT DISTINCT Breite FROM tblMaterialpreise ORDER BY Breite", dbOpenDynaset)
    Do While Not rstZeilen.EOF
        rstSpaltenkoepfe.MoveFirst
        Debug.Print rstZeilen!Breite,
        Set rstWerte = db.OpenRecordset("SELECT Preis, Hoehe FROM tblMaterialpreise WHERE Breite = " & rstZeilen!Breite & " ORDER BY Hoehe", dbOpenDynaset)
        Do While Not rstWerte.EOF
            If rstSpaltenkoepfe!Hoehe = rstWerte!Hoehe Then
                Debug.Print rstWerte!Preis,
                rstWerte.MoveNext
            Else
                Debug.Print ,
            End If
            rstSpaltenkoepfe.MoveNext
        Loop
        Debug.Print
        rstZeilen.MoveNext
    Loop
End Sub
```

Listing 2: Ausgabe der Spaltenköpfe, Zeilenköpfe und Preise im Direktbereich auch mit leeren Kombinationen

ein, welche den Datensatzzeiger des Recordsets, das wir zum Einfügen der Spaltenköpfe verwendet haben, für jeden Datensatz des Recordsets **rstZeilen** auf den ersten Datensatz zurücksetzt (**MoveFirst**). Warum das? Weil wir sicherstellen wollen, dass die Markierung im Direktfenster für jeden Spaltenkopf in jeder Zeile auch um eine Tabulatorposition vorrückt – unabhängig davon, ob es auch einen Wert für die Kombination aus den im Spalten- und Zeilenkopf angezeigten Werten gibt. Innerhalb der inneren **Do While**-Schleife, in der wir bisher einfach nur die Werte des Feldes **Preis** in den Direktbereich geschrieben haben, fügen wir nun eine **If...Then**-Bedingung ein, die prüft, ob der Wert des Feldes **Hoehe** des Recordsets **rstSpaltenkoepfe** mit dem des Recordsets **rstWerte** übereinstimmt. Ist dies der Fall, wird der Inhalt des Feldes **Preis** ausgegeben und wir gehen mit der **MoveNext**-Methode einen Schritt weiter im Recordset **rstWerte**. Falls nicht, geben wir nur einen Tabulatorschritt im Direktfenster aus. In beiden Fällen bewegen wir den Datensatzzeiger

des Recordsets **rstSpaltenkoepfe** um eine Position weiter. Auf diese Weise geben wir für Kombinationen, für die es keinen Preis gibt, einen Tabulatorsprung im Direktbereich aus. Das Ergebnis sieht nun wie in Bild 5 aus.

Damit können wir die grundlegende Ausgabe als erledigt betrachten und uns der ersten Erweiterung zuwenden – der Ausgabe im HTML-Format im Webbrowser-Steuererelement von Access.

Webbrowser-Steuererelement anlegen

Dazu legen wir ein neues Formular namens **frmKreuztabelleHTML** an und fügen diesem ein **Webbrowser**-Steuerelement

	700	800	900	1000
800	56	64	72	80
900		72	81	90
1000	70	80	90	100

Bild 5: Ausgabe der neuen Beispieldaten mit Leerstellen

HTML-Kreuztabelle 2: Werte bearbeiten

Im Beitrag »Kreuztabelle per HTML« haben wir gezeigt, wie Sie die Daten einer Tabelle in Form einer Kreuztabelle ausgeben können. Das ist natürlich auch per Kreuztabelleabfrage möglich, aber wir haben in diesem Fall das Webbrowser-Steuer-element mit einer entsprechenden HTML-Seite verwendet. Der Hintergrund ist, dass wir so Funktionen zum direkten Bearbeiten der Einträge hinzufügen können – vorausgesetzt, dass die Kreuztabelle nur die Werte einer Kombination anzeigt und nicht etwa Domänenfunktionen wie Summen oder Mittelwerte. Wir wollen also die bereits vorhandene Darstellung noch um Funktionen zum Bearbeiten sowie zum Hinzufügen neuer Spalten oder Zeilen erweitern.

Ausgangspunkt ist die im bereits genannten Beitrag **HTML-Kreuztabelle 1: Basics** (www.access-im-unternehmen.de/1161) genannte Ansicht aus Bild 1. Hier wollen wir nun folgende Erweiterungen hinzufügen:

- Wenn der Benutzer auf einen der Einträge für die Zahlenwerte klickt, soll er den angezeigten Wert bearbeiten können. Das Verlassen der Zelle soll den Wert in der zugrunde liegenden Tabelle speichern.
- In der oberen Zeile mit den Spaltenköpfen wollen wir rechts ein Plus-Zeichen anzeigen, mit dem der Benutzer eine neue Spalte hinzufügen kann.
- In der linken Zeile mit den Zeilenköpfen wollen wir unten ebenfalls ein Plus-Zeichen unterbringen, das per Mausklick die Möglichkeit zum Anlegen einer neuen Zeile ermöglicht.

	700	800	900	1000	1200
800	56	64	72	80	
900		72	81	90	
1000	70	80	90	100	
1200					1440

Bild 1: Diese Ansicht soll um Bearbeitungsfunktionen erweitert werden.

- Die leeren Zellen wollen wir auf irgendeine Weise anklickbar machen, damit der Benutzer dort auch Werte eintragen kann.
- Ein Doppelklick auf einen der Werte soll den entsprechenden Datensatz der Tabelle **tblMaterialpreise** in einem eigenen Formular öffnen.

Tabellenelemente mit ID und der Eigenschaft contentEditable versehen

Um die erste Aufgabe zu erfüllen, müssen wir die unter den Spaltenüberschriften und rechts von den Zeilenüberschriften befindlichen Zellen, die bereits Werte anzeigen, mit eindeutigen Markierungen versehen. Da jeder Datensatz einen Wert der Tabelle **tblMaterialpreise** repräsentiert, können wir hier mit dem Primärschlüsselwert der entsprechenden Datensätze arbeiten. Dieser landet dann in Form des Attributs **id** in einem **div**-Element, welches im **td**-Element steckt und den eigentlichen Zahlenwert einschließt. Außerdem wollen wir hier die Grundlage für die Bearbeitbarkeit der Elemente legen, indem wir für das gleiche **div**-Element das Attribut **contentEditable** auf den Wert **true** einstellen:

```
<table>
  <tbody>
    <tr>
      <td class="columnHeader rowHeader"></td>
```

```
<td class="columnHeader">700</td>
<td class="columnHeader">800</td>
...
</tr>
<tr>
<td class="rowHeader">800</td>
<td>
<div id="10" contenteditable="true">12</div>
</td>
<td>
<div id="7" contenteditable="true">64</div>
</td>
<td>
<div id="6" contenteditable="true">72</div>
</td>
...
</tr>
...
</tbody>
</table>
```

Diese Änderungen erreichen wir leicht, indem wir die Prozedur **KreuztabelleErstellen**, welche das HTML-Dokument aus **objDocument** mit dem HTML-Inhalt füllt. Allerdings führen wir gleich noch ein paar weitere Schritte durch, welche die Funktion zum Bearbeiten der Inhalte vervollständigen.

Änderung der Prozedur KreuztabelleErstellen

Die neue Version der Prozedur **KreuztabellenErstellen** finden Sie in Listing 1.

Hier verwenden wir ein **Collection**-Objekt, das wir im allgemeinen Teil des Klassenmoduls des Formulars deklarieren wollen:

```
Dim colDivs As Collection
```

Die Prozedur **Kreuztabellen** deklariert dann auch ein paar neue Elemente. Das erste ist ein Objekt des Typs **clsDivWrapper**. Was das ist, erläutern wir gleich weiter unten:

```
Dim objDivWrapper As clsDivWrapper
```

Außerdem deklarieren wir eine Variable des Typs **HTMLDivElement** namens **objDiv**:

```
Dim objDiv As MSHTML.HTMLDivElement
```

Damit das **Collection**-Objekt einsatzbereit ist, füllen wir es mit einer neuen Collection:

```
Set colDivs = New Collection
```

Die folgenden Abschnitte sind unverändert, daher haben wir sie im Listing nicht abgebildet. Die Magie bereiten wir in der inneren der beiden verschachtelten **Do While**-Schleifen vor. Hier erstellen wir mit der **createElement**-Methode von **objDocument** zunächst ein neues Objekt des Typs **HTMLDivElement** und referenzieren es mit der Variablen **objDiv**. Dann stellen wir dessen Attribut **id** auf den Primärschlüsselwert des aktuellen Datensatzes des Recordsets **rstWerte** ein sowie das Attribut **contentEditable** auf den Wert **true**.

Außerdem hängen wir das frisch erstellte Element mit der **appendChild**-Methode an das Objekt **objDiv** an.

```
Set objDiv = objDocument.createElement("Div")
objDiv.id = rstWerte!MaterialpreisID
objDiv.contentEditable = "true"
objCell.appendChild objDiv
```

Was dann geschieht, erklärt sich erst später vollständig, wenn wir die Klasse **clsDivWrapper** erläutern, die einen Großteil der Arbeit erledigt: Wir erstellen ein neues Objekt auf Basis der Klasse **clsDivWrapper** und weisen dessen Eigenschaft **Div** das **HTMLDivElement**-Objekt aus **objDiv** zu. Danach hängen wir das Objekt **objDivWrapper** an die Auflistung **colDivs** an:

```
objDiv.innerText = rstWerte!Preis
Set objDivWrapper = New clsDivWrapper
```



```

Private Sub KreuztabelleErstellen()
    ...
    Dim objDivWrapper As clsDivWrapper
    Dim objDiv As MSHTML.HTMLDivElement
    Set colDivs = New Collection
    ...
    Set rstZeilen = db.OpenRecordset("SELECT DISTINCT Breite FROM tblMaterialpreise ORDER BY Breite", dbOpenDynaset)
    Do While Not rstZeilen.EOF
        Set objRow = objTable.insertRow
        rstSpaltenkoepfe.MoveFirst
        Set objCell = objRow.insertCell
        objCell.innerText = rstZeilen!Breite
        objCell.className = "rowHeader"
        Set rstWerte = db.OpenRecordset("SELECT MaterialpreisID, Preis, Hoehe FROM tblMaterialpreise WHERE Breite = " & rstZeilen!Breite & " ORDER BY Hoehe", dbOpenDynaset)
        Do While Not rstWerte.EOF
            If rstSpaltenkoepfe!Hoehe = rstWerte!Hoehe Then
                Set objCell = objRow.insertCell
                Set objDiv = objDocument.createElement("Div")
                objDiv.id = rstWerte!MaterialpreisID
                objDiv.contentEditable = "true"
                objCell.appendChild objDiv
                objDiv.innerText = rstWerte!Preis
                Set objDivWrapper = New clsDivWrapper
                Set objDivWrapper.Div = objDiv
                colDivs.Add objDivWrapper
                rstWerte.MoveNext
            Else
                Set objCell = objRow.insertCell
            End If
            rstSpaltenkoepfe.MoveNext
        Loop
        rstZeilen.MoveNext
    Loop
End Sub

```

Listing 1: Ausschnitt aus der Prozedur **KreuztabelleErstellen**

```

Set objDivWrapper.Div = objDiv
colDivs.Add objDivWrapper

```

Damit kommen wir zum Inhalt und der Funktion der Klasse **clsDivWrapper**.

In der Prozedur **KreuztabelleErstellen** haben wir die Vorbereitung erschaffen und zu jedem **Div**-Element mit einem Wert des Feldes **Preis** der Tabelle **tblMaterial-**

preise eine Instanz der Klasse **clsDivWrapper** erstellt und diese in der global deklarierten Variablen **colDivs** gespeichert. Diese Klasse nun soll die fehlende Aufgabe übernehmen, nämlich das Speichern geänderter Werte in der Kreuztabelle.

Dazu legen Sie ein neues Klassenmodul namens **cls-DivWrapper** an, dem Sie zunächst zwei Deklarationen hinzufügen:

```
Private WithEvents m_Div As MSHTML.HTMLDivElement
Private strText As String
```

Die Variable **strText** soll den aktuell in der Zelle der Kreuztabelle befindlichen Text speichern, die Variable **m_Div** nimmt den von der Prozedur **KreuztabelleErstellen** übergebenen Verweis auf das Objekt des Typs **HTMLDivElement** auf. Die Deklaration haben wir um das Schlüsselwort **WithEvents** ergänzt, da wir für dieses Objekt ein oder mehrere Ereignisprozeduren implementieren wollen.

Damit wir der Klasse wie in der **KreuztabelleErstellen** programmiert den Verweis auf **objDiv** übergeben können, definieren wir eine **Property Set**-Prozedur, die wie folgt aussieht:

```
Public Property Set Div(Div As MSHTML.HTMLDivElement)
    Set m_Div = Div
End Property
```

Nun müssen wir aus den verschiedenen Ereignissen, die für das Element **m_Div** zur Verfügung stehen, die für uns interessanten herausfinden.

Die Ereignisse finden Sie, wenn Sie im linken Kombinationsfeld des Klassenmoduls den Eintrag **m_Div** auswählen und dann das rechte Kombinationsfeld öffnen (siehe Bild 2).

Wir entscheiden uns für die folgenden beiden Ereignisse:

- **onkeyup**: Dieses Ereignis wird ausgelöst, wenn der Benutzer eine Taste der Tastatur loslässt, also soeben ein Zeichen eingegeben oder gelöscht hat.

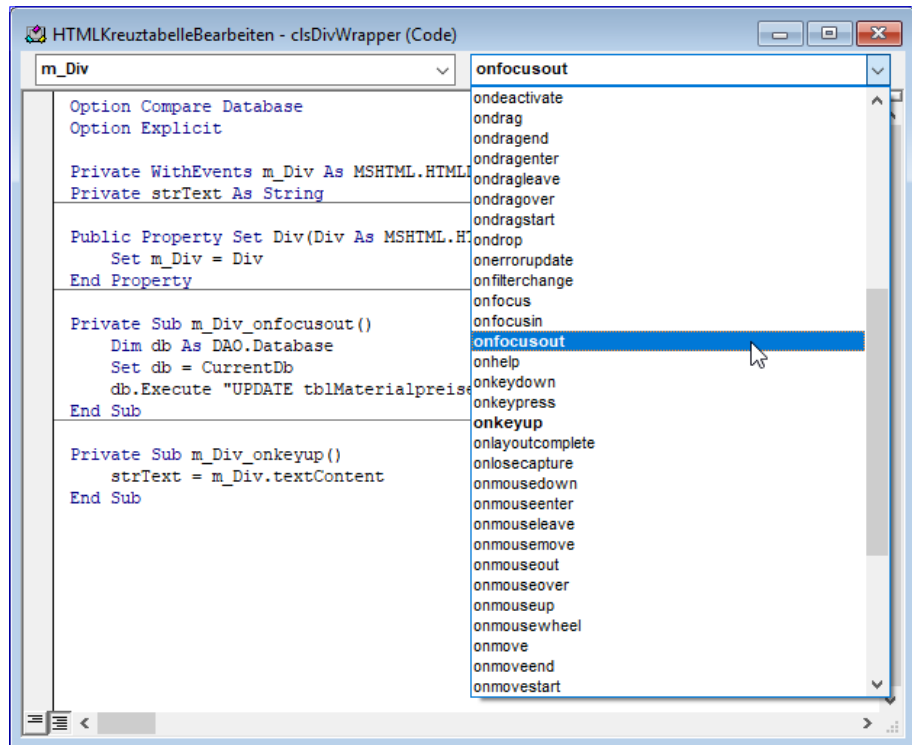


Bild 2: Implementieren der Ereignisse für **m_Div**

- **onfocusout**: Dieses Ereignis wird ausgelöst, wenn das **div**-Element den Fokus verliert.

Das Ereignis **onkeyup** nutzen wir, um nach der Eingabe eines jeden Zeichens den aktuellen Inhalt des **div**-Elements, den wir über die Eigenschaft **textContent** ermitteln, in die Variable **strText** schreiben:

```
Private Sub m_Div_onkeyup()
    strText = m_Div.textContent
End Sub
```

Das Ereignis **onfocusout** soll nach dem Verlassen des Feldes den aktuellen Inhalt in den entsprechenden Datensatz der Tabelle **tblMaterialpreise** eintragen.

Den aktuellen Inhalt können wir nun der Variablen **strText** entnehmen, allerdings müssen wir vor dem Ändern des Datensatzes natürlich prüfen, ob der Wert auch tatsächlich geändert wurde. Nur in diesem Fall enthält **strText** nämlich einen Inhalt – anderenfalls ist es leer:

SQL Server: Tabellendefinition ändern

Unter Access sind Sie es gewohnt, nach Lust und Laune am Entwurf einer Tabelle zu arbeiten. Probleme gibt es nur, wenn Sie einmal die Feldgröße oder den Felddatentyp ändern wollen – etwa, weil die Daten dann nicht mehr in das Feld passen könnten. Das können Sie jedoch durch einen Mausklick bestätigen und weitermachen. Beim SQL Server sieht das etwas anders aus. Relevante Änderungen, die sich auf bestehende Felder beziehen, sieht das SQL Server Management Studio nicht so gern und blockiert dies – Sie müssen dann die Tabelle neu anlegen. Wer einen kleinen Trick nicht kennt, macht sich auf diese Weise viel unnötige Arbeit.

Schauen wir uns kurz ein Beispiel an: Sie haben eine Tabelle namens **tblObjekte** erstellt, die aus den beiden Feldern **ObjektID** (Datentyp **int**) und **Bezeichnung** (Datentyp **varchar(255)**) besteht. Der Entwurf sieht dann wie in Bild 1 aus.

Nun haben Sie allerdings vergessen, einen Autowert für das Primärschlüsselfeld zu hinterlegen. Also öffnen Sie die Tabelle nochmals im Entwurf und bearbeiten die Spalteneigenschaften der Spalte **ObjektID**.

Hier wählen Sie für die Eigenschaft **Identitätsspezifikation** nach dem Erweitern der Eigenschaft den Wert **Ja** aus – die Felder **ID-Ausgangswert** und **ID-Inkrement** werden automatisch auf **1** eingestellt (siehe Bild 2).

Beim Versuch, die Tabelle nach den Änderungen zu speichern, schlägt allerdings fehl. Es erscheint die Meldung aus Bild 3. Diese besagt, dass die vorgenommenen Änderungen eine Neuerstellung der Tabelle erfordern.

Das gelingt allerdings nicht automatisch – zumindest nicht mit den standardmäßig aktivierten Optionen einer

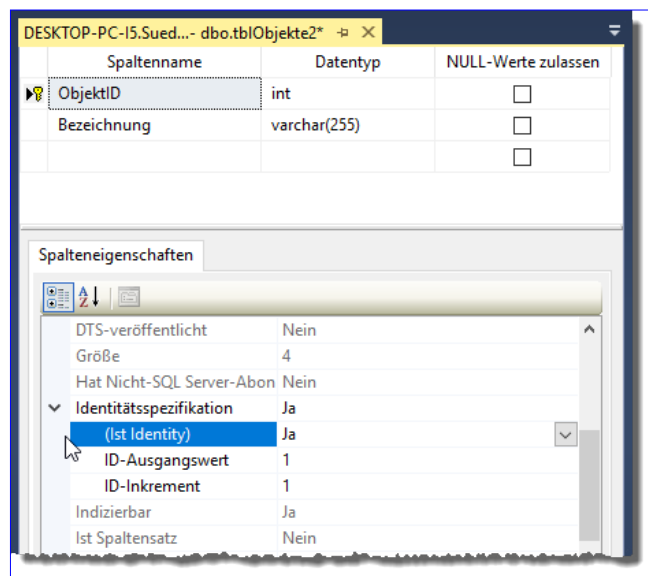


Bild 2: Anpassen der Identitätsspezifikation

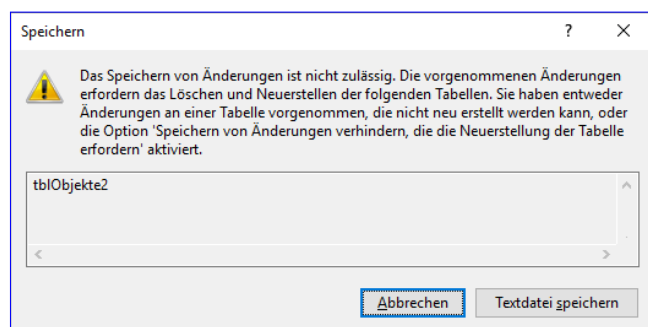


Bild 3: Fehlermeldung beim Versuch, die Änderung zu speichern

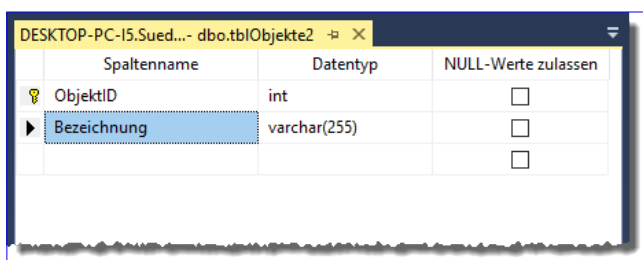


Bild 1: Erster Entwurf der Tabelle **tblObjekte**

Installation des SQL Server Management Studios. Wenn man nicht genau weiterliest, ist man versucht, die Tabelle tatsächlich zu löschen und dann nochmal mit den gewünschten Einstellungen neu anzulegen.

Allerdings gibt die Meldung auch gleich einen Hinweis darauf, dass man sich das Leben durch Anpassen einer bestimmten Option ein wenig leichter machen kann. Diese heißt **Speichern von Änderungen verhindern, die die Neuerstellung der Tabelle erfordern**.

Um diese Option zu aktivieren, klicken Sie auf den Menübefehl **Extras** **Optionen**. Dies öffnet den reichhaltigen Optionen-Dialog des SQL Server Management Studios (siehe Bild 4).

Nun müssen wir nur noch die gesuchte Option finden. Leider ergibt die Eingabe der Option oder auch von Teilen der Bezeichnung in das Suchfeld kein hilfreiches Ergebnis. Also müssen wir uns selbst auf die Suche machen.

Dabei kommen wir allerdings schnell zum Ziel – es handelt sich offensichtlich um eine Einstellung, die zum Tabellen-Designer gehört. Und tatsächlich finden wir die Option unter **Designer** **Tabellen- und Datenbank-Designer** (siehe Bild 5).

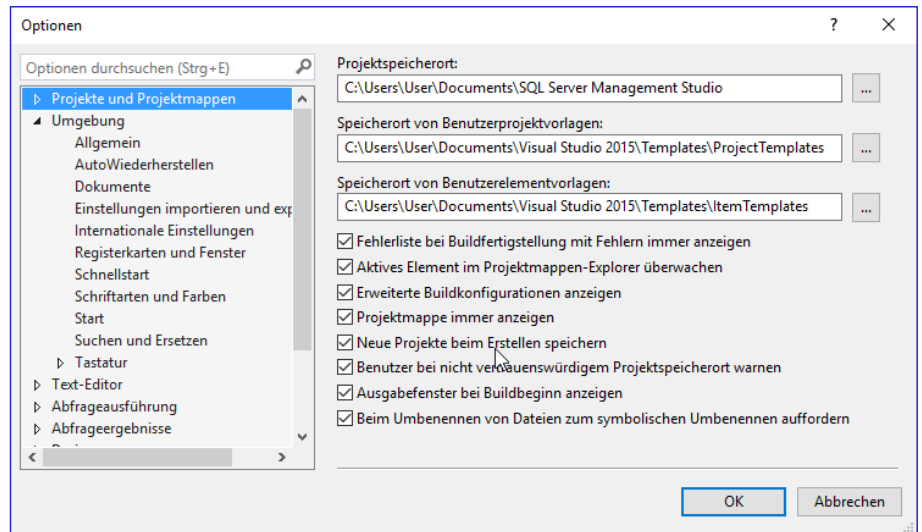


Bild 4: Der Optionen-Dialog des SQL Server Management Studios

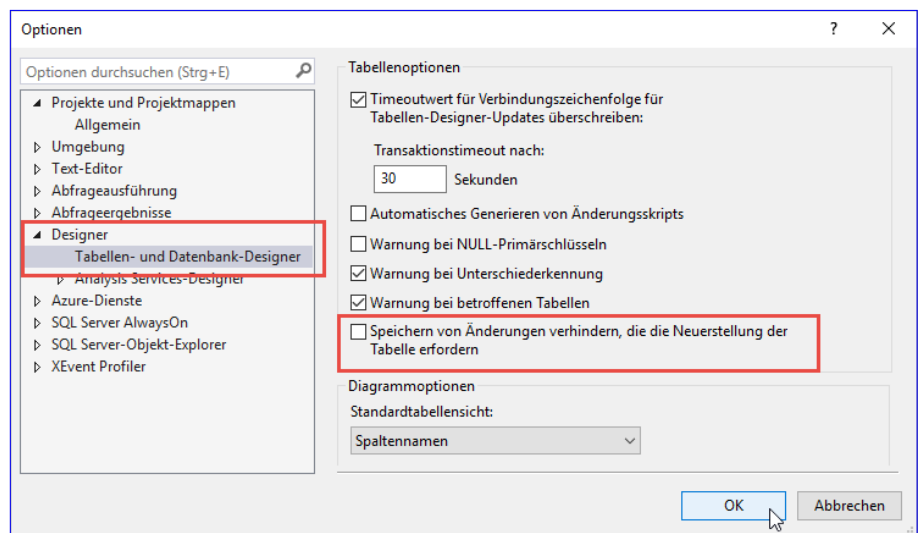


Bild 5: Option, mit der das Neuerstellen von Tabellen beim Speichern verhindert wird

Deaktivieren wir diese Option und klicken auf **OK**, können wir im Anschluss auch die gewünschten Änderungen durchführen – ohne selbst die Tabellen zu löschen und neu erstellen zu müssen. Dies geschieht dann beim Betätigen der Tastenkombination **Strg + S** für das zu ändernde Tabellen-Objekt automatisch im Hintergrund.

ODBC-Verknüpfungen in Formularen

Es gibt verschiedene Möglichkeiten, auf die Daten einer SQL Server-Datenbank zuzugreifen. Die einfachste Variante, wenn es um die Migration einer reinen Access-Datenbank in eine Kombination aus Access-Frontend und SQL Server-Backend geht, ist der Einsatz ODBC-verknüpfter Tabellen. Dabei werden die Tabellen einfach nur zum SQL Server migriert und man greift dann über ODBC-Verknüpfungen auf die Tabellen zu. Für den Zugriff auf diese Daten von Formularen aus ergeben sich so gut wie keine Änderungen – und was sich doch ändern kann, erfahren Sie in diesem Beitrag.

Wenn Sie eine Datenbank von Access zum SQL Server migrieren, übertragen Sie die Tabellen im Optimalfall 1:1 auf die SQL Server-Datenbank. Ist das der Fall, haben Sie recht gute Karten, dass Sie Ihre Anwendung recht schnell auch mit der neuen Datenherkunft läuft.

Wie das geht? Indem Sie einfach die Tabellen der SQL Server-Datenbank per ODBC mit der Access-Anwendung verknüpfen. Wenn Sie dabei darauf achten, dass die Verknüpfungen anschließend die gleichen Namen wie die zuvor verwendeten Tabellen haben, sollten die Abfragen, Formulare, Berichte und auch der VBA-Code, der auf die Daten zugreift, anstandslos funktionieren.

Wir schauen uns das einmal an. In unserer Beispieldatenbank finden Sie zwei Formulare, die Ihnen bei der

Verknüpfung der Tabellen behilflich sind und Ihnen viel Zeit sparen können. Das erste Formular heißt **frmVerbindungszeichenfolgen**. Hier legen Sie mit der Schaltfläche **Neu** eine neue Verbindungszeichenfolge an und geben dann ihre Eigenschaften ein. Ein Beispiel finden Sie in Bild 1 – hier testen wir auch gleich noch die Funktionalität der Verbindungszeichenfolge.

Das zweite hilfreiche Formular heißt **frmTabellenVerknuepfen**. Mit diesem wählen Sie als Erstes die soeben definierte Verbindungszeichenfolge aus und selektieren dann bei gedrückter **Strg**-Taste die zu verknüpfenden Tabellen – in diesem Fall **tblArtikel**, **tblBestelldetails**, **tblBestellungen**, **tblKategorien**, **tblKunden**, **tblLieferanten**, **tblPersonal** und **tblVersandfirmen** (siehe Bild 2).

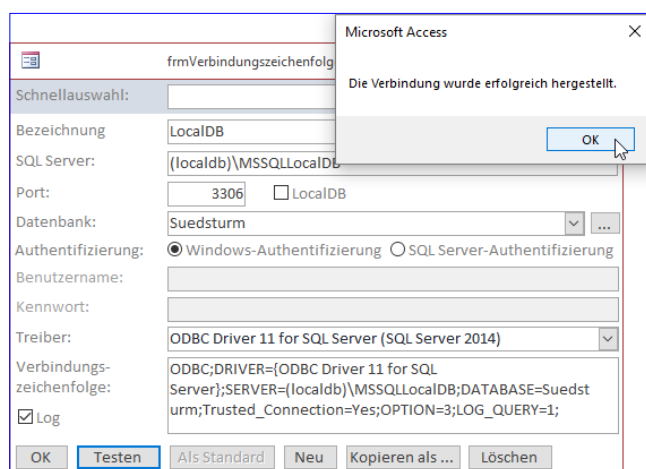


Bild 1: Verbindungszeichenfolge definieren und testen

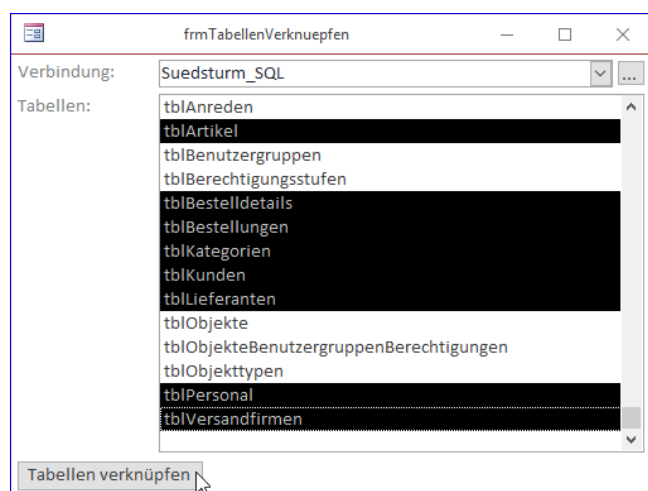


Bild 2: Hinzufügen der zu verknüpfenden Tabellen

Nach einem Klick auf die Schaltfläche **Tabellen verknüpfen** landen die Verknüpfungen dann im Navigationsbereich, wo sie mit dem entsprechenden Symbol angezeigt werden (siehe Bild 3).

Zeitstempel

Die Tabellen wurden mit dem SQL Server Migrations-Assistenten für Access in eine SQL Server-Datenbank migriert. Dabei stellt dieser sicher, dass jede Tabelle ein Timestamp-Feld erhält. Dieses sieht in einer frisch migrierten Datenbank erst einmal wie in Bild 4 aus – das Feld ist zunächst scheinbar leer.

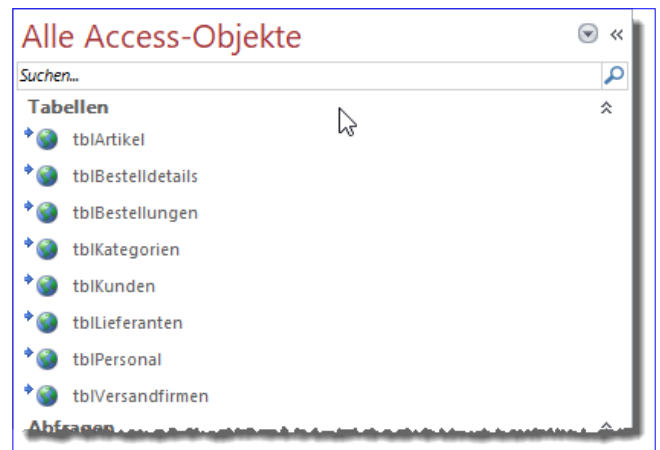


Bild 3: Die per ODBC verknüpften Tabellen

Wenn Sie sich dieses Feld jedoch im SQL Server Management Studio ansehen, sind die Werte dieses Feldes mit Daten des Typs **<Binärdaten>** gefüllt (siehe Bild 5).

KategorieID	Kategorienname	Beschreibung	Abbildung	SSMA_TimeStamp
6	Fleischprodukte	Fleisch-Fertiggerichte	OLE-Objekt	
1	Getränke	Alkoholfreie Getränke, Kaffee, Tee, Bier	OLE-Objekt	
5	Getreideprodukte	Brot, Kräcker, Nudeln, Müsli	OLE-Objekt	
2	Gewürze	Süße und saure Soßen, Gewürze	OLE-Objekt	
8	Meeresfrüchte	Meerespflanzen und -früchte, Fisch	OLE-Objekt	
4	Milchprodukte	Käsesorten	OLE-Objekt	
7	Naturprodukte	Getrocknete Früchte, Tofu usw.	OLE-Objekt	
3	Süßwaren	Desserts und Süßigkeiten	OLE-Objekt	
*	(Neu)		OLE-Objekt	

Bild 4: Tabelle mit Timestamp-Feld

Warum aber wollen wir unsere SQL Server-Tabellen mit einem Zeitstempel versehen – und was ist der Zeitstempel, unter SQL Timestamp, überhaupt? Zunächst einmal: Das Timestamp-Feld ist nicht etwa ein Feld, welches die aktuelle Systemzeit etwa beim Ändern oder Anlegen eines Datensatzes speichert.

KategorieID	Kategorienname	Beschreibung	Abbildung	SSMA_TimeStamp
1	Getränke	Alkoholfreie Ge...	<Binärdaten>	<Binärdaten>
2	Gewürze	Süße und saure...	<Binärdaten>	<Binärdaten>
3	Süßwaren	Desserts und Sü...	<Binärdaten>	<Binärdaten>
4	Milchprodukte	Käsesorten	<Binärdaten>	<Binärdaten>
5	Getreideprodukte	Brot, Kräcker, N...	<Binärdaten>	<Binärdaten>
6	Fleischprodukte1	Fleisch-Fertigg...	<Binärdaten>	<Binärdaten>
7	Naturprodukte	Getrocknete Fr...	<Binärdaten>	<Binärdaten>
8	Meeresfrüchte	Meerespflanzen...	<Binärdaten>	<Binärdaten>
*	NULL	NULL	NULL	NULL

Bild 5: Tabelle mit Timestamp-Feld im SQL Server Management Studio

Es hat auch nicht das Format eines Datumsfeldes. Es ist vielmehr ein Feld, das einen binären Wert speichert, der beim Anlegen oder Ändern eines Datensatzes eingetragen wird und der einzigartig ist.

Damit sorgen Sie für verschiedene Effekte:

- Sie vereinfachen die Untersuchung, ob ein Datensatz seit dem Beginn der Bearbeitung durch einen anderen

Benutzer geändert wurde und ob eine entsprechende Meldung angezeigt werden muss.

- Sie verhindern Fehler, die durch unterschiedliche Fließkomma-Datentypen auftreten können.

Datensatzänderungen prüfen

Bevor Sie einen Datensatz im SQL Server nach der Bearbeitung etwa in einem Formular speichern können, muss der SQL Server prüfen, ob eventuell ein anderer Benutzer in der Zwischenzeit ebenfalls Änderungen an diesem Datensatz durchgeführt hat. Ist das der Fall, erhalten Sie eine Meldung wie in Bild 6.

Wie lösen Sie diesen Fehler aus? Dazu haben wir die Tabelle **tblKategorien**, die in der Beispieldatenbank bereits ein Timestamp-Feld enthält, nochmals in eine neue Tabelle namens **tblKategorien1** kopiert, die kein Timestamp-Feld enthält. Außerdem haben wir für diese Tabelle eine ODBC-Verknüpfung namens **tblKategorien1** in der Access-Datenbank erstellt. Dann haben wir zwei Formulare namens **frmKategorien1** und **frmKategorien2** erstellt, die beide die ODBC-Verknüpfung **tblKategorien1** referenzieren.

Beide Formulare öffnen Sie und platzieren diese so, dass beide Formulare sichtbar sind. Dann ändern Sie den Datensatz im ersten Formular, speichern die Änderung aber noch nicht. Hier setzen wir den Wert des Feldes **Kategorienname** auf **Kategorie 2**. Achten Sie darauf, dass der Datensatzmarkierer das Bearbeiten-Symbol anzeigt (siehe Bild 7).

Nun ändern Sie den Wert des gleichen Feldes im zweiten Formular auf **Kategorie 3** und speichern die Änderung dort durch einen Klick auf den Datensatzmarkierer (siehe Bild 8). Schließlich versuchen Sie im Anschluss, auch die Änderungen im ersten Formular per Klick auf den Datensatzmarkierer zu speichern. In diesem Moment erscheint die oben vorgestellte Meldung.

Die Meldung hält zwei Optionen bereit:

- **In Zwischenablage kopieren:** Aktualisiert den Datensatz im Formular, wodurch die im zweiten Formular geänderte Version angezeigt wird. Die Version des Datensatzes, die im ersten Formular gespeichert wurde,

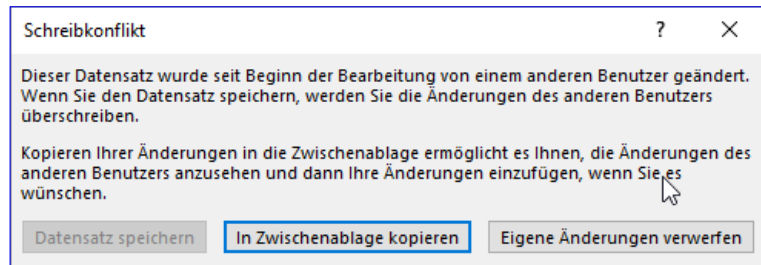


Bild 6: Fehlermeldung bei Schreibkonflikt

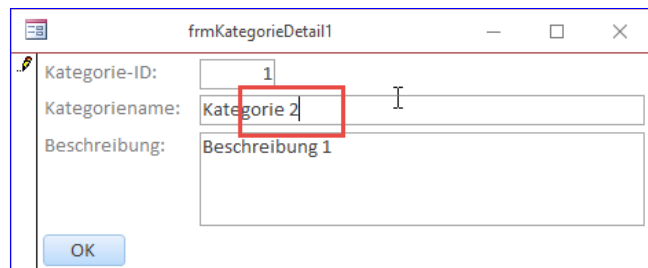


Bild 7: Ändern des Datensatzes im ersten Formular



Bild 8: Ändern und Speichern des Datensatzes im zweiten Formular

landet dann in der Zwischenablage. Von dort können Sie diesen dann über den aktuellen Datensatz schreiben und doch noch speichern.

- **Eigene Änderungen verwerfen:** Diese Option verwirft einfach die Änderungen im ersten Formular und aktualisiert dieses mit dem im zweiten Formular geänderten Datensatz.

Noch eine Fehlermeldung

Interessanterweise gibt es noch eine zweite Konstellation, unter der eine Fehlermeldung beim Ändern eines Datensatzes angezeigt wird – und zwar schon beim Versuch, auch nur den Inhalt eines Feldes zu bearbeiten. Um dies nachzustellen, verwenden Sie wieder die gleichen beiden Formulare **frmKategorien1** und **frmKategorien2**. Geben

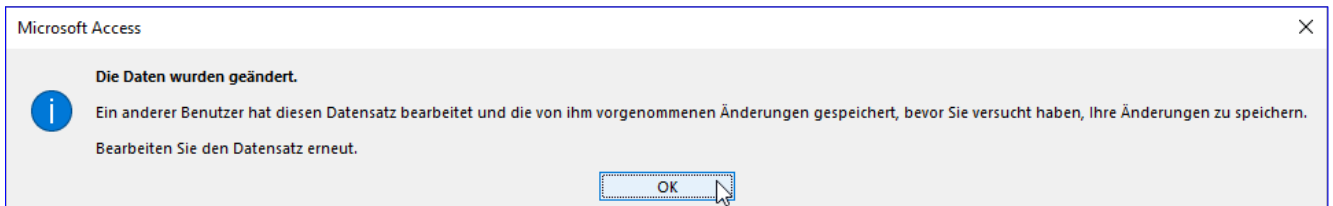


Bild 9: Fehlermeldung beim Versuch, einen bereits geänderten Datensatz von einem anderen Formular aus nochmals zu ändern

Sie nun in **frmKategorien1** für das Feld **Kategorienname** beispielsweise den Wert **Kategorie 0** ein. Speichern Sie den Datensatz diesmal direkt.

Anschließend wechseln Sie zum Formular **frmKategorien2** und versuchen, den Inhalt des Feldes **Kategorienname** anzupassen. Schon beim Eintippen des ersten Zeichens erscheint die Fehlermeldung aus Bild 9. Dies gilt übrigens für alle Steuerelemente des Formulars.

Wichtig: Um dies reproduzieren zu können, muss der im anderen Formular editierte und gespeicherte Datensatz bereits im ersten Formular angezeigt werden. Wenn Sie den Datensatz erst nach der Bearbeitung im anderen Formular laden oder diesen erst aktualisieren, können Sie ihn ohne Probleme bearbeiten.

Fehlernummer ermitteln

Aber woher kommt dieser Fehler? Wir wollen erst einmal die Fehlernummer ermitteln, um dann weitere Nachforschungen anstellen zu können. Fehler, die in Formularen auftreten, aber nicht per VBA ausgelöst werden, kommt man nicht auf herkömmliche Weise auf die Spur – also indem man einfach die Fehlernummer per **Err.Number**

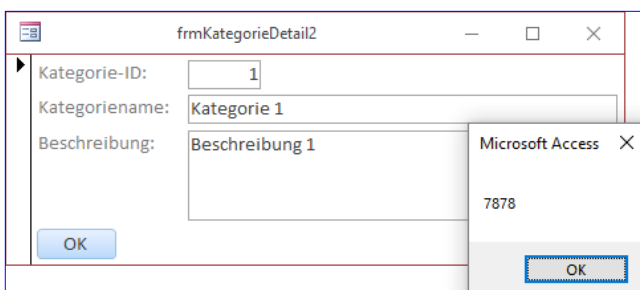


Bild 10: Ausgabe der Fehlernummer bei Reproduzierung des zweiten Fehlers

abfragt. Stattdessen müssen wir in diesem Fall die Ereignisprozedur **Form_Error** bemühen, die wir für die Eigenschaft **Bei Fehler** des Formulars anlegen (in diesem Fall nur für das Formular **frmKategorien2**).

Die Ereignisprozedur stellen wir zunächst nur mit einer einzigen Anweisung aus:

```
Private Sub Form_Error(DataErr As Integer, 7  
                        Response As Integer)  
    MsgBox DataErr  
End Sub
```

Danach führen wir die oben angegebenen Schritte erneut aus. Dies liefert die Meldung aus Bild 10 mit der Nummer **7878**.

Wenn wir schon einmal dabei sind, können wir auch gleich die Fehlernummer des weiter oben ausgelösten Fehlers ermitteln. Dieser lautet, wie Bild 11 zeigt, nur etwas anders, nämlich **7787**.

Um die Fehlertexte der beiden Fehler zu ermitteln, können wir die Funktion **AccessError** im Direktbereich des

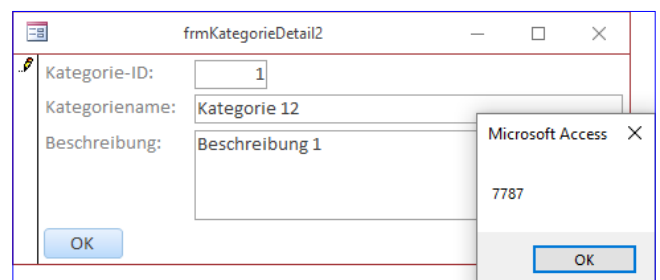


Bild 11: Ausgabe der Fehlernummer bei Reproduzierung des ersten Fehlers

```
Private Sub Form_Error(DataErr As Integer, Response As Integer)
    Select Case DataErr
        Case 7878
            MsgBox "7878"
        Case 7787
            MsgBox "Der Datensatz wurde in der Zwischenzeit an anderer Stelle geändert." & vbCrLf & vbCrLf _
                & "Ihre Änderungen werden verworfen, aber in der Zwischenablage gespeichert." & vbCrLf & vbCrLf _
                & "Um Ihre Version zu übernehmen, markieren Sie den Datensatz und drücken Sie Strg + V."
            Response = acDataErrContinue
        Case Else
    End Select
End Sub
```

Listing 1: Neue Fehlermeldung für den Fehler mit der Nummer 7787

VBA-Fensters absetzen. Für den Fehler **7878** ergibt sich Folgendes:

? AccessError(7878)

Die Daten wurden geändert.@Ein anderer Benutzer hat diesen Datensatz bearbeitet und die von ihm vorgenommenen Änderungen gespeichert, bevor Sie versucht haben, Ihre Änderungen zu speichern.@Bearbeiten Sie den Datensatz erneut.@1@@@1

Für den Fehler **7787** liefert diese Funktion interessanterweise keine Fehlermeldung.

Fehlermeldungen durch benutzerdefinierte Fehlermeldung ersetzen

Wenn Sie denken, dass die Benutzer Ihrer Datenbank Anwendung nicht mit der vom System erzeugten Meldung arbeiten können, lässt sich diese natürlich auch durch eine benutzerdefinierte Meldung ersetzen.

Für den Fehler **7787**, der beim zwischenzeitlichen Ändern des gleichen Daten-

satzes an anderer Stelle ausgelöst wird, erledigen wir dies wie in Listing 1.

Dabei prüfen wir in einer **Select Case**-Bedingung den Wert des Parameters **DataErr**. Lautet dieser auf **7787**, zeigen wir eine Meldung an, die den Benutzer über das Problem und die Möglichkeiten aufklärt (siehe Bild 12). Aber wo sind hier im Vergleich zur Systemmeldung die beiden Auswahlmöglichkeiten? Ganz einfach: Die haben wir herausgekürzt. In diesem Fall haben wir nämlich nicht die Möglichkeit, individuell auf die Benutzereingabe zu reagieren. Wie aber können wir dem Benutzer die Möglichkeit offen halten, dass er trotz der übernommenen Änderungen durch einen anderen Benutzer dennoch seine eigenen Änderungen durchsetzen kann? Und geht

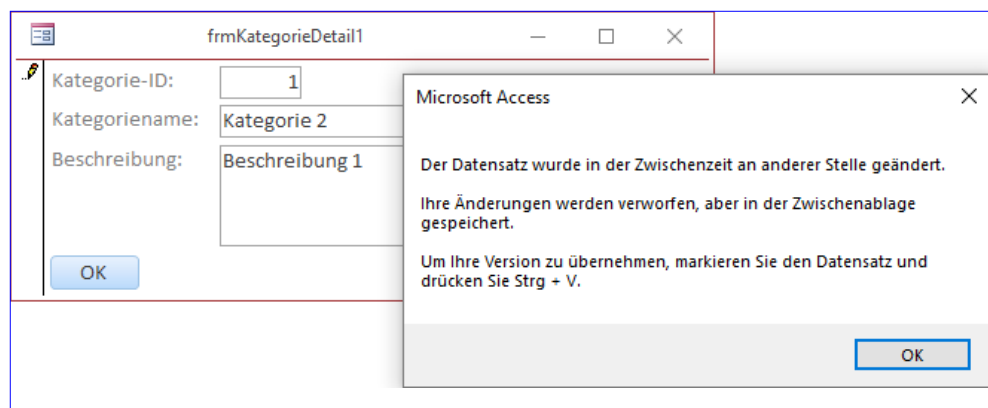


Bild 12: Neue, benutzerdefinierte Meldung beim Versuch, einen Datensatz zu ändern, der zwischenzeitlich an anderer Stelle geändert wurde.

Access und SQL Server im Web

Wenn Sie die Daten Ihrer Access-Datenbank über das Internet für verschiedene Access-Frontends verfügbar machen wollen, müssen Sie zwei Schritte erledigen: Der erste ist die Migration der Tabellen in eine SQL Server-Datenbank, der zweite die Bereitstellung dieser Datenbank über das Internet. Letzteres gelingt mit den Azure-Diensten von Microsoft. Wie das funktioniert, zeigt der vorliegende Beitrag im Detail.

Wir wollen die Dienste von Microsoft in Form von Microsoft Azure anhand unserer Beispieldatenbank **Suedsturm_SQL** ausprobieren. Als Erstes wollen wir allerdings ein Azure-Konto bei Microsoft einrichten. Dazu besuchen Sie die Seite azure.microsoft.com. Hier können Sie in der aktuellen Version über den Link **Jetzt kostenlos einsteigen** zunächst einmal experimentieren (siehe Bild 1).

Microsoft-Konto?

An dieser Stelle stellt sich die Frage, ob Sie bereits ein Microsoft-Konto besitzen. Wenn Sie bereits mit Office 365 arbeiten, stehen die Chancen gut. Falls nicht, gehen Sie zu login.live.com und geben einfach Ihre E-Mail-Adresse ein. Falls es noch kein passendes Konto gibt, können Sie dort eines erstellen.

Azure-Konto erstellen

Ist dies geschehen, kehren Sie zu azure.microsoft.com zurück und klicken auf **Jetzt kostenlos einsteigen**. Nun werden Sie auf eine Seite weitergeleitet, auf der Sie Ihre Konten bei Microsoft zur Auswahl vorfinden (siehe Bild 2).

Haben Sie das Konto ausgewählt, für das Sie das Microsoft Azure-Konto anlegen möchten, landen Sie auf einer weiteren Seite, die ein paar Informationen abfragt

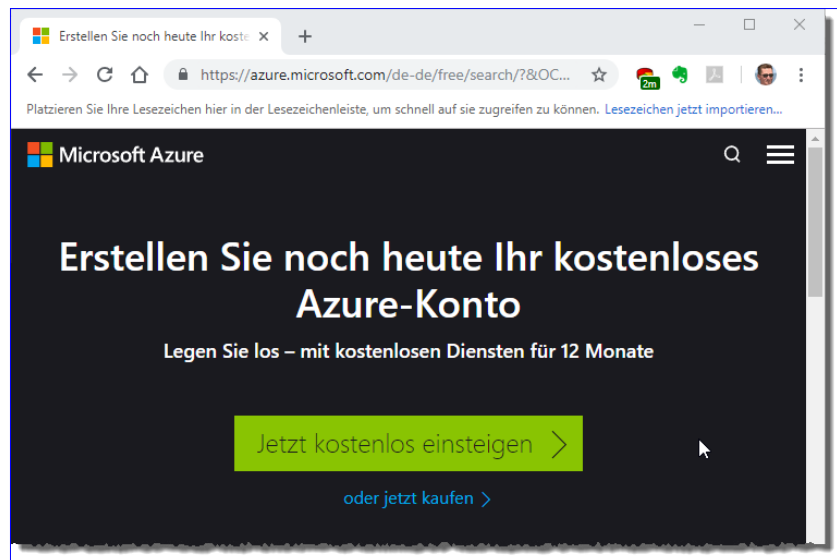


Bild 1: Einstieg in Azure

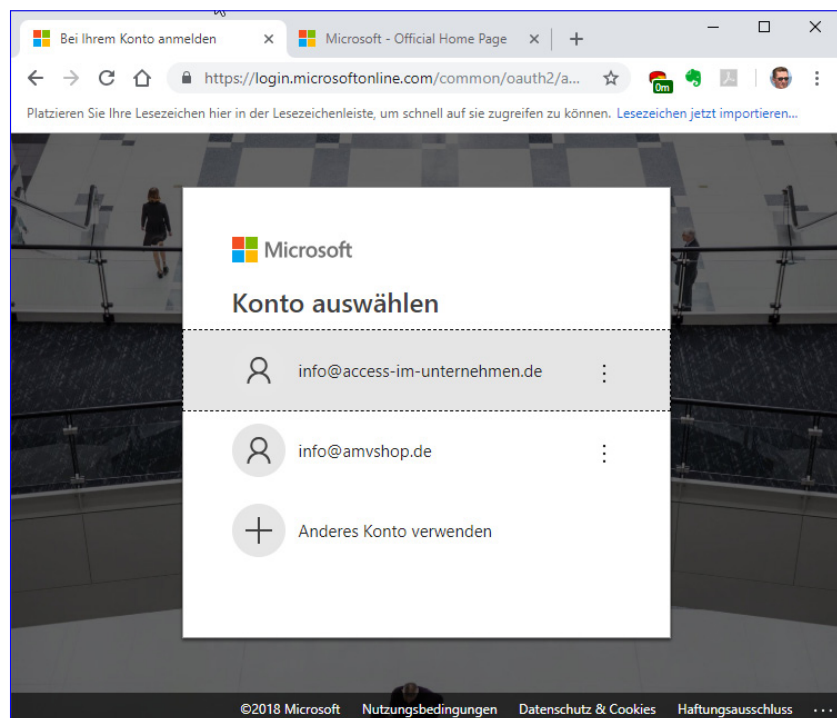


Bild 2: Auswählen des gewünschten Kontos

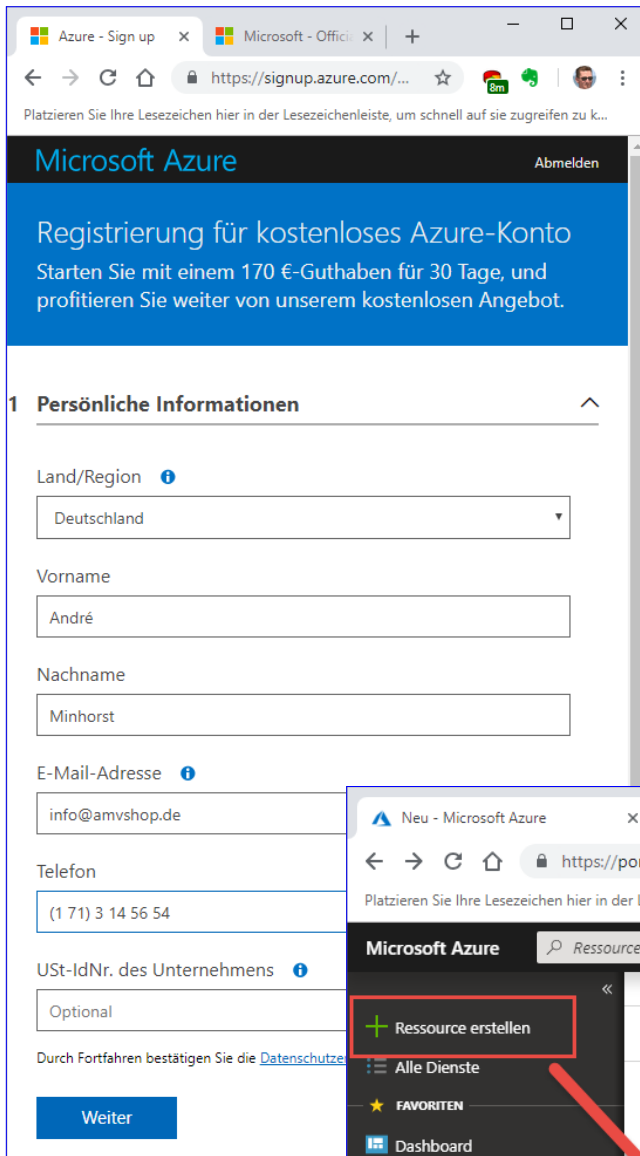


Bild 3: Angabe einiger Informationen

(siehe Bild 3). Im nächsten Schritt überprüft Microsoft Ihre Identität beispielsweise durch das Senden einer Nachricht an Ihre Mobilfunknummer, die einen Prüfcode enthält. Diesen geben Sie dann im nächsten Dialog ein, um Ihre Identität zu bestätigen.

Anschließend benötigt Microsoft noch die Daten einer Kreditkarte. Nachdem Sie diese angegeben haben und noch dem Abonnementvertrag, den Angebotsdetails und der Datenschutzerklärung zugestimmt haben und auf Registrieren geklickt haben, ist Ihr Konto angelegt.

Ressource für die Datenbank erstellen

Danach gelangen Sie in das Azure-Portal für Ihr Konto. Hier finden Sie links oben den Eintrag **Ressource erstellen**. Klicken Sie diesen Eintrag an, erscheint eine Liste der möglichen Elemente (siehe Bild 4). Hier wählen Sie den Eintrag **SQL Database** aus.

Danach finden Sie ein neues Element vor, für das Sie einige Daten eingeben müssen. Als Name legen wir **Suedsturm_SQL** fest. Als Abonnement wählen Sie Ihr kostenloses Abonnement aus – im Screenshot sehen Sie den Eintrag **Nutzungsbasierte Bezahlung** (mein Konto ist

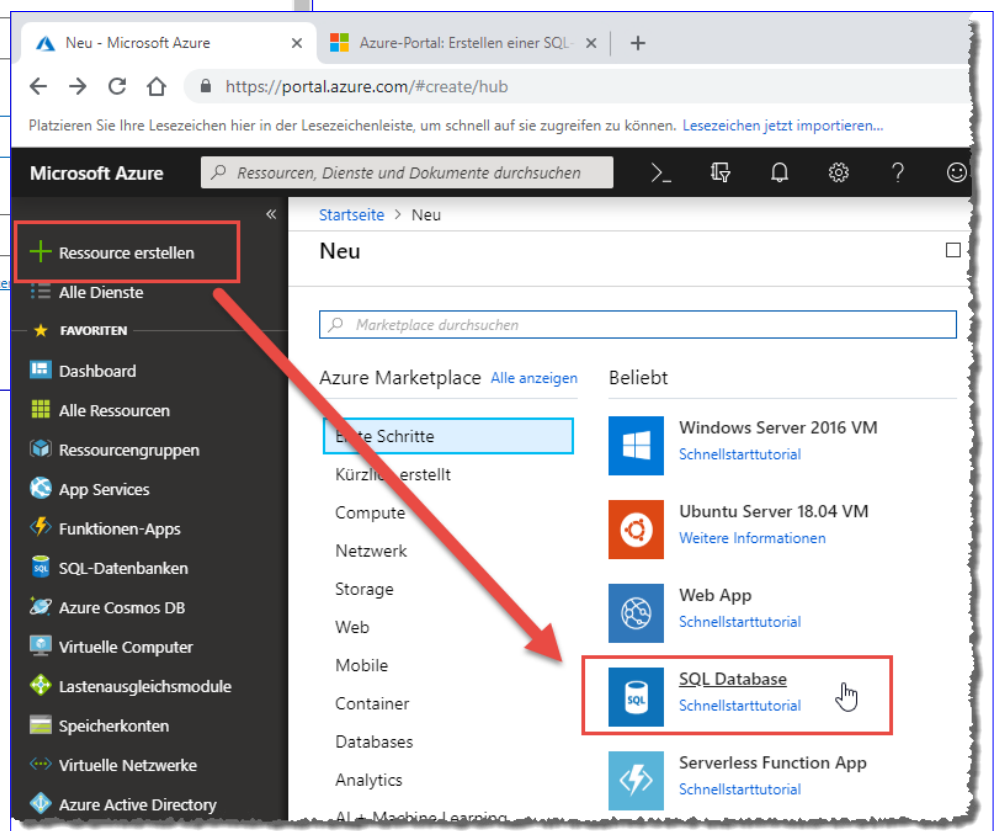


Bild 4: Hinzufügen einer neuen Ressource

schon aus der Testphase entwachsen ...). Wir legen außerdem eine neue Ressourcengruppe namens **AccessBackends** fest (siehe Bild 5). Unter **Quelle auswählen** behalten wir den Eintrag **Leere Datenbank** bei.

Server erstellen

Nun benötigen wir auch noch einen Server für die Datenbank. Diesen erstellen Sie, indem Sie auf den Eintrag **Server - Erforderliche Einstellung konfigurieren** klicken. Dadurch öffnet sich rechts ein Bereich mit der Überschrift **Server**.

Hier klicken Sie auf **Neuen Server erstellen**. Daraufhin erscheinen rechts die Textfelder zur Eingabe der Eigenschaften des Servers – also der Servername, der Name der Anmeldung und das Kennwort mit Bestätigung (siehe Bild 6). Den Server nennen wir **accessbackends**.

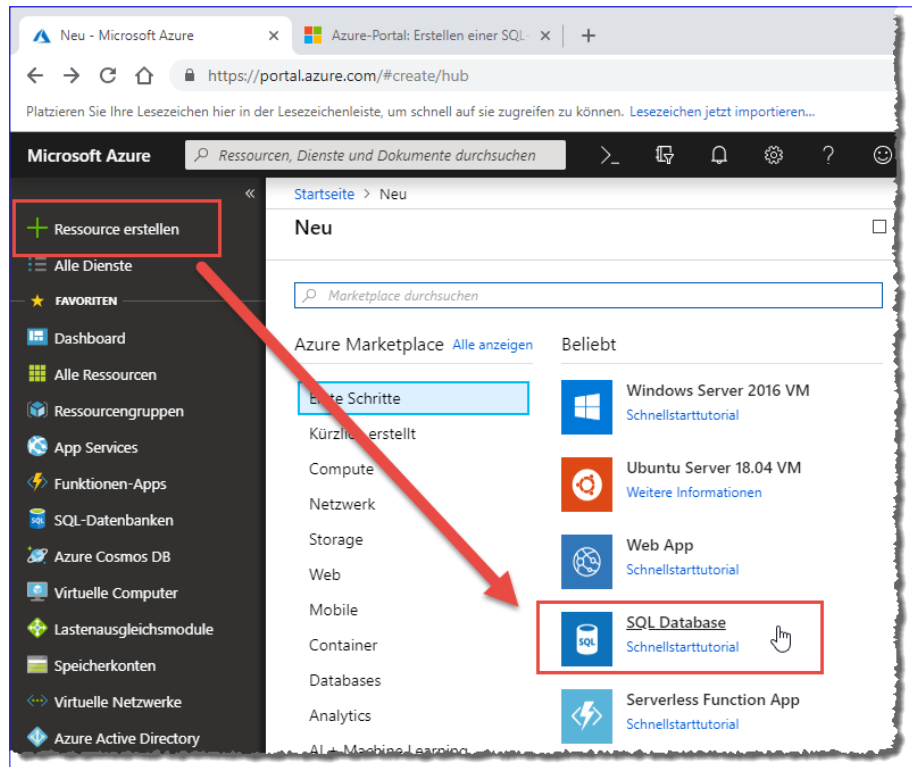


Bild 5: Anlegen einer neuen SQL-Datenbank

Nach der Eingabe der Informationen klicken Sie unten auf **Auswählen**. Daraufhin verschwinden die beiden Bereiche **Server** und **Neuer Server** wieder und wir können mit der Eingabe der Daten in der linken Spalte fortfahren.

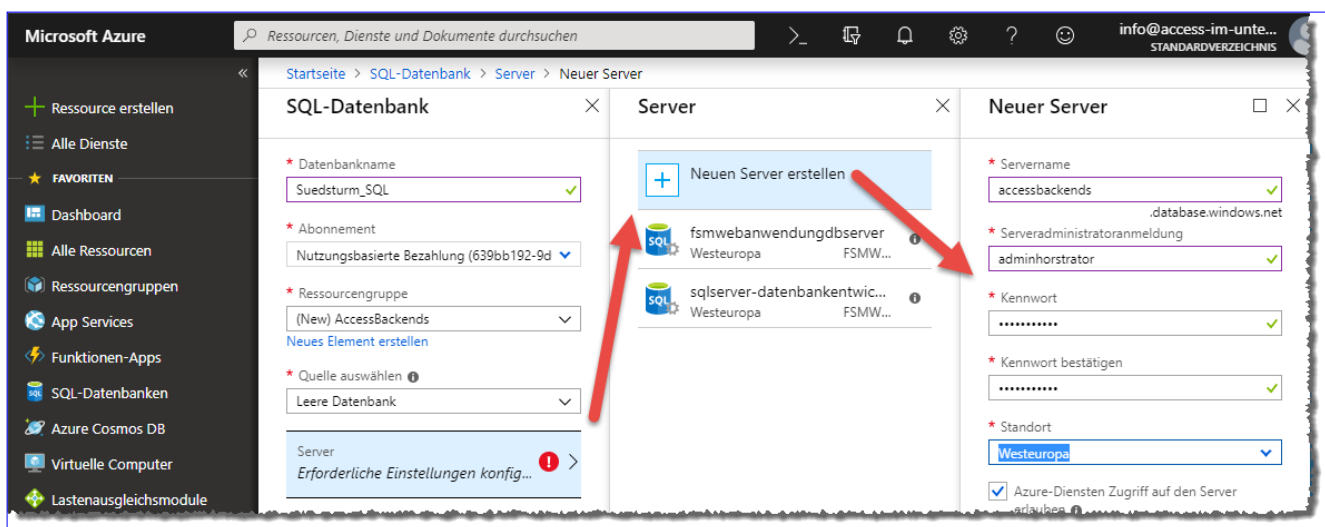


Bild 6: Anlegen eines neuen SQL Servers

Wenn Sie soeben das erste Mal ein Azure-Konto angelegt haben und noch in den Genuss des kostenlosen Testmonats kommen, können Sie nun einfach auf **Erstellen** klicken. Anderenfalls schauen Sie sich auf der rechten Seite die voraussichtlichen Kosten für den Betrieb der SQL Server-Datenbank an und wählen gegebenenfalls eine kleinere oder größere Konfiguration aus. Ich wähle für dieses Beispiel, da meine Testphase bereits vorbei ist, die **Basic**-Variante für weniger anspruchsvolle Workloads aus, die aktuell 4,21 EUR pro Monat kostet.

Auf der linken Seite sehen wir nun, dass die Datenbank maximal zwei Gigabyte groß werden darf. Die nächstgrößere Variante etwa würde ab 12,65 EUR pro Monat kosten und bietet bereits bis zu 250 GB Speicherplatz.

Nachdem wir die gewünschte Variante ausgewählt haben, können wir unten im Bereich **SQL-Datenbank** auf **Erstellen** klicken (siehe Bild 7). Die Webseite arbeitet nun wenige Sekunden und zeigt dann wieder die Übersicht von Microsoft Azure an.

Hier finden wir nun unter **Alle Ressourcen** auch die beiden Elemente, die wir soeben angelegt haben – also den SQL Server namens **accessbackends** sowie die Datenbank namens **Suedsturm_SQL** (siehe Bild 8).

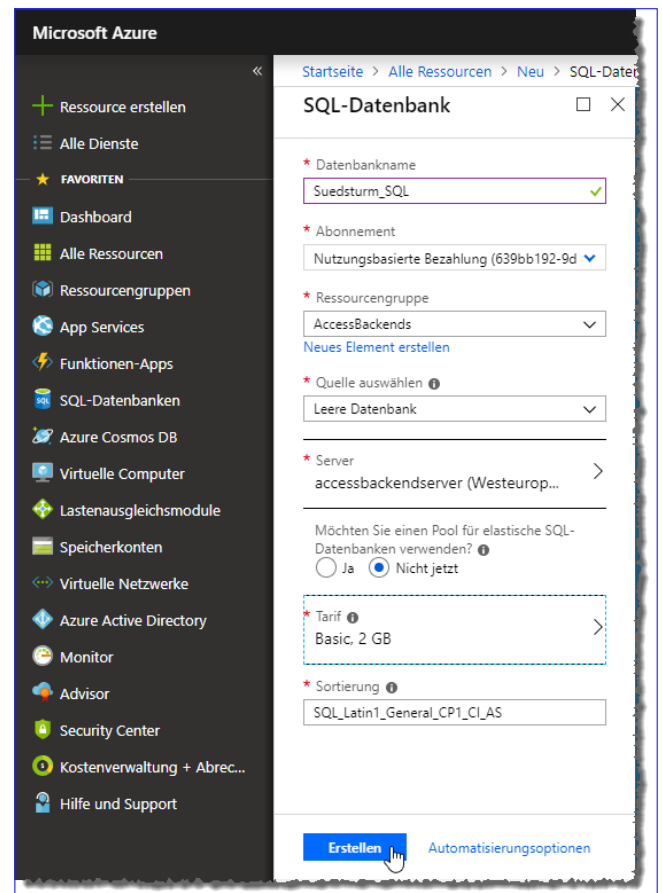


Bild 7: Abschluss der Erstellung der Datenbank

Details der Datenbank

Klicken Sie nun auf die Datenbank **Suedsturm_SQL**, zeigt Microsoft Azure die Übersicht über die Datenbank an (siehe Bild 9).

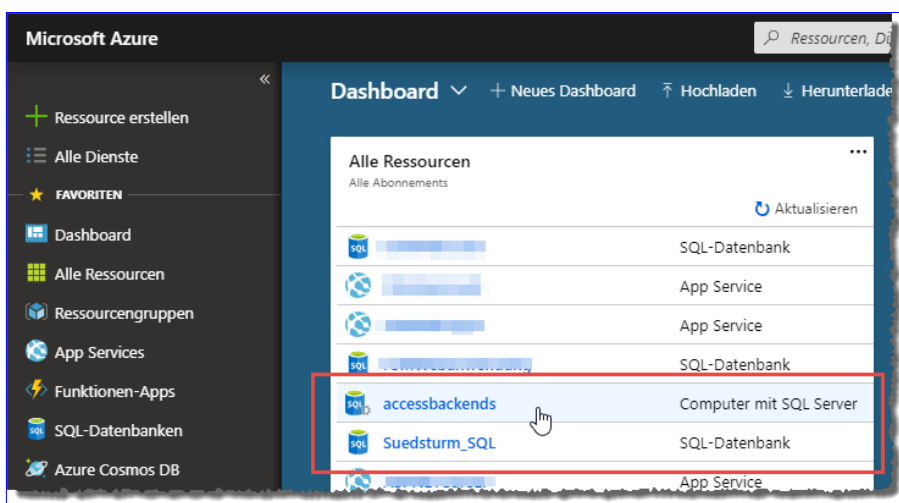


Bild 8: Der SQL Server und die Datenbank

Verbindungszeichenfolge ermitteln

Da wir mit Access zunächst per ODBC auf die Azure-Datenbank zugreifen wollen, interessiert uns natürlich die Verbindungszeichenfolge. Diese erhalten wir, indem wir links auf den Menüeintrag **Verbindungszeichenfolgen** klicken. Hier finden wir vier Registerreiter vor, hinter denen sich die Verbindungszeichenfolgen für

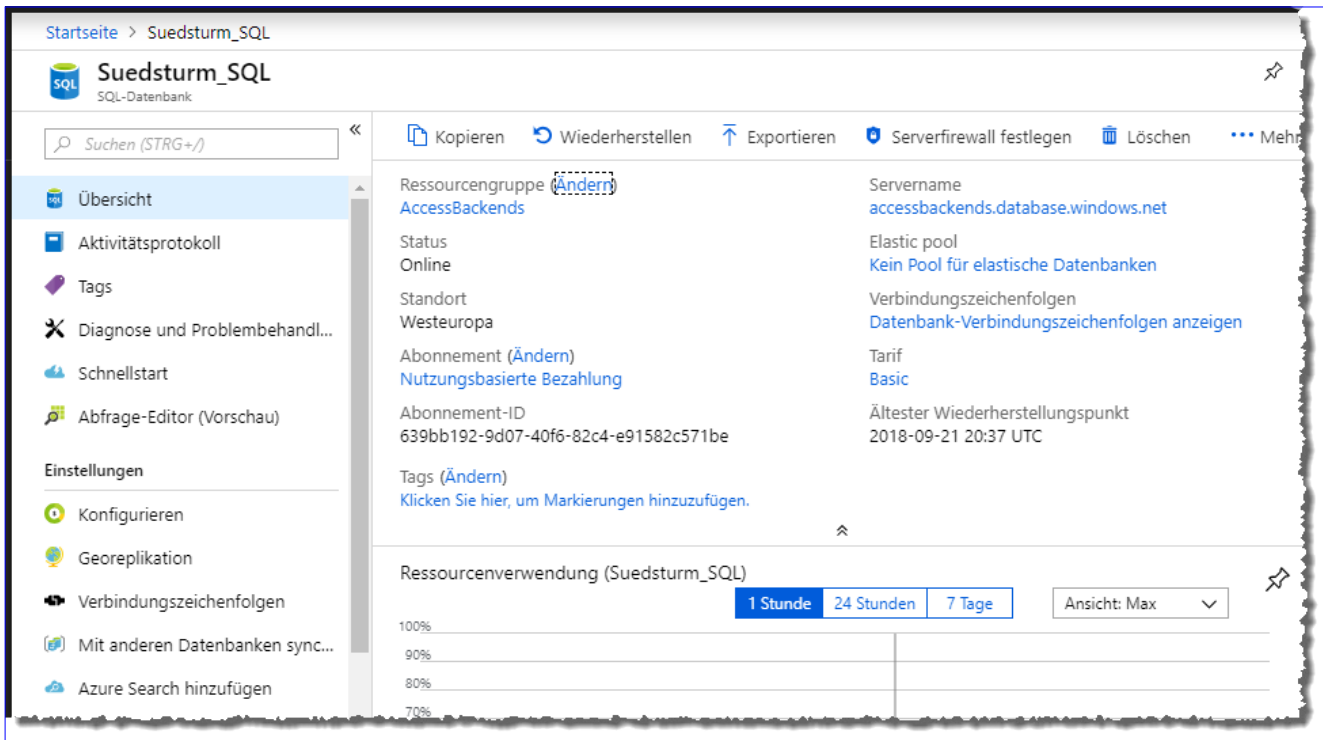


Bild 9: Eigenschaften des SQL Servers

ADO.NET, JDBC, ODBC und PHP verbergen. Die für uns interessante Verbindungszeichenfolge für ODBC sieht schließlich wie in Bild 10 aus.

Bearbeiten der Datenbank

Azure bietet rudimentäre Mittel, um den Entwurf der Datenbank zu bearbeiten und zum Beispiel Tabellen zu erstellen. Dazu klicken Sie links auf den Eintrag **Abfrage-Editor (Vorschau)**. Dies liefert eine arg abgespeckte Variante des Abfrage-Editors des SQL Server Management Studios (siehe Bild 11). Hier kann man mal kleine Aktionen durchführen, aber unsere Datenbank wollen wir dann doch im SQL Server Management Studio erstellen.

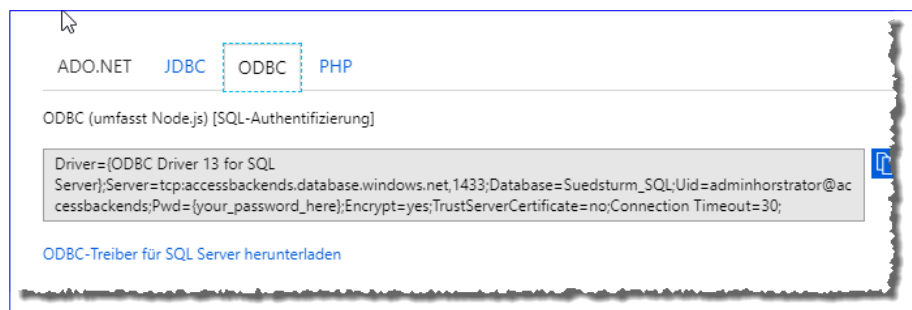


Bild 10: Verbindungszeichenfolge für den Zugriff über ODBC

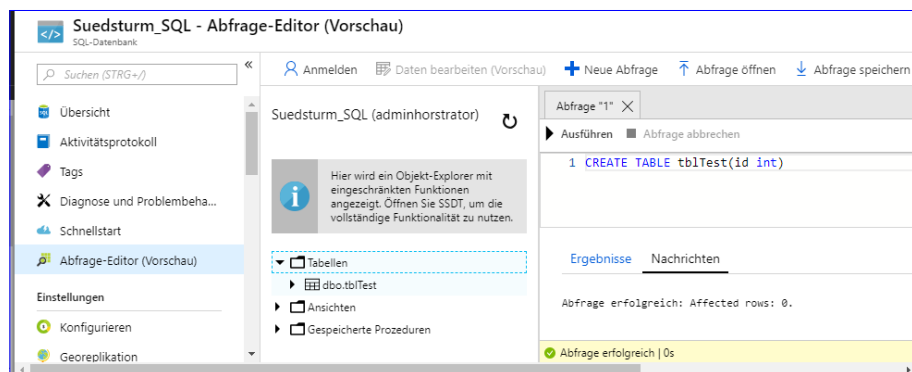


Bild 11: Bearbeiten der Datenbank, hier Anlegen einer kleinen Beispieldatenbank

Berechtigungen für Access-Objekte per SQL Server 1: Datenmodell

Im Beitrag »SQL Server: Sicherheit und Benutzerverwaltung« haben wir gezeigt, wie Sie die Sicherheitsfunktionen für den Zugriff auf SQL Server und SQL Server-Datenbanken für Benutzergruppen und Benutzer einrichten. Damit ist der Zugriff auf die Elemente des SQL Servers geregelt. Was aber ist mit den Objekten im Access-Frontend – also beispielsweise mit Formularen, Schaltflächen, Ribbon-Einträgen und so weiter? Wenn wir schon Benutzer und Benutzergruppen im SQL Server einrichten, möchten wir diese ja auch nutzen, um die Elemente der Benutzeroberfläche je nach der Anmeldung für Benutzergruppe oder Benutzer freizugeben oder auch nicht. Ein Beispiel dafür, wie Sie dies realisieren können, liefert der vorliegende Beitrag. In diesem ersten Teil erstellen wir das Datenmodell für die Berechtigungsverwaltung.

Mit dem Anlegen von Anmeldungen für Benutzer und Benutzergruppen auf Windows-Basis oder für Benutzer auf SQL-Server-Basis, deren Zuweisung an die verschiedenen Datenbanken und dem Einstellen der Berechtigungen für die Elemente dieser Datenbanken haben Sie schon einen großen Schritt getan: Sie haben dafür gesorgt, dass jeder Benutzer nur auf die Daten zugreifen darf, die für die jeweilige Anmeldung freigegeben sind.

Nun besitzt eine moderne Access-Anwendung ein Ribbon, mit dem sich die einzelnen Funktionen der Anwendung aufrufen lassen. Normalerweise sind alle Befehle des Ribbons aktiviert und sichtbar. Wenn Sie eine Anwendung erstellen, die von verschiedenen Benutzergruppen genutzt wird, wollen Sie aber möglicherweise nicht immer alle Funktionen für alle Benutzergruppen freischalten. So könnte ein Datenbank-Frontend gleichzeitig die Formulare, Berichte und Funktionen enthalten, die nur für die Bestellannahme erforderlich sind. Die dortigen Mitarbeiter sollen aber vielleicht nicht die Auswertungen sehen, die etwa für das Management interessant sind. Also wollen wir verschiedene Dinge erreichen: Erstens sollen Bereiche, die für bestimmte Benutzergruppen nicht vorgesehen sind, nicht zu öffnen sein. Zweitens sollen auch gar keine Steuerelemente

im Ribbon oder auch in Formularen vorhanden sein, um diese Bereiche zu öffnen – oder zumindest sollen diese deaktiviert werden.

Berechtigungen je Benutzergruppe

Die folgende Lösung orientiert sich an der Windows-Authentifizierung, wie wir sie im Beitrag **SQL Server: Sicherheit und Benutzerverwaltung** (www.access-im-unternehmen.de/1154) vorgestellt haben. Das heißt, dass wir die Berechtigungen mit der Benutzergruppen abgleichen, denen die jeweiligen Benutzer angehören.

Im Einzelnen soll der Ablauf so aussehen, dass der Benutzer sich über seine Windows-Anmeldung und die zugeordnete Benutzergruppe an den SQL Server anmeldet. Dort haben wir bereits ausschließlich Anmeldungen auf Basis der vorhandenen Benutzergruppen vorgesehen. Wenn nun das Ribbon für einen Benutzer geladen wird, soll die Anwendung in einer Tabelle auf dem SQL Server prüfen, ob die Benutzergruppe, welcher der Benutzer angehört, die Berechtigungen zur Anzeige der Steuerelemente des Ribbons besitzt. Dazu verwenden wir drei Tabellen. Die erste enthält alle Ribbon-Steuerelemente, Formulare, Formular-Steuerelemente und Berichte, für die Berechtigungen geprüft werden sollen. Die zweite enthält die verschiede-

nen Berechtigungsstufen, in diesem Fall Keine, **Lesen** und **Alle**. Damit sollten alle wichtigen Fälle abgedeckt sein. Sie können diese Tabelle aber nach Bedarf selbst erweitern. Die dritte Tabelle ist eine Verknüpfungstabelle zwischen den beiden ersten Tabellen und legt für jedes in der ersten Tabelle aufgeführte Element die Berechtigung aus der zweiten Tabelle fest.

Nun benötigen wir noch auf Access-Seite entsprechende Prozeduren, die vor dem Anzeigen eines Ribbons, eines Formulars oder eines Berichts aufgerufen werden und für die jeweiligen Elemente prüfen, ob die Benutzergruppe des aktuellen Benutzers diese Elemente nutzen darf. Voraussetzung für die Sicherheit dieser Vorgehensweise ist, dass der Benutzer den Quellcode in der Access-Anwendung nicht einsehen kann.

Benutzergruppen und Benutzer unter Windows anlegen

Damit Sie die folgenden Beispiele reproduzieren können, legen Sie zwei Benutzergruppen namens **Bestellannahme** und **Management** an sowie zwei Benutzer namens **Harry Klein** und **Peter Gross**. Diese ordnen Sie jeweils einer Benutzergruppe zu. Dabei brauchen Sie nicht umständlich in der Windows-Systemsteuerung zu arbeiten, sondern können dies bequem über die Kommandozeile erledigen. Diese öffnen Sie wie im Beitrag **SQL Server: Sicherheit und Benutzerverwaltung** im Administrator-Modus. Danach brauchen Sie nur noch die folgenden sechs Anweisungen einzugeben und per Eingabetaste auszuführen:

```
net.exe USER /ADD "Harry Klein" password
net.exe USER /ADD "Peter Gross" password
net.exe LOCALGROUP /ADD Bestellannahme
net.exe LOCALGROUP /ADD Management
net.exe LOCALGROUP Management /ADD "Peter Gross"
net.exe LOCALGROUP Bestellannahme /ADD "Harry Klein"
```

Dann legen Sie wie im gleichen Beitrag beschrieben zwei Anmeldungen im SQL Server Management Studio für die

beiden Benutzergruppen **Bestellannahme** und **Management** an. Das können Sie über den Dialog erledigen, aber Sie haben auch die Möglichkeit, schnell ein Skript zu schreiben (oder dieses hier zu kopieren – nicht vergessen, den Platzhalter **<Server>** zu ersetzen). Das folgende Skript legt eine neue Anmeldung für die Windows-Benutzergruppe **Management** an und stellt die Standarddatenbank auf **Suedsturm_SQL** ein. Dann erstellt das Skript einen neuen Benutzer für die Datenbank **Suedsturm_SQL** für die Anmeldung der Benutzergruppe **Management**. Das Skript führen Sie im SQL Server Management Studio in einem neuen Abfragefenster für die Datenbank **Suedsturm_SQL** aus:

```
USE [master]
GO
CREATE LOGIN [<Server>\Management] FROM WINDOWS WITH DE-
FAULT_DATABASE=[Suedsturm_SQL]
GO
USE [Suedsturm_SQL]
GO
CREATE USER [<Server>\Management] FOR LOGIN [<Server>\
Management]
GO
ALTER ROLE [db_datareader] ADD MEMBER [<Server>\Manage-
ment]
GO
ALTER ROLE [db_datawriter] ADD MEMBER [<Server>\Manage-
ment]
GO
```

Das gleiche Skript können Sie noch einmal in leicht geänderter Form ausführen. Diesmal verwenden wir die Anmeldung **<Server>\Bestellannahme**.

Berechtigungen einstellen

Danach legen wir einige Berechtigungen für die beiden Benutzergruppen fest. Die Benutzergruppe **Management** soll alle Daten lesen können, aber nicht ändern. Nicht, dass die Manager was kaputtmachen ... Die Benutzergruppe **Bestellverwaltung** soll einige Tabellen nur lesen

können – zum Beispiel **tblAnreden**, **tblArtikel** oder **tblKategorien** zum Beispiel. Andere Tabellen müssen die Mitglieder dieser Benutzergruppe natürlich auch bearbeiten können – zum Beispiel **tblBestellungen** oder **tblBestelldetails**. Wenn Sie keine Lust haben, in den Dialogen zu arbeiten, können Sie auch hier einige Anweisungen absetzen, was vielleicht übersichtlicher ist – zumal Sie in den Dialogen immer nur die Berechtigungen für eine Tabelle gleichzeitig festlegen können. Um beispielsweise Leserechte für die Tabelle **tblArtikel** für die Benutzergruppe Bestellannahme festzulegen, reicht in einem neuen Abfragefenster der Datenbank **Suedsturm_SQL** die folgende Anweisung:

```
GRANT SELECT ON [dbo].[tblArtikel] TO [<Server>\Bestell-
annahme]
GO
```

Um das Lesen, Anlegen, Ändern oder Löschen von Datensätzen in der Tabelle **tblBestellungen** festzulegen, verwenden Sie diese Anweisungen:

```
GRANT UPDATE ON [dbo].[tblBestellungen] TO [<Server>\Be-
stellannahme]
GO
GRANT SELECT ON [dbo].[tblBestellungen] TO [<Server>\Be-
stellannahme]
GO
GRANT INSERT ON [dbo].[tblBestellungen] TO [<Server>\Be-
stellannahme]
GO
GRANT DELETE ON [dbo].[tblBestellungen] TO [<Server>\Be-
stellannahme]
GO
```

Tabellen für die Berechtigungen der Benutzergruppen an den Objekten definieren

Als Nächstes benötigen wir die drei bereits weiter oben beschriebenen Tabellen, in denen wir die Elemente der

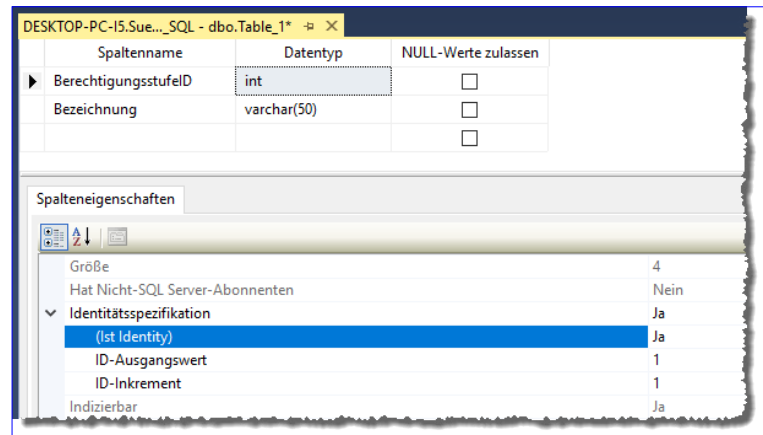


Bild 1: Anlegen der Tabelle **tblBerechtigungsstufen**

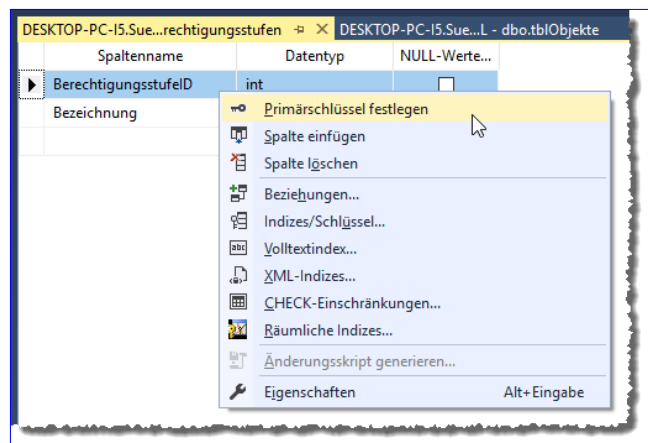


Bild 2: Festlegen des Primärschlüsselfeldes per Kontextmenü

Benutzeroberfläche, die Berechtigungsstufen und die Verknüpfungen zwischen diesen Elementen speichern.

Die Tabelle **tblBerechtigungsstufen** soll die Felder **BerechtigungsstufID** und **Bezeichnung** enthalten. Die Tabelle legen wir im SQL Server Management Studio wie in Bild 1 an. Um diesen Dialog zu öffnen, wählen Sie im Kontextmenü **Tabelle...** des Eintrags **Suedsturm_SQLTabellen** aus. Nachdem Sie die beiden Felder wie in der Abbildung hinzugefügt haben, legen Sie in den Eigenschaften noch die Identitätsspezifikation für das Feld **BerechtigungsstufID** fest. Auf diese Weise versehen Sie das Feld mit einem Autowert.

Außerdem müssen Sie das Feld **BerechtigungsstufID** noch als Primärschlüsselfeld markieren. Das erledigen Sie

PDF-Dokumente im Griff mit PDFtk

PDFtk ist ein kostenloses bis günstiges Tool, mit dem Sie verschiedene Manipulationen an PDF-Dateien vornehmen können. Damit ist es eine tolle Ergänzung zu Anwendungen, die ihre Dateien wie etwa Rechnungen oder anderen Dokumente per Bericht formatieren und dann als PDF speichern. Vor der Weiterverarbeitung etwa durch das Versenden per E-Mail können Sie die PDFs dann mit PDFtk optimieren, zum Beispiel in dem Sie diese mit einem Kennwort schützen. Dieser Beitrag zeigt, wie Sie PDFtk, das neben der grafischen Benutzeroberfläche auch einen Aufruf per Kommandozeile erlaubt, zum automatisierten Bearbeiten Ihrer PDFs per VBA nutzen.

PDFtk

PDFtk kommt in verschiedenen Versionen, zum Beispiel in der kostenlosen Version **PDFtk Free** und in der sehr günstigen Variante **PDFtk Pro**. **PDFtk Pro** kostet zum Zeitpunkt der Erstellung dieses Beitrags 3,99 \$, was ein echter Schnapper ist. Erst diese Version bietet etwa die Möglichkeit, PDF-Dokumente mit Wasserzeichen zu versehen oder diese per Kennwort zu schützen, daher wollen wir uns in diesem Beitrag mit dieser kostenpflichtigen Version beschäftigen.

Nach dem Kauf und dem Download installieren Sie die Anwendung, die Sie dann auch direkt starten können, um sich vorab einen Überblick über die Features in der Benutzeroberfläche zu verschaffen. Später schauen wir uns dann an, wie Sie die verschiedenen Funktionen von der Kommandozeile aus erledigen können – und somit auch per VBA.

PDF mit Wasserzeichen

Zum Einarbeiten in die Benutzeroberfläche schauen wir uns an, wie Sie ein

PDF-Dokument mit einem Wasserzeichen versehen – in diesem Fall, um einen Artikel im PDF-Format mit dem Text **Persönliche Kopie für André Minhorst** zu versehen. Viele Verlage, die E-Books verkaufen, machen dies so, um zu verhindern, dass die Käufer ihre E-Books weitergeben. Also wählen wir in der Benutzeroberfläche als Erstes die Datei aus, die wir mit dem Wasserzeichen versehen wollen. Diese können Sie entweder über die Schaltfläche **Add PDF...** und den damit zu öffnenden Dateiauswahl-Dialog

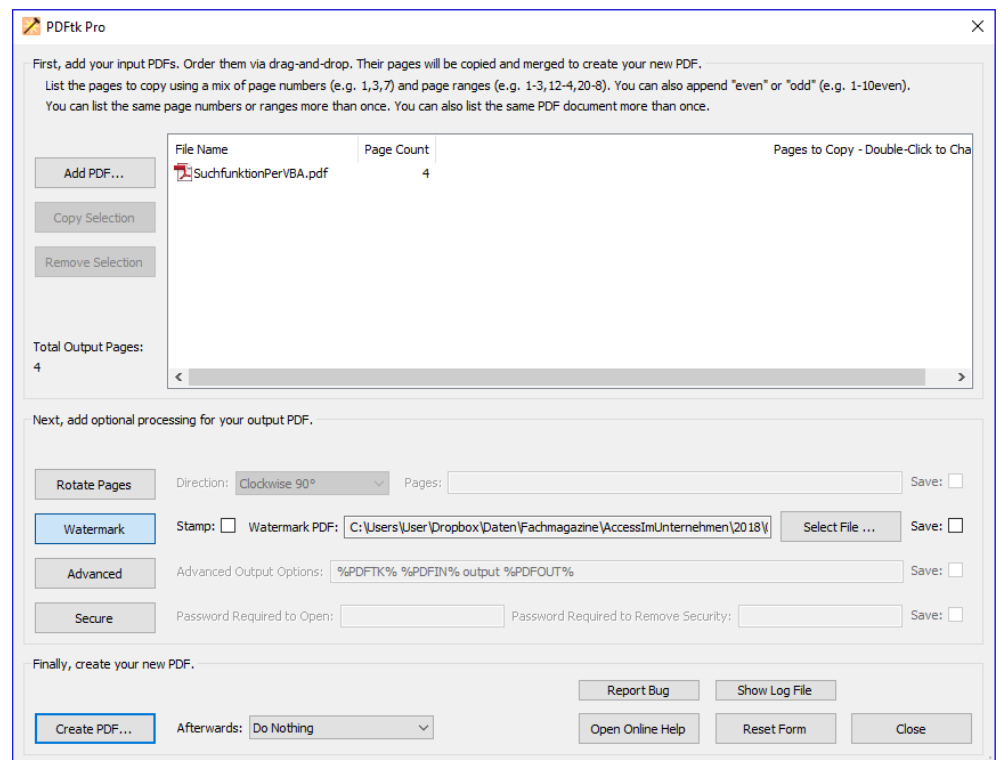


Bild 1: Die Benutzeroberfläche von PDFtk Pro

auswählen. Das Ergebnis ist in beiden Fällen, dass die zu bearbeitende PDF-Datei in der Liste der PDF-Dateien angezeigt wird (siehe Bild 1).

Nun benötigen wir eine Datei, die das Wasserzeichen – in diesem Fall einen Text – beisteuert. Dazu erstellen wir ein PDF, das den gewünschten Text einfach oben rechts enthält – dies können Sie zum Beispiel von Access aus erledigen (siehe Bild 2). Diese fügen wir über die Benutzeroberfläche von PDFtk Pro hinzu, indem wir die Schaltfläche **Watermark** betätigen und die Wasserzeichen-Datei auswählen. Danach brauchen Sie nur noch auf **Create PDF...** zu klicken und die Zieldatei festzulegen.

Das Ergebnis sieht dann wie in Bild 3 aus – das Wasserzeichen wurde wie gewünscht im oberen

Bereich integriert. Wenn das Wasserzeichen nicht sichtbar ist, weil sich im Originaldokument Elemente davor befinden, können Sie die Option **Stamp** aktivieren. Das Wasserzeichen wird übrigens zu jeder einzelnen Seite des Zieldokuments hinzugefügt.

PDFs zusammenführen

Auch praktisch ist die Funktion, mehrere PDF-Dokumente zu einem einzigen zusammenzuführen. Dazu fügen Sie ein-

fach alle benötigten Dokumente zu der Liste hinzu und klicken dann auf **Create PDF...** – die angegebenen Dokumente werden dann zu einem Dokument verschmolzen. Wenn Sie die Dokumente hinzugefügt haben und diese werden nicht in der gewünschten Reihenfolge angezeigt, passen Sie die Reihenfolge in der Liste per Drag and Drop an.

Hier kann man auch noch feiner arbeiten und zum Beispiel nur bestimmte Seiten der Dokumente berücksichtigen.

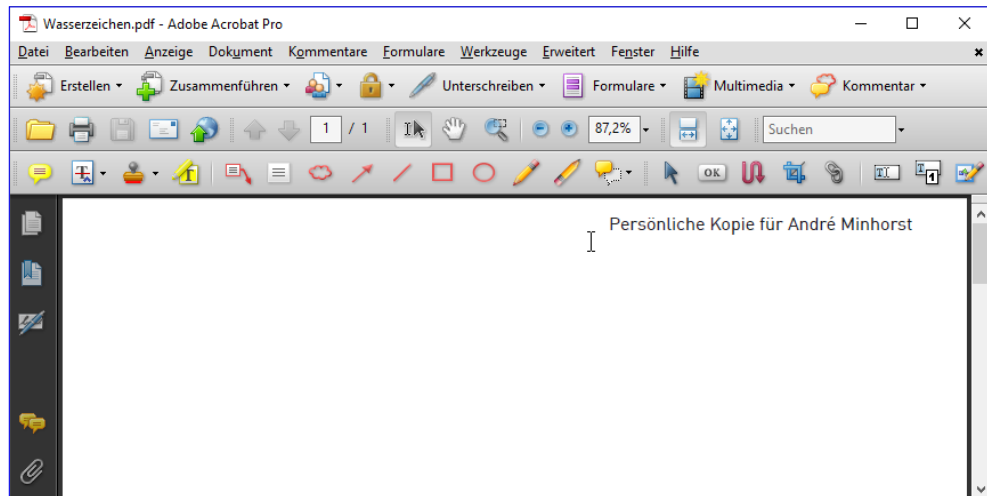


Bild 2: Dokument mit dem Wasserzeichen

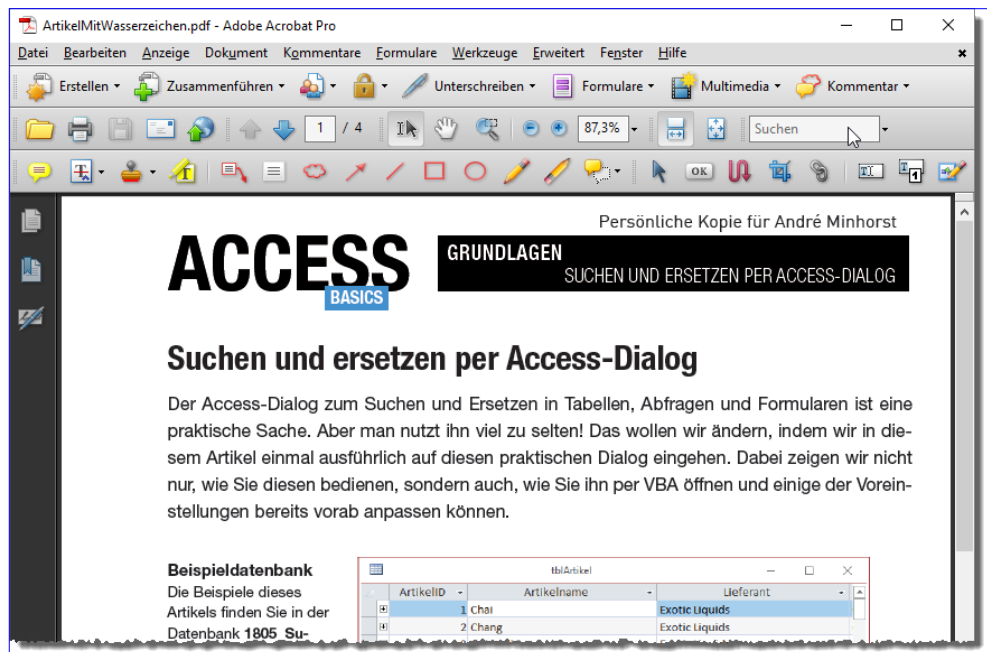


Bild 3: Original und Wasserzeichen in einer Datei

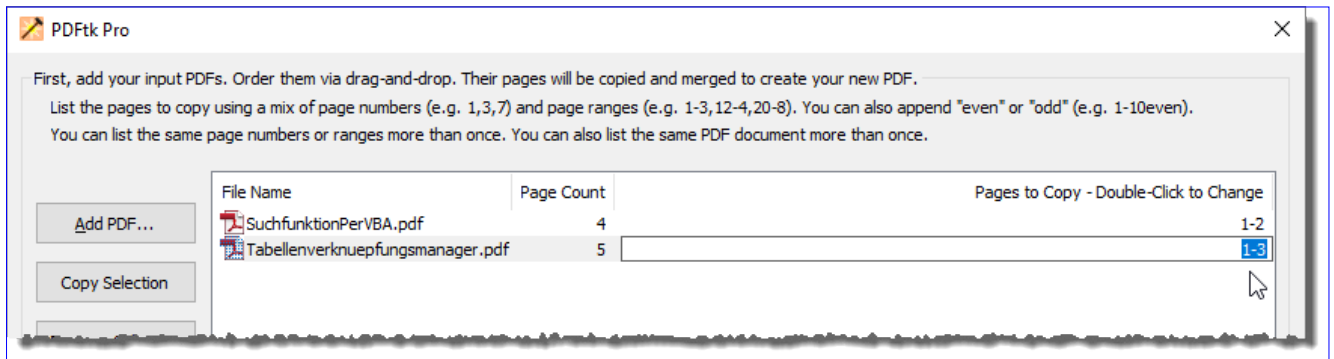


Bild 4: Anpassen der zu berücksichtigenden Seitenzahlen

Dazu klicken Sie doppelt auf den jeweiligen Eintrag und tragen dann für die Seitenzahlen im rechten Bereich der Liste die gewünschten Seiten ein – zum Beispiel:

- **1** für die erste Seite,
- **2-4** für die zweite bis vierte oder
- **1,4** für die erste und vierte (siehe Bild 4).
- Wenn Sie etwa nur die geraden oder die ungeraden Seiten berücksichtigen wollen, hängen Sie die Zeichenfolgen **even** (gerade Zahlen) oder **odd** (ungerade Zahlen) an die Angabe der Seitenzahlen an (zum Beispiel **1-4odd** für die ungeraden Seiten im Bereich der ersten bis vierten Seite).
- Von einer bestimmten Seite bis zum Ende definieren Sie zum Beispiel für alle Seiten ab der dritten mit **3-end**.
- Wenn Sie nur die hinteren paar Seiten berücksichtigen wollen, verwenden Sie das Präfix **r** vor der Seitenzahl.

r1 ist die letzte Seite, **r3** die drittletzte, **r2-r1** die letzten beiden.

Mit der Schaltfläche **Rotate Pages** können Sie angeben, welche Seiten gedreht werden sollen. Hier geben Sie die Seiten des gesamten Dokuments an, die gedreht werden sollen.

PDF mit Kennwort schützen

Wenn Sie ein PDF-Dokument mit einem Kennwort schützen wollen, klicken Sie auf die Schaltfläche **Secure**.

Damit aktivieren Sie die Textfelder **Password Required to Open** und **Password Required to Remove Security**.

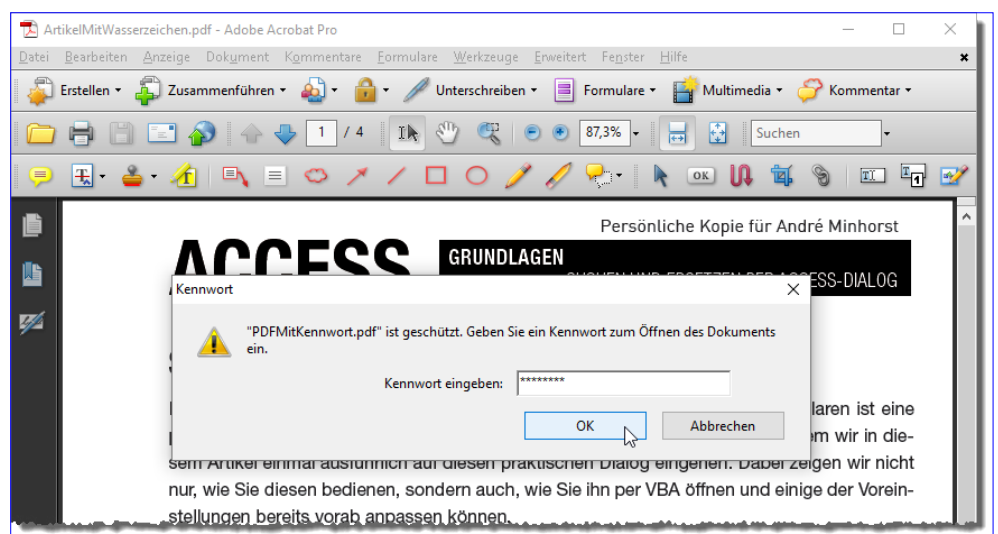


Bild 5: Beim Öffnen des PDFs wird ein Kennwort abgefragt.

VB.NET-DLL für Access programmieren

Visual Studio bietet viel mehr Bibliotheken und Möglichkeiten als Access. Dafür ist es auch viel komplizierter. Zum Glück können wir beide Welten kombinieren, zum Beispiel durch Erstellen einer DLL mit Visual Studio, die Funktionen bereitstellt, die nur unter .NET zur Verfügung stehen, die Sie dann von Access aus per VBA referenzieren und nutzen können. Dieser Beitrag zeigt anhand eines Beispiels, wie das gelingt.

Ohne Probleme möglich wird dies dadurch, dass Sie sich die Community Edition von Visual Studio, beispielsweise in der Version 2017, kostenlos herunterladen und diese nutzen können. Den Download finden Sie, wenn Sie bei Google nach **Visual Studio 2017 Community** suchen.

Nach dem Installieren und Starten von Visual Studio öffnen Sie über den Menüeintrag **DateiNeuProjekt...** den Dialog zum Erstellen eines neuen Projekts. Hier wählen Sie links den Eintrag **InstalliertVisual BasicWindows Desktop** aus und in der Mitte dann **Klassenbibliothek (.NET Framework)** – siehe Bild 1. Geben Sie den Namen des Projekts ein, hier **CMD**, und wählen Sie den Ordner aus, in dem das Projektverzeichnis erstellt werden soll.

DLL als Administrator erstellen

Ein Hinweis vorab: Wenn Sie die DLL erstellen möchten, müssen Sie Visual Studio als Administrator öffnen. Dazu geben Sie beispielsweise Visual Studio im Suchfeld von Windows ein, warten, bis der Eintrag Visual Studio 2017 auftaucht, klicken dann mit der rechten Maustaste auf diesen Eintrag und wählen aus dem nun erscheinenden

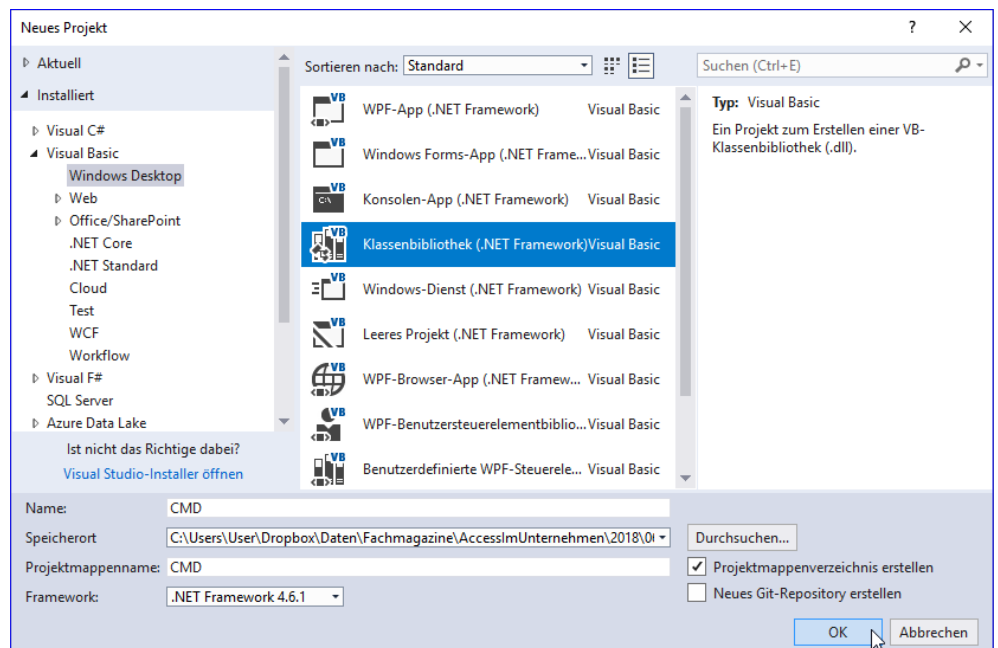


Bild 1: Erstellen einer Klassenbibliothek

Kontextmenü den Befehl **Als Administrator ausführen** aus. Mit einer so gestarteten Instanz von Visual Studio können Sie nun auch DLLs auf dem aktuellen Rechner erstellen und registrieren.

Visual Studio empfängt Sie nun mit der im Codefenster geöffneten Klasse **Class1**. Sie können hier nun erste Methoden eintragen.

Unser Ziel ist es, eine DLL zu erstellen, die Methoden einer Bibliothek, die nur unter .NET, nicht aber unter VBA zur Verfügung steht. Zu Testzwecken wollen wir zuerst einmal nur eine einfache Funktion über die DLL in VBA bereitstellen, welche nach dem Aufruf eine einfache Meldung anzeigt.

Wir ändern nun erst einmal den Namen der zu verwendenden Klasse, indem wir **Class1.vb** im Projektmappen-Explorer in **CMD.vb** umbenennen. Dazu klicken Sie diesen Eintrag im Projektmappen-Explorer mit der rechten Maustaste an und wählen dann den Befehl **Umbenennen...** aus dem Kontextmenü aus. Geben Sie den neuen Namen ein und bestätigen Sie auch die nun erscheinende Meldung, wodurch Sie alle Verweise auf den vorherigen Namen auf den neuen Namen ändern (siehe Bild 2). In diesem Fall wird auch beispielsweise der Name der Klasse im Klassenmodul angepasst:

```
Public Class CMD
```

```
End Class
```

Einfache Meldung ausgeben

Um eine einfache Meldung auszugeben, nachdem die entsprechende Methode der Klasse aufgerufen wurde, legen Sie zunächst die gewünschte **Sub**-Methode an:

```
Public Class CMD
```

```
    Public Sub MeldungAusgeben()
```

```
    End Sub
```

```
End Class
```

Nun heißt der Befehl zum Ausgeben eines Meldungsfensters unter VB.NET etwas anders als unter VBA, also nicht **MsgBox**. Stattdessen verwendet man unter VB.NET die Methode **Show** der Klasse **MessageBox**. Diese wiederum ist in DLL-Klassen nicht standardmäßig als Verweis eingebunden, was wir allerdings schnell nachholen können.

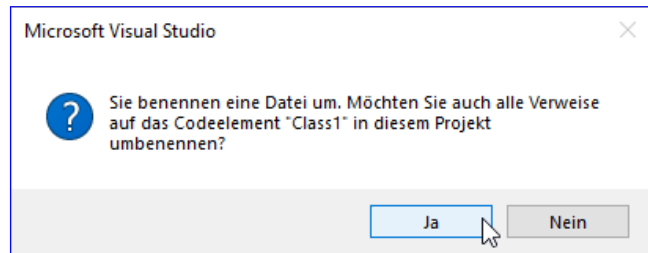


Bild 2: Umbenennen aller Verweise entsprechen des neuen Elementnamens

Verweis auf Bibliothek hinzufügen

Dazu rufen Sie mit dem Menübefehl **Projekt|Verweis hinzufügen...** den Dialog **Verweis-Manager** auf. Hier klicken Sie links auf **Assemblies**. Rechts oben im Suchfenster können Sie etwa **System.Windows** eintippen. Die Einträge werden nach jedem Zeichen aktualisiert, sodass Sie schon bald den Eintrag **System.Windows.Forms** in der Liste entdecken. Diesen markieren Sie durch einen Haken und schließen den Dialog mit einem Klick auf die Schaltfläche **OK** (siehe Bild 3).

Diese Bibliothek machen wir nun in unserer Klasse verfügbar, indem wir diese mit der **Imports**-Anweisung ganz oben einfügen:

```
Imports System.Windows.Forms
```

Die Methode **MeldungAusgeben** erweitern wir nun um den Aufruf der **Show**-Methode der **MessageBox**-Klasse:

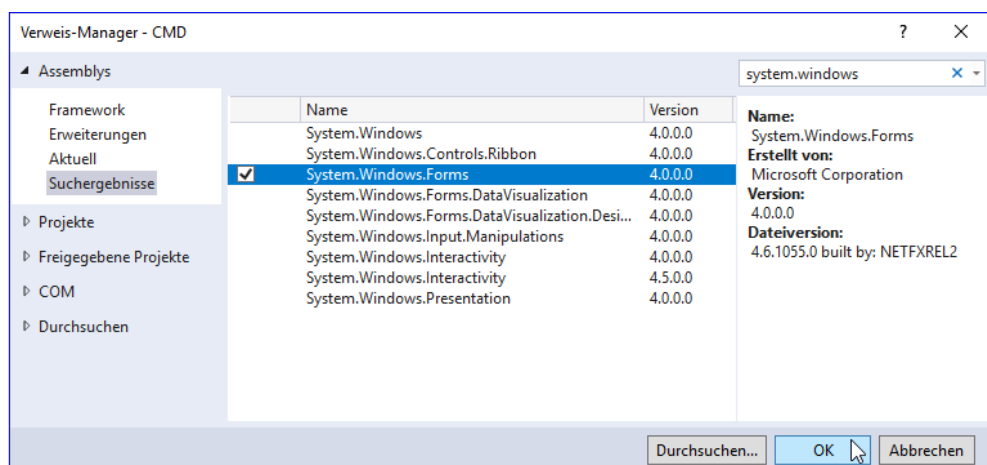


Bild 3: Hinzufügen eines Verweises für die **MessageBox**-Klasse

```
Public Class CMD
    Public Sub MeldungAusgeben()
        MessageBox.Show("Dies ist eine Meldung von einer 7
            VB-DLL.")
    End Sub
End Class
```

die Option **Assembly COM-sichtbar machen**, die Sie nun aktivieren. Anschließend können Sie den Dialog wieder schließen (siehe Bild 4).

Danach bleiben Sie direkt bei den Projekteigenschaften und aktivieren Sie den Bereich **Kompilieren**. Hier finden Sie ganz unten den Eintrag **Für COM-Interop registrieren**, den Sie ebenfalls aktivieren. Mit dieser Einstellung

Außerdem benötigen wir noch eine spezielle Auszeichnung der Klasse **CMD**, damit diese später unter VBA auch verfügbar ist. Dazu fügen Sie einen weiteren Namespace hinzu:

```
Imports System.Runtime.InteropServices
```

Außerdem statten Sie die Klasse mit einer Zusatzinformation aus, welche diese sichtbar macht:

```
<ClassInterface(ClassInterfaceType.AutoDual)>
Public Class CMD
    ...
End Class
```

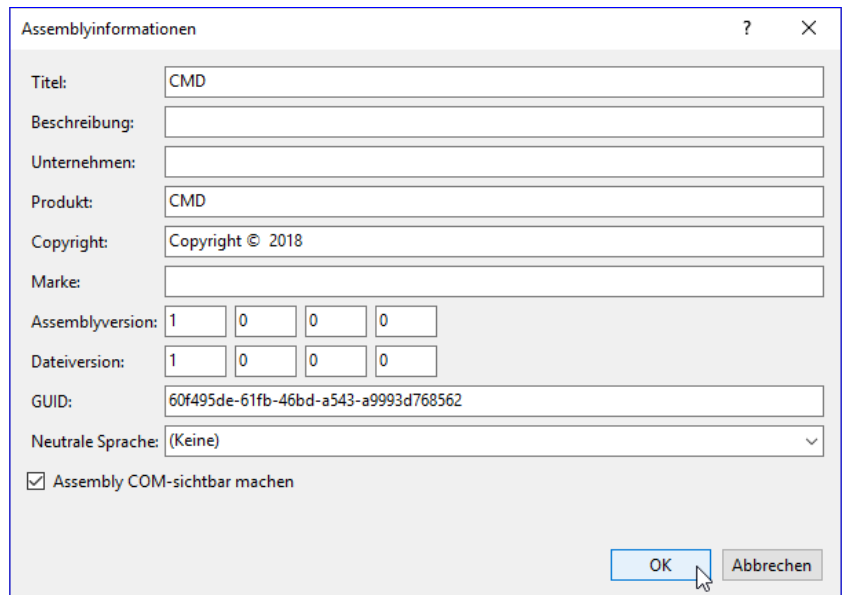


Bild 4: Einstellen der Eigenschaft **Assembly COM-sichtbar machen**

DLL-Klassen und -Methoden sichtbar machen

Nun fehlen noch zwei Einstellungen, bevor wir die DLL erstmalig testen können. Um diese vorzunehmen, klicken Sie doppelt auf den Eintrag **MyProject** im Projektmappen-Explorer. Im Bereich **Anwendung** des nun erscheinenden Fensters klicken Sie nun auf **Assemblyinformationen....** Im nun erscheinenden Dialog **Assemblyinformationen** finden Sie unten

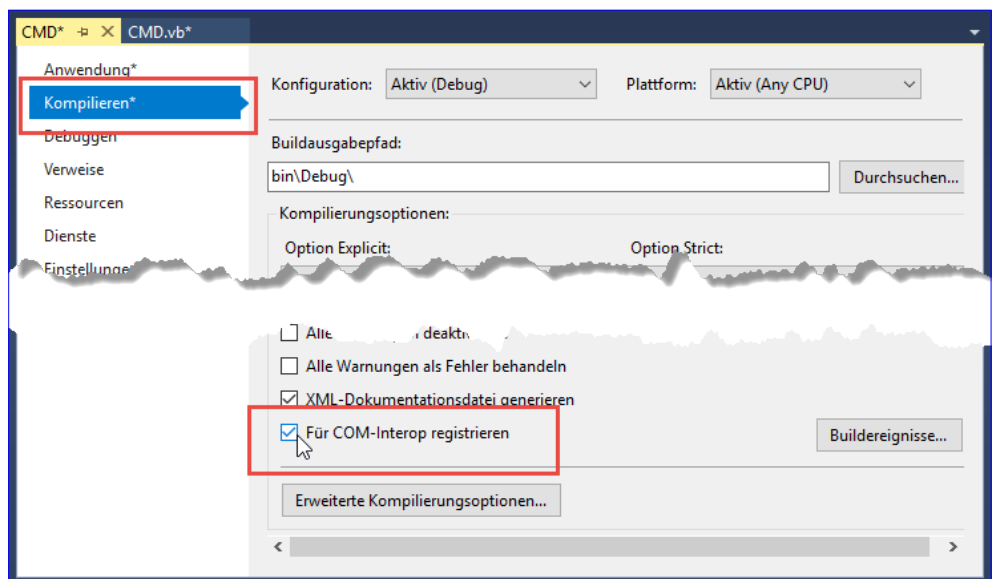


Bild 5: Registrieren für COM-Interop

Kommandozeile per DLL

Wenn Sie von Access aus Befehle ausführen wollen, die Sie normalerweise über die Kommandozeile beziehungsweise die Eingabeaufforderung eingeben würden, ist das kein Problem – das erledigen Sie ganz einfach über die Shell-Anweisung. Interessanter wird es, wenn Sie die Ausgabe der Anwendung einlesen und verarbeiten wollen, um beispielsweise auszuwerten, ob der Aufruf erfolgreich war oder welches Ergebnis dieser geliefert hat. Mit VBA beziehungsweise API-Funktionen gelingt dies nicht immer zuverlässig. Also schauen wir uns in der .NET-Bibliothek um, ob es dort Alternativen gibt, und machen uns diese über eine DLL unter VBA verfügbar.

Im Beitrag **VB.NET-DLL für Access programmieren** (www.access-im-unternehmen.de/1167) haben wir gezeigt, wie Sie eine VB-DLL programmieren, die Sie vom VBA-Projekt einer Access-Anwendung aus einbinden können, um auf ihre Methoden und Eigenschaften zuzugreifen. Auf dieser DLL setzen wir nun auf, um uns die zum Absetzen von Programmaufrufen und zum Erfassen der Rückgabewerte erforderlichen Funktionen verfügbar zu machen.

Debugging

Um das Debugging zu erleichtern, wollen wir während des Aufrufs von Access aus den Code der COM-DLL debuggen. Dazu sind ein paar kleine Einstellungen nötig, die

wir in den Projekteigenschaften im Bereich **Debuggen** vornehmen. Hier stellen wir für die Eigenschaft Externes Programm starten den Pfad zur **MSAccess.exe** ein (siehe Bild 1). Dieser Pfad lautet beispielsweise unter Office 365 wie folgt:

```
C:\Program Files (x86)\Microsoft Office\root\Office16\MSACCESS.EXE
```

Unter Befehlszeilenargument geben Sie den Pfad zu der zu verwendenden Datenbankdatei ein.

Wenn Sie beim Starten nun noch wollen, dass das VBA-Modul mit den Anweisungen zum Testen der DLL aufge-

rufen wird, fügen Sie der Datenbank ein einfaches Formular hinzu, für dessen Ereigniseigenschaft **Beim Laden** Sie die folgende Ereignisprozedur hinterlegen:

```
Private Sub Form_Load()  
    DoCmd.OpenModule "mdlDLL"  
End Sub
```

Speichern Sie das Formular unter dem Namen **frmStart**. Hinterlegen Sie dieses Formular in den Access-Optionen für die

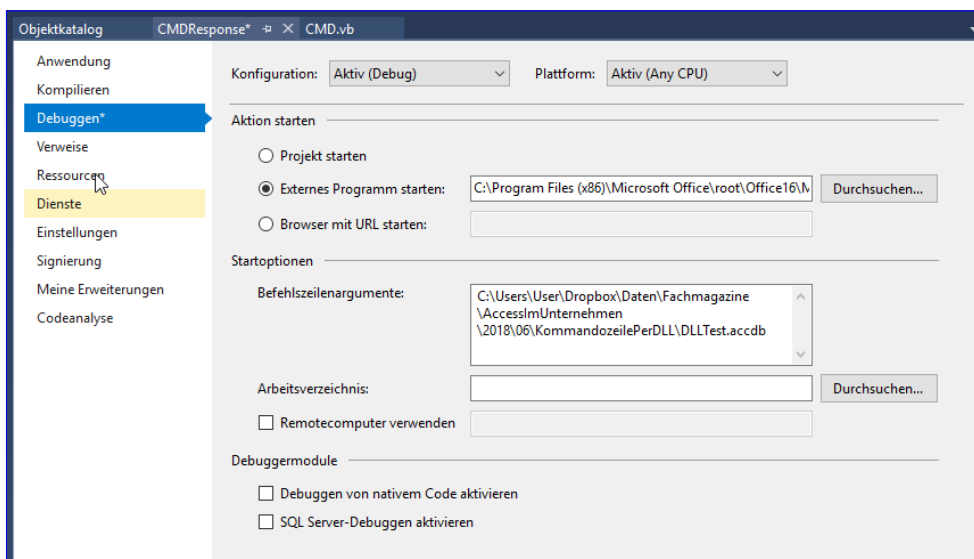


Bild 1: Pfad zur MSAccess.exe festlegen, die beim Debuggen gestartet werden soll

Eigenschaft **Formular anzeigen** (siehe Bild 2).

Wenn Sie die .NET-DLL nun von Visual Studio aus mit dem Menübefehl **Debugging/Debugging starten**, wird direkt die angegebene Access-Datenbank geöffnet. Diese zeigt direkt das Formular **frmStart** an, welches wiederum den VBA-Editor mit dem Modul **mdlDLL** anzeigt.

Test mit PDFTK

Im Beitrag **PDF-Dokumente im Griff mit PDFtk** (www.access-im-unternehmen.de/1166) zeigen wir, wie Sie mit dem Kommandozeilentool **PDFtk** auf PDF-Dokumente zugreifen, um diese zu kombinieren, zu schützen und vieles mehr. Dies gelingt allerdings nur über die Eingabeaufforderung. Wir können die notwendigen Befehle zwar über die **Shell**-Anweisung absetzen, allerdings können wir die Rückgabewerte beziehungsweise die Ausgabe von **PDFtk.exe** nicht einlesen oder auswerten.

Das wollen wir nun mit den unter .NET viel flexibleren Bibliotheken erledigen, in diesem Fall mit einige Elementen der Bibliothek **System.Diagnostics**. Um die DLL zunächst mit einer einfachen Funktion auszustatten, die **PDFTK** heißt und einfach nur den Befehl **PDFTK** absetzt und die resultierende Ausgabe abfängt und als Rückgabewert liefert, füllen Sie die Klassendatei des Projekts wie in Listing 1.

Der erste Teil ist das Interface, das wir anlegen müssen, damit wir nur unsere eigenen Member in der DLL angezeigt bekommen (siehe auch unter **VB.NET-DLL für Access programmieren**). Die eigentliche Klasse namens **CMD** implementiert die einzige Funktion der Schnittstelle **ICMD** namens **PDFTK**. Diese Funktion soll einen Wert mit dem Datentyp **String** zurückliefern. Sie erstellt ein neues

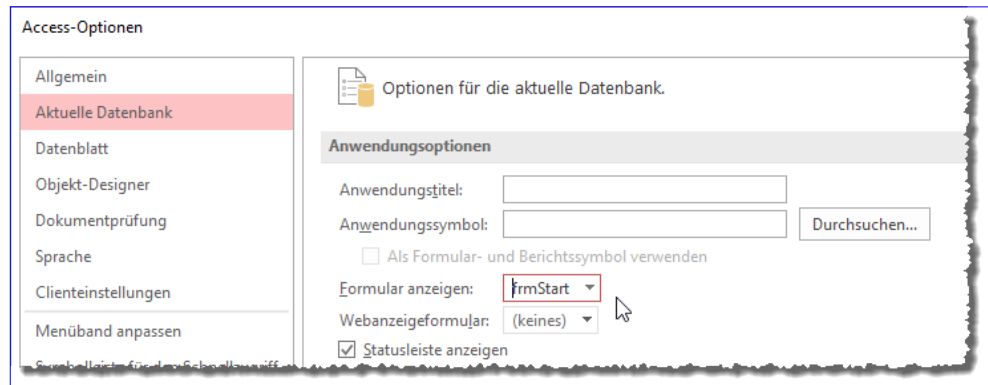


Bild 2: Einstellen des Startformulars unserer Beispielanwendung

Objekt des Typs **Process** und weist dem Objekt **StartInfo** dieses Objekts verschiedene Daten zu – zum Beispiel den Namen der aufzurufenden Datei **PDFtk.exe** mit der Eigenschaft **FileName**. Andere Parameter wie **Arguments** oder **WorkingDirectory** verwenden wir in dieser Version noch nicht. Allerdings stellen wir die Eigenschaft **RedirectStandardOutput** auf **True** ein.

Der Start des Aufrufs erfolgt dann mit der **Start**-Methode. Den Output des Prozesses ermitteln wir mit der Methode **ReadToEnd** des Objekts **Standardoutput** des Prozesses und schreiben diesen in die Variable **strOutput**. Nachdem wir das Ende mit der Methode **WaitForExit** abgewartet haben, können wir den Inhalt von **strOutput** als Rückgabewert zurück an die aufrufende Instanz liefern.

Wenn wir nun das Debuggen dieses Projekts starten, müssen Sie gegebenenfalls noch einen Verweis im VBA-Editor auf die Bibliothek **CMDResponse** hinzufügen (wie in **VB.NET-DLL für Access programmieren** beschrieben).

Anschließend können Sie die Methode mit der folgenden Prozedur aufrufen:

```
Public Sub TestDLL()
    Dim objCMD As CMDResponse.CMD
    Set objCMD = New CMDResponse.CMD
    Debug.Print objCMD.PDFTK
End Sub
```