HAUFE.

Ausgabe 01/2019

ACCESS

IM UNTERNEHMEN

KOMFORTABEL SUCHEN

Suchen Sie nach verschiedenen Suchbegriffen in unterschiedlichen Feldern – und das mit nur einem einzigen Steuerelemenet! (ab S. 61)

In diesem Heft:

GESPEICHERTE PROZEDUREN

Führen Sie gespeicherte Prozeduren von Access aus per Pass-through-Abfrage aus.

SEITE 2

KOMBINATIONSFELD PER TASTE

Wählen Sie Kombinationsfeld-Einträge ganze einfach mit der Nach oben- oder Nach unten-Taste aus.

SEITE 9

COM-ADD-INS FÜR DEN VBA-EDITOR

Erstellen Sie Add-Ins mit Visual Studio für den VBA-Editor

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-Steuerelement, 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.

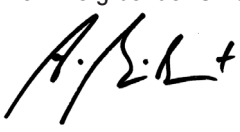
Wie Sie diese Daten dann bearbeiten können, beschreibt der Beitrag **HTML-Kreuztabelle 2: Werte bearbeiten** ab Seite 21. Hier zeigen wir natürlich nicht nur, wie Sie die Werte bearbeiten, sondern auch, welcher Mechanismus dafür sorgt, dass die geänderten Daten auch in die Tabelle mit den Preisen geschrieben 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

Gespeicherte Prozeduren mit Pass-Through-Abfragen

Für den Zugriff auf die Daten einer SQL Server-Datenbank gibt es mehrere Methoden. Die erste ist das Einbinden der Tabellen per ODBC. Sie greifen dann – oberflächlich betrachtet – genau wie auf lokale Daten zu. Die andere, performantere und auch für den Mehrbenutzerbetrieb effizientere Variante, ist der Zugriff über Pass-Through-Abfragen auf gespeicherte Prozeduren. Gespeicherte Prozeduren sind Skripte, die Anweisungen auf dem SQL Server ausgeben und die ihre Ergebnisse, also zum Beispiel gefundene Datensätze, zurückgeben können. Zugriff auf solche gespeicherten Prozeduren erhalten Sie über die Nutzung sogenannter Pass-Through-Abfragen. Wie Sie die gespeicherten Prozeduren und Pass-Through-Abfragen kombinieren, um Daten vom SQL Server in Ihre Access-Datenbank zu bekommen, zeigt dieser Beitrag.

Voraussetzungen

Für die Beispiele dieses Beitrags benötigen Sie die im Download befindliche Access-Datenbank, das SQL-Skript zum Erstellen des Backends auf dem SQL Server sowie eine Instanz des SQL Servers (das kann auch LocalDb oder die Express-Version sein) und das SQL Server Management Studio. Wie Sie die Datenbank mithilfe des SQL-Skripts aus dem Download erstellen, erfahren Sie im Beitrag **SQL Server-Datenbanken kopieren** (**www.access-im-unternehmen.de/1153**).

Gespeicherte Prozeduren

Gespeicherte Prozeduren, im Englischen **Stored Procedures**, sind eine der flexibelsten Objektarten im SQL Server. Sie können damit nicht nur Daten abfragen, sondern auch Daten manipulieren. Wenn Sie Daten abfragen, liefern gespeicherte Prozeduren die abgefragten Daten als Ergebnis zurück, wenn Sie Daten manipulieren, können Sie die gespeicherte Prozedur so programmieren, dass diese etwa die Anzahl der geänderten Datensätze zurückliefert. Sie können dabei auf den gesamten Sprachumfang von T-SQL zurückgreifen und nicht nur einfache Auswahlabfragen oder Aktionsabfragen definieren wir unter Access. Stattdessen können Sie sogar Strukturen wie Bedingungen oder Schleifen nutzen und in einer gespeicherten Prozedur mehrere Auswahl- oder Aktionsabfragen durchführen.

Pass-Through-Abfragen

Pass-Through-Abfragen sind eine Möglichkeit, von Access auf die Daten eines SQL Servers zuzugreifen. Sie können damit zum Beispiel einfach die Daten einer kompletten Tabelle zurückliefern, was dann in etwa dem Verknüpfen der Tabelle per ODBC entspricht – mit dem Unterschied, dass Sie über eine Pass-Through-Abfrage keine Daten ändern können. Die von Pass-Through-Abfragen gelieferten Ergebnisse sind also immer schreibgeschützt. Davon ab können Sie mit einer Pass-Through-Abfrage aber nicht nur Daten abfragen, sondern alle denkbaren SQL-Anweisungen an die Zieldatenbank schicken.

Dabei gibt es einen sehr wichtigen Punkt zu beachten: Pass-Through-Abfragen werden direkt an den jeweiligen SQL Server geschickt, sie müssen daher auch im SQL-Dialekt des jeweiligen Servers formuliert sein und nicht etwa in dem von Access/Jet verwendeten Dialekt.

Ein Beispiel: Während Sie in Access etwa das Sternchen (*****) als Platzhalter für beliebige Zeichen verwenden, nutzen die meisten relationalen Datenbankmanagementsysteme das Prozent-Zeichen (**%**) als Platzhalter für beliebige Zeichen. Die Suche nach allen Artikeln, die mit A beginnen, würde also so lauten:

```
SELECT * FROM tblArtikel WHERE Artikelname LIKE 'A%'
```

Einer Pass-Through-Abfrage geben Sie über die Eigenschaften die Verbindungszeichenfolge mit, welche die Zieldatenbank für die Abfrage definiert.

Gespeicherte Prozeduren und Pass-Through-Abfragen

Wenn wir nun alle denkbaren SQL-Anweisungen über eine Pass-Through-Abfrage an den Server schicken können – warum sollten wir dann die Abfragen erst auf dem Server als gespeicherte Prozeduren definieren und diese dann von Access aus per Pass-Through-Abfrage aufrufen, statt diese direkt von Access aus an den SQL Server zu schicken? Die Antwort ist einfach: Die Performance einer Abfrage an den SQL Server richtet sich nicht nur nach der Menge der über das Netz zu transportierenden Daten (diese ist in beiden Fällen etwa gleich). Sie richtet sich auch danach, wie lange der SQL Server dafür braucht, diese Daten zusammenzustellen.

Und hier kommt der Abfrageoptimierer des SQL Servers ins Spiel: Wenn Sie eine Abfrage als gespeicherte Prozedur anlegen und diese ausführen, erstellt SQL Server automatisch einen Ausführungsplan, der in der Folge für weitere Aufrufe dieser Abfrage genutzt wird. Das gilt auch, wenn Sie für die Abfrage einen oder mehrere Parameter nutzen.

Wenn Sie jedoch eine Abfrage, deren Inhalt dem einer gespeicherten Prozedur gleichzusetzen ist, auf der Access-Seite formulieren und diese über eine Pass-Through-Abfrage an den SQL Server schicken, muss diese jedes Mal vom Abfrageoptimierer analysiert werden. Bei Abfragen, die nur einmalig genutzt werden, ergibt sich somit kein nennenswerter Unterschied, aber sobald Sie eine Abfrage mehr als einmal nutzen, haben Sie Performance-Vorteile, wenn Sie die Abfrage als gespeicherte Prozedur auf dem SQL Server speichern.

Wir werden uns also in der Regel auf die Kombination aus Pass-Through-Abfrage und gespeicherter Prozedur konzentrieren.

Gespeicherte Prozedur anlegen

Als Erstes legen wir eine einfache gespeicherte Prozedur auf unserer SQL Server-Datenbank **Suedsturm_SQL** an. Dabei belassen wir es für das erste Beispiel bei einer einfachen Abfrage, die alle Datensätze der Tabelle **tblArtikel** zurückliefern soll. Diese Abfrage legen wir im SQL Server Management Studio an.

Dazu öffnen wir SQL Server Management Studio und wechseln zur Datenbank **Suedsturm_SQL**. Hier navigieren wir zum Eintrag **Suedsturm_SQL|Programmierbarkeit|Gespeicherte Prozeduren** und wählen aus dem Kontextmenü dieses Eintrags den Befehl **Gespeicherte Prozedur...** aus (siehe Bild 1).

Dies öffnet die Vorlage für gespeicherte Prozeduren, die es uns vereinfachen soll, schnell neue gespeicherte Prozeduren anzulegen (siehe Bild 2).

Hier sehen Sie gleich, dass die eigentliche Anweisung mit **CREATE PROCEDURE** beginnt. In diesem Abfragefenster geben wir also nicht den Code der gespeicherten Abfrage selbst ein, sondern den Code, den SQL Server zum

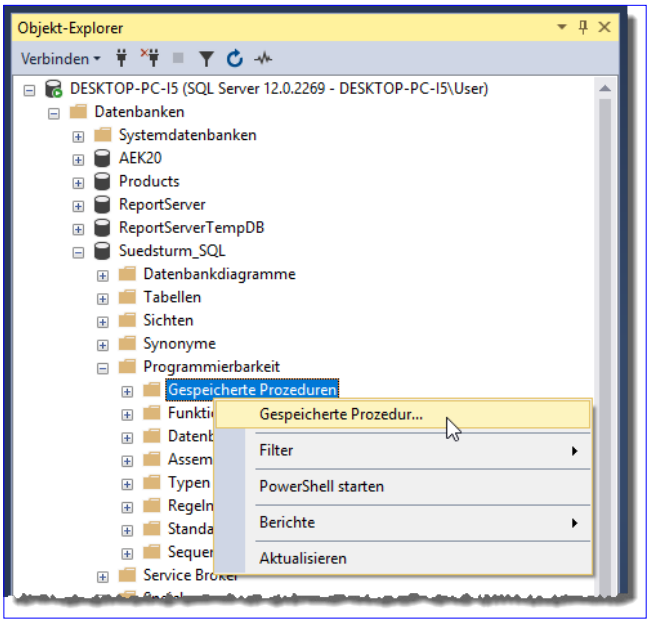


Bild 1: Neue gespeicherte Prozedur anlegen

Erstellen der gespeicherten Prozedur benötigt. Auf die gleiche Weise lautet der Code zum Verändern einer bestehenden gespeicherten Prozedur **ALTER PROCEDURE**. Die Vorlage ist für unsere Zwecke etwas überdimensioniert, sodass wir den enthaltenen Code etwas zusammenschrumpfen und schließlich folgendes SQL-Skript erhalten:

```
CREATE PROCEDURE dbo.spArtikelAlle
AS
SET NOCOUNT ON;
SELECT * FROM tblArtikel;
```

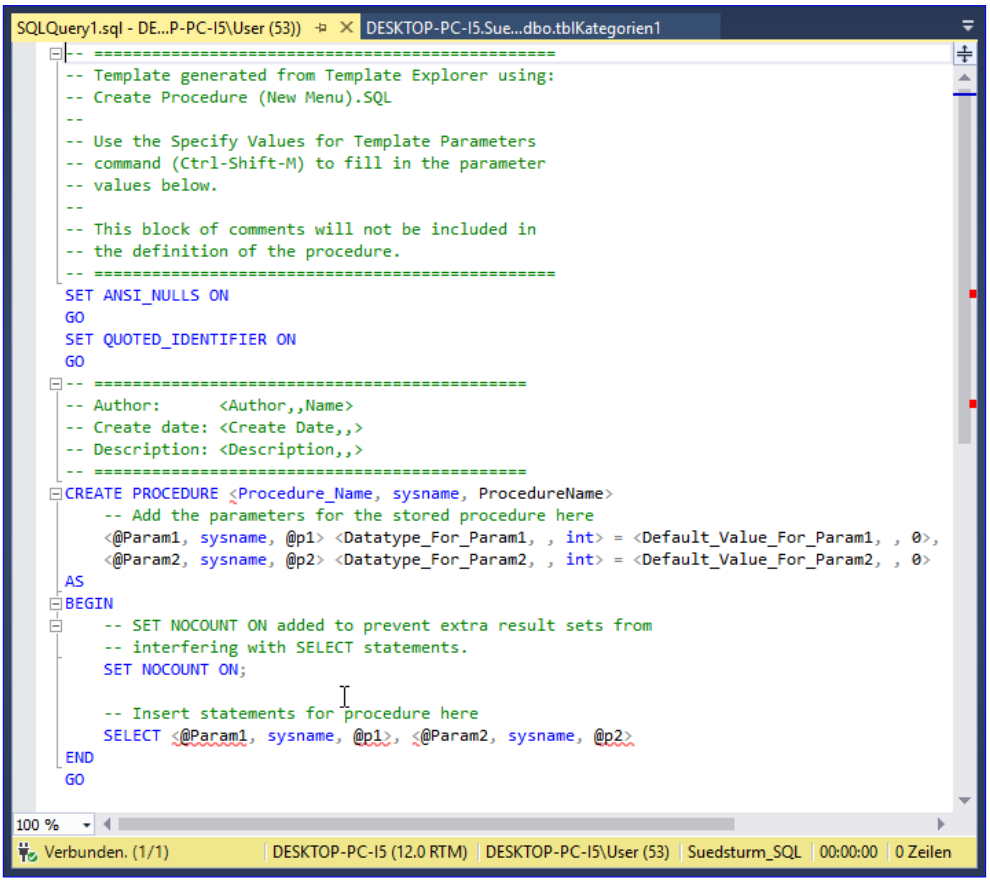


Bild 2: Vorlage für eine neue gespeicherte Prozedur

Mit einem Klick auf die Taste **F5** führen Sie die Abfrage zum Erstellen der gespeicherten Prozedur aus. Dies legt einen neuen Eintrag im Objekt-Explorer unter **Programmierbarkeit\Gespeicherte Prozeduren** an – siehe Bild 3. Wenn Sie die gespeicherte Prozedur anpassen können, rufen Sie den Kontextmenü-Befehl **Ändern** auf.

Namen für gespeicherte Prozeduren

Gespeicherte Prozeduren erhalten normalerweise das Präfix **sp**, also zum Beispiel **spArtikelAlle**. Die Variante mit dem Unterstrich, also etwa **sp_ArtikelAlle**, sollten Sie nicht verwenden, da SQL Server dann beim Aufruf erst die gespeicherten Systemprozeduren durchsucht und nicht direkt in der Zieldatenbank. Dem Namen für die gespeicherte Prozedur stellen wir als Präfix das Standardschema voran, hier **dbo.spArtikelAlle**. Wenn Sie mit benutzerdefinierten Schemas arbeiten, können Sie auch dieses hier angeben (mehr zu solchen Schemas im Beitrag **SQL Server: Sicherheit mit Schema**, **www.access-im-unternehmen.de/1179**).

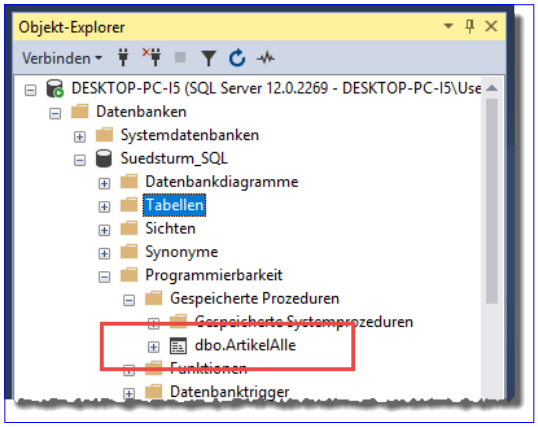


Bild 3: Die neue gespeicherte Prozedur im Objekt-Explorer

Gespeicherte Prozedur ausführen

Um die gespeicherte Prozedur auszuführen, wählen Sie den Kontextmenü-Eintrag **Gespeicherte Prozedur ausführen...** aus. Als Erstes erscheint ein Dialog, der die Eingabe von Parametern für die gespeicherte Prozedur

erlaubt. Da wir keine Parameter für diese gespeicherte Prozedur definiert haben, ist dieser Dialog allerdings unnötig und Sie können ihn gleich mit **OK** bestätigen.

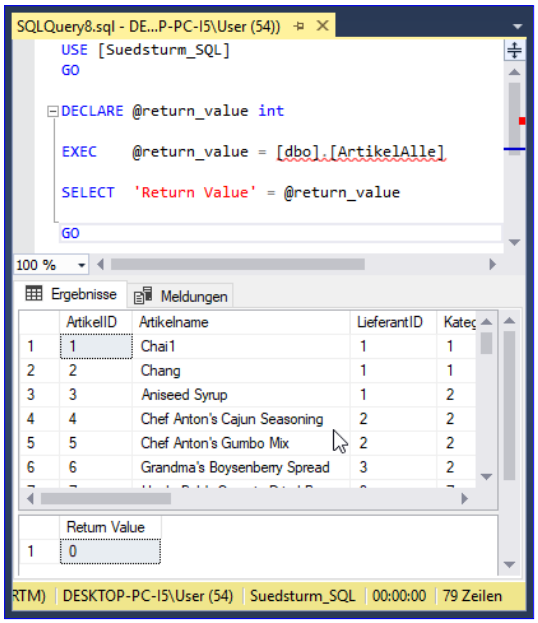


Bild 4: Ausführen der gespeicherten Prozedur

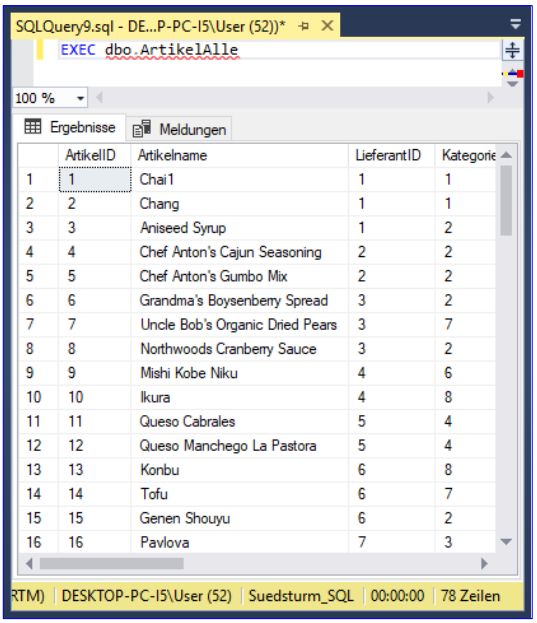


Bild 5: Ausführen der gespeicherten Prozedur per **EXEC**

Anschließend liefert SQL Server Management Studio das Ergebnis der Abfrage samt eines kleinen, durch den Aufruf des Kontextmenü-Befehls erstellten Skripts im mittleren Bereich (siehe Bild 4).

Das bekommen Sie auch etwas übersichtlicher hin, indem Sie einfach den Befehl **EXEC dbo.ArtikelAlle** in einer neuen Abfrage ausführen (siehe Bild 5). Damit erhalten wir auch gleich den Befehl, den wir in unserer gleich zu erstellenden Pass-Through-Abfrage verwenden wollen.

Gespeicherte Prozedur per Pass-Through-Abfrage

Mit der soeben erstellten gespeicherten Prozedur wollen wir nun eine Pass-Through-Abfrage in Access füttern und damit die gewünschten Daten in Access anzeigen.

Dazu gibt es in Access 2016 noch nicht einmal mehr einen passenden Assistenten – wir müssen also mit dem herkömmlichen Abfrageentwurf auskommen.

Nachdem Sie also über den Ribbon-Eintrag **Erstellen|Abfragen|Abfrageentwurf** eine neue Abfrage erstellt haben, schließen Sie zunächst den automatisch erscheinenden Dialog **Tabelle anzeigen**. Oben im Ribbon finden Sie den Befehl **Entwurf|Abfragetyp|Pass-Through**, mit dem Sie die Abfrage in eine Pass-Through-Abfrage umwandeln können. Auf den ersten Blick ist die einzige Änderung,

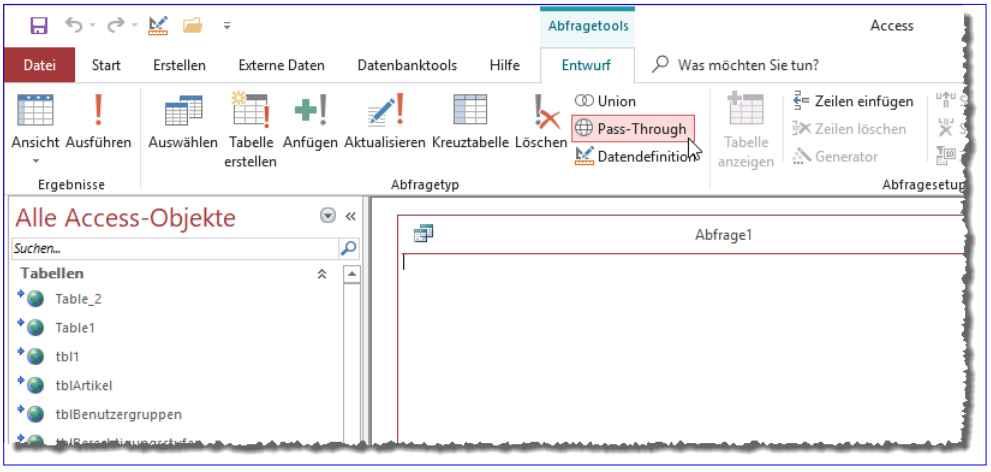


Bild 6: Umwandeln der Abfrage in eine Pass-Through-Abfrage

Kombinationsfeld per Taste steuern

Kombinationsfelder enthalten viele Einträge, die Sie durch das Aufklappen des Kombinationsfeldes anzeigen und per Mausklick auswählen können. Wenn Sie mit der Tastatur arbeiten, müssen Sie das Kombinationsfeld erst mit der Taste F4 öffnen und dann mit der Nach oben- oder der Nach unten-Taste den gewünschten Eintrag auswählen, den Sie dann durch Verlassen des Steuerelements bestätigen. Das geht auch einfacher, nämlich mit den beiden Tasten »Nach oben« und »Nach unten«. Es bedarf allerdings einiger Zeilen VBA-Code, die wir in diesem Beitrag vorstellen.

Vorbereitung

Um ein Formular mit einem Beispielkombinationsfeld bereitzustellen, legen Sie zunächst ein neues Formular an. Dann legen wir die Tabelle **tblArtikel** aus der Beispieldatenbank **Suedsturm** als Datensatzquelle des Formulars fest. Aktivieren Sie dann die Feldliste und ziehen Sie alle Felder aus der Feldliste in den Detailbereich der Entwurfsansicht des Formulars. Die beiden Felder **KategorieID** und **LieferantID** werden dann direkt als Kombinationsfelder angelegt.

Für die beiden Kombinationsfelder legen wir die Namen **cboLieferantID** und **cboKategorieID** fest. Speichern Sie das Formular dann unter dem Namen **frmKombinatoinsfeldMitTaste** (siehe Bild 1).

Wenn Sie in die Formularansicht wechseln, können Sie bei Tastaturbetrieb den Fokus mit der Tabulator-Taste auf eines der Kombinationsfelder verschieben. Dann öffnen Sie dieses mit der Taste **F4** und können dann mit den Tasten **Nach oben** und **Nach unten** einen der Einträge auswählen (siehe Bild 2).

Dies wollen wir nun vereinfachen, indem wir das Blättern durch die Einträge ohne Öffnen des Kombinationsfeldes ermöglichen. Wie das gelingt, erfahren Sie in den folgenden Abschnitten.

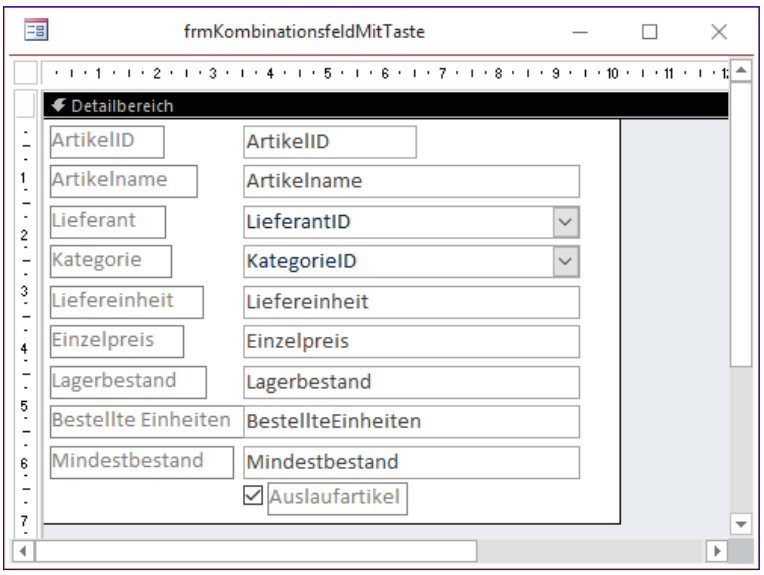


Bild 1: Beispielformular mit zwei Kombinationsfeldern

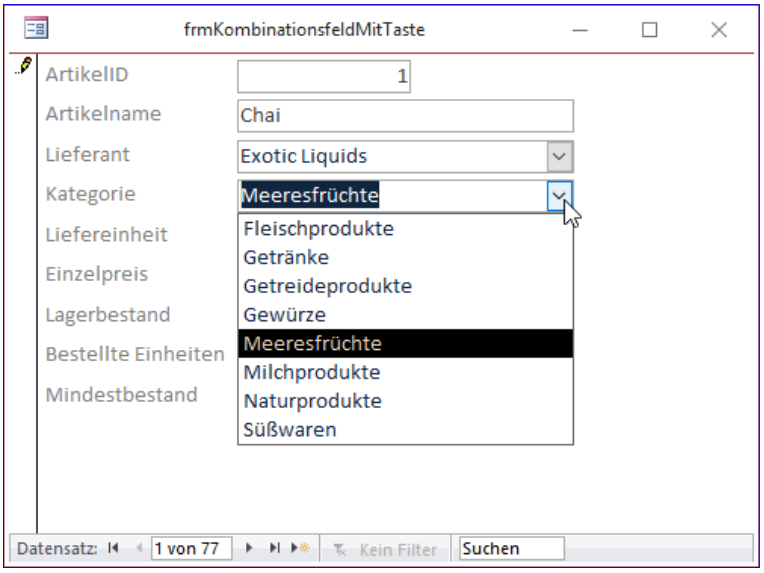


Bild 2: Beispielformular mit zwei Kombinationsfeldern in der Formularansicht

Kombinationsfeldeintrag per Taste auswählen

Wir wollen das Verhalten nachbilden, das wir sehen, wenn wir uns bei aufgeklapptem Kombinationsfeld mit der **Nach oben**- und der **Nach unten**-Taste durch die Einträge bewegen. Hier stellen wir fest, dass jeweils der vorherige oder nächste Eintrag ausgewählt wird, sobald der Benutzer eine der beiden Tasten **Nach oben** oder **Nach unten** herunterdrückt.

Damit steht schon einmal das Ereignis fest, dass wir programmieren müssen, nämlich **Bei Taste ab**. Für dieses Ereignis legen wir nun eine Ereignisprozedur an. Diese sieht ohne eigene Anweisungen nun wie folgt aus:

```
Private Sub cboLieferantID_KeyDown(KeyCode As Integer, 
                                   Shift As Integer)

End Sub
```

Mit dem Parameter **KeyCode** erhalten wie eine Zahl, welche die soeben gedrückte Taste repräsentiert, **Shift** liefert einen Zahlencode für die Tasten **Umschalt**, **Strg** und **Alt**.

Wir wollen zunächst herausfinden, welche Werte der Parameter **KeyCode** liefert, wenn wir im Kombinationsfeld eine der Tasten **Nach oben** oder **Nach unten** betätigen. Dazu fügen wir die folgende **Debug.Print**-Anweisung zur Ausgabe von **KeyCode** hinzu:

```
Private Sub cboLieferantID_KeyDown(KeyCode As Integer, 
                                   Shift As Integer)
    Debug.Print KeyCode
End Sub
```

Betätigen wir nun in der Formularansicht die **Nach oben**-Taste, wird der Wert **38** im Direktbereich ausgegeben, für die **Nach unten**-Taste der Wert **40**.

Wir müssen also auf die beiden Werte **38** und **40** reagieren, was wir in einer **Select Case**-Bedingung erledigen.

Die Prozedur sieht danach wie in Listing 1 aus. Sie enthält genau genommen sogar zwei **Select Case**-Bedingungen. Die erste prüft, ob der Benutzer überhaupt eine der beiden Tasten **Nach oben** oder **Nach unten** betätigt hat. In diesem Fall ermitteln wir zunächst den aktuellen Indexwert des Kombinationsfeldes und speichern diesen in der Variablen **lngIndex**.

```
Private Sub cboLieferantID_KeyDown(KeyCode As Integer, Shift As Integer)
    Dim lngIndex As Long
    Select Case KeyCode
        Case 38, 40
            lngIndex = Me!cboLieferantID.ListIndex
            Select Case KeyCode
                Case 38
                    If lngIndex > 0 Then
                        Me!cboLieferantID.ListIndex = lngIndex - 1
                    End If
                Case 40
                    If lngIndex < Me!cboLieferantID.ListCount - 1 Then
                        Me!cboLieferantID.ListIndex = lngIndex + 1
                    End If
            End Select
            KeyCode = 0
    End Select
End Sub
```

Listing 1: Kombinationsfeldeinträge auswählen mit den **Nach oben**- und **Nach unten**-Tasten

Dann folgt die zweite **Select Case**-Bedingung, in der wir nach den Werten **38** (**Nach oben**) und **40** (**Nach unten**) unterscheiden.

Im Falle der Taste **Nach oben** prüfen wir, ob der aktuelle Index größer als **0** ist. Dazu der Hinweis, dass der Wert der Eigenschaft **ListIndex** bei noch nicht

HTML-Kreuztabelle 3: Neue Zeilen und Spalten

In den ersten beiden Teilen der Beitragsreihe haben wir gezeigt, wie Sie die Daten einer Tabelle in Form einer Kreuztabelle ausgeben können. Das ist natürlich auch per Kreuztabellenabfrage möglich, aber wir haben in diesem Fall das Webbrowser-Steuerelement 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 in den bereits genannten Beiträgen **HTML-Kreuztabelle 1: Basics** (**www.access-im-unternehmen.de/1161**) und **HTML-Kreuztabelle 2: Werte bearbeiten** (**www.access-im-unternehmen.de/1162**) erarbeitete Lösung. Hier wollen wir nun folgende Erweiterungen hinzufügen:

- 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.
- Ein Doppelklick auf einen der Werte soll den entsprechenden Datensatz der Tabelle **tblMaterialpreise** in einem eigenen Formular öffnen.

Neue Zeilen und Spalten anlegen

Für das Anlegen neuer Zeilen und Spalten haben wir uns weiter oben vorgenommen, rechts neben den Spaltenüberschriften und unter den Zeilenüberschriften jeweils ein Plus-Zeichen anzuzeigen, über das wir eine neue Zeile oder Spalte hinzufügen können. Wir könnten dann, wenn der Benutzer auf eines der beiden Plus-Zeichen klickt, einfach die neue Zeile hinzufügen. Aber geht das wirklich so einfach? Nein: Denn unser Algorithmus aus der Prozedur **KreuztabelleAnlegen** fügt nur Spalten und Zeilen an für Höhen- oder Breitenangaben, zu denen es auch bereits mindestens eine Preisangabe gibt. Dementsprechend haben wir auch gar keine Möglichkeit, Daten so in der Tabelle **tblMaterialpreise** zu speichern, dass diese allein in der Anzeige einer neuen Zeile oder Spalte münden, ohne selbst einen Preis für die Kombination aus Höhe und Breite zu enthalten.

Also müssen wir uns einen alternativen Weg ausdenken, der wie folgt aussieht könnte: Wenn der Benutzer einen Preis für eine Kombination aus Höhe und Breite angeben möchte, von denen entweder die Höhe oder die Breite oder auch beides noch nicht vorhanden ist, muss er eben einen neuen Datensatz in der Tabelle **tblMaterialpreise** einfügen, der die gewünschten Werte für Höhe und Breite enthält.

Letztlich wäre es aber intuitiver, wenn der Benutzer einfach auf ein Plus-Zeichen zum Erstellen einer neuen Zeile oder Spalte klickt und dann durch einen Klick auf die entstehenden Zellen mit dem Minus-Zeichen den neuen Preis eingibt.

Was aber benötigen wir, um dies zu realisieren? Richtig, wir müssten die verfügbaren Werte für die Höhen und Breiten in jeweils einer weiteren Tabelle vorhalten. In

der aktuellen Konstellation ist das auch möglich, denn wir gehen ja erstmal von einem einzigen Material aus. In der Praxis ist unser Ansatz aber natürlich zu kurz gedacht – wir werden auch verschiedene Materialien berücksichtigen müssen, die wir in einer weiteren Tabelle speichern, die etwa **tblMaterialien** heißt. Auf diese Tabelle verweisen wir dann von der Tabelle **tblMaterialpreise** aus per Fremdschlüsselfeld. Ebenso können wir auf diese Tabelle dann von den beiden Tabellen zum Speichern von Höhen und Breiten verweisen. Also legen wir diese Tabellen an. Auf das Hinzufügen neuer Zeilen und Spalten kommen wir später zurück – vorher sind noch einige Vorbereitungen und Erweiterungen zu erledigen.

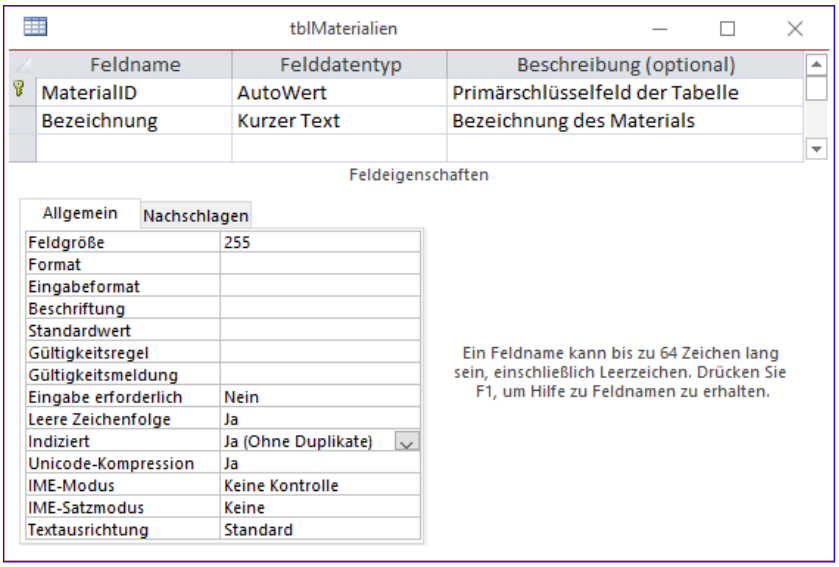


Bild 1: Die Tabelle **tblMaterialien**

Tabelle zum Speichern der Materialien

Die Tabelle **tblMaterialien** soll die verschiedenen Materialien aufnehmen, damit wir die Materialpreise nicht nur in Abhängigkeit der Kombinationen von Höhe und Breite, sondern auch in Abhängigkeit von verschiedenen Materialien erfassen können, also etwa 17 mm und 19 mm dicke Spanplatten. Die Tabelle enthält zwei Felder: Das Feld **MaterialID** ist das Primärschlüsselfeld, das Feld **Bezeichnung** legt den Namen des Materials fest. Für dieses Feld legen wir einen eindeutigen Index fest, damit jede Bezeichnung nur einmal eingegeben werden kann (siehe Bild 1).

Tabellen zum Speichern von Höhen und Breiten

Die Tabelle zum Speichern der Höhen heißt **tblHoehen**. Sie enthält ein Primärschlüsselfeld namens **HoeheID**, das Feld **Hoehe** zum Angeben der Höhe und das Fremdschlüsselfeld **MaterialID**, über das die Höhe einem Datensatz der Tabelle **tblMaterialien** zugeordnet wird. Für die beiden Felder **Hoehe** und **MaterialID** legen wir einen eindeutigen, zusammengesetzten Schlüssel fest. So verhindern wir, dass für ein Material die gleiche Höhe doppelt angelegt wird (siehe Bild 2).

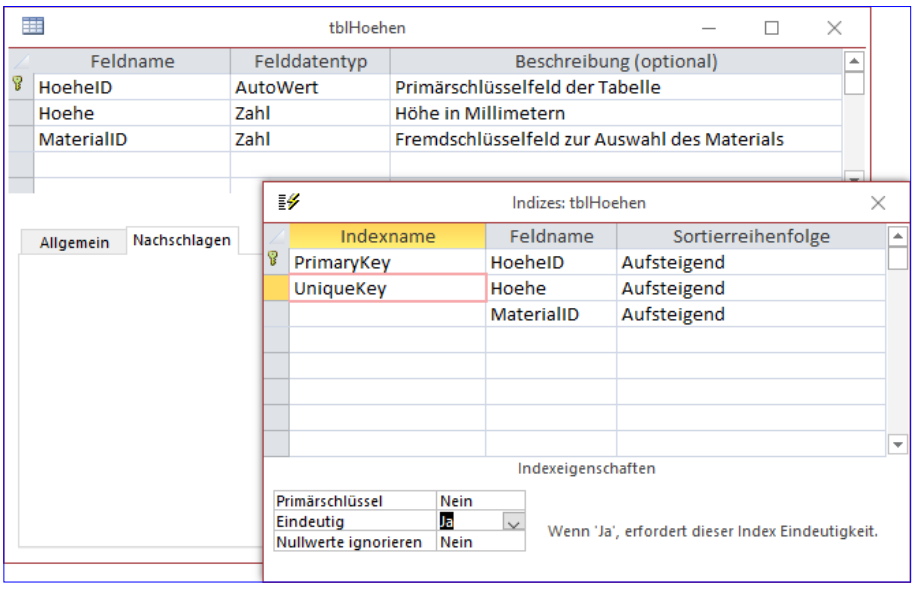


Bild 2: Die Tabelle **tblHoehen**

Die Tabelle **tblBreiten** bauen wir analog zur Tabelle **tblHoehen** auf.

Materialpreis nach Material

Schließlich wollen wir der Tabelle **tblMaterialpreise** noch ein Fremdschlüsselfeld namens **MaterialID** hinzufügen, mit dem wir den entsprechenden Eintrag der Tabelle **tblMaterialien** auswählen. Auch hier müssen wir das neue Feld **MaterialID** in den eindeutigen Index integrieren. Die neue Version der Tabelle sieht nun wie in Bild 3 aus.

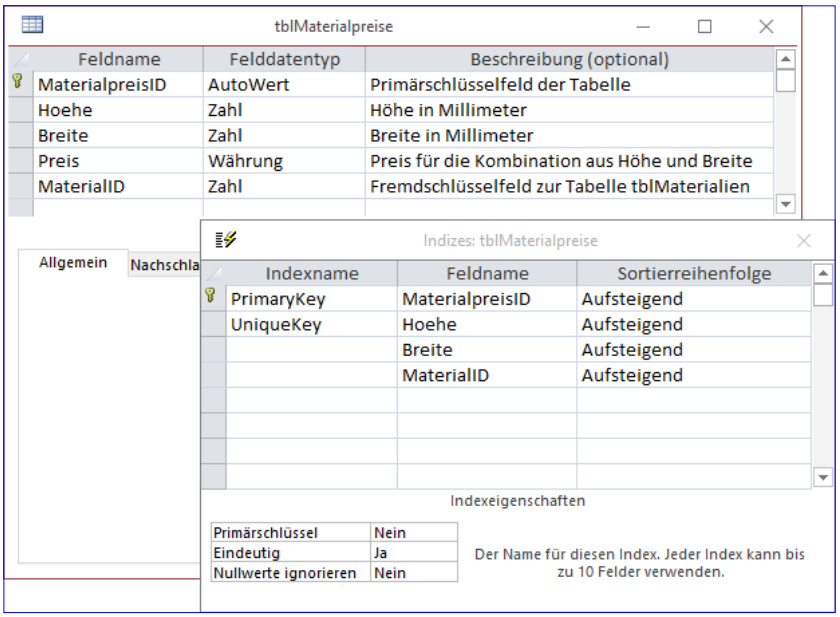


Bild 3: Die angepasste Tabelle **tblMaterialpreise**

Das Datenmodell enthält nun vier Tabellen, die wie in Bild 4 miteinander verknüpft sind.

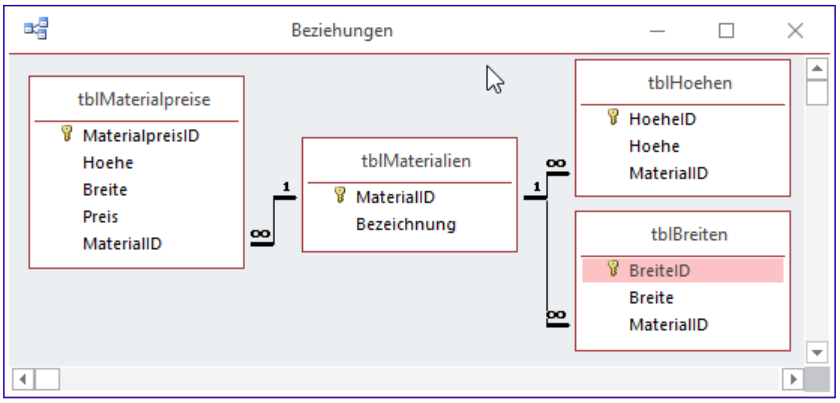


Bild 4: Aktueller Stand des Datenmodells

Material anlegen

Nun fügen wir einige Einträge zur Tabelle **tblMaterialien** hinzu, zum Beispiel **Spanplatte 17 mm** und **Spanplatte 19 mm**. Anschließend verbinden wir alle Datensätze der Tabelle **tblMaterialpreise**, die wir bisher angelegt haben, über das Feld **MaterialID** mit dem ersten der beiden neu angelegten Datensätze der Tabelle **tblMaterialien**. Das Ergebnis sieht in der Tabelle **tblMaterialpreise** dann etwa wie in Bild 5 aus.

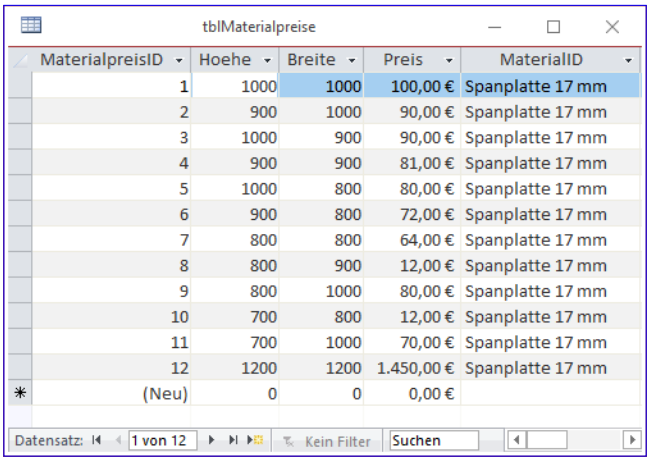


Bild 5: Inhalt der Tabelle **tblMaterialpreise**

Außerdem fügen wir in den beiden Tabellen **tblHoehen** und **tblBreiten** die Werte ein, die bereits in der Tabelle **tblMaterialpreise** vorliegen.

Höhen und Breiten verknüpfen?

An dieser Stelle fragen wir uns auch: Sollten wir nicht die Felder **Hoehe** und **Breite** der Tabelle **tblMaterialpreise** mit den entsprechenden Datensätzen der Tabelle **tblHoehen** und **tblBreiten** verknüpfen? Und würde es dann nicht sogar Sinn machen, hier einmal auf einen

Autowert in den Primärschlüsselfeldern der beiden Tabellen **tblHoehen** und **tblBreiten** zu verzichten und direkt die Werte der Höhen und Breiten in das Primärschlüsselfeld einzutragen? Dann hätten wir die für uns interessanten Werte direkt in den gebundenen Spalten der Fremdschlüsselfelder und müssten gar keine Lookup-Abfragen tätigen, um auf die Werte der Tabellen **tblHoehen** und **tblBreiten** zugreifen – diese würden dann lediglich zur Beschränkung der möglichen Werte dienen, die in den Fremdschlüsselfeldern **Hoehe** und **Breite** eingegeben beziehungsweise ausgewählt werden können.

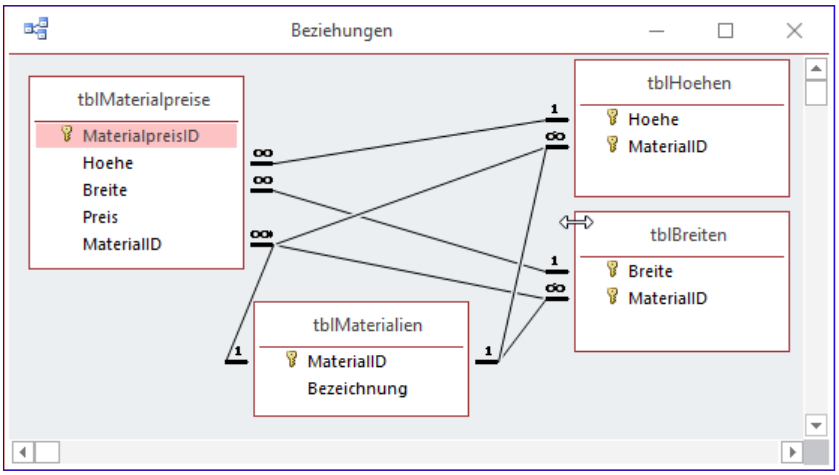


Bild 6: Neue Verknüpfungen im Datenmodell

Das erledigen wir schließlich, indem wir die Beziehungen wie in Bild 6 hinzufügen. Vorher haben wir die Primärschlüsselfelder **HoeheID** und **BreiteID** aus den Tabellen **tblHoehen** und **tblBreiten** entfernt und die Felder **Hoehe** und **Breite** in Primärschlüsselfelder ohne Autowert umgewandelt.

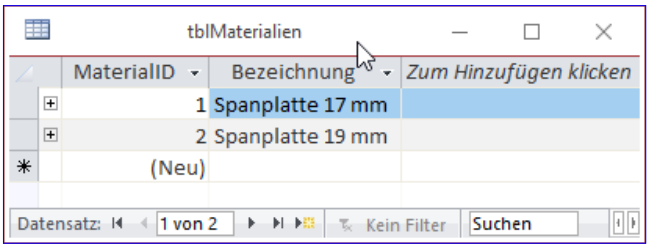


Bild 7: Neuer Datensatz in der Tabelle **tblMaterialien**

Werte für neues Material hinzufügen

Damit wir neue Daten zum Spielen haben, fügen wir der Tabelle **tblMaterialien** einen zweiten Datensatz hinzu (siehe Bild 7).

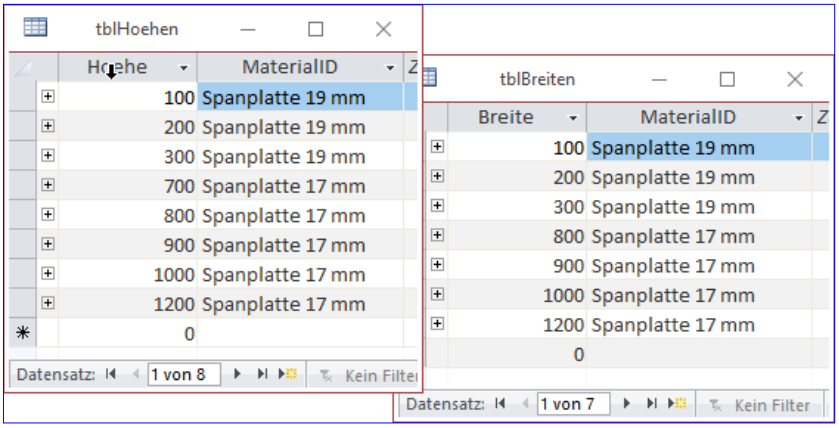


Bild 8: Neue Werte für Höhe und Breite für das zuletzt hinzugefügte Material

Außerdem fügen wir in den beiden Tabellen **tblHoehen** und **tblBreiten** einige neue Datensätze mit Werten für die neue Materialart ein. Zur Unterscheidung von den Datensätzen für die bereits vorhandene Materialart nutzen wir hier kleinere Werte (100 bis 300) – siehe Bild 8.

Schließlich legen wir einige neue Datensätze in der Tabelle **tblMaterialpreise** für das neue Material an (siehe Bild 9).

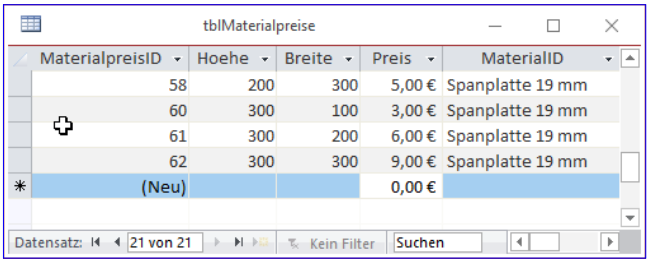


Bild 9: Materialpreise für neues Material

Wenn wir nun das Formular mit der Kreuztabelle öffnen, sieht dieses wie in Bild 10 aus. Hier werden nun alle Maße und alle Preise angezeigt – sowohl für die 17 mm als auch für die 19 mm dicken Spanplatten. Leider kann man diese aktuell nur dadurch auseinanderhalten, dass wir für die eine Spanplatte die kleineren Höhen und Breiten und für die andere die größeren festgelegt haben.

Damit nur die Werte für eines der Materialien angezeigt werden, fügen wir dem Formular nun ein Kombinationsfeld hinzu, mit dem der Benutzer das Material auswählen kann. Das neue Kombinationsfeld nennen wir **cboMaterialID**. Für seine Eigenschaft **Datensatzherkunft** legen wir die folgende Abfrage fest:

```
SELECT MaterialID, Bezeichnung FROM tblMaterialien
ORDER BY Bezeichnung;
```

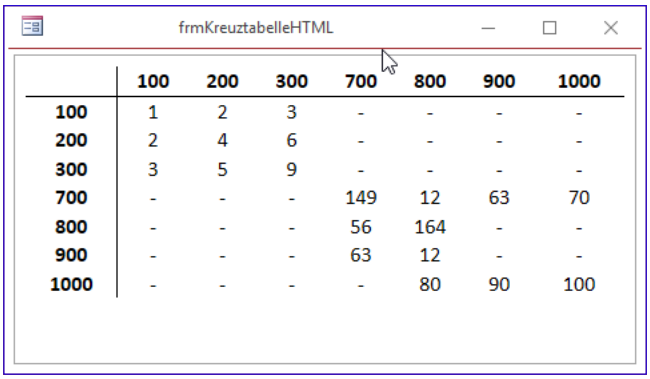
Damit nur das Feld **Bezeichnung** im Kombinationsfeld angezeigt wird und nicht das Primärschlüsselfeld **MaterialID**, stellen wir die Eigenschaft **Spaltenanzahl** auf **2** und **Spaltenbreiten** auf **0cm** ein.

Nun benötigen wir noch eine Ereignisprozedur, die nach der Auswahl eines neuen Eintrags im Kombinationsfeld die Kreuztabelle neu füllt. Der dazu aufgerufenen Prozedur **KreuztabelleErstellen** müssen wir auch noch an irgendeiner Stelle die Information mitgeben, für welches Material das Webbrowser-Steuerelement die Preise anzeigen soll.

Die Ereignisprozedur implementieren wir wie folgt:

```
Private Sub cboMaterialID_AfterUpdate()
    KreuztabelleErstellen Me!cboMaterialID
End Sub
```

Hier haben wir bereits vorgegriffen, auf welche Art wir das betroffene Material übergeben – nämlich als Parameter der Prozedur **KreuztabelleErstellen**. Diese erweitern wir um den entsprechenden Parameter. Außerdem fügen wir der Abfrage, welche die anzuzeigenden Daten für die



	100	200	300	700	800	900	1000
100	1	2	3	-	-	-	-
200	2	4	6	-	-	-	-
300	3	5	9	-	-	-	-
700	-	-	-	149	12	63	70
800	-	-	-	56	164	-	-
900	-	-	-	63	12	-	-
1000	-	-	-	-	80	90	100

Bild 10: Kreuztabelle mit neuen Daten

Kreuztabelle in der Prozedur **KreuztabelleErstellen** ermittelt, ein **WHERE**-Kriterium mit dem Wert des Parameters hinzu. Das sieht dann gekürzt wie folgt aus:

```
Private Sub KreuztabelleErstellen(lngMaterialID As Long)
    ...
    Set rstSpaltenkoepfe = db.OpenRecordset("SELECT _
                DISTINCT Hoehe FROM tblMaterialpreise _
            WHERE MaterialID = " & lngMaterialID & " _
                    ORDER BY Hoehe", dbOpenDynaset)
    ...
    Set rstZeilen = db.OpenRecordset("SELECT DISTINCT _
                    Breite FROM tblMaterialpreise _
            WHERE MaterialID = " & lngMaterialID & " _
                    ORDER BY Breite", dbOpenDynaset)
    Do While Not rstZeilen.EOF
        ...
        Set rstWerte = db.OpenRecordset("SELECT _
                    MaterialpreisID, Preis, Hoehe _
            FROM tblMaterialpreise WHERE MaterialID = " _
                & lngMaterialID & " AND Breite = " & _
      rstZeilen!Breite & " ORDER BY Hoehe", dbOpenDynaset)
        ...
    Loop
End Sub
```

Außerdem müssen wir noch die Prozedur anpassen, die beim Öffnen des Formulars das Webbrowser-Steuerelement füllt und dazu die Prozedur **KreuztabelleErstellen** aufruft:

1:n-Beziehung als Restriktion für Feldwerte

Normalerweise verwenden Sie 1:n-Beziehungen, um die Datensätze zweier Tabellen zu verknüpfen und so Kombinationen aus diesen Datensätzen zu ermöglichen. Oder Sie nutzen die Beziehung für die Verknüpfung mit einer Lookup-Tabelle, welche die Werte für ein Feld liefert. Es geht aber noch anders: Wir wäre es, wenn Sie etwa für ein Feld, dass nur bestimmte Zahlenwerte annehmen können soll, auch eine per 1:n-Beziehung verknüpfte Tabelle hinterlegen? Dieser Beitrag zeigt, welchen praktischen Zweck dies hat und wie Sie dies umsetzen.

Im Beitrag **HTML-Kreuztabelle 2: Werte bearbeiten** (**www.access-im-unternehmen.de/1162**) haben wir eine Tabelle namens **tblMaterialpreise** verwendet, um die Preise für ein Material mit bestimmten Maßen, in diesem Fall Höhe und Breite, zu definieren. Um schon Werte für Höhe und Breite vorzudefinieren, die noch gar nicht in der Tabelle **tblMaterialpreise** enthalten sind, haben wir zwei Tabellen namens **tblHoehen** und **tblBreiten** erstellt. Diese haben ein Primärschlüsselfeld namens **HoeheID**/**BreiteID**, ein Feld für die Angabe des jeweiligen Wertes namens **Hoehe**/**Breite** und ein Fremdschlüsselfeld namens **MaterialID** zur Auswahl des Materials, auf das sich die Höhe oder Breite bezieht, enthalten (siehe Bild 1).

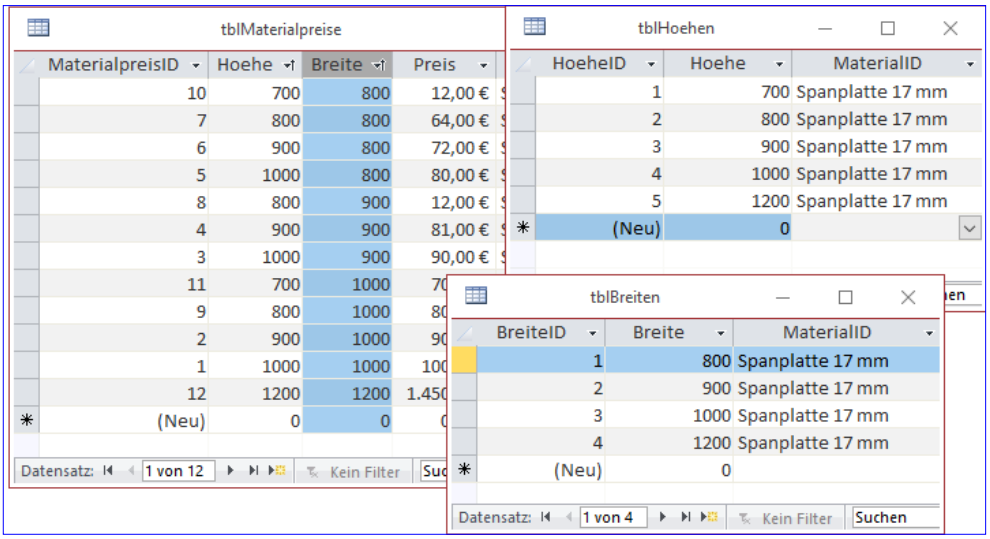


Bild 1: Bisherige Ausstattung der Tabellen **tblMaterialpreise**, **tblHoehen** und **tblTiefen**

Eigentlich wollte ich der Tabelle **tblMaterialpreise** dann statt der bisher vorhandenen Felder **Hoehe** und **Breite**, welche die Angaben mit dem Datentyp **Zahl** in Millimeter gespeichert haben, zwei Fremdschlüsselfelder namens **HoeheID** und **BreiteID** hinzufügen, die jeweils mit der Tabelle **tblHoehen** beziehungsweise **tblBreiten** verknüpft werden sollten. Auf diese Weise hätte man dann über ein Nachschlagefeld die jeweils für den Preis ausschlaggebenden Maße auswählen können.

Wert als Primärschlüssel

Allerdings fiel mir dann auf, dass dies möglicherweise einfacher geht und auf eine andere Weise sogar noch Performance-Vorteile mit sich bringen könnte. Wenn ich die Verknüpfungen wie geplant realisieren würde, müsste Access immer, wenn die Werte der Tabelle **tblMaterialpreise** mit den Werten für die Höhe und Breite ausgegeben werden sollen, die entsprechenden Informationen über die Verknüpfung aus den Tabellen **tblHoehen** und **tblBreiten** abfragen. Das ist aber in diesem Fall gar nicht nötig.

Wenn wir uns ansehen, welche Werte das Feld **Hoehe** in der Tabelle **tblHoehen** und das Feld **Breite** in der Tabelle **tblBreite** annehmen soll, könnte man annehmen, dass jeder Wert in jeder der beiden Tabellen nur einmal vorkommen soll. Das ist jedoch nicht der Fall, denn jeder Wert kann für jede Kombination mit dem über das Fremdschlüsselfeld **MaterialID** maximal einmal vorkommen – bei zwei Materialien kann also auch zweimal der Wert 1.000 mm vorkommen.

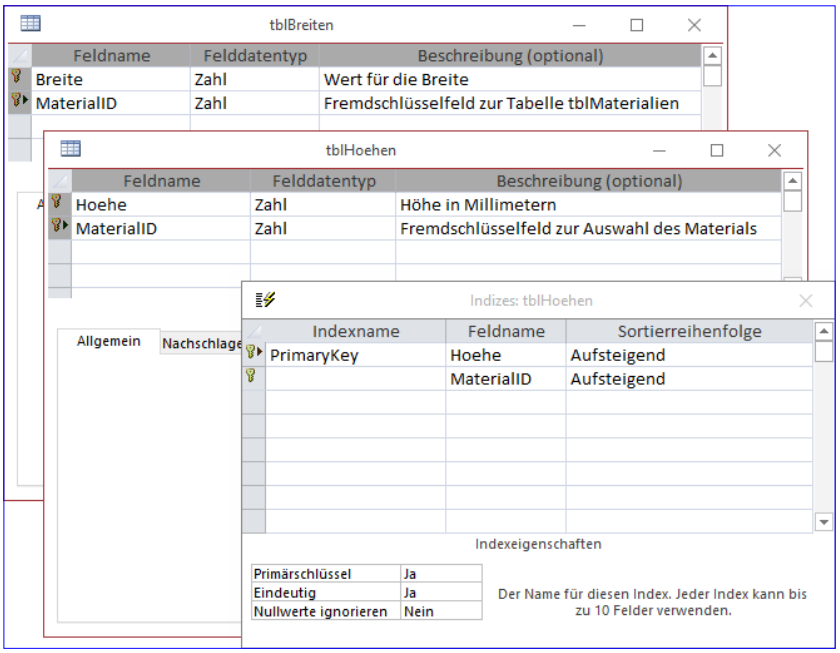


Bild 2: Ersetzen des einfachen Primärschlüsselfeldes durch einen zusammengesetzten Primärschlüssel aus den übrigen Feldern

Wenn die Unterscheidung über das Feld **MaterialID** nicht wäre und wir die Werte nur für ein einziges Material angeben müssten, könnten wir die Tabellen **tblHoehen** und **tblBreiten** mit dem Feld **Hoehe** beziehungsweise **Breite** als Primärschlüsselfeld ausstatten. Da wir allerdings das Feld **MaterialID** berücksichtigen müssen und wir einen zusammengesetzten, eindeutigen Index über die beiden Felder **Hoehe** und **MaterialID** beziehungsweise **Breite** und **MaterialID** haben, müssten wir aus diesen zusammengesetzten, eindeutigen Indizes einen zusammengesetzten Primärschlüssel machen. Das Ergebnis sieht wie in Bild 2 aus. Die noch in **Indizes**-Fenster vorhandenen zusammengesetzten, eindeutigen Indizes aus den Feldern **Hoehe** und **MaterialID** beziehungsweise **Breite** und **MaterialID** entfernen wir.

Verknüpfungen erstellen

Dann haben wir aber auch leider kein einfaches Primärschlüsselfeld mehr, mit dem wir die noch zu erstellenden Fremdschlüsselfelder der Tabelle **tblMaterialpreise** verknüpfen können. Aber das ist kein Problem: Wir erstellen einfach zwei Verknüpfungen!

Die erste legen wir auf dem gewohnten Wege an. Dazu nutzen wir das bereits in der Tabelle **tblMaterialpreise** vorhandene Feld **Hoehe** und wählen als neuen Datentyp in der Entwurfsansicht den Eintrag **Nachschlagefeld** aus. Hier wählen wir die Tabelle **tblHoehen** als Quelltabelle aus und das Feld **Hoehe** als einziges ausgewähltes Feld.

Für dieses Feld legen wir eine aufsteigende Sortierung fest. Interessanterweise finden wir in dem Schritt, der normalerweise die Möglichkeit anbietet, referenzielle Integrität zu definieren, keine solche Einstellung (siehe Bild 3).

Also verzichten wir zunächst darauf und schließen den Dialog über die Schaltfläche **Fertigstellen**. Das Gleiche erledigen für analog für das Feld **Breite** der Tabelle **tblMaterialpreise** und die Tabelle **tblBreiten**.

Schauen wir uns nun das Datenmodell im **Beziehungen**-Fenster an, sieht dies wie in Bild 4 aus. Nun soll der Benutzer etwa für das Feld **Hoehe** der Tabelle **tblMaterialpreise** nur diejenigen Felder der Tabelle **tblHoehen** auswählen können, die im Feld **MaterialID** den gleichen Wert wie die Tabelle **tblMaterialpreise** aufweist.

Um dies zu gewährleisten, müssen wir dann auch noch eine Beziehung zwischen den beiden Feldern **MaterialID** der Tabelle **tblMaterialpreise** und **MaterialID** der Tabelle **tblHoehen** beziehungsweise **tblBreiten** herstellen.

Wie erledigen wir das? Wenn wir es probieren wie beim herkömmlichen Erstellen einer Beziehung zwischen zwei Tabellen im Beziehungen-Fenster, also durch das Ziehen des Fremdschlüsselfeldes der einen Tabelle auf die Zieltabelle, erhalten wir die Meldung aus Bild 5.

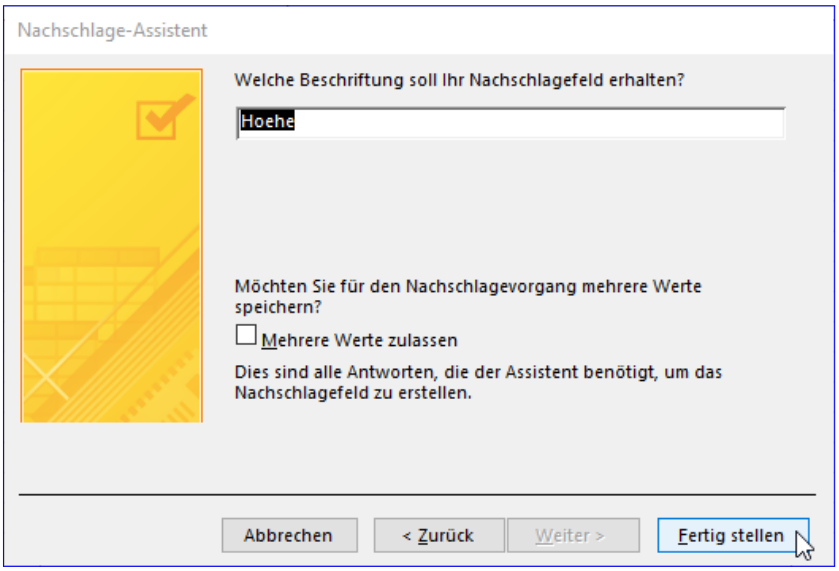


Bild 3: Fehlende Einstellung zum Festlegen referenzieller Integrität

Betätigen wir hier nun die Schaltfläche **Ja**, erscheint der Dialog **Beziehung bearbeiten** aus Bild 6. Hier fügen wir die zweite Beziehung hinzu, indem wir sowohl für die Tabelle **tblHoehen** als auch für die Tabelle **tblMaterialpreise** das Feld **MaterialID** auswählen.

Außerdem aktivieren wir hier die Option **Mit referenzieller Integrität**. Das Gleiche erledigen wir für die Tabellen **tblBreiten** und **tblMaterialpreise**.

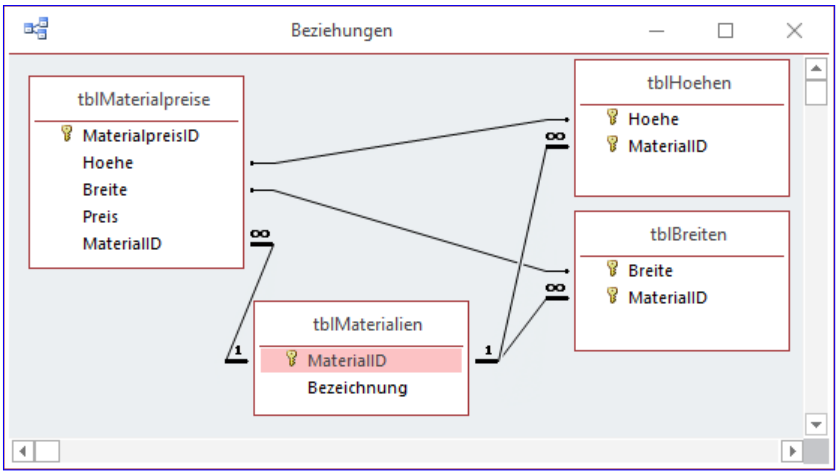


Bild 4: Datenmodell für die Zuordnung der Höhe und der Breite

Wenn wir nun einige weitere Datensätze zur Tabelle **tblHoehen** hinzufügen, die wir einem anderen Wert der Tabelle **tblMaterialien** zuordnen, sieht diese Tabelle wie in Bild 7 aus.

Was wollen wir damit erreichen? Nun, wir wollen sehen, ob wir damit die Auswahl im Nachschlagefeld der Tabelle **tblMaterialpreise** so einschränken können, dass nur die Werte der verknüpften Tabelle **tblHoehen** im Nach-

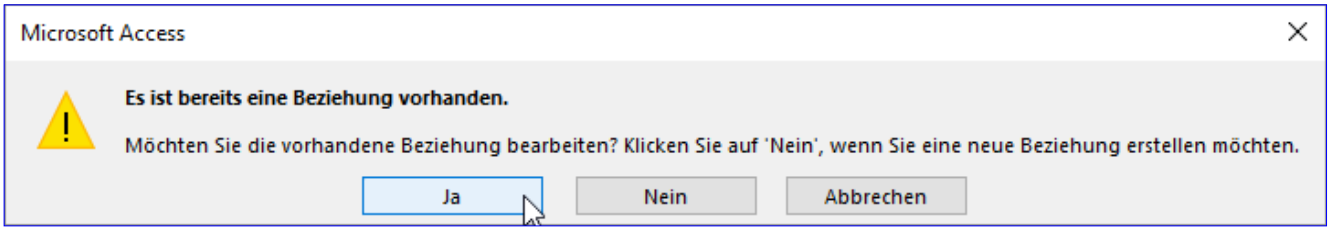


Bild 5: Meldung bei vorhandener Beziehung

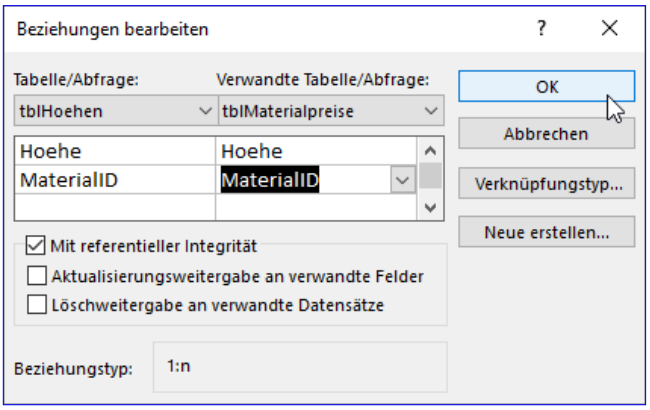


Bild 6: Doppelte Verknüpfung herstellen

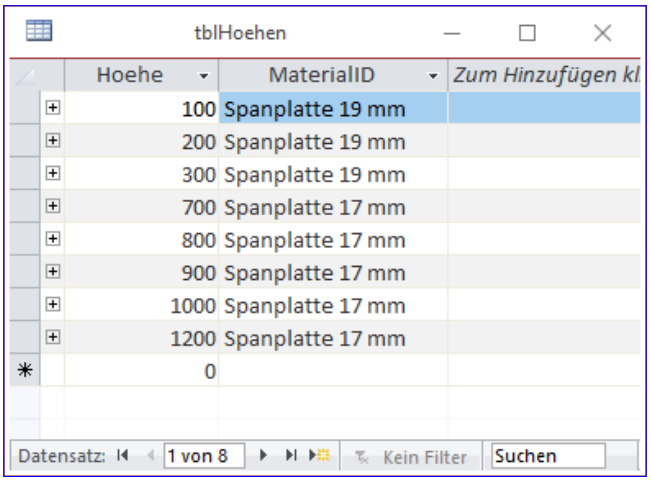


Bild 7: Höhen für verschiedene Spanplattendicken

schlagefeld **Hoehe** angezeigt werden, die auch dem aktuell im Feld **MaterialID** ausgewählten Material entsprechen. Das ist nicht der Fall, wie Bild 8 zeigt.

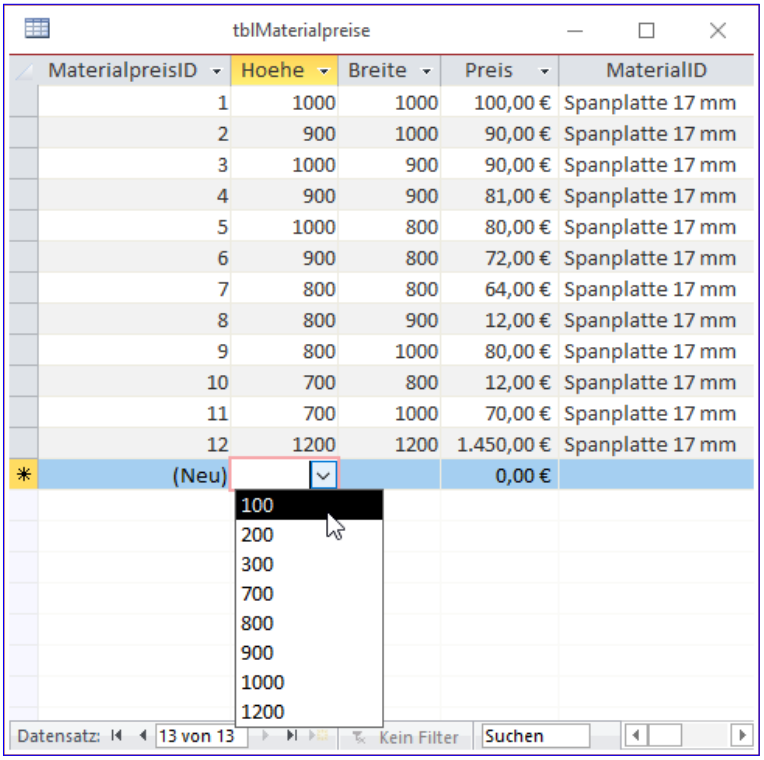


Bild 8: Die Auswahl der Höhen bietet zu viele Datensätze an.

Einschränkung per Abfrage

Diese Einschränkung bekommen wir nur durch die Erstellung einer geeigneten Abfrage hin. Diese sieht im Entwurf wie in Bild 9 aus. Wir fügen der Abfrage die beiden Tabellen **tblMaterialpreise** und **tblHoehen** hinzu. Die beiden vorhandenen Beziehungen werden automatisch in den Abfrageentwurf übernommen. Was noch fehlt, ist die Anpassung der Datensatzherkunft des Nachschlagefeldes **Hoehe** der Tabelle **tblMaterialpreise**. Diese wollen wir so einrichten, dass diese nur solche Einträge der Tabelle **tblHoehen** anzeigt, die den gleichen Wert im Feld **MaterialID** aufweist wie der aktuelle Datensatz der Tabelle **tblMaterialpreise**.

Das realisieren wir durch die Formulierung der Eigenschaft wie folgt:

```
SELECT Hoehe FROM tblHoehen WHERE tblMaterialpreise.MaterialID = tblHoehen.MaterialID;
```

Wenn wir nun in die Datenblattansicht der Abfrage wechseln, sind die Datensätze im Nachschlagefeld **Hoehe** zwar nach den Werten der Tabelle **tblHoehen** für einen der beiden angegebenen Werte von **MaterialID** gefiltert – allerdings immer nur für den Wert von **MaterialID** für den oben befindlichen Datensatz (siehe Bild 10).

Wenn wir zu einem Datensatz wechseln, der mit dem Eintrag **Spanplatte 19 mm** der Tabelle **tblMaterialien**

verknüpft ist und hier das Nachschlagefeld für **Hoehe** öffnen, erhalten wir wieder die Höhen für den Eintrag **Spanplatte 17 mm** (siehe Bild 11).

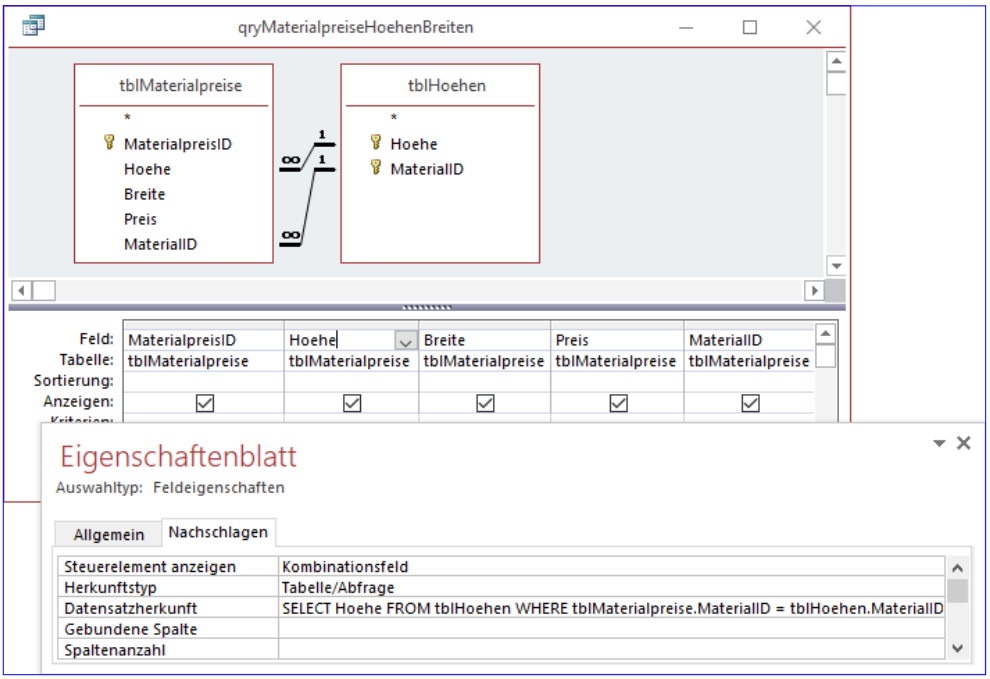


Bild 9: Formulierung der Datensatzherkunft des Nachschlagefeldes

Das ändert sich allerdings, wenn wir nun auf die Schaltfläche **F5** zum Aktualisieren klicken. Dies aktualisiert nicht nur die Abfrage, sondern offensichtlich auch die Datensatzherkunft des Nachschlagefeldes entsprechend des Wertes des zuletzt markierten Datensatzes der Abfrage – auch wenn der Datensatzmarkierer danach wieder auf den ersten Datensatz zurückfällt. Bild 12 zeigt, dass das Nachschlagefeld nun den richtigen Datensatz anzeigt.

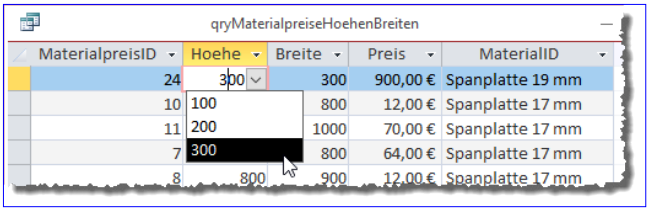


Bild 10: Für diesen Datensatz zeigt das Nachschlagefeld die richtigen Werte an ...

Neuen Datensatz in der Abfrage anlegen

Damit könnte man später im Formular eventuell leben – die Datensatzherkunft ließe sich ja direkt ansteuern und aktualisieren, ohne dass der aktuelle Datensatz verlassen werden muss. Aber was geschieht, wenn wir nun einen über die Abfrage **qryMaterialpreiseHoehenBreiten** einen neuen Datensatz in der Tabelle **tblMaterialpreise** hinzufügen? Wenn wir den Datensatzzeiger auf einen neuen Datensatz einstellen, mit **F5** aktualisieren und dann wieder zum neuen, leeren Datensatz wechseln, ist das Nachschlagefeld leer (siehe Bild 13). Kein Wunder, denn das Feld **MaterialID** ist leer und die Tabelle **tblHoehen** hat keine Datensätze, in denen das Feld auch leer ist.

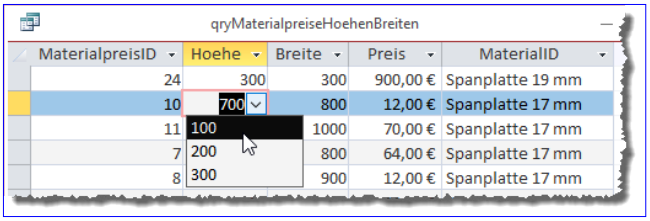


Bild 11: ... für den nächsten nicht mehr.

Also wählen wir hier erst einen Wert für **MaterialID** aus und aktualisieren dann mit **F5**. Dadurch will Access den Datensatzzeiger auf den ersten Datensatz verschieben. Das gelingt allerdings nicht, da der neue Datensatz nicht

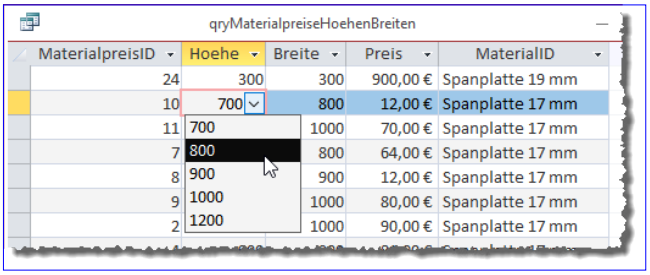


Bild 12: Nach der Aktualisierung klappt es dann.

vollständig ist – die beiden Felder **Hoehe** und **Breite** müssen mit Datensätzen der Tabellen **tblHoehen** und **tblBreiten** verknüpft werden. Also erhalten wir die Meldung aus Bild 14.

Nachdem wir diese bestätigt haben und nochmal das Nachschlagefeld aufklappen, finden wir dort die gewünschten Daten vor (siehe Bild 15). Da wir nun das gewünschte Ergebnis haben, nehmen wir zwei Schritte vor:

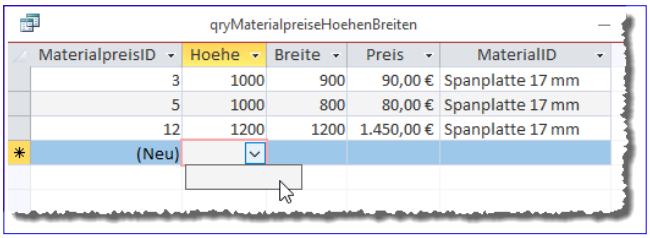


Bild 13: Beim Einfügen eines neuen Datensatzes, bei dem man zuvor mit **F5** die Datensatzherkunft des Nachschlagefeldes aktualisiert hat, ist dieses leer.

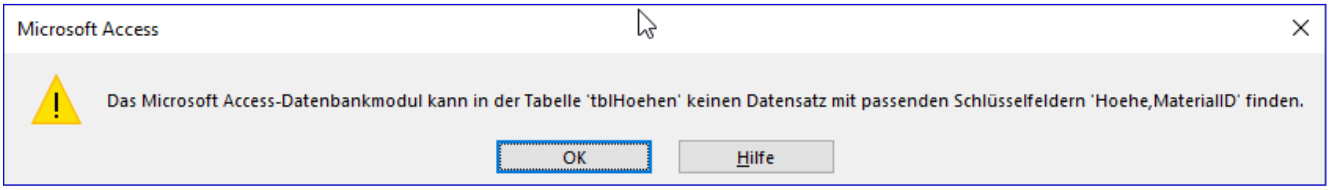


Bild 14: Fehlermeldung beim Verlassen eines unvollständigen Datensatzes

- Erstens die Anpassung des Nachschlagefeldes für das Fremdschlüsselfeld **Breite** und
- zweitens den Bau eines Formulars, das die hier vorkommenden Fehlermeldungen abfängt beziehungsweise die Daten der Nachschlagefelder aktualisiert, ohne Datensatz zu wechseln und somit die Fehlermeldungen überhaupt gar nicht anzeigt.

Den ersten Schritt erledigen wir wie in Bild 16. Wir fügen die Tabelle **tblBreiten** zum Abfrageentwurf hinzu und stellen die Eigenschaften im Bereich **Nachschlagen** wie in der Abbildung ein.

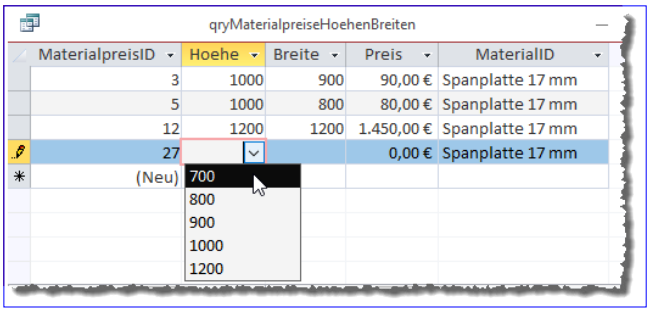


Bild 15: Korrekte Darstellung der gewünschten Daten nach Aktualisierung

Formular erstellen

Für den zweiten Teil erstellen wir ein neues Formular namens **frmMaterialpreiseHoehenBreiten** und öffnen es in der Entwurfsansicht. Stellen Sie die Abfrage **qryMaterialpreiseHoehenBreiten** als Datenherkunft für das Formular ein. Außerdem legen Sie für die Eigenschaft **Standardansicht** den Wert **Datenblatt** fest. Ziehen Sie alle Felder der Datenherkunft aus der Feldliste in den Entwurf des Formulars (siehe Bild 17).

Wechseln Sie nun in die Datenblattansicht, verhält sich das Formular genauso wie die Abfrage. Sie zeigt für die Nachschlagefelder **Hoehe** und **Breite** die Werte an, die zu der **MaterialID** passen, die für den beim Öffnen oben angezeigten Datensatz ausgewählt ist.

Wir wollen nun, dass beim Wechseln des Wertes für das Feld **MaterialID** oder auch beim Wechseln des Datensatzes die zur jeweiligen **MaterialID** gehörenden Werte der beiden Tabellen **tblHoehen** und **tblTiefen** angezeigt werden.

Dazu müssen wir für zwei Ereignisprozeduren entsprechende Prozeduren hinterlegen. Das erste ist das Ereignis **Beim Anzeigen** des Formulars, die beim Wechseln des Datensatzes ausgelöst wird, das zweite das Ereignis **Nach**

Aktualisierung des Nachschlagefeldes **MaterialID**. Dieses benennen wir zuvor in **cboMaterialID** um, genau wie wir die beiden Kombinationsfelder **Hoehe** und **Breite** in **cboHoehe** und **cboBreite** umbenennen. Beide resultierenden Ereignisprozeduren statten wir mit einem Aufruf der Prozedur **NachschlagefelderAktualisieren** aus:

```
Private Sub cboMaterialID_
AfterUpdate()
    NachschlagefelderAktua-
lisieren
End Sub

Private Sub Form_Current()
    NachschlagefelderAktualisieren
End Sub
```

Diese sieht schließlich wie folgt aus:

```
Private Sub NachschlagefelderAktualisieren()
    Dim strSQLHoehe As String
    Dim strSQLBreite As String
    strSQLHoehe = "SELECT Hoehe FROM tblHoehen ⤵
        WHERE tblHoehen.MaterialID = " & Me!cboMaterialID
    strSQLBreite = "SELECT Breite FROM tblBreiten ⤵
        WHERE tblBreiten.MaterialID = " & Me!cboMaterialID
    Me!cboHoehe.RowSource = strSQLHoehe
    Me!cboBreite.RowSource = strSQLBreite
End Sub
```

Die Prozedur stellt jeweils eine SQL-Abfrage für die beiden Kombinationsfelder zusammen, deren Kriterium das Feld **MaterialID** nach dem Wert des Kombinationsfeldes **cboMaterialID** filtert.

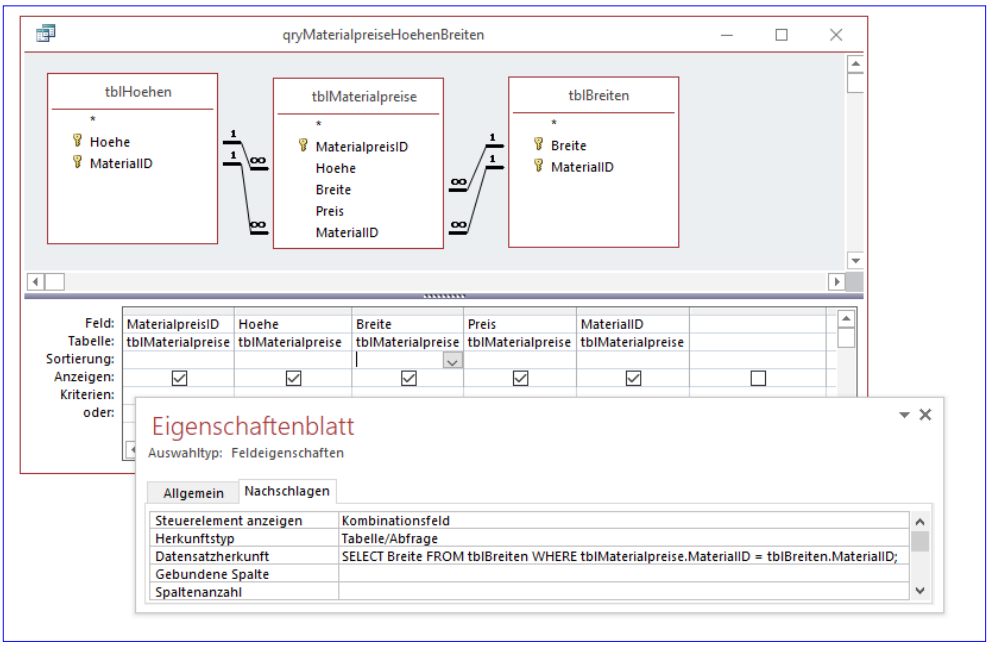


Bild 16: Erweiterung der Abfrage um die Tabelle **tblBreiten** und Einstellung des Nachschlagefeldes

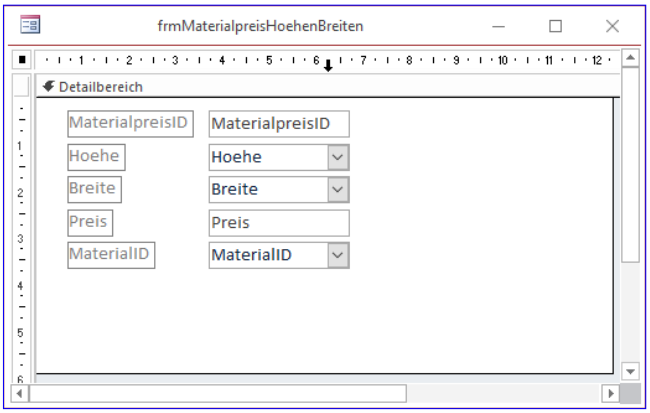


Bild 17: Entwurf des Formulars **frmMaterialpreiseHoehenBreiten**

Wenn wir das Formular **frmMaterialpreiseHoehenBreiten** nun in der Datenblattansicht öffnen, funktioniert die Auswahl bereits wie gewünscht. In Bild 18 haben wir das Öffnen der Auswahl für zwei Datensätze mit verschiedenen Einstellungen für das Feld **MaterialID** dargestellt.

Wenn wir allerdings eines der beiden Kombinationsfelder **cboHoehe** oder **cboBreite** für den neuen, leeren Datensatz öffnen, erhalten wir den Fehler aus Bild 19. Das ist logisch, denn wenn für **cboMaterialID** kein Wert ausgewählt ist, stellt die Prozedur **NachschlagefelderAktua-**

COM-Add-In für den VBA-Editor programmieren

Der VBA-Editor hinkt Visual Studio um Lichtjahre hinterher. Doch es gibt gute Nachrichten: Mit einem COM-Add-In auf Basis von .NET können Sie auch den VBA-Editor noch um Funktionen erweitern. Das zeigen ja auch andere Werkzeugkästen wie etwa die MZ-Tools. Wir wollen in diesem Artikel einmal zeigen, wie Sie ein COM-Add-In in Visual Studio programmieren, das dann beim Öffnen des VBA-Editors zu seiner Erweiterung zur Verfügung steht.

In diesem Artikel schauen wir uns zunächst an, welche Schritte grundsätzlich zum Erstellen eines COM-Add-Ins für den VBA-Editor nötig sind. Die Funktion des COM-Add-Ins beschränken wir auf das Nötigste – darauf gehen wir dann in einem weiteren Artikel ein und fügen dem Add-In dann nützliche Funktionen hinzu.

Neues Projekt erstellen

Visual Studio 2017 stellt zwar Projektvorlagen für COM-Add-Ins für die Office-Anwendungen Excel, Outlook, Word et cetera bereit, jedoch nicht für Access oder den VBA-Editor der Office-Anwendung. Also müssen wir uns selbst behelfen und die notwendigen Elemente selbst zusammenstellen.

Dazu legen Sie unter Visual Studio zunächst ein neues Projekt mit der Vorlage **Visual Basic\|Windows Desktop\|Klassenbibliothek** an und speichern dieses unter dem Namen **VBECOMAddIn** (siehe Bild 1).

Damit erstellen Sie ein neues Projekt, dass zunächst lediglich die Klasse **Class1.vb** enthält.

Verweise hinzufügen

Danach fügen wir dem Projekt einige Verweise hinzu, wobei Sie jeweils über den Menüeintrag **Projekt\|Verweise hinzufügen...** starten. Hier wollen wir als einen Verweis auf die **Extensibility**-Bibliothek hinzufügen. Diese finden wir nur über den **Durchsuchen**-Dialog des Verweis-Managers, den Sie über die entsprechende Schaltfläche öffnen (siehe Bild 2).

Hier finden Sie dann etwa unter dem Verzeichnis **C:\Program Files (x86)\Microsoft Visual Studio 14.0\Visual Studio Tools for Office\PIA\Common** die Datei **Extensibility.dll** (siehe Bild 3).

Achtung: Wenn Sie einfach **Extensibility** in das Suchfeld eingeben, werden Sie vermutlich die falsche Version erwischen, was dazu führen kann, dass das Add-In nicht geladen werden kann!

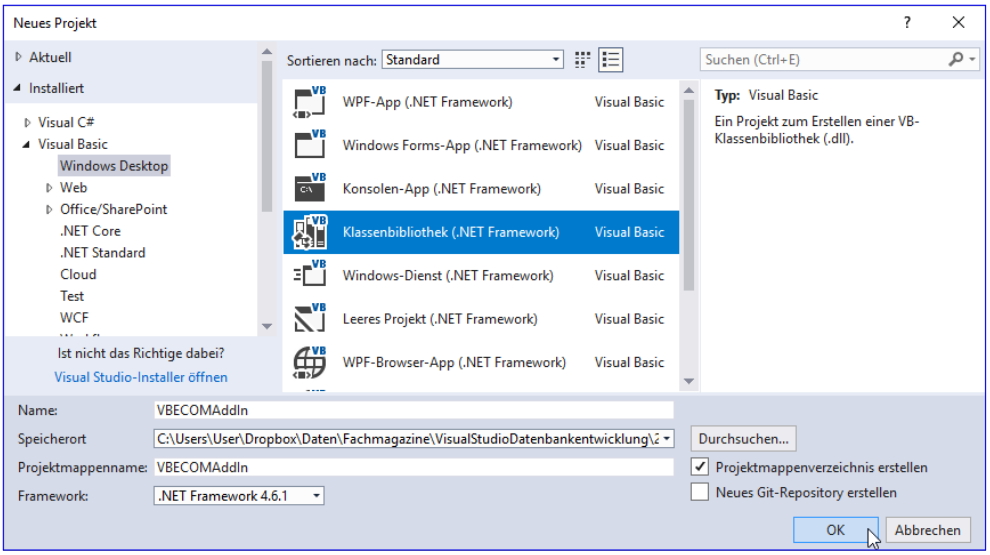


Bild 1: Anlegen des neuen Projekts

Weitere Verweise hinzufügen

Danach ergänzen wir das Projekt um weitere Verweise, wozu wir ebenfalls den **Verweis-Manager** benötigen. Die weiteren Verweise finden Sie im Bereich **Assemblys**, wo Sie am einfachsten den passenden Suchbegriff in das Suchen-Feld eingeben. Die erste so zu ergänzende Bibliothek heißt beispielsweise

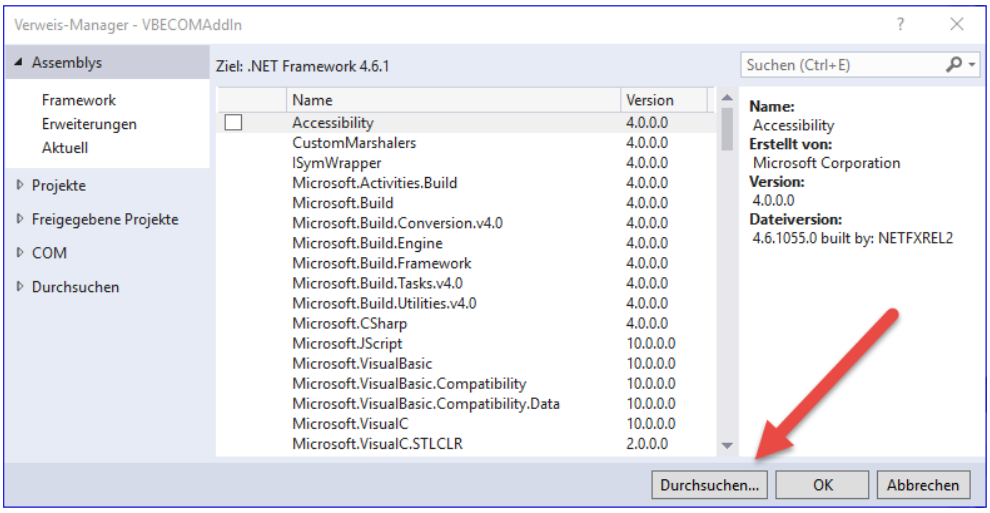


Bild 2: Verwalten der Verweise

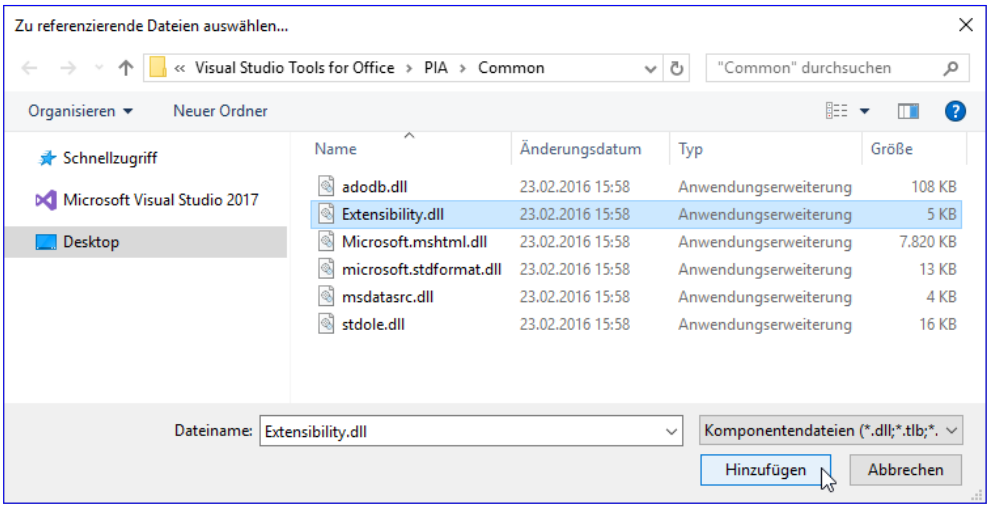


Bild 3: Auswahl der Datei **Extensibility.dll**

Sie ist in zwei verschiedenen Verzeichnissen verfügbar, nämlich in **C:\Program Files (x86)\Microsoft Visual Studio 14.0\Visual Studio Tools for Office\PIA\Office15** und **C:\Program Files (x86)\Microsoft Visual Studio\Shared\Visual Studio Tools for Office\PIA\Office15**. Wir konnten keinen Unterschied bei den beiden DLLs feststellen. Es spielt also keine Rolle, welche Sie verwenden. Also wählen wir einfach einen der beiden aktuelleren Einträge aus (siehe Bild 4). Warum aber überhaupt einen Verweis auf die Access-Bibliothek, obwohl wir doch ein COM-Add-In für den VBA-Editor erstellen wollen? Ganz einfach: So haben wir auch die Mög-

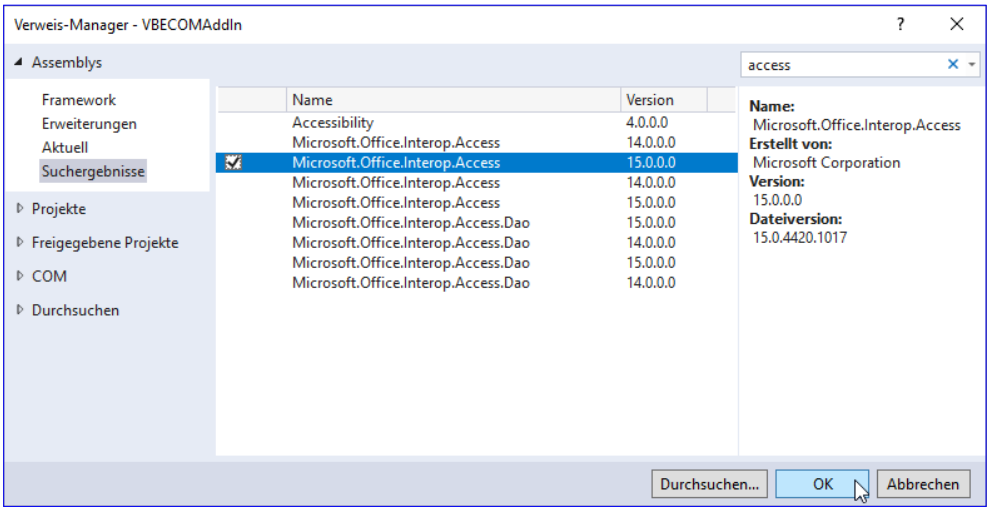


Bild 4: Einbinden des Namespaces **Microsoft.Office.Interop.Access**

lichkeit, einmal auf die Objekte der Access-Datenbank zuzugreifen, um gegebenenfalls etwa VBA-Klassen auf Basis von Tabellen oder Formularen erstellen zu lassen. Auf die gleiche Weise fügen wir nun auch noch Verweise auf die Bibliotheken **Microsoft.Office.Interop.Access.Dao**, **Microsoft.VBE.Interop** und **Microsoft.VBE.Interop.Forms** hinzu. Außerdem benötigen wir etwa für die Anzeige von Meldungsfenstern noch den Namespace **System.Windows.Forms**.

Insgesamt sieht die Liste der Verweise nun wie in Bild 5 aus.

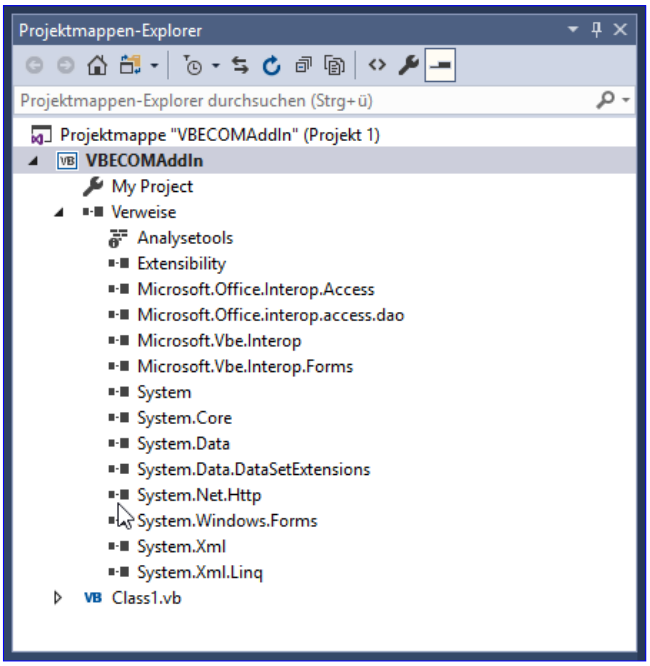


Bild 5: Liste der Verweise des Projekts

Klasse Connect erstellen

Nun fügen wir dem Projekt eine neue Klasse namens **Connect** hinzu. Dazu wählen Sie im Kontextmenü des Eintrags **VBECOMAddIn** des Projektmappen-Explorers den Befehl **Hinzufügen|Klasse...** aus.

Für die neue Klasse vergeben Sie im nun erscheinenden Dialog **Neues Element hinzufügen** den Namen **Connect** (siehe Bild 6).

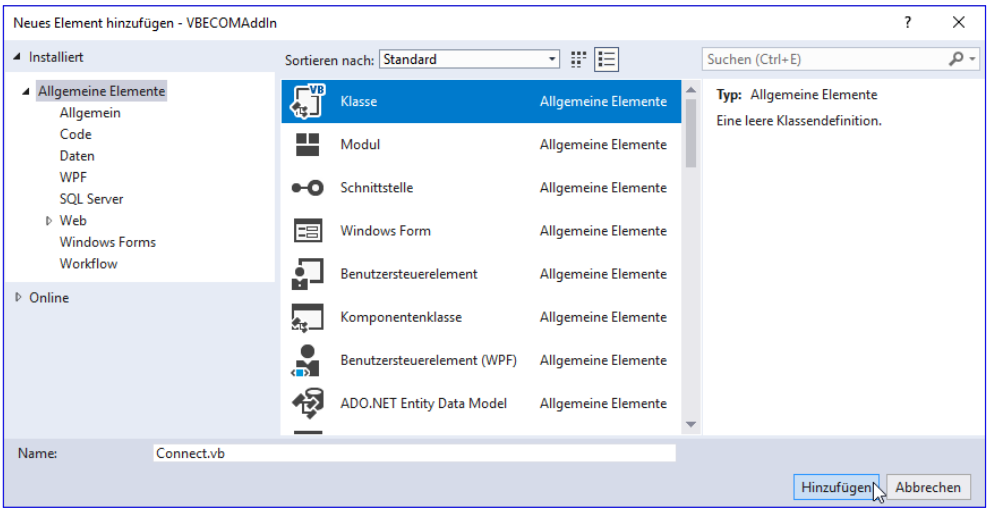


Bild 6: Hinzufügen der Klassendatei **VBECOMAddIn**

Namespaces hinzufügen

Der neuen Klassendatei fügen wir nun zunächst die Verweise auf die Namespaces der soeben hinzugefügten Bibliotheken hinzu, und zwar im Kopf des Moduls:

```
Imports Microsoft.Office.Interop
Imports Extensibility
Imports Microsoft.Office.Interop.Access
Imports Microsoft.Office.Interop.Access.
Dao
Imports System.Windows.Forms
```

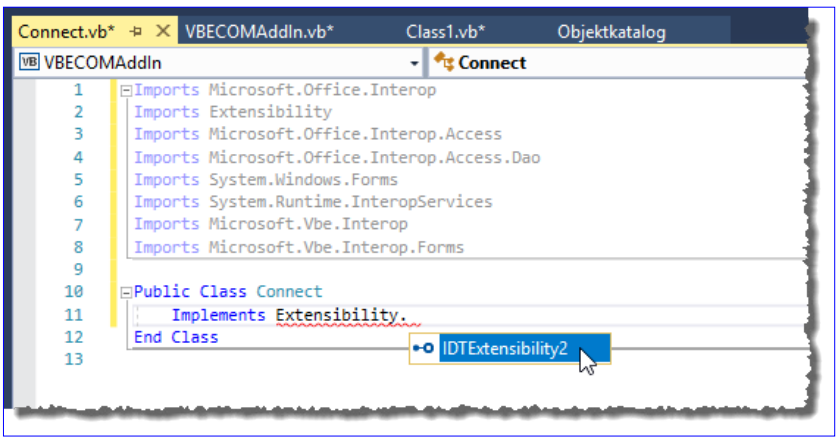


Bild 7: Implementieren der Schnittstelle **Extensibility.IDTExtensibility2**

```
Imports System.Runtime.InteropServices
Imports Microsoft.Vbe.Interop
Imports Microsoft.Vbe.Interop.Forms
```

Extensibility-Schnittstelle implementieren

Danach folgt ein Schritt, mit dem wir die Schnittstelle **Extensibility.IDTExtensibility2** für die Klasse **Connect** implementieren. Dazu fügen wir in der Zeile unterhalb der Zeile **Public Class Connect** die Anweisung **Implements** ein und ergänzen dann wie in Bild 7 zu erkennen auf **Implements Extensibility.IDTExtensibility2**.

Diese Anweisung wird nun noch rot unterstrichen markiert. Der Grund ist einfach: Wir haben zwar angegeben, dass wir die Schnittstelle implementieren wollen, allerdings haben wir nicht alle Member dieser Schnittstelle implementiert. Einen Hinweis darauf liefert auch die Fehlerliste (siehe Bild 8).

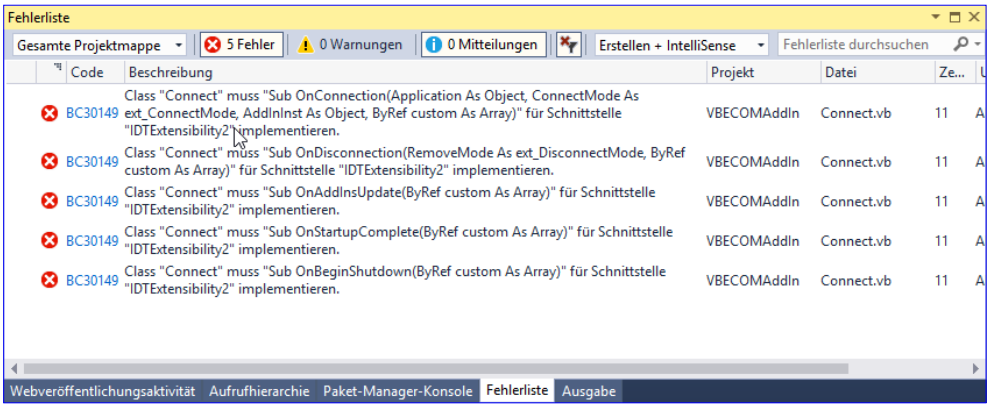


Bild 8: Fehlende Member der Implementierung von **Extensibility.IDTExtensibility2**

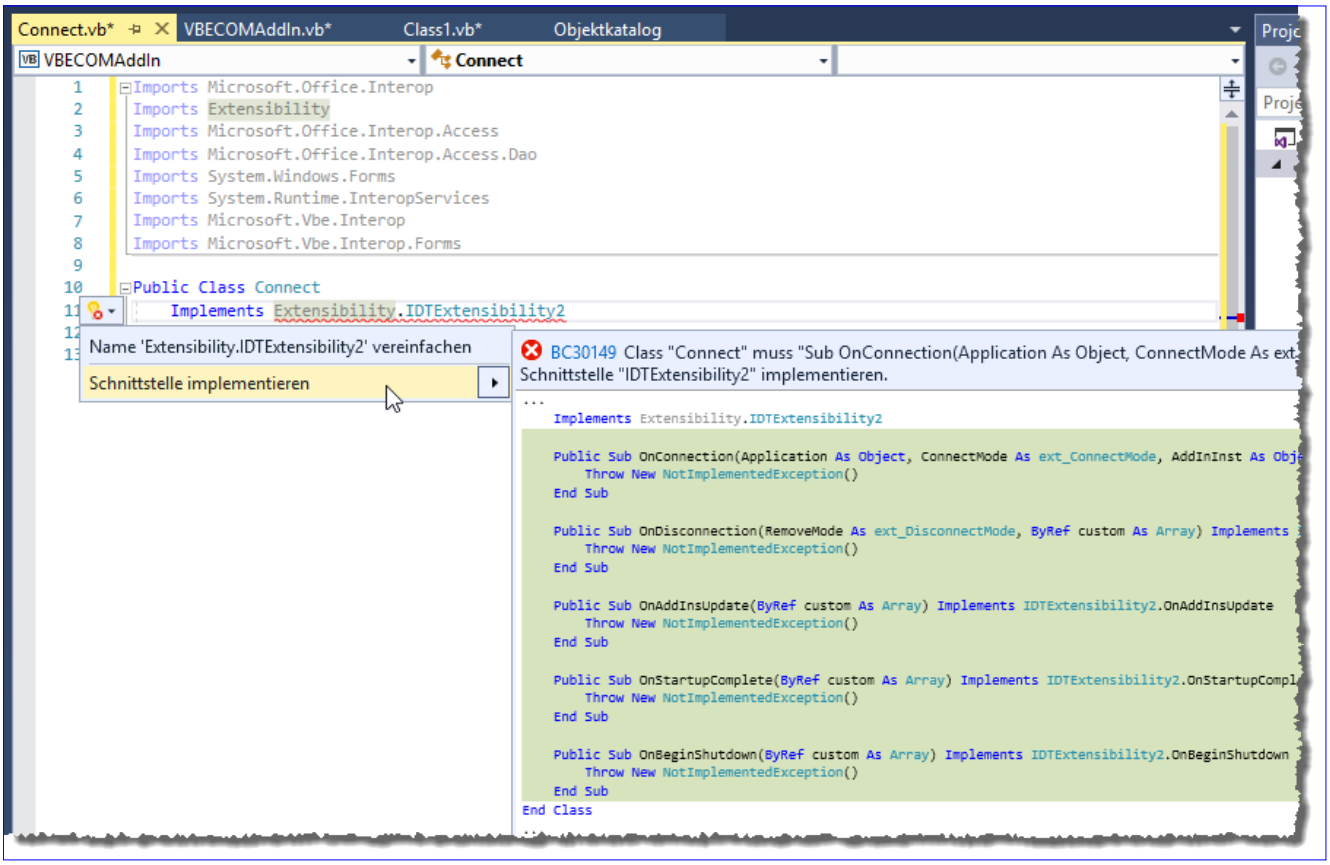


Bild 9: Implementieren per Schnellaktionen und Refactorings

Diesen Fehler können wir allerdings recht schnell beheben. Dazu klicken Sie zunächst mit der rechten Maustaste auf den markierten Code und wählen aus dem Kontextmenü den Eintrag **Schnellaktionen und Refactorings...** aus. Daraufhin erscheint eine Auswahl mit zwei Möglichkeiten, von denen wir den zweiten Eintrag namens **Schnittstelle implementieren** auswählen. Dies zeigt alle Member an, die durch diesen Schritt automatisch zur Klasse hinzugefügt würden (siehe Bild 9).

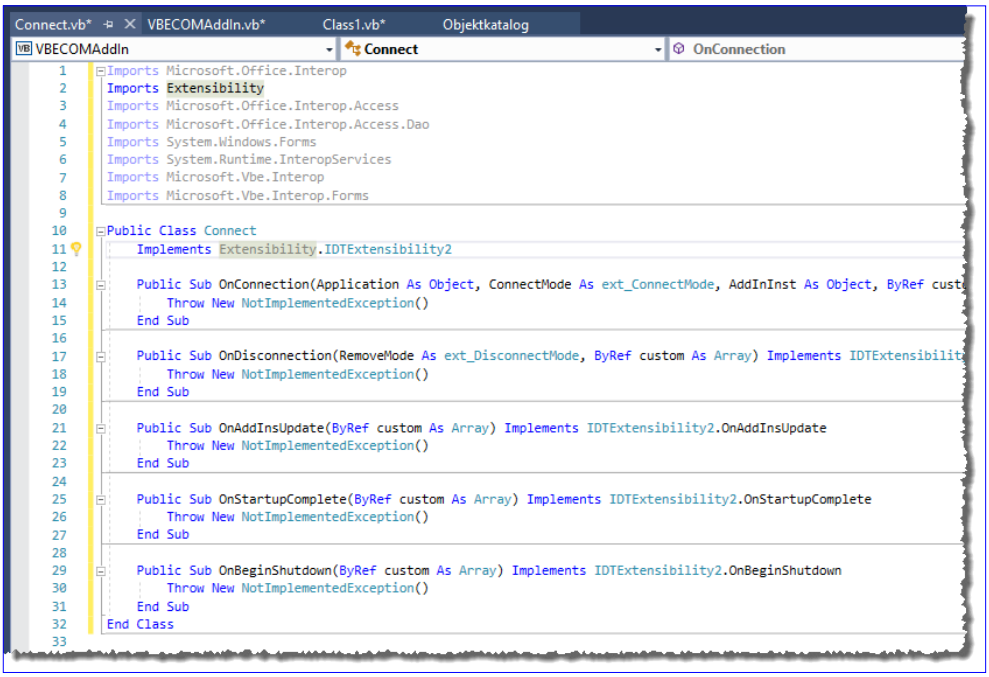


Bild 10: Die Klasse mit allen Membern der Implementierung der Schnittstelle **Extensibility.IDTExtensibility2**

Wählen Sie diesen Eintrag aus, ergänzt diese Schnellaktion die Klasse wie in Bild 10.

Neue GUID ermitteln

Nun benötigen wir eine eindeutige GUID, mit der wir das Add-In versehen. Diese ermitteln wir, indem wir den Menüpunkt **Extras|GUID erstellen** aufrufen. Im nun erscheinenden Dialog wählen wir die letzte Option aus und klicken zum Erstellen der neuen GUID auf die Schaltfläche **Neue GUID** (siehe Bild 11).

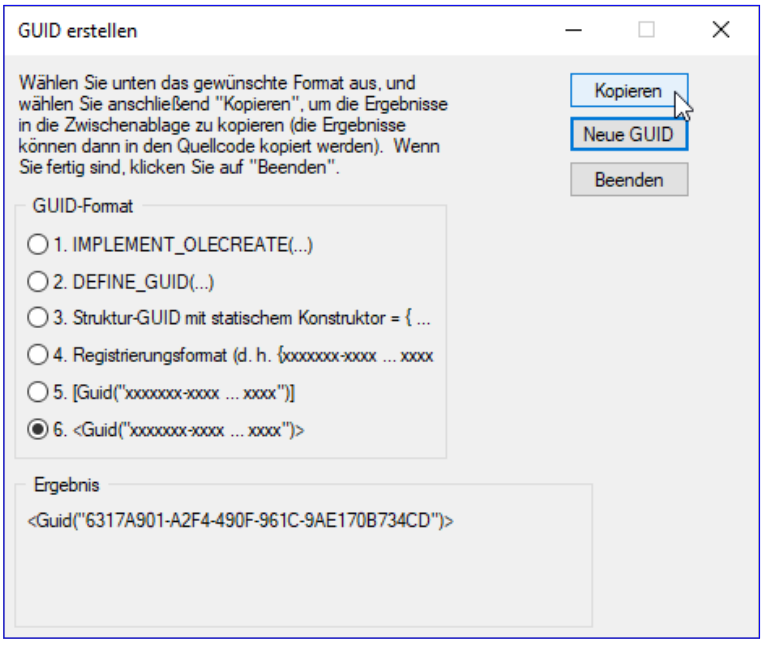


Bild 11: Ermitteln einer neuen GUID

Dann kopieren Sie diese mit der Schaltfläche **Kopieren** in die Zwischenablage. Die GUID hat nicht exakt die Form, die wir benötigen, sodass wir diese noch etwas anpassen müssen.

Dann fügen wir diese in die Attribut-Klasse **ComVisible** direkt über der Zeile **Public Class Connect** ein und legen die **ProgID** auf **VBECOMAddIn.Connect** fest, also auf **<Projektname>.<Klasse>**. Das Ergebnis sieht dann wie folgt aus:

```
<ComVisible(True), Guid("6317A901-
A2F4-490F-961C-9AE170B734CD"),
```

Menüs für das COM-Add-In für den VBA-Editor

Im Beitrag »COM-Add-In für den VBA-Editor« haben wir uns zunächst darum gekümmert, überhaupt eine COM-DLL zu programmieren und diese so in die Registry einzutragen, dass sie beim Starten des VBA-Editors geladen wird und die dort angelegten Ereignisprozeduren ausgelöst wurden. Damit sind wir noch lange nicht fertig. Im vorliegenden Artikel schauen wir uns an, wie Sie dem VBA-Editor Menüeinträge für den Aufruf der im COM-Add-In enthaltenen Funktionen hinzufügen – und zwar für die Menüleiste, die Symbolleiste sowie für Kontextmenüs.

Vorbereitung

Als Vorbereitung für die Beispiele dieses Artikels führen Sie die Schritte aus dem Artikel **COM-Add-In für den VBA-Editor** oder verwenden das dort enthaltene Beispielprojekt als Ausgangspunkt. Wir haben das Projekt einfach neu erstellt, und zwar unter dem Namen **COMAddInMenues**, und eine neue Registrierungsdatei namens **COMAddInMenues.reg** und eine neue GUID erzeugt, die wir an den entsprechenden Stellen eingefügt haben (also in der **.reg**-Datei und über der **Connect**-Klasse). Achten Sie auch darauf, den Namen **COMAddInMenues** an den entsprechenden Stellen der **.reg**-Datei einzufügen. Die Datei sieht für das neue Add-In wie in Listing 1 aus.

Das Ladeverhalten stellen wir gleich auf den Wert **3** ein, damit das Add-In beim Starten des VBA-Editors aufgerufen wird und wir das Ladeverhalten nicht erst noch im Add-In-Manager anpassen müssen.

Die übrigen Schritte wie das Hinzufügen der Verweise und die benötigten Einstellungen haben wir im Artikel **COM-Add-In für den VBA-Editor** beschrieben.

```
Windows Registry Editor Version 5.00
[HKEY_CURRENT_USER\Software\Microsoft\VBA\VBE\6.0\Addins\COMAddInMenues.Connect]
"CommandLineSafe"=dword:00000000
"Description"="Grundgerüst für ein COM-Add-In für den VBA-Editor"
"LoadBehavior"=dword:00000003
"FriendlyName"="COMAddInMenues"
[HKEY_CLASSES_ROOT\WOW6432Node\CLSID\{ABA0B519-CB8C-4A03-9815-D99DFE3312C9}]
@="COMAddInMenues.Connect"
[HKEY_CLASSES_ROOT\WOW6432Node\CLSID\{ABA0B519-CB8C-4A03-9815-D99DFE3312C9}\Implemented Categories]
[HKEY_CLASSES_ROOT\WOW6432Node\CLSID\{ABA0B519-CB8C-4A03-9815-D99DFE3312C9}\InprocServer32]
@="mscoree.dll"
"ThreadingModel"="Both"
"Class"="COMAddInMenues.Connect"
"Assembly"="COMAddInMenues"
"RuntimeVersion"="v2.0.50727"
"CodeBase"="file:///C:\...\COMAddInMenues.dll"
[HKEY_CLASSES_ROOT\WOW6432Node\CLSID\{ABA0B519-CB8C-4A03-9815-D99DFE3312C9}\ProgId]
@="COMAddInMenues.Connect"
```

Listing 1: Inhalt der Datei für die Registry

Denken Sie vor dem Erstellen des Projekts immer daran, Visual Studio im Administrator-Modus zu öffnen, da das COM-Add-In sonst nicht erstellt werden kann.

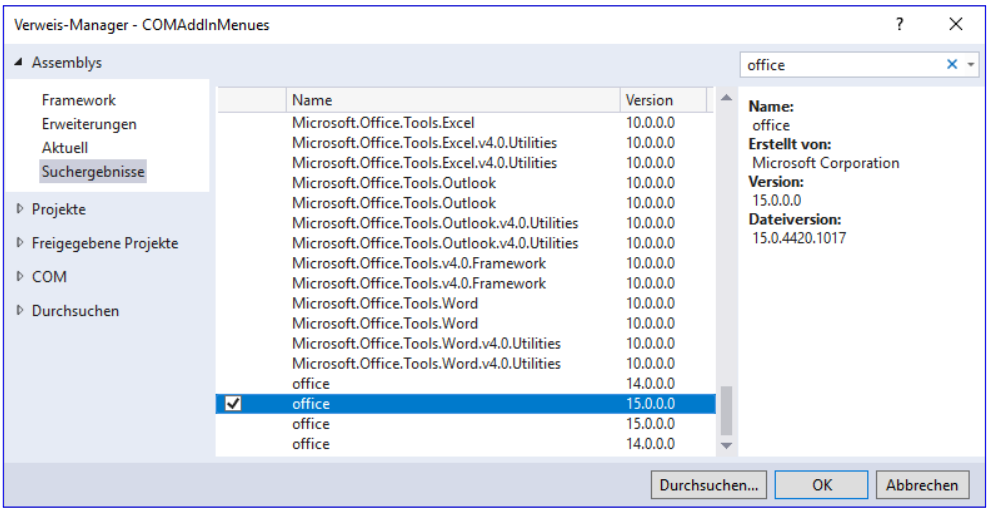


Bild 1: Hinzufügen eines Verweises auf die Office-Bibliothek

Zugriff auf die eingebauten Menüs

Bevor wir eigene Menüpunkte anlegen können, wollen wir uns zunächst ansehen, wie wir überhaupt auf die Menüs des VBA-Editors zugreifen können.

Die Menüs und die für den Zugriff benötigten Objekte wie die **CommandBars**-Auflistung und das **CommandBar**-Objekt sowie deren untergeordnete Elemente und Eigenschaften sind Bestandteil der Office-Bibliothek. Diese fügen Sie über den Verweis-Manager hinzu, indem Sie nach dem Schlüsselwort **Office** suchen und dann den Eintrag aus Bild 1 auswählen.

Außerdem importieren wir den passenden Namespace in die Klasse **Connect.vb**:

```
Imports Microsoft.Office.Core
```

Die Befehle, die sich beim Start des COM-Add-Ins mit den Menüeinträgen befassen, wollen wir gleich in eine eigene Prozedur auslagen. Diese rufen wir gleich beim Start des COM-Add-Ins in der Methode **OnConnection** auf:

```
Public Sub OnConnection(Application As Object, _
        ConnectMode As ext_ConnectMode, _
        AddInInst As Object, _
        ByRef custom As Array) _
        Implements IDTExtensibility2.OnConnection
    _VBE = DirectCast(Application, VBE)
    _AddIn = DirectCast(AddInInst, AddIn)
    Select Case ConnectMode
        Case Extensibility.ext_ConnectMode.ext_cm_Startup
            CommandBarsAusgeben()
    End Select
End Sub
```

Als Erstes wollen wir einfach die Auflistung der **CommandBar**-Elemente durchlaufen und diese in Meldungsfenstern ausgeben. Dazu füllen wir die Prozedur **CommandBarsAusgeben** wie folgt:

```
Private Sub CommandBarsAusgeben()
    Dim cbrs As CommandBars
    Dim cbr As CommandBar
    cbrs = _VBE.CommandBars
    For Each cbr In cbrs
        MessageBox.Show(cbr.Name)
    Next
End Sub
```

Diese Prozedur verwendet die bereits beim Laden des COM-Add-Ins mit einem Verweis auf das Objekt **VBE** gefüllte Member-Variablen **_VBE**. **VBE** steht stellvertretend für das Objektmodell des VBA-Editors.

Diese stellt unter anderem die Auflistung **CommandBars** bereit, die alle **CommandBar**-Elemente des VBA-Editors

enthält. Die Variable **cbrs** mit dem Typ **CommandBars** füllen wir mit einem Verweis auf die **CommandBars**-Auflistung des **VBE**-Objekts. In einer **For Each**-Schleife durchlaufen wir dann alle Elemente dieser Auflistung und geben den Namen des aktuellen Elements in einer Meldung aus.

Debuggen des Add-Ins

Das ist allerdings nicht wirklich komfortabel – immerhin gibt es einige Menüs in der Auflistung **CommandBars**. Wir wollen diese lieber in Visual Studio ausgeben, statt für jedes Menü ein neues Meldungsfenster zu öffnen. Wenn wir in Visual Studio den Menüeintrag **Debuggen|Debugging starten...** wählen, führt dies allerdings zur Meldung aus Bild 2.

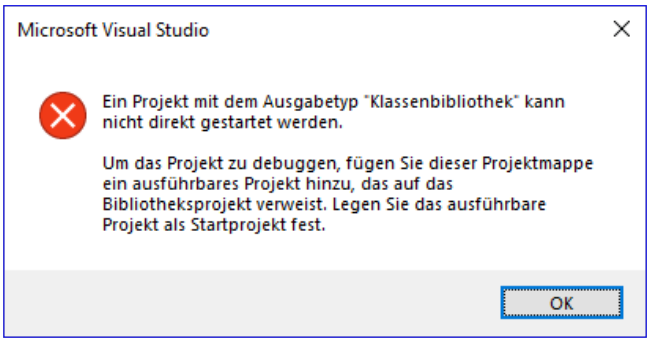


Bild 2: Fehlermeldung beim Versuch, das Projekt zu debuggen

Um dies zu ändern, müssen wir in den Eigenschaften des Projekts eine Anwendung festlegen, die zusammen mit dem Add-In gestartet wird. Das Add-In selbst ist nämlich keine ausführbare Datei. Wir wollen das Add-In mit dem VBA-Editor unter Access testen, also legen wir Access als Startanwendung fest.

Dazu öffnen Sie die Eigenschaften des Projekts und wechseln dort zum Bereich **Debuggen**. Hier wählen Sie unter **Projekt starten** die Option **Externes Programm starten** aus und wählen über den mit der Schaltfläche **Durchsuchen...** zu öffnenden Dialog die Datei **MSAccess.Exe** aus (siehe Bild 3).

Danach können Sie mit dem Menübefehl **Debuggen|Debugging starten** das Debuggen beginnen. Visual Studio startet dann Microsoft Access und wird, wenn Sie den VBA-Editor von Access aus aufrufen, die für das Laden des Add-Ins bestimmten Ereignisse auslösen. Sie können nun also beispielsweise Haltepunkte im Code platzieren und den Code so schrittweise durchlaufen. Beenden Sie das Debugging von Visual Studio aus, wird die gestartete Instanz von Visual Studio ebenfalls beendet.

Gegebenenfalls stoßen Sie dabei auf eine Fehlermeldung, dass die DLL nicht erzeugt werden konnte. Dies kann der Fall sein, wenn Sie bereits zuvor den VBA-Editor für eine Anwendung geöffnet haben und diese die bestehende DLL referenziert hat. Visual Studio kann diese dann nicht neu erstellen.

Wenn wir den Befehl in der Schleife wie folgt ändern, erhalten wir die Liste aller Menüs des VBA-Editors wie in Bild 4:

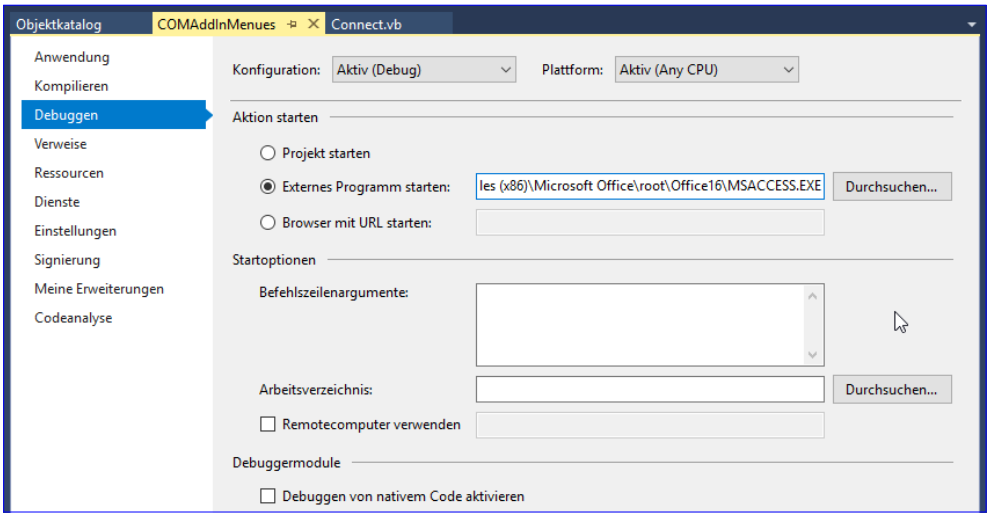


Bild 3: Startanwendung festlegen

```
For Each cbr In cbrs
    Debug.Print(cbr.Name)
Next
```

Nun erweitern wir die Schleife so, dass auch noch die Steuerelemente eines jeden Menüs ausgeben werden. Dazu deklarieren wir eine Variable für die Steuerelemente mit dem Typ **CommandBarControl**:

```
Dim cbc As CommandBarControl
```

In der Schleife fügen wir eine weitere Schleife über alle Elemente der **Controls**-Auflistung hinzu und geben die enthaltenen Elemente ebenfalls im Direktbereich von Visual Studio aus:

```
For Each cbr In cbrs
    Debug.Print(cbr.Name)
    For Each cbc In cbr.Controls
        Debug.Print("  " + cbc.Caption)
    Next
Next
```

Für den Eintrag **Menüleiste** sieht das Ergebnis etwa wie folgt aus:

```
Menüleiste
  &Datei
  &Bearbeiten
  ...
  Add-&Ins
  &Fenster
  &?
```

Genau genommen handelt es sich hierbei gar nicht um Schaltflächen, sondern um Untermenüs, die wiederum Control-Elemente enthalten – hier finden sich dann die eigentlichen Befehle.

Eine weitere Schleife wollen wir nun nicht mehr unterbringen, das System sollte nun verständlich sein.

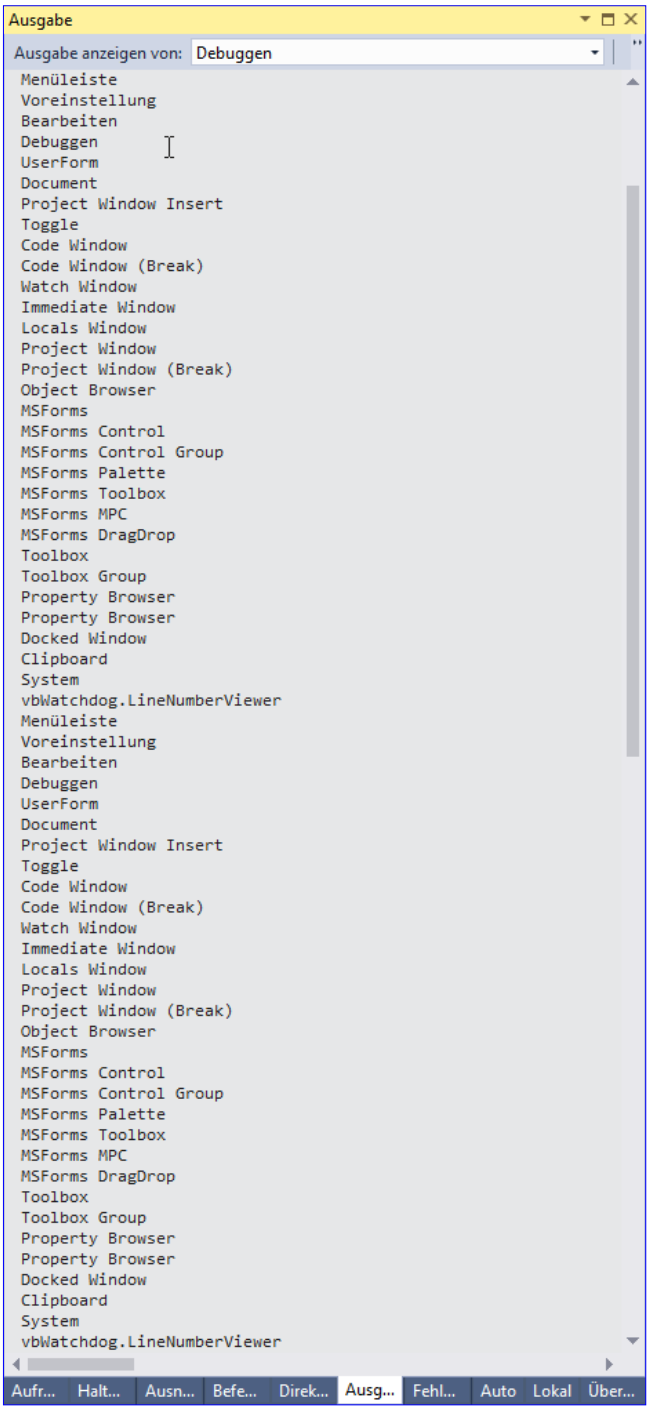


Bild 4: Ausgabe der Menüs des VBA-Editors

Menü hinzufügen oder nur Steuerelemente?

Wir können nun eigenen Menüs hinzufügen, bestehenden Menüs neue Untermenüs zuweisen oder auch bestehenden und neuen Menüs neue Steuerelemente zuweisen. In

der Regel werden wir jedoch neue Steuerelemente zu bestehenden Menüs hinzufügen, um diese um die Funktionen des COM-Add-Ins zu erweitern.

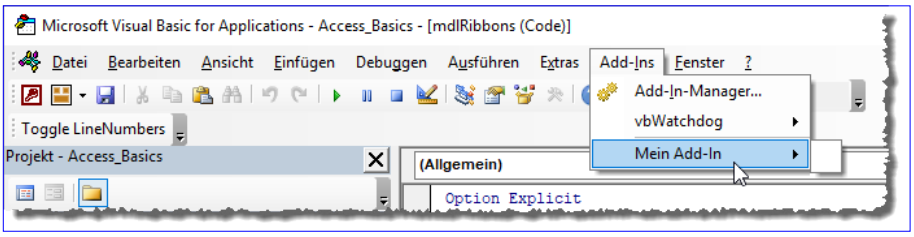


Bild 5: Unser neues Untermenü

Steuerelement hinzufügen

Dazu müssen wir zunächst einmal herausfinden, welchem Menü wir unseren Befehl hinzufügen wollen. In diesem Fall wollen wir, da wir ja noch keinen konkreten Anwendungszweck haben, einfach einmal einen Befehl zum Menüpunkt **Add-Ins** hinzufügen.

Damit es nicht zu leicht wird, soll dieses Menü noch ein Untermenü erhalten, dem wir dann unsere Schaltfläche hinzufügen. Gerade beim Auflisten des eingebauten Menüs namens **Menüleiste** haben wir ja bereits den Namen ermittelt.

Diesen können wir nun zum Referenzieren des entsprechenden **CommandBar**-Elements verwenden. Darunter finden wir das Untermenü namens **Add-&Ins**, das wir ebenfalls referenzieren. Schließlich legen wir darin mit der **Add**-Methode der **Controls**-Auflistung ein neues Unterelement des Typs **msoControlPopup** an, was dem Untermenü entspricht, und weisen diesem die Beschriftung **Mein Add-In** zu.

Außerdem soll vor diesem Untermenü ein Trennstrich eingefügt werden, was wir mit dem Wert **True** für die Eigenschaft **BeginGroup** erreichen:

```
Private Sub SteuerelementHinzufuegen()
    Dim cbr As CommandBar
    Dim cbrAddIns As CommandBarPopup = Nothing
    Dim cbrSub As CommandBarPopup = Nothing
    cbr = _VBE.CommandBars("Menüleiste")
    cbrAddIns = cbr.Controls.Item("Add-&Ins")
    cbrSub = cbrAddIns.Controls.Add(MsoControlType.msoControlPopup)
    With cbrSub
        .Caption = "Mein Add-In"
        .BeginGroup = True
    End With
End Sub
```

Den Aufruf dieser Methode fügen wir schließlich noch der Ereignismethode **OnStartupComplete** zu:

```
Public Sub OnStartupComplete(ByRef custom As Array)
        Implements IDTExtensibility2.OnStartupComplete
    SteuerelementHinzufuegen()
End Sub
```

Danach starten wir das Projekt, was automatisch Access öffnet, und laden dann eine Datenbank, für die wir den VBA-Editor öffnen. Das Add-Ins-Menü sieht dann nach dem Aufklappen wie in Bild 5 aus. Dieses zeigt dann natürlich noch keine Befehle an, da wir noch keine hinzugefügt haben.

Schaltfläche zum Untermenü hinzufügen

Die Schaltfläche zum Ausführen eines benutzerdefinierten Befehls fügen wir nun hinzu. Dazu deklarieren wir zunächst eine Variable namens **cbc** mit dem Typ **CommandBarButton** in der Methode **SteuerelementHinzufuegen**. Diese fügen wir über die **Add**-Methode der **Controls**-Auflistung des soeben erstellten Untermenüs aus **cbrSub** hinzu und legen dabei den Typ des Steuerelements auf **msoControlButton** fest. Schließlich füllen wir die Eigenschaft **Caption** mit dem Wert **Meine Schaltfläche**:

```
Private Sub SteuerelementHinzufuegen()
    ...
```

```
    Dim cbc As CommandBarButton = Nothing
    ...
    With cbrSub
        ...
        cbc = .Controls.Add(MsoControlType.msoControlButton)
        With cbc
            .Caption = "Meine Schaltfläche"
        End With
    End With
End Sub
```

Die Schaltfläche erscheint nun so wie in Bild 6.

Bild 6: Neue Schaltfläche im Untermenü

Schaltfläche mit Funktion versehen

Nun soll die Schaltfläche auch noch eine Aufgaben ausführen – in diesem Fall beispielsweise einfach das Anzeigen eines Meldungsfensters. Dazu müssen wir irgendwie festlegen, dass wir für das mit **cbc** deklarierte Element Ereignisse implementieren können.

Dazu versehen wir die Deklaration der Variablen mit dem zusätzlichen Schlüsselwort **WithEvents**.

Eine solche Deklaration ist allerdings innerhalb einer Methode unzulässig, die Variable muss daher unabhängig von der Methode deklariert werden. Wir verschieben die folgende Zeile daher in den allgemeinen Teil der Klasse:

```
Dim WithEvents cbc As CommandBarButton = Nothing
```

Das Anlegen der Ereignisprozedur, die beim Anklicken des Menüeintrags ausgelöst wird, legen wir nun auch auf einfache Weise an.

Dazu wählen Sie im mittleren Kombinationsfeld oben im Codefenster den Eintrag **cbc** aus und selektieren dann im rechten Kombinationsfeld den Eintrag für die gewünschte Ereignismethode – in diesem Fall gibt es nur eine einzige Auswahlmöglichkeit, nämlich **Click** (siehe Bild 7).

```
Private Sub cbc_Click(Ctrl As CommandBarButton, 7
        ByRef CancelDefault As Boolean) Handles cbc.Click
    MessageBox.Show("Klick auf meinen Menübefehl")
End Sub
```

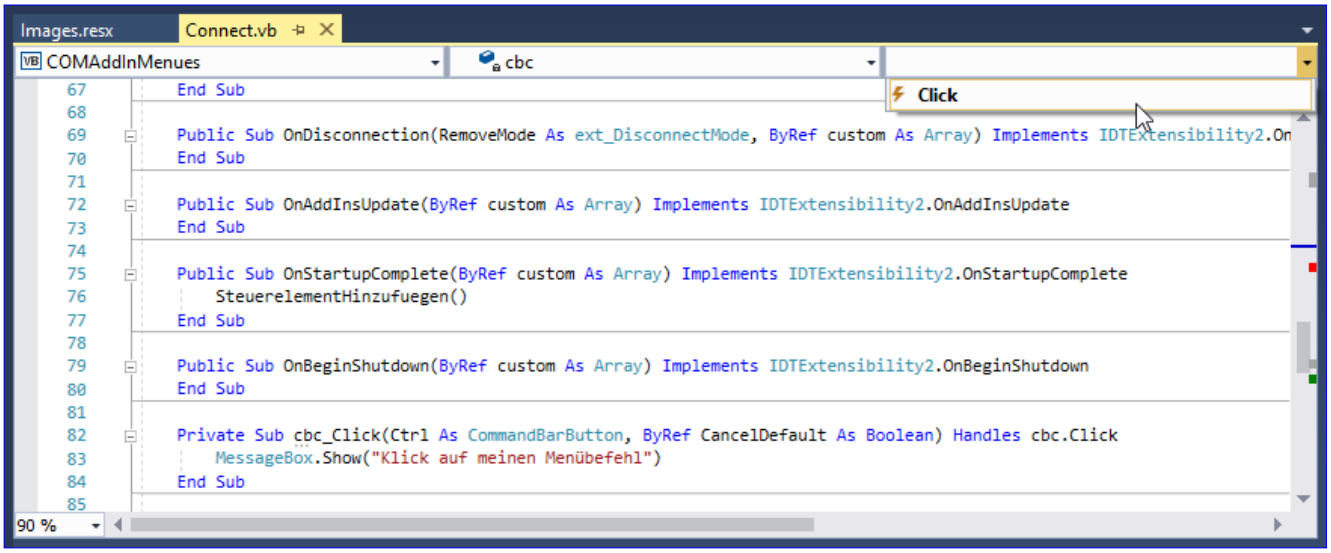


Bild 7: Anlegen der neuen Ereignisprozedur

Mehrere Felder gleichzeitig durchsuchen

Normalerweise legen ein Textfeld an, in das Sie einen Suchbegriff eingeben und die Daten dann in einem oder mehreren Feldern nach diesem Suchbegriff durchsuchen. Oder Sie haben mehrere Suchfelder etwa für Vorname, Nachname et cetera. Dieser Beitrag stellt ein Suchsteuerelement vor, mit dem Sie gezielt nach den Inhalten verschiedener Felder gleichzeitig suchen können. Dabei gibt es mehrere Vorlagen, die vorab festgelegt werden und die der Benutzer dann einstellt, um die Suchbegriffe einzugeben.

Vorbereitung

Als Vorbereitung statten wir eine Beispieldatenbank mit den Tabellen der Datenbank **Suedsturm** aus, von der wir vor allem die Tabelle **tblKunden** benötigen.

Danach erstellen wir ein Unterformular namens **sfmKunden**, dessen Eigenschaften **Datensatzquelle** wir mit der Tabelle **tblKunden** füllen. Danach ziehen wir alle Felder dieser Tabelle aus der Feldliste in den Detailbereich der Entwurfsansicht. Stellen Sie die Eigenschaft **Standardansicht** auf den Wert **Datenblatt** ein und speichern und schließen Sie das Formular.

Anschließend erstellen wir ein weiteres Formular namens **frmKundenEinfacheSuche**. Diesem fügen wir ein Kombinationsfeld namens **cboEinfacheSuche** hinzu. Darunter fügen wir das Unterformular ein, indem wir das Formular **sfmKunden** aus dem Navigationsbereich in den Formularentwurf ziehen. Für dieses Steuerelement stellen wir die beiden Eigenschaften **Horizontaler Anker** und **Vertikaler Anker** auf den Wert **Beide** ein. Das Ergebnis sieht wie in Bild 1 aus.

Einfache Suchfunktion

Zum Warmwerden bauen wir eine einfache Suchfunktion ein, die nur die Datensätze des Feldes **Firma** nach dem eingegebenen Begriff durchsucht. Diese soll durch das Betätigen der Eingabetaste nach der Eingabe des Suchbegriffs beziehungsweise beim Verlassen des Kombinationsfeldes **cboEinfacheSuche** ausgelöst werden. Dafür ist das Ereignis **Nach Aktualisierung** verantwortlich, für das wir die folgende Ereignisprozedur hinterlegen:

```
Private Sub cboEinfacheSuche_AfterUpdate()
    Dim strFilter As String
    Dim strVergleichsausdruck As String
    strVergleichsausdruck = Nz(Me!cboEinfacheSuche, "")
    If Len(strVergleichsausdruck) > 0 Then
        strFilter = "Firma LIKE '" _
            & strVergleichsausdruck & "'"
```

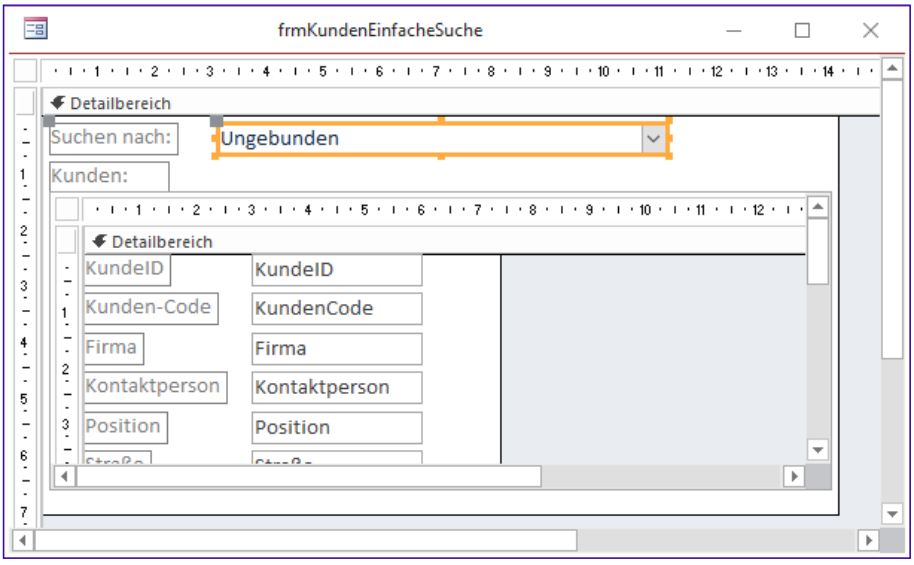


Bild 1: Suchkombinationsfeld und Unterformular

```
    End If
    With Me!sfmKunden.Form
        .Filter = strFilter
        .FilterOn = True
    End With
End Sub
```

Diese einfache Suche liest den Inhalt des Kombinationsfeldes in die Variable **strVergleichsausdruck** ein, wobei der Benutzer Platzhalter wie das Sternchen (*) selbst eingeben muss. Dann prüft die Prozedur, ob die Länge des in **strVergleichsausdruck** enthaltenen Strings größer als **0** ist. Falls ja, wird dieser zu einem Filterausdruck wie **Firma LIKE '<Suchausdruck>'** zusammengesetzt, wobei **<Suchausdruck>** dem eingegebenen Ausdruck entspricht. Anderenfalls wird der Filterausdruck in **strFilter** gar nicht gefüllt. In beiden Fällen wird der Inhalt von **strFilter** der Eigenschaft **Filter** des **Form**-Objekts des Unterformulars zugewiesen und die Eigenschaft **FilterOn** auf **True** eingestellt. Das Ergebnis etwa nach Eingabe des Ausdrucks **B*** sieht wie in Bild 2 aus. Das Leeren des Kombinationsfeldes und das Bestätigen dieser Eingabe zeigt wieder alle Einträge der Datensatzquelle im Unterformular an.

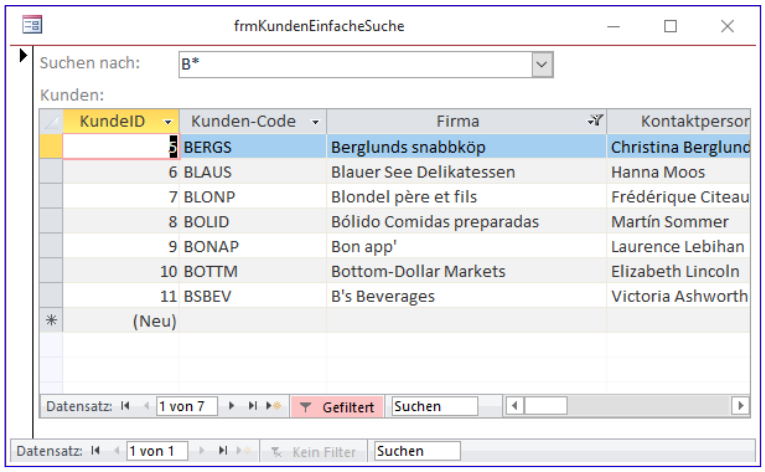


Bild 2: Die einfache Suche in Aktion

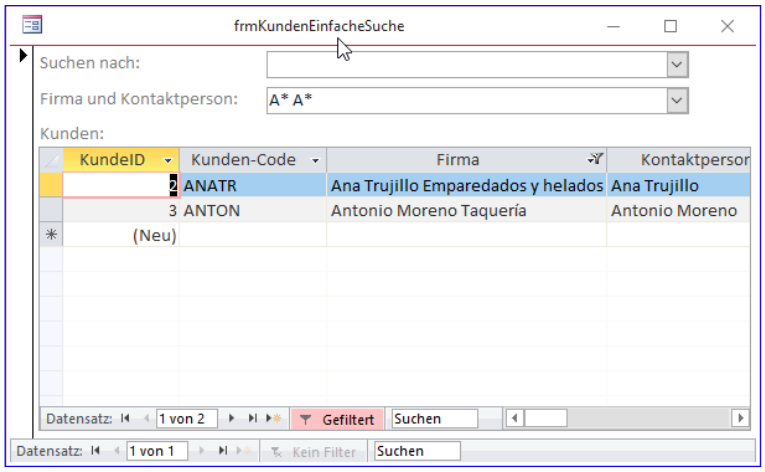


Bild 3: Suche nach zwei Begriffen in zwei Feldern

Zwei Vergleichsausdrücke für zwei Felder gleichzeitig

Nun wollen wir uns dem gewünschten Ergebnis annähern. Wir wollen ja Vergleichsausdrücke für mehrere Felder gleichzeitig eingeben können. Dazu fügen wir dem Formular ein weiteres Kombinationsfeld namens **cboFirmaUndKontaktpersonFiltern** hinzu.

Das Ergebnis nun soll sein, dass wir zwei durch Komma getrennte Vergleichswerte für das Feld **Firma** und das Feld **Kontaktperson** eingeben können und dass die beiden Felder dann nach diesen beiden Ausdrücken gefiltert werden. Das Ergebnis soll dann etwa wie in aussehen, wo wir alle Kunden ermitteln wollen, bei denen sowohl die Firma als auch der Name der Kontaktperson mit **A** beginnt (siehe Bild 3).

Wie sieht nun die Prozedur aus, die durch das Ereignis **Nach Aktualisierung** des Kombinationsfeldes **cboFirmaUndKontaktpersonFiltern** ausgelöst wird? Diese beginnt ähnlich wie die vorherige Prozedur – mit der Ausnahme, dass wir ein paar weitere Variablen deklarieren:

```
Private Sub cboFirmaUndKontaktpersonFiltern_AfterUpdate()
    Dim strFilter As String
    Dim strVergleichsausdruck As String
    Dim strFirma As String
```

```
    Dim strKontaktperson As String
    strVergleichsausdruck = _
            Nz(Me!cboFirmaUndKontaktpersonFiltern, "")
```

Enthält die Variable **strVergleichsausdruck** einen Wert, weisen wir der Variablen **strFirma** den Teil aus **strVergleichsausdruck** bis zum ersten Leerzeichen zu und stellen den ersten Teil des Filterausdrucks in **strFilter** zusammen:

```
    If Not Len(strVergleichsausdruck) = 0 Then
        strFirma = Split(strVergleichsausdruck, " ")(0)
        strFilter = "Firma LIKE '" & strFirma & "'"
```

Dann prüfen wir in einer **If**-Bedingung, ob **strVergleichsausdruck** noch einen zweiten Vergleichswert enthält. Dazu schauen wir, ob sich darin ein Leerzeichen befindet. Ist das der Fall, können wir auf den zweiten Wert des mit der **Split**-Funktion ermittelten Arrays zugreifen und diesen in der Variablen **strKontaktperson** speichern:

```
        If InStr(1, strVergleichsausdruck, " ") > 0 Then
            strKontaktperson = _
                    Split(strVergleichsausdruck, " ")(1)
            strFilter = strFilter & " AND Kontaktperson _
                    LIKE '" & strKontaktperson & "'"
        End If
    End If
```

Danach aktivieren wir wie gehabt den Filter für diesen Filterausdruck:

```
    With Me!sfmKunden.Form
        .Filter = strFilter
        .FilterOn = True
    End With
End Sub
```

Nun weiß der Benutzer im Gegensatz zu uns allerdings nicht, wie man die Suchbegriffe in das Textfeld eingeben muss, damit es das gewünschte Ergebnis liefert. Deshalb fügen wir nun eine Vorlage zum Kombinationsfeld hinzu, anhand derer der Benutzer erkennen kann, welche Daten er dort eingeben kann. Dazu stellen wir das Kombinationsfeld mit dem Wert **Wertliste** für die Eigenschaft **Herkunftsart** auf die Anzeige einer Wertliste um. Diese geben wir wie folgt für die Eigenschaft **Datensatzherkunft** an:

```
"[Firma] [Ansprechpartner]"
```

Damit das Kombinationsfeld diesen Eintrag nun auch anzeigt, wählen wir gleich beim Laden des Formulars den ersten Eintrag des Kombinationsfeldes aus. Dazu hinterlegen wir die folgende Anweisung für das Ereignis **Beim Laden** des Formulars:

```
Private Sub Form_Load()
    Me!cboFirmaUndKontaktpersonFiltern = _
        Me!cboFirmaUndKontaktpersonFiltern.ItemData(0)
End Sub
```

Wenn der Benutzer den Fokus auf das Kombinationsfeld verschiebt, soll der Inhalt komplett angezeigt werden. Auf diese Weise kann der Benutzer dann gleich mit der Eingabe des Suchbegriffs beziehungsweise der Suchbegriffe beginnen. Dazu stellen wir den Wert der Eigenschaft **SelStart** des Kombinationsfeldes auf **0** ein und den Wert der Eigenschaft **SelLength** auf **999**. Das erledigen wir in der Ereignisprozedur, die durch das Ereignis **Bei Fokuserhalt** des Kombinationsfeldes ausgelöst wird:

```
Private Sub cboFirmaUndKontaktpersonFiltern_GotFocus()
    Me!cboFirmaUndKontaktpersonFiltern.SelStart = 0
    Me!cboFirmaUndKontaktpersonFiltern.SelLength = 999
End Sub
```

Das Ergebnis beim Anzeigen des Formulars sieht nun wie in Bild 4 aus.

Wenn wir nun die zwei gewünschten Suchbegriffe eingeben, erhalten wir das gleiche Ergebnis wie im vorherigen Beispiel.

Verschiedene Kombinationen für Suchbegriffe

Nun möchte der Benutzer sicher nicht nur nach der Kombination aus **Firma** und **Kontaktperson** suchen, sondern vielleicht auch einmal nach **Kunden-Code** und **Firma** oder **Kunden-Code** und **Kontaktperson**. Damit wird die Lösung nun interessant, denn wir müssen uns überlegen, wie wir mehrere Varianten behandeln.

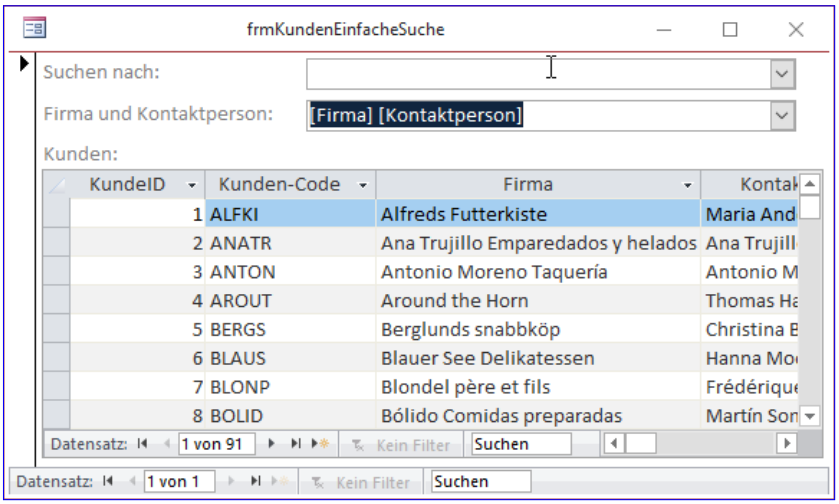


Bild 4: Anzeige des Schemas für den Suchbegriff

Die grundlegende Idee ist, dass wir die verschiedenen Kombinationen alle im Kombinationsfeld zur Auswahl anbieten und dass der gewünschte Eintrag einfach per **Nach oben**- und **Nach unten**-Taste angesteuert werden kann. Außerdem wollen wir nicht für jede neue Kombination die Eigenschaft **Datensatzherkunft** des Kombinationsfeldes ändern und schon gar nicht jedes Mal den Code für die neue Kombination anpassen.

Also überlegen wir uns, wie wir das Ganze etwas flexibler handhaben können. Die erste Idee ist, die verschiedenen Kombinationen plus die beteiligten Felder in einer Tabelle zu hinterlegen, aus der das Kombinationsfeld dann seine Daten zur Auswahl bezieht.

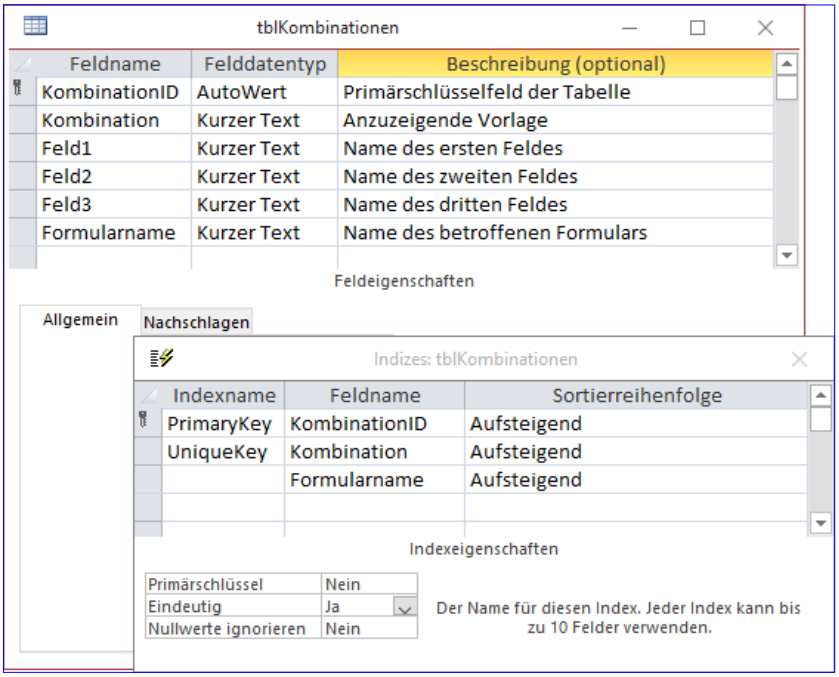


Bild 5: Entwurf der Tabelle für die Kombinationen

Diese Tabelle könnte im Entwurf etwa wie in Bild 5 aussehen.

Das Feld **Kombination** nimmt den anzuzeigenden Platzhalter an, also etwa **[Firma] [Kontaktperson]**. Die Felder **Feld1**, **Feld2** und **Feld3** nehmen die einzelnen Namen der Felder, die durchsucht werden sollen, auf. Damit ist auch klar, dass wir die Anzahl der zu durchsuchenden Felder auf drei begrenzen wollen.

Das Feld **Formularname** soll den Namen des Formulars aufnehmen, in dem diese Kombination zum Einsatz kommt. Auf diese Weise können wir Kombinationen für verschiedene Formulare speichern. Für die beiden Felder Kombination und Formularname haben wir einen eindeutigen, zusammengesetzten Index hinterlegt, damit jede Kombination nur einmal je Formular angezeigt werden kann.

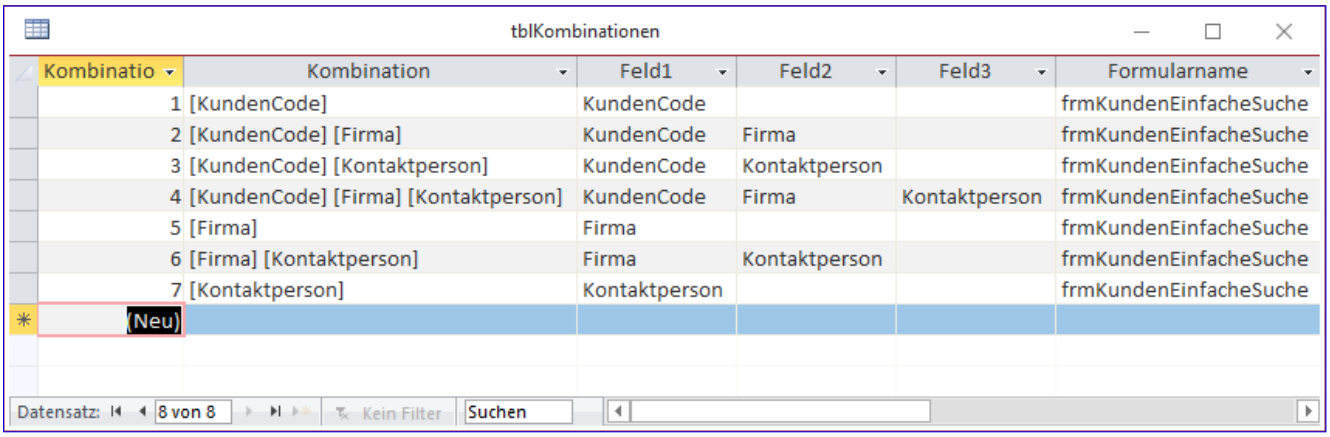


Bild 6: Beispielkombinationen in der Tabelle **tblKombinationen**

Wir hinterlegen zunächst einige Kombinationen wie in Bild 6 in der Tabelle **tblKombinationen**.

Kombinationen im Kombinationsfeld anzeigen

Nun fügen wir dem Formular ein neues Kombinationsfeld namens **cboSuchkombinationen** hinzu.

Diese verwendet als **Herkunftstyp** wieder **Tabelle/Abfrage** und zwar eine Abfrage, die wie in Bild 7 aussieht und alle Felder der Tabelle **tblKombinationen** enthält. Für das Feld **Kombination** legen wir eine aufsteigende Reihenfolge fest, für das Feld **Formularname** den Namen unseres Formulars als Parameter. Wir wollen wieder in der Ereignisprozedur für das Ereignis **Beim Laden** des Formulars das Kombinationsfeld auf den ersten Eintrag einstellen:

```
Private Sub Form_Load()
    Me!cboSuchkombinationen = _
        Me!cboSuchkombinationen.ItemData(0)
End Sub
```

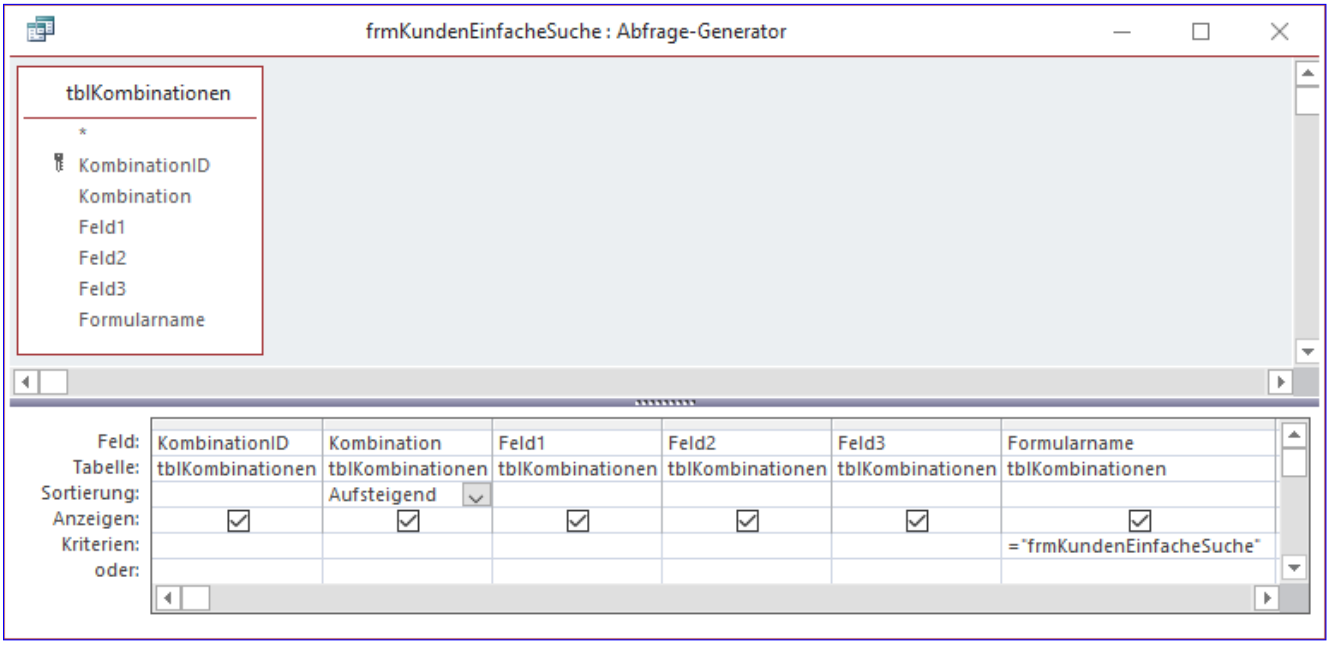


Bild 7: Datensatzherkunft des Formulars zur Eingabe der Suchkombinationen

Wenn wir das Formular nun öffnen und das Kombinationsfeld aufklappen, sieht dies wie in Bild 8 aus.

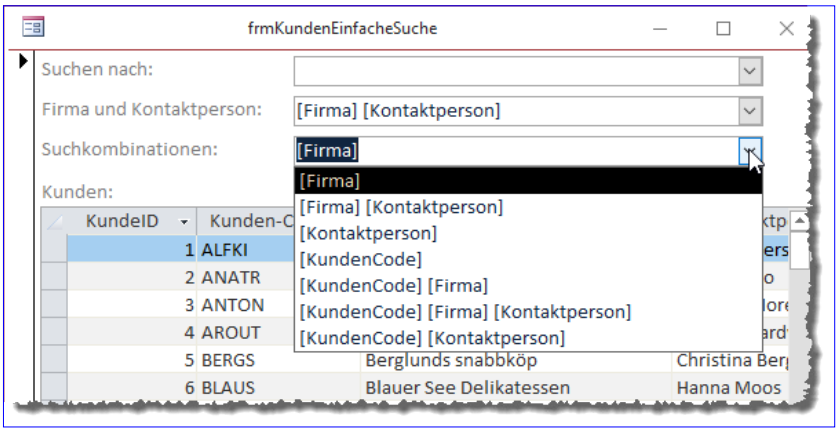


Bild 8: Kombinationsfeld mit Suchkombinationen

Wir wollen nun zunächst die Auswahl nur über das Aufklappen uns Selektieren eines Eintrags erledigen – um das Auswählen mit der **Nach oben**- und der **Nach unten**-Taste kümmern wir uns später.

Das erste Problem, dem wir uns ausgesetzt sehen, ist die Meldung, die beim Eingeben eines neuen Wertes in das Kombinationsfeld erscheint und die wie in Bild 9 aussieht.

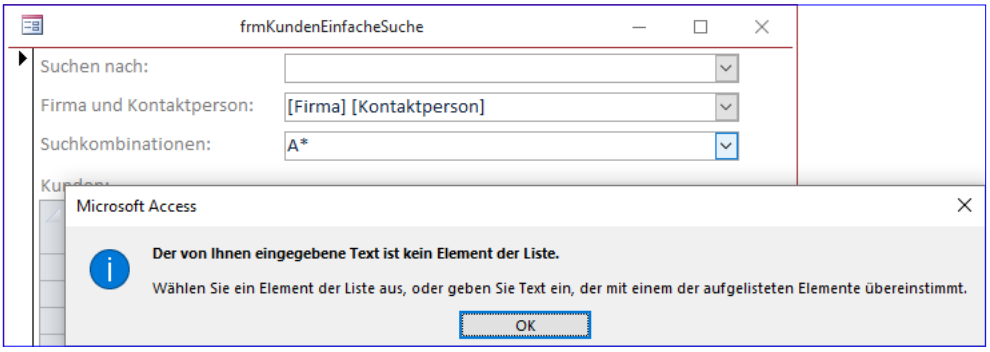


Bild 9: Meldung nach Bestätigung der Eingabe des Suchbegriffs

Die Meldung ist logisch: Wenn wir in ein Kombinationsfeld, das an eine Datensatzherkunft gebunden ist, einen neuen Wert eingeben und die Eingabe abschließen, prüft Access, ob der eingegebene Eintrag bereits vorhanden ist.

Um diese Meldung zu umgehen, nutzen wir das Ereignis **Bei nicht in Liste**. Hier wollen wir einfach dafür sorgen, dass die Meldung nicht angezeigt wird, was wir durch Einstellen des Wertes **acDataErrAdded** für den Parameter **Response** erledigen:

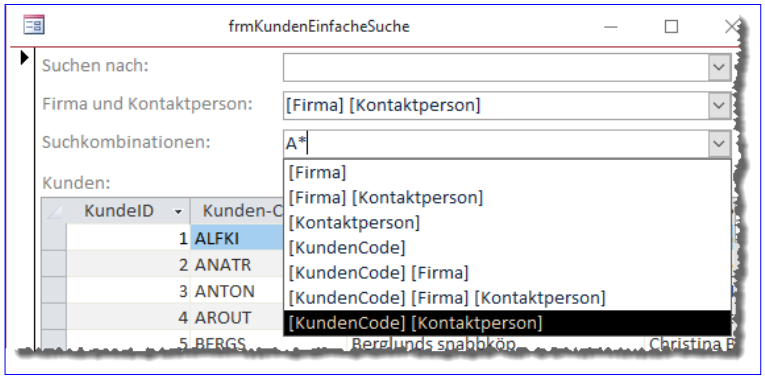


Bild 10: Der neue Eintrag kann nicht ohne weiteres eingegeben werden.

```
Private Sub cboSuchkombinationen_NotInList( _
        NewData As String, Response As Integer)
    Response = acDataErrAdded
End Sub
```

Dies führt aber auch nicht zu einem zufriedenstellenden Ergebnis, wie Bild 10 zeigt – wir können keinen neuen Wert eingeben und das Kombinationsfeld verlassen.

Inhalt der Tabelle tblKombinationen in Wertliste einlesen

Beim vorherigen Beispiel, bei dem wir die Vorlage für die Suche als Element einer Wertliste zum Kombinationsfeld hinzugefügt haben, war dies jedoch problemlos möglich. Also kehren wir doch zum Ansatz mit der Wertliste zurück. Allerdings füllen wir diese Wertliste beim Anzeigen des