HAUFE.
Ausgabe 04/2024

ACCESS
IM UNTERNEHMEN

MEHRWERTIGE FELDER LOSWERDEN

Ersetzen Sie mehrwertige Felder und machen Sie Ihre Anwendung SQL Server-kompatibel (ab Seite 2).

In diesem Heft:

Mat-Nr. 01583-5144

Mehrwertige Felder loswerden?

Mit Access 2010 hat Microsoft die sogenannten mehrwertigen Felder eingeführt. Damit lassen sich zum Beispiel m:n-Beziehungen abbilden, ohne dass man explizit eine m:n-Beziehung definieren muss. Es reicht, wenn man eine 1:n-Beziehung beispielsweise zur Zuordnung einer Ausstattung zu einem Fahrzeug hat. Das macht so keinen Sinn, denn man möchte einem Fahrzeug ja mehr als eine Ausstattung zuweisen. Mit mehrwertigen Feldern erlaubte Microsoft aber genau dies – plötzlich konnten wir, obwohl wir nur eine herkömmliche 1:n-Beziehung hatten, mehrere Einträge der verknüpften Tabelle zuweisen. Das fliegt dem Anwender irgendwann um die Ohren, und zwar spätestens, wenn man seine Datenbank zum SQL Server migrieren möchte.

Der SQL Server unterstützt nämlich nicht alle Features, die sich Microsoft für Access ausgedacht hat. Dazu gehören neben den Anlagefeldern die mehrwertigen Felder. Und da wir uns im Moment immer mal wieder mit dem Thema Migration zum SQL Server beschäftigen, haben wir uns auch überlegt, wie wir in diesem Zusammenhang mit mehrwertigen Feldern umgehen können. Dazu haben wir gleich zwei Beiträge produziert. Der erste heißt **Mehrwertige Felder mit Wertliste loswerden** (ab Seite 2) und zeigt, wir man die intern verwaltete m:n-Beziehung durch eine echte m:n-Beziehung ersetzt.

Allerdings verlieren wir damit auch die praktische Möglichkeit, die verfügbaren zuzuordnenden Elemente in einem Listenfeld anzuzeigen und diese über einen Haken in einem Kontrollkästchen zum aktuellen Datensatz zuzuordnen. Wir wir das auch für eine echte m:n-Beziehung abbilden können, zeigen wir ab Seite 16 im Beitrag **m:n-Daten wie im mehrwertigen Feld selektieren**.

Das Beispiel der m:n-Beziehung zwischen Fahrzeugen und Ausstattungen greifen wir anschließend nochmals auf. Solche Beziehungen können wir noch auf eine andere Art abbilden, nämlich durch eine Kombination aus Listenfeld und Unterformular in der Datenblattansicht. Mit dem Listenfeld können wir schnell per Doppelklick Einträge zuordnen, die dann in der Datenblattansicht mit ihren Details angezeigt werden. Wie das gelingt, lesen Sie ab Seite 25 im Beitrag **m:n-Beziehung mit Listenfeld und Datenblatt**.

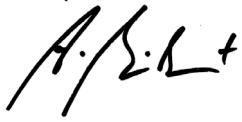
Dieses Beispiel erweitern wir dann nochmals, indem wir dem Listenfeld eine sehr ergonomische Suchfunktion hinzufügen. Diese beschreiben wir im Beitrag **Schnellsuche im Listenfeld mal anders** ab Seite 42.

Auch zum Thema Ribbon haben wir eine praktische Lösung. Manchmal ist es praktisch, wenn das zuletzt verwendete Ribbon-Tab gleich beim Öffnen der Anwendung wieder angezeigt wird. Wie das gelingt, zeigt der Beitrag **Letztes geöffnetes Ribbon-Tab merken** ab Seite 49.

Diese und viele andere Einstellungen, die mit dem aktuellen Benutzer zusammenhängen, können wir in der Registry speichern. Dazu braucht man keine komplizierten API-Funktionen, sondern wir können dies mit Bordmitteln erledigen – mehr dazu unter dem Titel **Registry-Einträge für VBA-Anwendungen** ab Seite 56.

Und schließlich schauen wir uns ab Seite 60 an, wie man schnell eine erste Migration der Tabellen einer Access-Datenbank zum SQL Server durchführt. All dies finden Sie unter dem Titel **Access und SQL Server: Der Migrations-Wizard**.

Viel Spaß beim Lesen!



Ihr André Minhorst

Mehrwertige Felder mit Wertliste loswerden

Mehrwertige Felder sind eine Erfindung von Microsoft, um den Umgang mit Datenkonstrukten, bei denen für ein Feld mehrere Werte ausgewählt werden können, zu vereinfachen. In einem mehrwertigen Feld können wir aus einer Liste von Werten, die entweder aus einer Wertliste oder aus einer anderen Tabelle stammen, keinen, einen oder mehrere Einträge auswählen. Aus einer verknüpfen Tabelle mehrere Werte zuordnen? Das hört sich ja eigentlich nach dem Einsatzzweck einer m:n-Beziehung an. Genau das bildet Microsoft intern ab. Allerdings funktioniert das nur innerhalb des Access-Biotops. Sollen die Daten einmal zum SQL Server oder einer anderen Datenbank wandern, wird es kompliziert. Hier können wir solche Konstrukte nämlich nicht mehr einfach abbilden – wir müssen diese also ersetzen. Wie wir die mehrwertigen Felder loswerden, zeigen wir in diesem Beitrag.

Microsoft hat in Access 2010 einige Neuerungen geliefert, die in Zusammenhang mit den gleichzeitig veröffentlichen sogenannten Webdaten standen. Neben den mehrwertigen Feldern sind das zum Beispiel die Anlagefelder. Wobei die Anlagefelder, wenn man sie richtig nutzt, zumindest noch einigermaßen sinnvoll sind. 1:1 zum SQL Server übertragen können wir diese jedoch ebenfalls nicht. Das ist allerdings Thema eines anderen Beitrags.

In diesem Beitrag wollen wir uns die mehrwertigen Felder ansehen. Davon gibt es grob zwei Ausführungen:

- Als Erstes mehrwertige Felder, die ihre Daten aus einer Wertliste beziehen und nicht aus einer Tabelle oder Abfrage. Das heißt, dass eine per Semikolon getrennte Liste von Einträgen vorliegt, aus denen der Benutzer keinen, einen oder mehrere Einträge auswählen kann.

- Als Zweites sehen wir die mehrwertigen Verknüpfungen, bei denen ein Nachschlagefeld auf Basis einer anderen Tabelle oder Abfrage erstellt wurde, für das wir keinen, einen oder mehrere Einträge je Datensatz auswählen können. Wir haben also prinzipiell eine 1:n-Beziehung, bei der Access intern eine m:n-Beziehung zwischenschaltet.

Beide sind praktisch, bis man mal einen der Einträge aus der Liste entfernt – diese bleiben nämlich in der Auswahl erhalten. In diesem Beitrag schauen wir uns zunächst an, was geschieht, wenn wir die erstgenannten Felder zum SQL Server migrieren wollen und wie wir das Datenmodell und die enthaltenen Daten korrekt zum SQL Server übertragen.

Probleme bei Verwendung in SQL Server und Co.

Richtig problematisch wird es jedoch, wenn wir Tabellen mit solchen Feldern zum SQL Server migrieren wollen. Warum das der Fall ist, zeigen wir gleich nach dem Erstellen der Beispiele für diesen Beitrag.

Beispiel mehrwertige Wertliste: Produkte und Kategorien

Zu Beispielzwecken legen wir eine Tabelle namens **tblProdukte** an, welche die folgenden drei Felder enthält:

- **ProduktID**: Primärschüsselfeld der Tabelle

- **Produktname**: Name des Produkts

- **Kategorien**: Feld mit den Kategorien des Produkts

Für den Datentyp des dritten Feldes wählen wir **Nachschlage-Assistent...** aus (siehe Bild 1).

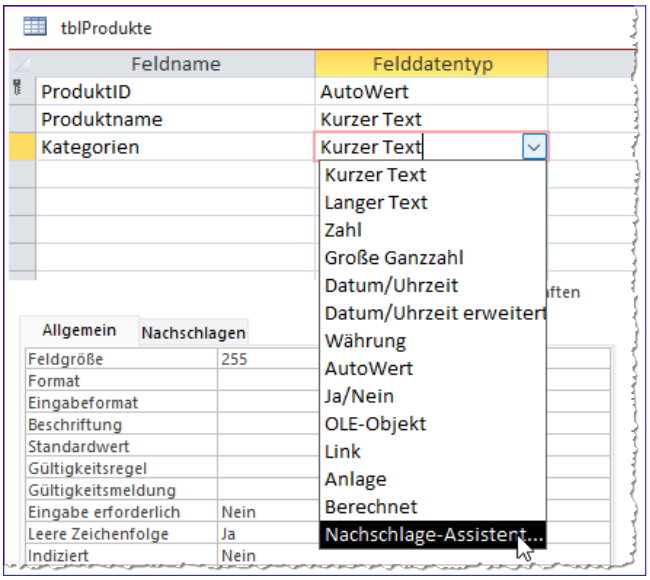


Bild 1: Anlegen eines Nachschlagefeldes

Im Assistenten führen wir folgende Schritte durch:

- Schritt 1: Auswahl von **Ich möchte selbst Werte in die Liste eingeben**
- Schritt 2: Anzahl der Spalten bleibt **1**
- Schritt 3: Beschriftung beibehalten, Option **Nur Listeneinträge** aktivieren, Option **Mehrere Werte zulassen** aktivieren (siehe Bild 2).

Danach können wir für das Feld **Kategorien** in den Eigenschaften unter **Nachschlagen** die gewünschten Kategorien durch Semikola separiert für die Eigenschaft **Datensatzherkunft** eingeben (siehe Bild 3).

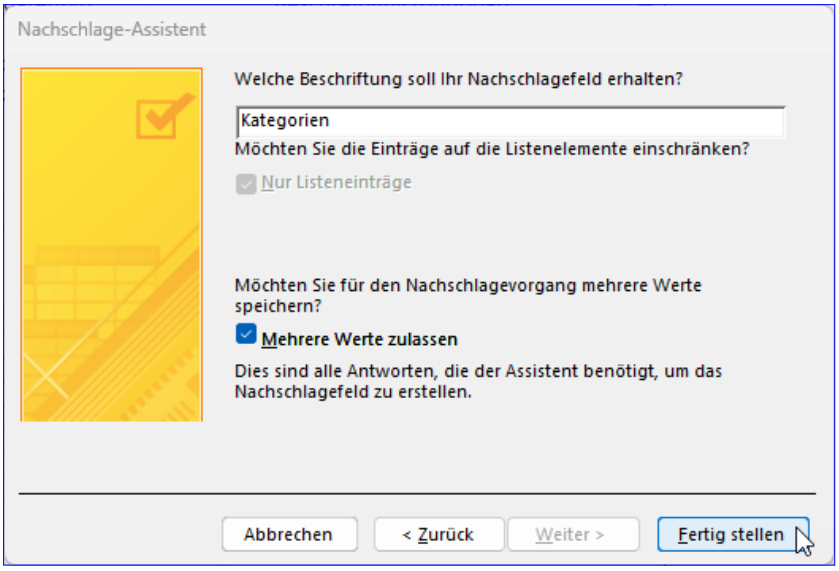


Bild 2: Einstellen der Nachschlagefeld-Eigenschaften

Wenn wir wollen, dass der Benutzer die Werte selbst ändern kann, stellen wir noch die Eigenschaft **Wertlistenbearbeitung zulassen** auf den Wert **Ja** ein.

Mehrwertige Wertliste im Formular

Wenn wir ein neues Formular mit dieser Tabelle als Datensatzquelle ausstatten und die drei Felder hinzufügen, sehen wir das Ergebnis aus Bild 4. In diesem Fall ist die Wertlistenbearbeitung aktiviert, was wir an der Schaltfläche erkennen, die beim Öffnen der Liste eingeblendet wird.

Analyse der Daten

Bevor wir prüfen, was beim Migrieren der mehrwertigen Wertliste zum SQL Server passiert, schauen wir uns an, was diese Felder eigentlich speichern. Zunächst einmal betrachten wir den im Feld **Kategorien** des ersten Daten-

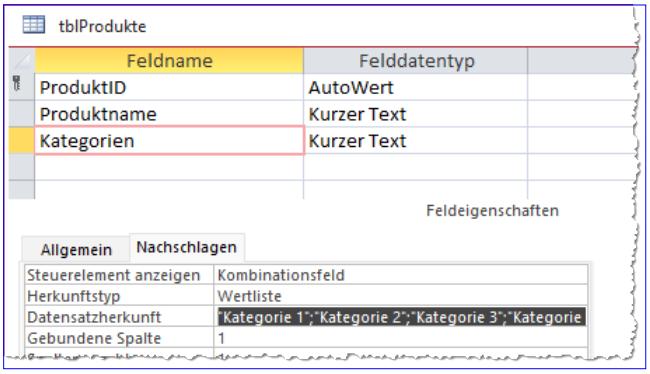


Bild 3: Hinzufügen der Einträge

satzes gespeicherten Wert. Dazu nutzen wir die **DLookup**-Funktion, die folgendes Ergebnis liefert:

```
Debug.Print DLookup("Kategorien", "tblProdukte")
Kategorie 1; Kategorie 3
```

Außerdem schauen wir uns noch an, wie die zugrunde liegenden Daten aussehen. Dazu lesen wir die Eigenschaft **RowSource** des Kombinationsfeldes aus:

```
Debug.Print CurrentDb.TableDefs("tblProdukte").Fields("Kategorien").Properties("RowSource")
"Kategorie 1";"Kategorie 2";"Kategorie 3";"Kategorie 4"
```

Wir erhalten also zwei Listen, von denen die erste nur die selektierten Einträge enthält.

Migration dieser Tabelle zum SQL Server

Nun migrieren wir diese Tabelle mit dem SQL Server Migration Assistant in eine neue Datenbank im SQL Server – mehr dazu im Beitrag **Access und SQL Server: Der Migrations-Assistent** (**www.access-im-unternehmen.de/1498**).

Dazu gehen wir nach dem Starten des SSMA wie folgt vor:

- Wir legen mit **File|New Project...** ein neues Projekt namens **MehrwertigeWertlisten** an.
- Wir wählen mit dem Befehl **Add Databases** die Beispieldatenbank dieses Beitrags aus.
- Im Bereich **Access Metadata Explorer** selektieren wir die zu migrierende Tabelle **tblProdukte** (siehe Bild 5).
- Nun klicken wir auf **Connect to SQL Server** und verbinden uns mit der SQL Server-Instanz, auf der wir die Datenbank anlegen wollen. Für **Database** geben wir den Namen der zu erstellenden Datenbank an, hier **MehrwertigeWertlisten** (siehe Bild 6). Gegebenenfalls erscheint eine Meldung, dass die Datenbank noch nicht

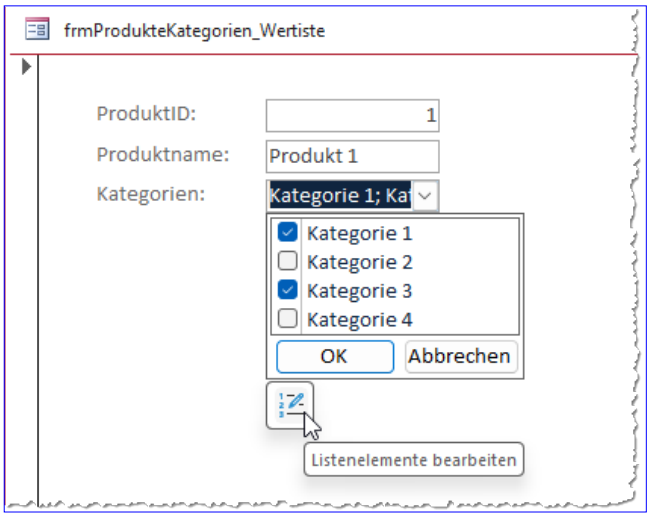


Bild 4: Einsatz der mehrwertigen Wertliste im Formular

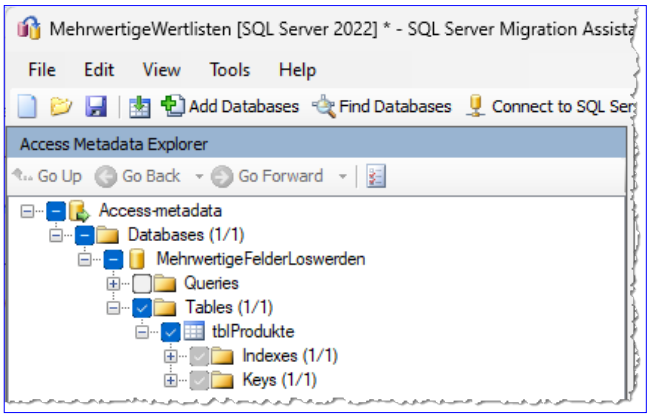


Bild 5: Zu migrierende Tabelle auswählen

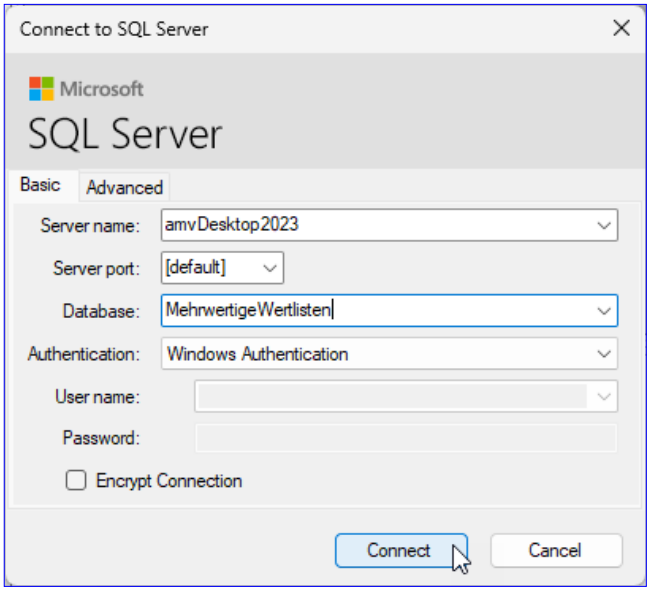


Bild 6: Verbinden mit der noch anzulegenden Datenbank

vorhanden ist und die fragt, ob die Datenbank angelegt werden soll. Dies bestätigen wir.

- Damit wird die Datenbank bereits angelegt, wovon wir uns im SQL Server Management Studio überzeugen können (siehe Bild 7).

- Dann markieren wir den Eintrag mit dem Namen der zu migrierenden Datenbank im Bereich **Access Metadata Explorer**.

- Schließlich rufen wir den Befehl **Convert, Load and Migrate** auf.

- Die Migration beginnt und es erscheint der Dialog **Synchronize with the Database**. Diesen Dialog können wir mit **Ok** direkt wieder schließen.

- Danach folgt der Dialog **Convert, Load and Migrate**. Dieser weist auf einen Fehler bei der Konvertierung hin (siehe Bild 8).

- Ein Klick auf diesen Link zeigt Details zum Fehler an. Der Datentyp **Complex Text** wird nicht unterstützt (siehe Bild 9).

- Schließlich können wir uns im SQL Server Management Studio das Ergebnis anschauen.

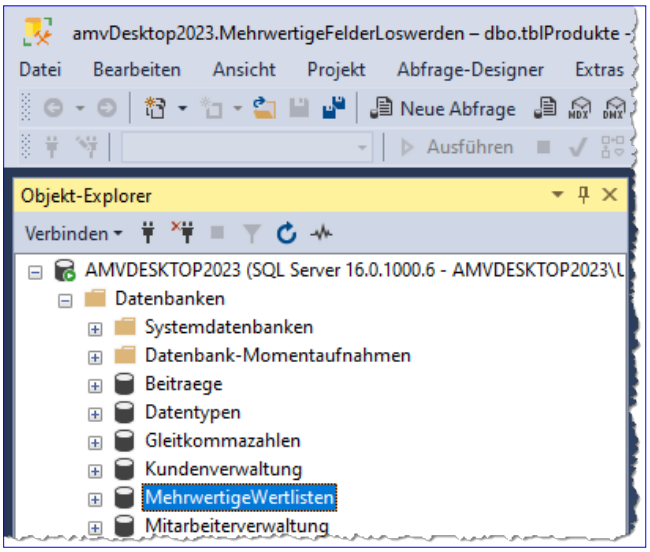


Bild 7: Die neue Datenbank ist bereits erstellt.

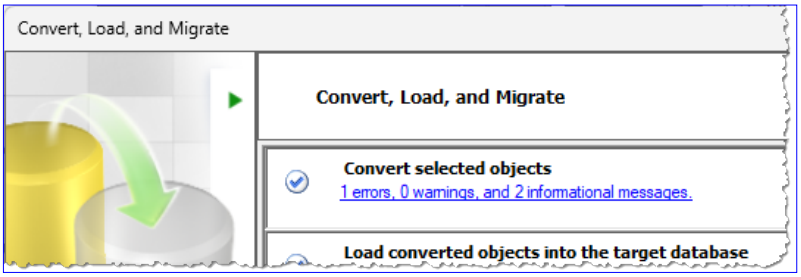


Bild 8: Es gibt einen Fehler.

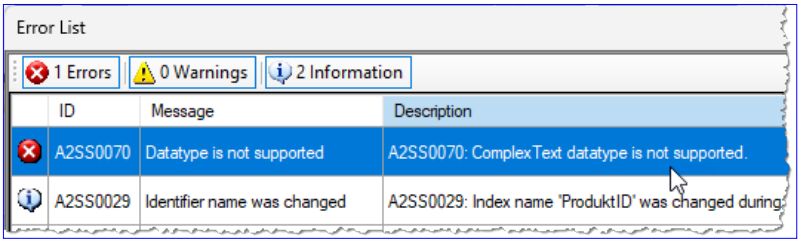


Bild 9: Ein Datentyp wird nicht unterstützt.

In diesem Fall sehen wir, dass das Feld **Kategorien** zwar migriert wurde.

Allerdings finden wir hier ein Feld mit dem Datentyp **varchar(8000)** vor. Das Feld enthält lediglich die für die jeweiligen Produkte ausgewählten Kategorien.

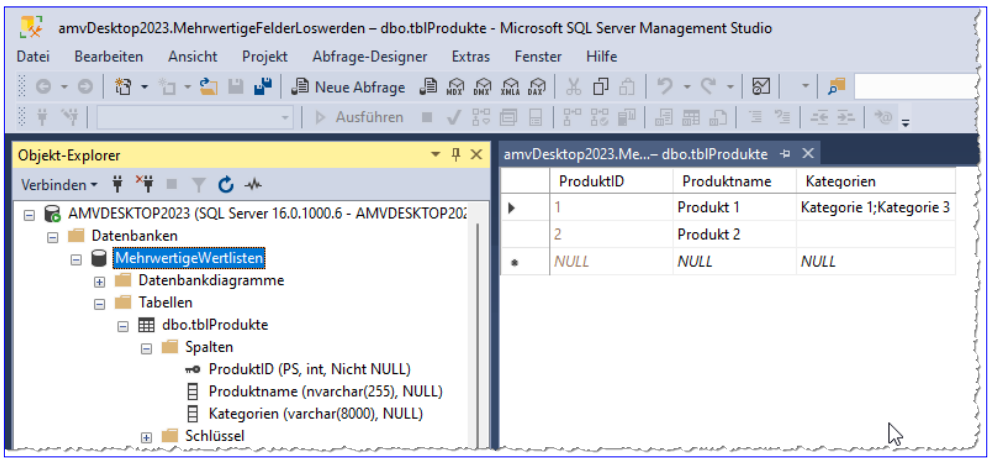


Bild 10: Das Ergebnis der Migration

Die verfügbaren Kategorien sind hier nicht mehr zu sehen (siehe Bild 10).

Und da die meisten Tabelleneigenschaften, vor allem aber die zum Thema Nachschlagefelder, nicht zum SQL Server übertragen werden, können wir nicht damit rechnen, dass die vollständige Liste der verfügbaren Kategorien irgendwo im SQL Server zu finden ist.

Wenn wir die zum SQL Server migrierte Tabelle **tblProdukte** nun per ODBC-Verknüpfung in die Beispieldatenbank einbinden, sehen wir dort im Feld **Kategorien** zwar noch die gleichen Werte, die auch in der ursprünglichen Tabelle enthalten waren, aber wir können diese nicht mehr selektieren.

Auch wenn wir eine Kopie des Beispielformulars von oben anlegen und dieses mit der verknüpften Tabelle **dbo_tblProdukte** versehen, sehen wir dort zwar die Liste der markierten Einträge. Wenn wir die Liste jedoch öffnen, finden wir keine Möglichkeit mehr, mehr als einen Eintrag auszuwählen (siehe Bild 11).

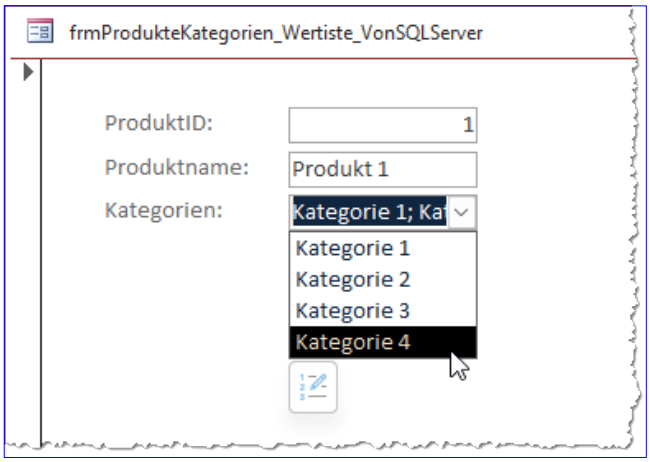


Bild 11: Bei der migrierten und neu verknüpften Tabelle werden die Kontrollkästchen für die Kategorien nicht mehr angezeigt.

Mehrwertige Wertlisten korrekt migrieren

Nun schauen wir uns an, wie wir diese mehrwertigen Wertlisten so migrieren können, dass die gewünschten Daten vom SQL Server bereitgestellt werden und auch zugewiesen werden können. Es führt dabei kein Weg daran vorbei, die Daten, die sich im Feld **Kategorien** der Tabelle **tblProdukte** befinden, in eine eigene Tabelle namens **tblKategorien** auszulagern. Diese weisen wir allerdings nicht der Eigenschaft **Datensatzherkunft** des Feldes **Kategorien** der Tabelle **tblProdukte** zu.

Dies wäre nur sinnvoll, wenn nur eine Kategorie je Produkt ausgewählt werden soll. Die Ausgangssituation ist jedoch, dass wir jedem Produkt eine, keine oder mehrere Kategorien zuweisen können. Um dies zu realisieren, benötigen wir also auch noch eine Verknüpfungstabelle zum Herstellen einer m:n-Beziehung. Diese nennen wir **tblProdukteKategorien**.

In weiteren Schritten sind zunächst die Einträge der Wertliste des Feldes **Kategorien** in die Tabelle **tblKategorien** zu übertragen. Und zu guter Letzt übertragen wir noch die ausgewählten Werte in die Verknüpfungstabelle **tblProdukteKategorien**.

Damit sind wir leider noch nicht am Ende der Migrationsarbeiten angelangt. Die einfache Darstellung aus dem mehrwertigen Feld mit der Anzeige aller Einträge, die man einfach per Kontrollkästchen selektieren kann, gibt es so für m:n-Beziehungen leider nicht.

Also werden wir in weiteren Beiträgen Alternativen dazu vorstellen – siehe **m:n-Daten wie im mehrwertigen Feld selektieren** (**www.access-im-unternehmen.de/1424**).

Von der mehrwertigen Wertliste zur m:n-Beziehung

Wir gehen in diesen Schritten vor:

- Tabelle für die Kategorien erstellen
- Verknüpfungstabelle erstellen
- Wertliste in Tabelle für Kategorien übertragen
- Gewählte Werte in Verknüpfungstabelle schreiben

- Mehrwertiges Feld **Kategorien** löschen

Erstellen der Tabelle tblKategorien

Diese Tabelle enthält nur zwei Felder – das Primärschlüsselfeld **KategorieID** und das Feld **Kategorie**. Für dieses legen wir einen eindeutigen Index fest, damit jede Kategorie nur einmal eingetragen werden kann (siehe Bild 12).

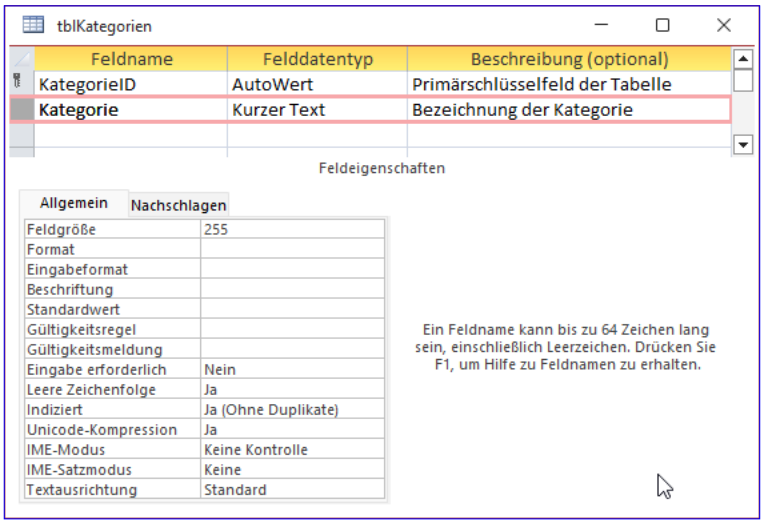


Bild 12: Tabelle zum Speichern der Kategorien

Erstellen der Tabelle tblProdukteKategorien

Die Tabelle **tblProdukteKategorien** soll die Verknüpfung jedes Eintrags der Tabelle **tblProdukte** mit jedem Eintrag der Tabelle **tblKategorien** erlauben.

Dazu fügen wir dieser Tabelle zunächst ein Primärschlüsselfeld namens **ProduktKategorieID** hinzu. Außerdem legen wir zwei Fremdschlüsselfelder an. Das erste heißt **ProduktID** und ermöglicht die Angabe des Produkts eines Datensatzes, das zweite heißt **KategorieID** und soll das Zuweisen einer Kategorie erlauben.

Für die beiden Felder **ProduktID** und **KategorieID** legen wir außerdem einen zusammengesetzten, eindeutigen Index an (siehe Bild 13).

Da diese Tabelle direkt zum SQL Server migriert werden soll, der bekanntermaßen solche Eigenschaften wie die zum Realisieren von Nachschlagefeldern in Access nicht erlaubt, brauchen wir uns gar nicht erst die Mühe zu machen, die Fremdschlüsselfelder als Nachschlagefelder einzurichten.

Einrichten der Beziehungen

Damit kommen wir zu den Beziehungen, die wir über das Beziehungen-Fenster anlegen. Dazu ziehen wir die drei Tabellen **tblProdukte**, **tblProdukteKategorien** und **tblKategorien** in das Beziehungen-Fenster. Die Beziehung zwischen der Tabelle **tblProdukte** und **tblProdukteKategorien** erzeugen wir, indem wir das Feld **ProduktID** von der Tabelle **tblProdukte** auf das gleichnamige Feld der Tabelle **tblProdukteKategorien** ziehen. Im nun erschei-

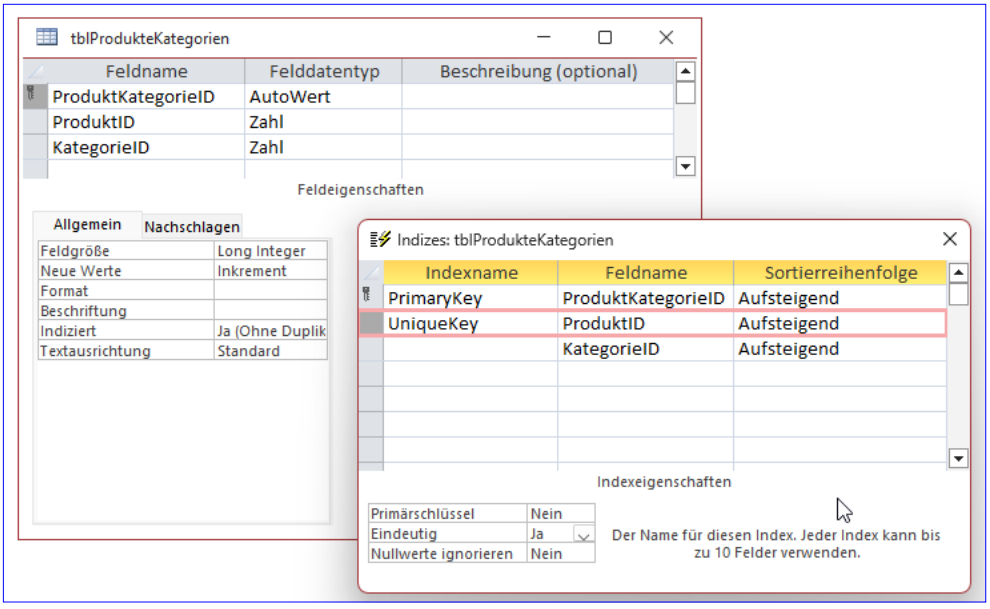


Bild 13: Verknüpfungstabelle zwischen den Tabellen **tblProdukte** und **tblKategorien**

m:n-Daten wie im mehrwertigen Feld selektieren

Im Beitrag namens »Mehrwertige Felder mit Wertliste loswerden« (www.access-im-unternehmen.de/1493) beschreiben wir, wie man die in Access 2010 eingeführten mehrwertigen Felder durch m:n-Beziehungen ersetzt. Es fehlt dann allerdings die praktische Liste mit Kontrollkästchen, mit der man einen oder mehrere der zu verknüpfenden Elemente einfach markieren kann. Auf diese Weise lassen sich schnell die Kategorien zu einem Produkt oder auch die Ausstattungen für Fahrzeuge zusammenklicken. Gemäß unserem Motto »Wer A sagt, muss auch B sagen«, liefern wir noch eine Lösung nach, um die aus den mehrwertigen Feldern in eine m:n-Beziehung überführten Daten auf praktische Weise anzuzeigen.

Dabei verwenden wir die Beispiele, die wir in dem oben genannten Beitrag **Mehrwertige Felder mit Wertliste loswerden** (**www.access-im-unternehmen.de/1493**) verwendet haben. Wir beginnen wieder mit den Produkten und ihren Kategorien. In der Beispieldatenbank finden wir in der Tabelle **tblProdukte** noch das Feld **Kategorien**, das als mehrwertiges Feld mit einer Wertliste als Datensatzherkunft ausgestattet ist (siehe Bild 1).

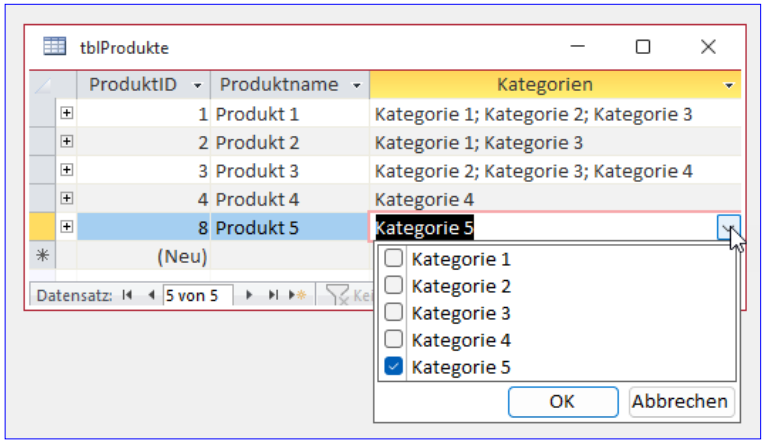


Bild 1: Beispiel für ein mehrwertiges Feld

Wenn wir diese Tabelle als Datensatzquelle für ein Formular verwenden, alle Felder in den Detailbereich ziehen und dann noch das Feld **Kategorien** in ein Listenfeld umwandeln, erhalten wir eine Ansicht wie in Bild 2. Das ist sehr praktisch und eine schöne Alternative zu den übrigen Möglichkeiten, eine m:n-Beziehung abzubilden.

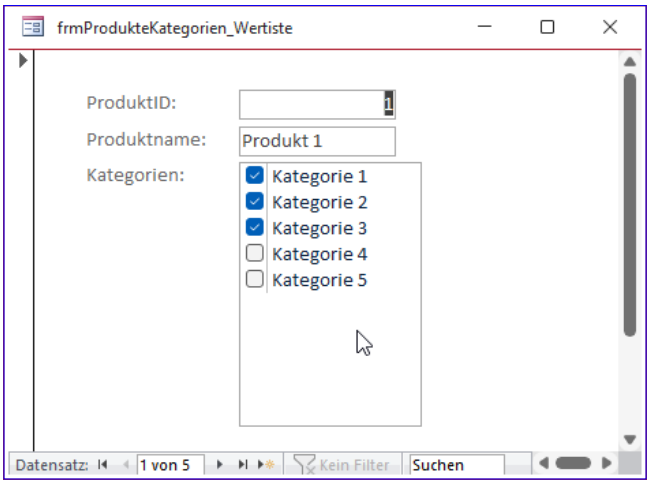


Bild 2: Beispiel für ein als Listenfeld abgebildetes mehrwertiges Feld

Nun haben wir allerdings das mehrwertige Feld durch eine m:n-Beziehung ersetzt, in der alle Kategorien in der Tabelle **tblKategorien** gespeichert werden und in der die Verknüpfungstabelle **tblProdukteKategorien** die Zuordnung von Produkten zu Kategorien aufnimmt (siehe Bild 3). Damit erhalten wir ein Datenmodell, das wir nicht nur komplett steuern können (das mehrwertige Feld wurde intern von Access verwaltet). Wir können ein solches Datenmodell auch zum SQL Server überführen. Dies wäre mit einem mehrwertigen Feld nicht möglich

gewesen – hier würden zwar die für ein Produkt ausgewählten Kategorien übernommen werden, aber die Liste aller Kategorien und die Auswahlmöglichkeit fallen unter den Tisch.

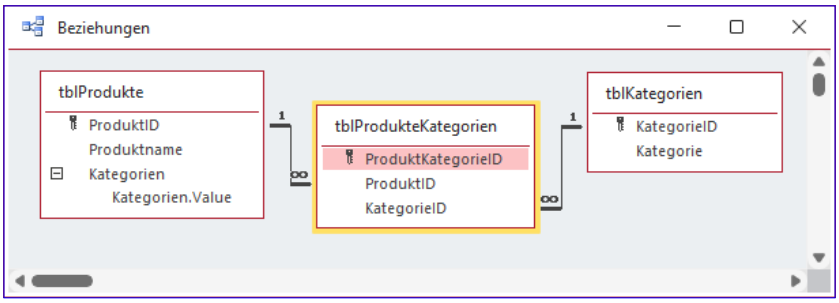


Bild 3: Die aufgeteilten Daten eines mehrwertigen Feldes

Das Ziel dieses Beitrags ist es nun, die Daten im Formular so abzubilden, wie es zuvor beim mehrwertigen Feld in der Ausführung als Listenfeld mit Kontrollkästchen der Fall war.

In einem Listenfeld werden wir das nicht abbilden können, denn dieses erlaubt nicht die Verwendung von Kontrollkästchen. Hier könnten wir lediglich alle für das aktuelle Produkt ausgewählten Kategorien markieren. Alternativ könnten wir auch noch das **ListView**-Steuerelement nutzen, das ein Kontrollkästchen je Eintrag anzeigen kann. Aber wir wollen in diesem Fall mit Access-Bordmitteln auskommen.

Wie also können wir das erledigen? Die einzigen beiden Möglichkeiten, Datensätze inklusive Kontrollkästchen darzustellen, sind die Endlosansicht und die Datenblattansicht eines Formulars beziehungsweise Unterformulars.

Bei der Datenblattansicht werden allerdings zwingend die Spaltenüberschriften angezeigt. Da wir eine möglichst genaue Abbildung des mehrwertigen Listenfeldes erhalten wollen, fällt diese Variante also weg. Es bleibt noch die Verwendung der Endlosansicht. Diese bietet außerdem alle gestalterischen Möglichkeiten.

Überlegungen zur Darstellung der zugeordneten Datei

Bevor wir loslegen, müssen wir uns überlegen, wie wir die Darstellung erhalten. In einer herkömmlichen Konstellation aus Haupt- und Unterformular zur Darstellung einer m:n-Beziehung landet die m-Tabelle, in diesem Beispiel die Tabelle **tblProdukte**, im Hauptformular. Im Unterformular bilden wir die Daten der Verknüpfungstabelle ab, gegebenenfalls erweitert um die Daten der n-Tabelle. Das Unterformular-Steuerelement wird so eingerichtet, dass das Unterformular immer nur die Datensätze der m:n-Tabelle anzeigt, die mit dem Datensatz im Hauptformular verknüpft sind.

Wir können üblicherweise mit Haupt- und Unterformular also immer nur alle verknüpften Datensätze der n-Tabelle anzeigen. Wir wollen aber alle Datensätze der n-Tabelle anzeigen und für diejenigen, die über die m:n-Verknüpfungstabelle dem aktuell angezeigten Datensatz der m-Tabelle zugeordnet sind, ein markiertes Kontrollkästchen hinzufügen. Das gelingt auf diesem zunächst eingeschlagenen Weg nicht.

Stattdessen müssen wir eine Darstellung finden, die alle Datensätze der n-Tabelle anzeigt, hier **tblKategorien**, und die noch ein Kontrollkästchen hinzufügt, das die für den aktuellen Datensatz im Hauptformular selektierten Kategorien liefert. Wir haben es hin- und hergedreht und mit diversen Abfragetypen experimentiert. Wie haben keine praktikable Lösung gefunden. Also wählen wir einen völlig anderen Ansatz.

Alle Kategorien plus Markierung der zugeordneten Kategorien

Die einfachere, da sichtbare Lösung ist die Verwendung einer temporären Tabelle. Diese nennen wir **tblKategorienSelektiert** und entwerfen sie wie in Bild 4. Neben den beiden Feldern **KategorieID** und **Kategorie** fügen wir dieser das **Ja/Nein**-Feld **Zugeordnet** hinzu. Das Feld

KategorieID statten wir nicht mit der **Autowert**-Funktion aus, da wir die Werte für dieses Feld später selbst eintragen wollen.

Diese Tabelle fügen wir dem Unterformular hinzu und sorgen per Code dafür, dass diese beim Anzeigen eines Produkt-Datensatzes im Hauptformular jeweils geleert und neu gefüllt wird.

Hauptformular erstellen

Diese Endlosansicht legen wir in einem Unterformular an. Als Erstes erstellen wir das Hauptformular für unsere Lösung. Dieses verwendet die Tabelle **tblProdukte** als Datensatzquelle. Aus der Tabelle ziehen wir die beiden Felder **ProduktID** und **Produktname** in den Detailbereich des Formulars (siehe Bild 5).

Unterformular erstellen

Nun erstellen wir das Unterformular namens **sfmProdukteKategorien**. Diesem weisen wir die Tabelle **tblKategorienSelektiert** als Datensatzquelle zu und ziehen die beiden Felder **Zugeordnet** und **Kategorie** in den Detailbereich des Formulars.

Die beiden Bezeichnungsfelder, die beim Hineinziehen der Felder in den Detailbereich automatisch angelegt wurden, entfernen wir. Dann ordnen wir die verbleibenden beiden Steuerelemente wie im mehrwertigen Listenfeld an (siehe Bild 6). Für das Textfeld **Kategorie** stellen wir die Eigenschaft **Horizontaler Anker** auf **Beide** ein, damit es beim Verbreitern des Forulars angepasst wird.

Schließlich stellen wir noch die Eigenschaft **Standardansicht** auf **Endlosformular** ein. Damit die

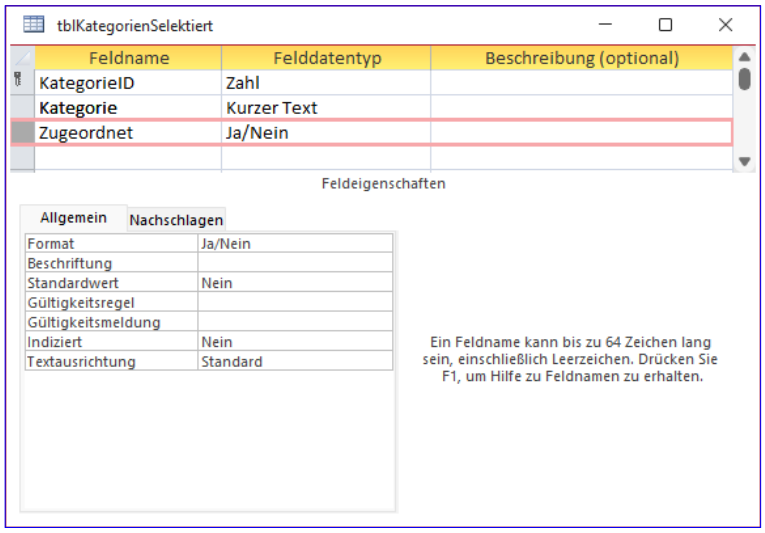


Bild 4: Temporäre Tabelle

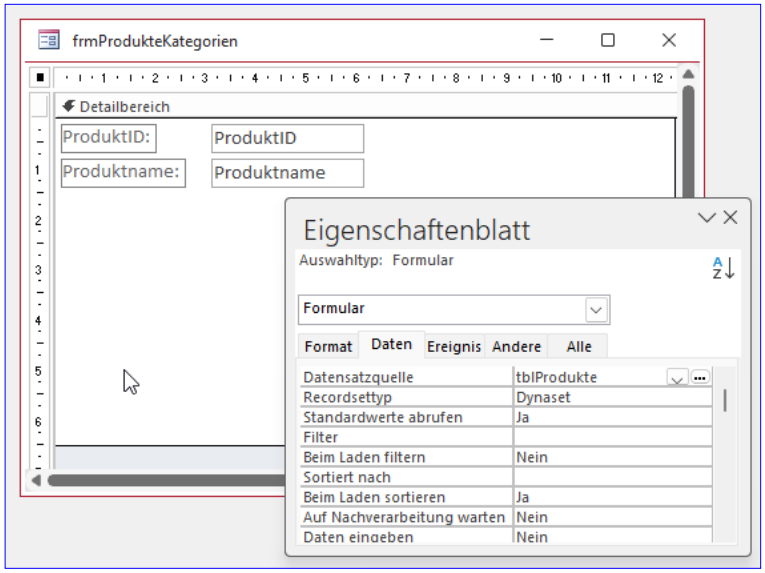


Bild 5: Anlegen des Hauptformulars

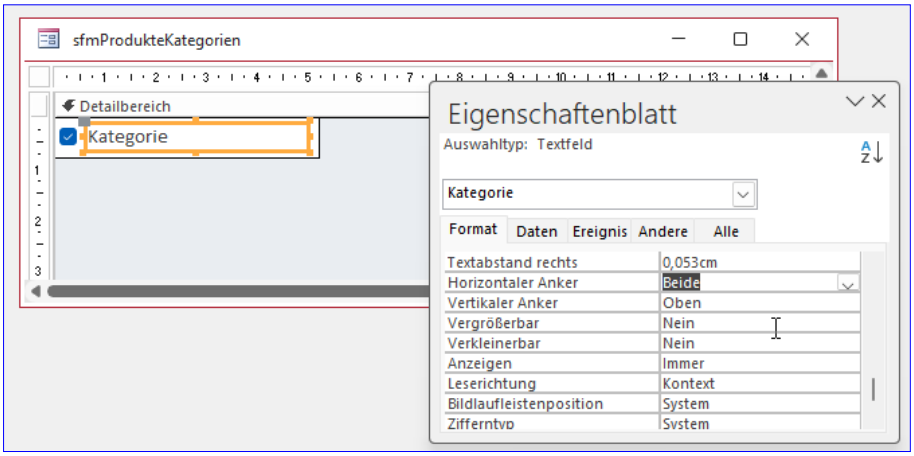


Bild 6: Unterformular mit den Kategorien und der Möglichkeit zum Selektieren

üblichen Elemente eines Formulars ausgeblendet werden, stellen wir außerdem die Eigenschaften **Datensatzmarkierer** und **Navigationsschaltflächen** auf **Nein** ein. Die Eigenschaft **Trennlinien** ist in neueren Access-Versionen ohnehin bereits deaktiviert.

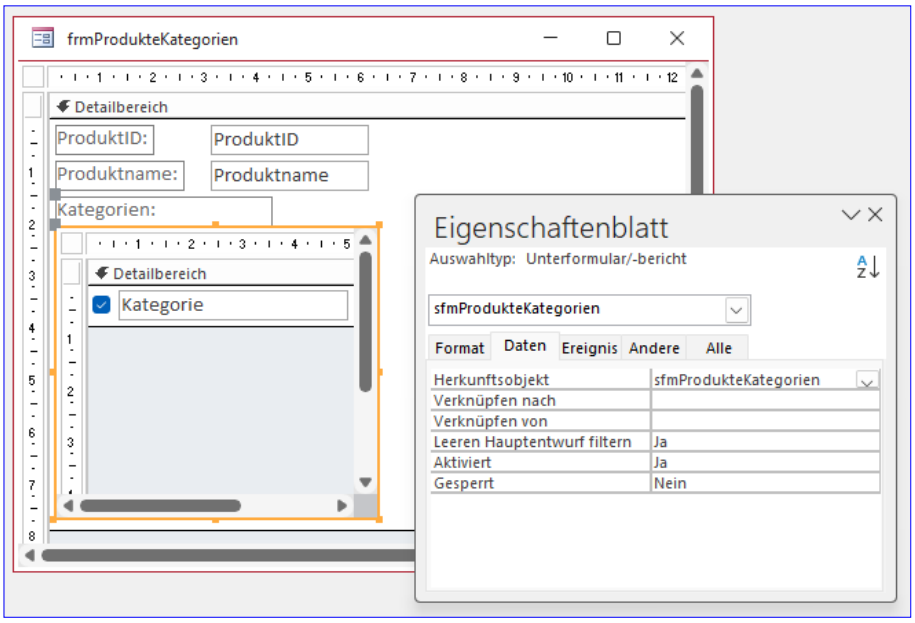


Bild 7: Einbau des Unterformulars

Außerdem zeigt das Formular seine Datensätze aktuell noch mit alternierender Hintergrundfarbe an. Das ändern wir, indem wir den Detailbereich des Unterformulars markieren und dort im Bereich **Format** die Eigenschaft **Alternative Hintergrundfarbe** auf den gleichen Wert einstellen wie die Eigenschaft **Hintergrundfarbe**.

Für die perfekte Optik entfernen wir noch den Rahmen des Textfeldes. Dazu stellen wir die Eigenschaft **Rahmenart** auf **Transparent** ein.

Dieses Formular ziehen wir nun als Unterformular in das Formular **frmProdukteKategorien**. Das Ergebnis sieht wie in Bild 7 aus. Die beiden Eigenschaften **Verknüpfen von** und **Verknüpfen nach** des Unterformular-Steuerelements sollten unbedingt leer sein.

Wechseln wir nun in die Formularansicht, erhalten wir schon einmal einen Einblick, wie das Formular später aussehen könnte (siehe Bild 8). Allerdings ist das Unterformular noch leer. Dies ändern wir allerdings jetzt.

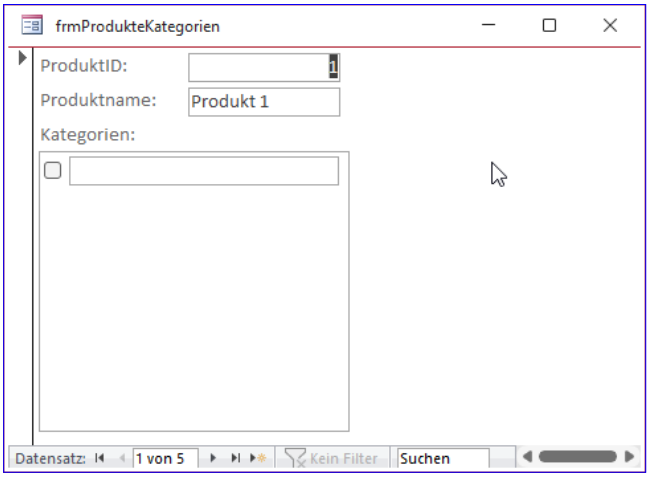


Bild 8: Das noch leere Unterformular in der Formularansicht

Unterformular füllen

Dazu hinterlegen wir für das Ereignis **Beim Anzeigen** des Hauptformulars die Ereignisprozedur aus Listing 1.

Diese Prozedur leert zunächst mit einer **DELETE**-Abfrage die Tabelle **tblKategorienSelektiert**, da sich in der Regel noch Daten vom zuvor angezeigten Datensatz darin befinden.

Dann fügt die Prozedur alle Einträge der Tabelle **tblKategorien** zur Tabelle **tblKategorienSelektiert** hinzu und sortiert die Kategorien dabei direkt alphabetisch.

Dazu nutzen wir eine **INSERT INTO**-Abfrage, welche eine entsprechende **SELECT**-Anweisung auf Basis der Tabelle **tblKategorien** als Quelle verwendet.

m:n-Beziehung mit Listenfeld und Datenblatt

Es gibt verschiedene Möglichkeiten, eine m:n-Beziehung zwischen zwei Tabellen in Formularen abzubilden: Mit zwei Listenfeldern, mit einem Haupt- und einem Unterformular und viele weitere. In diesem Beitrag schauen wir uns eine Kombination aus Listenfeld und Datenblatt an. Dabei betrachten wir das Beispiel von Fahrzeugen und Ausstattungsmerkmalen. Eigentlich sollte man meinen, das wäre eine reine m:n-Beziehung, in der die Verknüpfungstabelle nur die Zuordnung der Merkmale zu den Fahrzeugen vornimmt. Allerdings liefert einer unserer Kunden ein Beispiel, bei dem es etwas aufwendiger wird: Zusätzlich zur reinen Zuordnung soll auch noch festgelegt werden können, welche Ausstattungsmerkmale mit auf das Preisschild sollen und welche Serien- und welche Sonderausstattungen es gibt. Kein Problem: Dann bauen wir einfach eine ergonomische Lösung für diesen Fall, wie dieser Beitrag zeigt.

Man könnte annehmen, dass die Zuordnung von Ausstattungen zu Fahrzeugen über eine einfache m:n-Verknüpfungstabelle zu realisieren ist, welche jeweils ein Fremdschlüsselfeld zum Herstellen einer 1:n-Beziehung zur Tabelle der Fahrzeuge enthält und eines zum Herstellen einer 1:n-Beziehung zur Tabelle der Ausstattungsmerkmale. Doch einer unserer Kunden hatte eine Anforderung, die darüber hinausgeht: Er möchte nicht nur festlegen, ob ein Ausstattungsmerkmal zu einem Fahrzeug gehört oder nicht, sondern auch noch angeben können, ob dieses auf das Preisschild gelangen soll.

Falls Sie sich fragen, warum man nicht einfach alle Ausstattungen dort unterbringt: Manchmal hat ein Fahrzeug einfach so viele Ausstattungsmerkmale, dass diese nicht alle auf das Preisschild passen, das im Fahrzeug hinter die Windschutzscheibe gelegt wird. Dann heißt es aussortieren! Bisher hat der Kunde dies manuell erledigt.

Das heißt, er hat auf Basis der m:n-Beziehung zwischen den Fahrzeugen und den Ausstattungen eine Kommaseparierte Liste generiert, die er zunächst in einem Memofeld gespeichert hat. Dann hat er diesen Text in ein zweites Memofeld kopiert und die Merkmale, die am wenigsten Kaufanreiz bieten oder so selbstverständlich sind, dass sie nicht aufgeführt werden müssen, aus der zweiten Liste entfernt – solange, bis diese auf das Preisschild passte.

Das ist allerdings etwas umständlich und auch fehleranfällig:

- Erstens musste er, wenn er einmal versehentlich ein Ausstattungsmerkmal hinzugefügt hat, das nicht zum Fahrzeug gehört oder wenn er eine Ausstattung nachträglich hinzufügen wollte, alle Texte überarbeiten.
- Zweitens ist es grundsätzlich immer anzustreben, so wenig wie möglich von Hand Daten nachträglich zu bearbeiten.

Also mussten wir uns zwei Dinge überlegen:

- Wie können wir die Wünsche des Kunden im Datenmodell abbilden?
- Und wie setzen wir diese Änderungen in der Benutzeroberfläche um, sodass die Eingabe so intuitiv und einfach wie möglich ist – und möglichst wenig Nachbearbeitung erfordert?

Aktuelles Datenmodell

Bevor wir starten, schauen wir uns das aktuelle Datenmodell an. Eigentlich hatte der Kunde die Fahrzeuge und Ausstattungen in jeweils einer Tabelle gespeichert und die Zuordnung der Ausstattungen zu den Fahrzeugen über eine 1:n-Beziehung mit Mehrfachauswahl realisiert. Dies haben wir in einem ersten Schritt, den wir im Beitrag **m:n-Daten wie im mehrwertigen Feld selektieren** (**www.access-im-unternehmen.de/1424**) erläutern, in eine echte m:n-Beziehung umgewandelt.

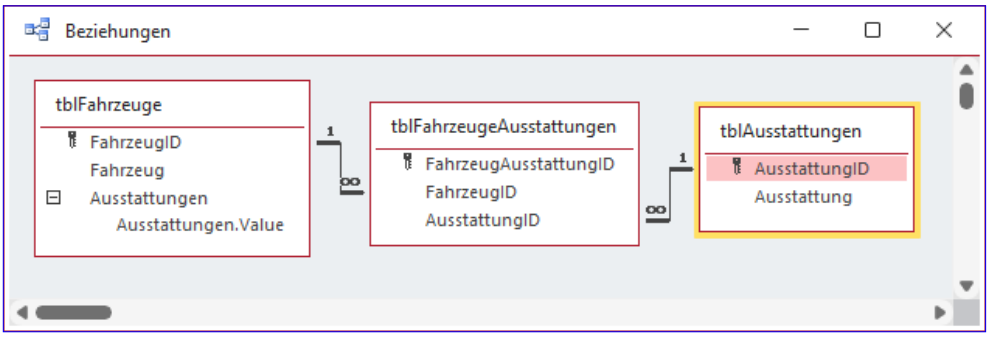


Bild 1: Ausgangsdatenmodell

Die Ausgangssituation sieht also wie in Bild 1 aus. Wir sehen zwei Tabellen namens **tblFahrzeuge** und **tblAusstattungen**. Diese werden über die Tabelle **tblFahrzeugeAusstattungen** über jeweils ein Fremdschlüsselfeld verbunden.

Wir haben außerdem einige Beispieldatensätze hinzugefügt. Diese sehen in den Tabellen in der Datenblattansicht wie in Bild 2 aus.

Anpassung des Datenmodells

Der erste Teil der Aufgabe besteht darin, folgende Informationen in das Datenmodell aufzunehmen:

- Soll eine Ausstattung auf dem Preisschild angezeigt werden?
- Wenn sie angezeigt wird, soll sie unter der Kategorie Serienausstattung oder Sonderausstattung erscheinen?

Wenn wir davon ausgehen, dass es nur die beiden Kategorien **Serienausstattung** oder **Sonderausstattungen** gibt, könnten wir in einem ersten Schnellschuss zwei **Ja/Nein**-Felder in der Tabelle **tblFahrzeugeAusstattungen** unterbringen, die beispielsweise die Namen **Serienausstattung** und **Sonderausstattung** tragen.

Aber meistens kommen dem Kunden im Verlauf immer noch weitere Ideen, was sich mit der neuen Funktion noch so alles anstellen lässt – in diesem Fall vielleicht noch eine weitere Kategorie. Außerdem hat die aktuelle Idee einen Haken: Wir können theoretisch festlegen,

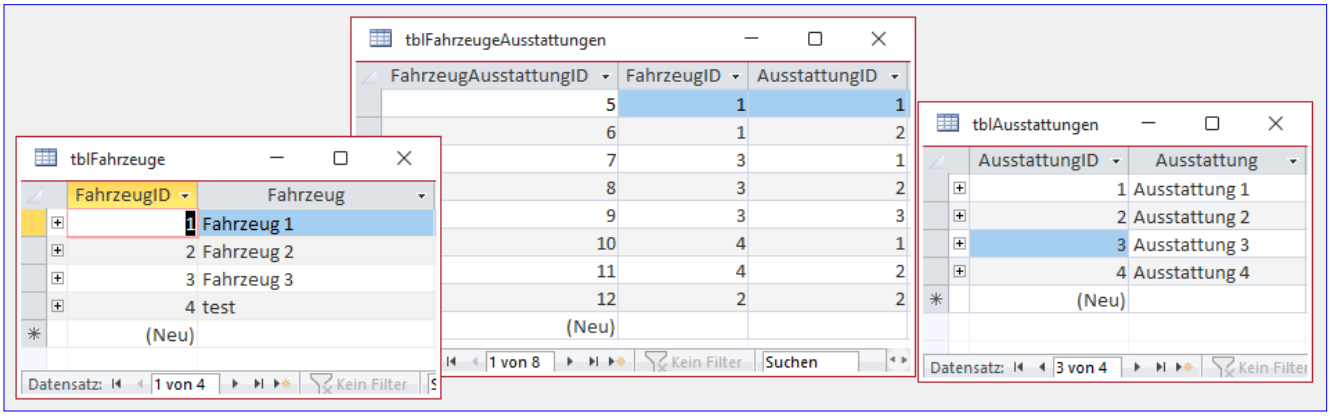


Bild 2: Beispieldaten und Zuordnung über die m:n-Verknüpfungstabelle

dass ein Ausstattungsmerkmal gleichzeitig zur Serienausstattung und zur Sonderausstattung gehört, indem wir beide **Ja/Nein**-Felder anhaken (siehe Bild 3).

Wir müssten schließlich bereits im Formular verhindern, dass beide Felder angekreuzt werden.

Zusammengefasst mit der Idee, dass neben Serienausstattung oder Sonderausstattung noch eine weitere Kategorie hinzukommen könnte, die im Bericht für das Preisschild aufgeführt werden soll, sollten wir diesen ersten Entwurf also verwerfen.

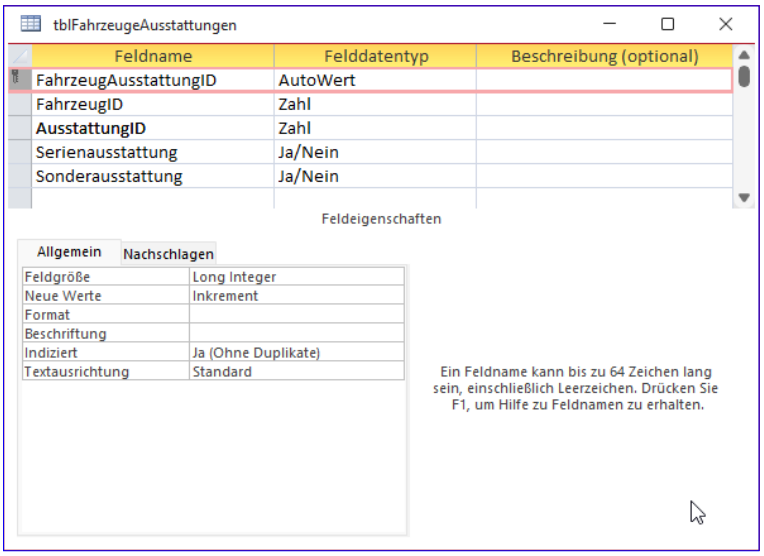


Bild 3: Erster Entwurf für die zusätzlichen Informationen

Die nächste Frage ist: Wird ein Ausstattungsmerkmal in mehr als einer Kategorie auf dem Preisschild erscheinen? Das hört sich wenig sinnvoll an. Ausgehend davon, dass eine Ausstattung nun entweder

- nicht auf dem Preisschild erscheint oder
- unter einer Kategorie wie Serienausstattung, Sonderausstattung oder einer anderen Kategorie erscheint,

können wir den folgenden Entwurf ins Auge fassen: Wir erstellen eine neue Tabelle namens **tblAusstattungskategorien** mit den Feldern **AusstattungskategorieID** und **Ausstattungskategorie** und mit Werten wie **Serienausstattung** oder **Sonderausstattung**. Diese sieht in der Entwurfsansicht wie in Bild 4 aus.

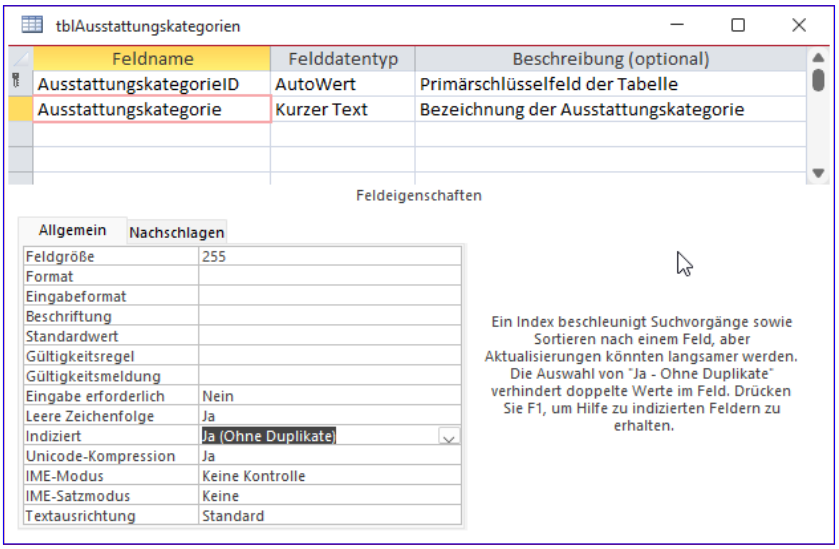


Bild 4: Entwurf der Tabelle **tblAusstattungskategorien**

Der m:n-Verknüpfungstabelle **tblFahrzeugeAusstattungen** fügen wir ein neues Feld namens **PreisschildAusstattungskategorieID** hinzu. Warum nennen wir dieses Feld nicht einfach **AusstattungskategorieID**? Weil wir flexibel bleiben wollen und es vorkommen kann, dass der Kunde noch für eine andere Ausgabeart als Preisschilder festlegen möchte, ob die Ausstattung dort ausgegeben werden soll und unter welcher Kategorie.

Das Feld legen wir als Nachschlagefeld an, die ihre Werte aus der Tabelle **tblAusstattungskategorien** entnimmt. Warum als Nachschlagefeld? Weil dadurch später, wenn wir das Feld in ein Formular ziehen, direkt ein Kombinationsfeld auf Basis

dieses Nachschlagefeldes erstellt wird.

Außerdem fügen wir der Tabelle noch ein **Ja/Nein**-Feld namens **Preisschild** hinzu. Die Tabelle sieht im Entwurf nun wie in Bild 5 aus.

Wir stellen außerdem noch die Eigenschaft **Beschriftung** der Felder wie folgt ein:

- **AusstattungID**: **Ausstattung**
- **PreisschildAusstattungskategorieID**: **Kategorie**
- **AufPreisschild**: **Auf Preisschild**

Diese Beschriftungen werden später bei der Übernahme in das Formular übernommen, sodass wir diese dort nicht nochmals anpassen müssen.

Im Datenmodell spiegeln sich die Erweiterungen nun wie in Bild 6 wieder.

Ausstattungen auswählbar machen

Schließlich wandeln wir das Feld **AusstattungID** noch in ein Nachschlagefeld um. Da wir dafür bereits im

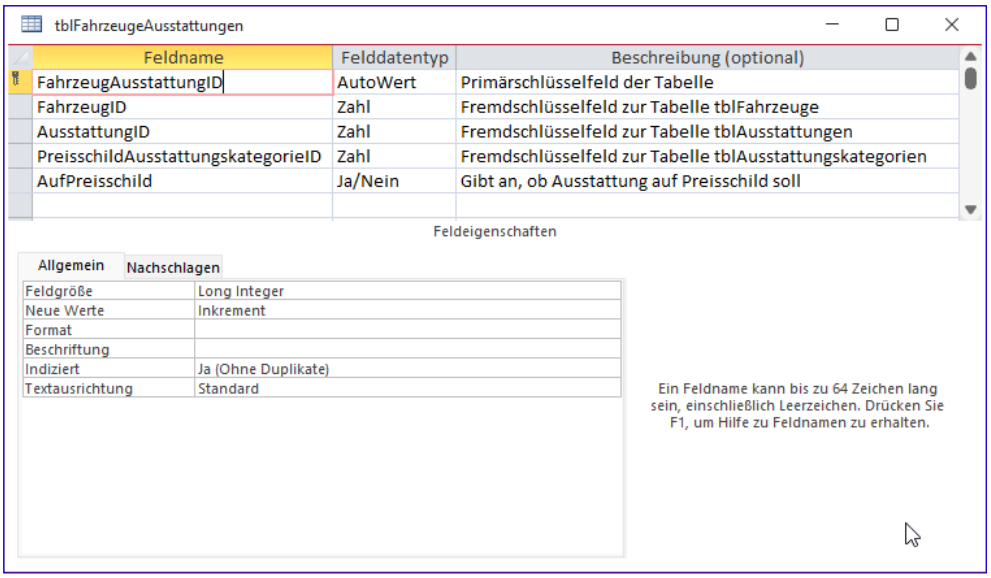


Bild 5: Entwurf der Tabelle **tblFahrzeugeAusstattungen** mit den neuen Feldern

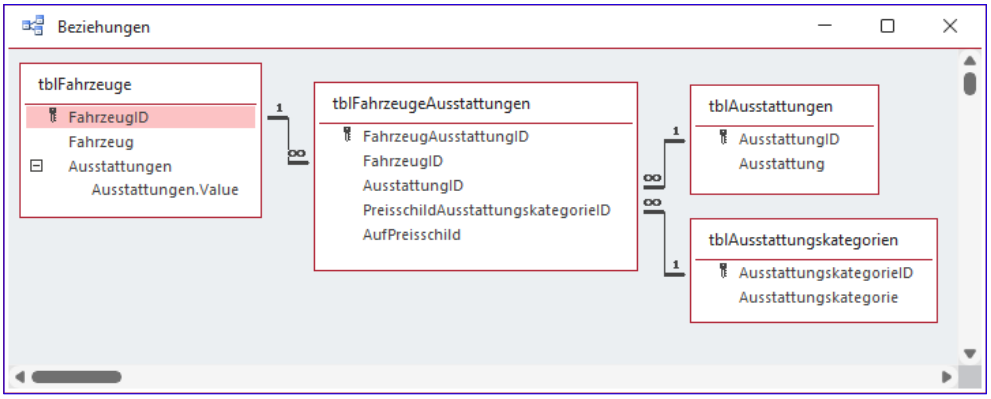


Bild 6: Das Datenmodell mit der Tabelle für die Ausstattungskategorien

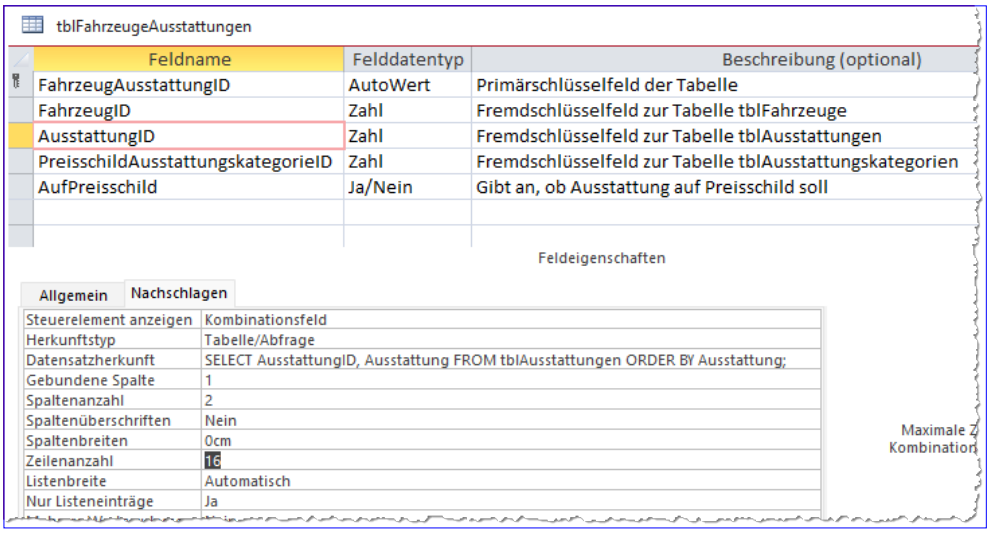


Bild 7: Einstellen des Feldes **AusstattungID** als Nachschlagefeld

oben genannten Beitrag eine Beziehung angelegt haben, fügen wir die Nachschlagefeld-Eigenschaften kurzerhand manuell hinzu.

Dazu aktivieren wir im Tabellenentwurf für das Feld **AusstattungID** in den Eigenschaften die Registerseite **Nachschlagen**.

Hier stellen wir als Erstes den Wert der Eigenschaft **Steuerelement anzeigen** auf **Kombinationsfeld** ein. Dadurch blendet Access einige weitere Eigenschaften ein, die wir wie in Bild 7 anpassen.

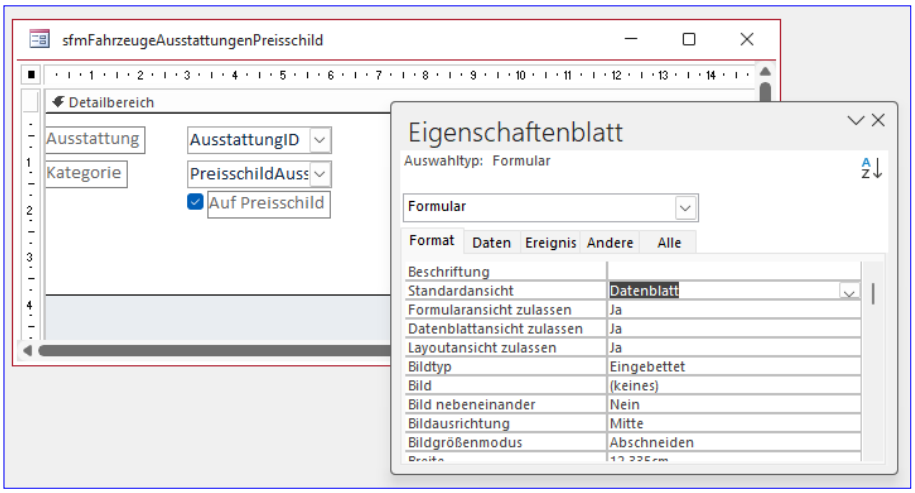


Bild 8: Das Unterformular **sfmFahrzeugeAusstattungenPreisschild**

Wichtig ist hier, dass die nachzuschlagenden Daten aus der Tabelle **tblAusstattungen** nach dem Namen der Ausstattungen sortiert ausgegeben werden.

Ideen für die Benutzeroberfläche

Wie aber wollen wir die Benutzeroberfläche gestalten? Wenn wir schon eine m:n-Verknüpfungstabelle verwenden, wie sie auch beispielsweise bei Bestellungen, Bestellpositionen und Artikeln verwendet wird, können wir diese dann nicht auch so abbilden? Also so, dass wir in einem Hauptformular das Fahrzeug abbilden und im Unterformular in der Datenblattansicht die zugeordneten Ausstattungen? Mit den weiteren Steuerelementen, mit denen wir angeben, ob die Ausstattung auf dem Preisschild erscheinen soll und ob es sich um eine Serien- oder eine Sonderausstattung handelt?

Das wird auf jeden Fall Bestandteil der Lösung sein, aber nicht allein. Wie starten jedoch erst einmal mit diesem Teil.

Fahrzeuge und Ausstattungen in Haupt- und Unterformular

Wir beginnen mit dem Unterformular namens **sfmFahrzeugeAusstattungenPreisschild**, dem wir die Tabelle **tblFahrzeugeAusstattungen** als **Datensatzquelle** zuweisen. Den Namen **sfmFahrzeugeAusstattungen** verwenden wir nicht, weil dieser in der Beispieldatenbank bereits für ein anderes Objekt vergeben ist.

Wir ziehen die drei Felder **AusstattungID**, **PreisschildAusstattungskategorieID** und **AufPreisschild** in den Formularentwurf und stellen die Eigenschaft **Standardansicht** auf den Wert **Datenblatt** ein (siehe Bild 8). Danach speichern und schließen wir das Formular.

Das Hauptformular erstellen wir direkt im Anschluss. Es soll den Namen **frmFahrzeugeAusstattungenPreisschild** erhalten und verwendet die Tabelle **tblFahrzeuge** als **Datensatzquelle**. Wir ziehen lediglich die beiden Felder **FahrzeugID** und **Fahrzeug** in den Detailbereich des Formularentwurfs.

Außerdem ziehen wir dort das Formular **sfmFahrzeugeAusstattungenPreisschild** aus dem Navigationsbereich als Unterformular hinein.

Wenn wir das Unterformular-Steuerelement markieren und im Eigenschaftenblatt die Seite **Daten** aktivieren, sollten die beiden Eigenschaften **Verknüpfen von** und **Verknüpfen nach** jeweils den Wert **FahrzeugID** enthalten (siehe Bild 9).

Wechseln wir nun in die Formularansicht, sind wir eigentlich schon fertig: Das Hauptformular zeigt die Fahrzeuge an und im Unterformular können wir komfortabel die Ausstattungen auswählen. Die übrigen beiden Steuerelemente erlauben es uns, die Kategorie auszuwählen und ob die Ausstattung auf dem Preisschild erscheinen soll (siehe Bild 10).

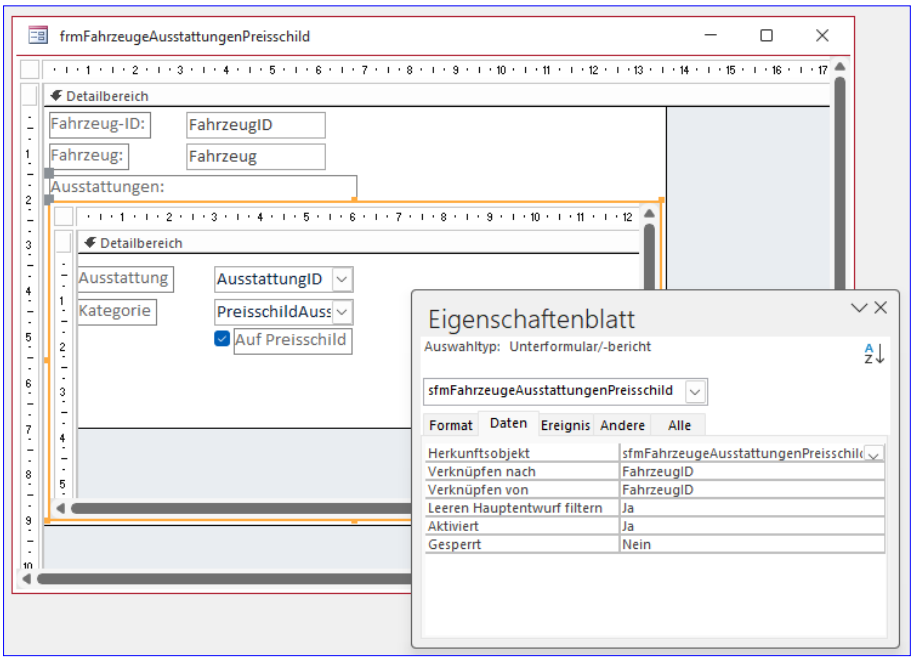


Bild 9: Haupt- und Unterformular in der Entwurfsansicht

Problem: Menge der Ausstattungsmerkmale

Allerdings haben wir ein Problem: Die Liste der Ausstattungsmerkmale enthält mehr als 150 Einträge. Das heißt also, dass das Zusammenstellen eines Fahrzeugs, das durchschnittlich vielleicht 30-50 Ausstattungen enthält, ungefähr so viel Spaß macht wie die Steuererklärung.

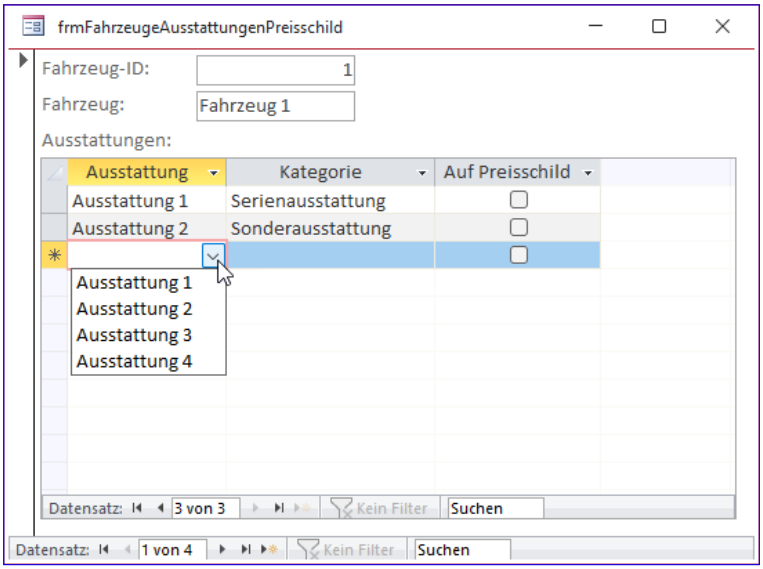


Bild 10: Das Formular in Aktion

Selbst wenn Mitarbeiter die Anfangsbuchstaben aller Ausstattungsmerkmale kennen und diese eingeben, dauert es doch noch eine Weile, bis man alle Daten eingegeben hat.

Einfachere Auswahl per Listenfeld

Also müssen wir hier noch nachlegen, um dem Benutzer ein schnelleres und effizienteres Arbeiten zu ermöglichen. Wir wollen dazu zunächst ein Listenfeld hinzufügen, das alle aktuell nicht ausgewählten Ausstattungsmerkmale anzeigt und mit dem wir per Doppelklick weitere Elemente zur Auswahl hinzufügen können.

Also verschieben wir das Unterformular in der Entwurfsansicht etwas nach rechts und fügen links davon ein Listenfeld namens **lstAusstattungen** hinzu. Dieses soll einfach nur die Ausstattungen in alphabetischer Reihenfolge anzeigen und den Wert des Feldes **AusstattungID** der Tabelle **tblAusstattungen** als unsichtbare erste Spalte liefern.

Die anzuzeigenden Ausstattungen hängen jedoch, da wir nur die noch nicht zugeordneten Ausstattungsmerkmale ausgeben wollen, von den Einträgen der Tabelle **tblFahr-**

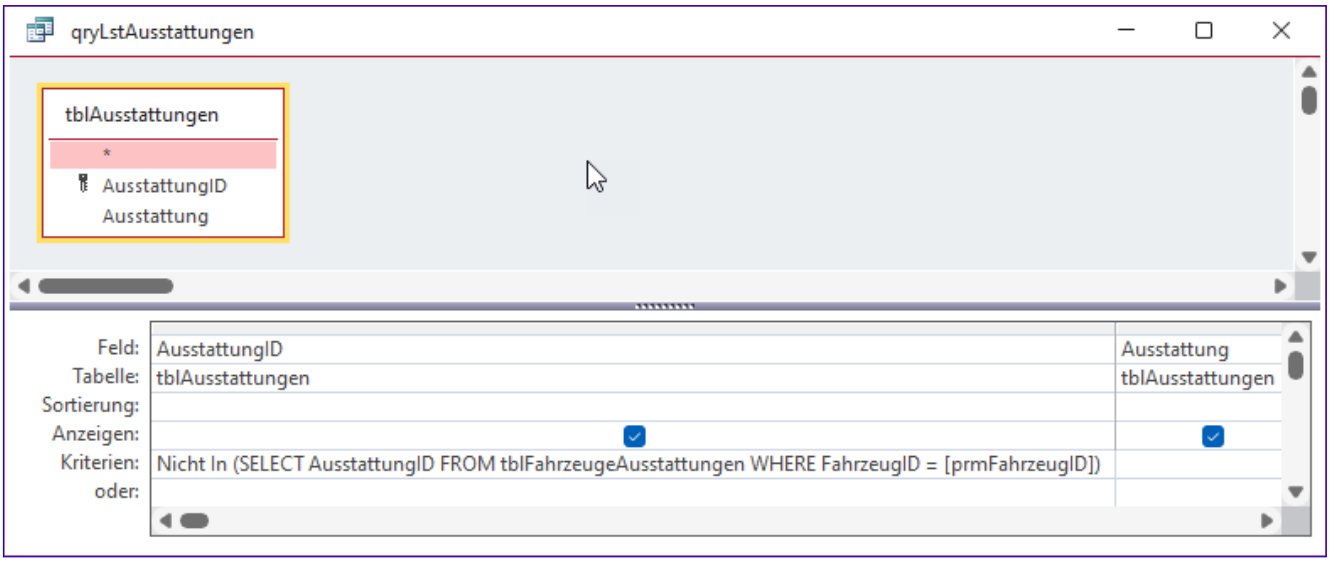


Bild 11: Abfrage zur Ermittlung der noch nicht verwendeten Ausstattungen

zeugeAusstattungen ab, die mit dem aktuell im Hauptformular angezeigten Fahrzeug verknüpft sind.

Die Datensatzherkunft wollen wir daher per VBA zuweisen. Die dazu verwendete Prozedur heißt **Ausstattungsliste-Fuellen**. Wir rufen diese zu verschiedenen Gelegenheiten auf – zum Beispiel beim Anzeigen eines neuen Datensatzes im Hauptformular oder nach dem Hinzufügen eines der **Ausstattungsmerkmale** zur Tabelle **tblFahrzeuge-Ausstattungen**.

Da die Abfrage, um die im Listenfeld anzuzeigenden Datensätze zu ermitteln, etwas komplizierter zu formulieren ist, nutzen wir den Abfrageeditor dazu. Wir wollen mit dieser Abfrage alle Datensätze der Tabelle **tblAusstattungen** liefern, die noch nicht mit einem Datensatz der Tabelle **tblFahrzeugeAusstattungen** verknüpft sind.

Dazu benötigen wir erst einmal eine Abfrage, die uns alle Ausstattungen liefert, die mit dem aktuellen Fahrzeug verknüpft sind.

Diese enthält die Tabelle **tblAusstattungen** als Datenquelle. Wir ziehen beide Felder der Tabelle in das Entwurfsraster. Für das Feld **AusstattungID** legen wir ein Kriterium fest, das alle Datensätze liefert, deren Wert im Feld **AusstattungID** noch nicht über die Tabelle **tblFahrzeugeAusstattungen** mit einem Fahrzeug verknüpft sind. Um welches Fahrzeug es sich handelt, geben wir zunächst über den Parameter **[prmFahrzeugID]** an (siehe Bild 11).

Wechseln wir in die Datenblattansicht, erscheint eine Inputbox für die Eingabe der **FahrzeugID** des zu unter-

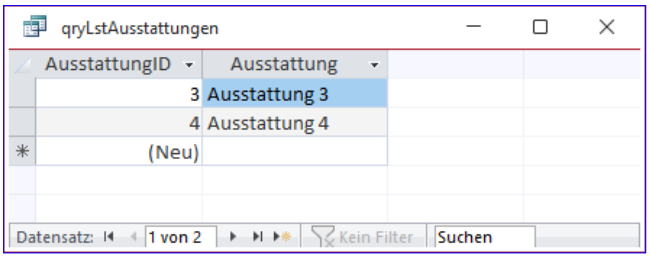


Bild 12: Die Abfrage **qryLstAusstattungen**

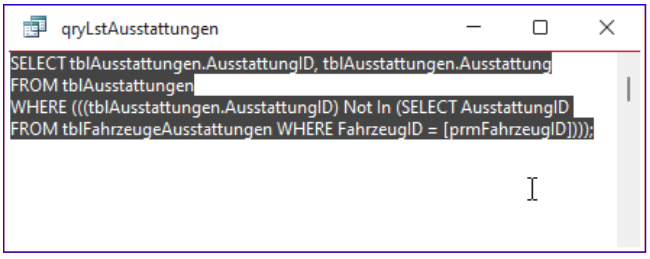


Bild 13: SQL-Ausdruck der Abfrage

Schnellsuche im Listenfeld mal anders

In vielen bisherigen Lösungen haben wir für die Schnellsuche im Listenfeld oder auch in Unterformularen in der Datenblattansicht ein Textfeld als Suchfeld verwendet. Direkt bei Eingabe eines jeden Zeichens wurde das Suchergebnis aktualisiert. In diesem Beitrag wollen wir einmal eine noch ergonomischere Version vorstellen. Der Unterschied soll so aussehen, dass man den Suchbegriff eingeben kann, während das Listenfeld den Fokus hat. Es soll also kein Wechseln vom Suchfeld zum Listenfeld und zurück nötig sein, wenn man durch die Suche den gewünschten Datensatz vorgefunden hat und diesen beispielsweise markieren und beispielsweise durch Betätigen der Eingabetaste eine Aktion für diesen Eintrag durchführen möchte. Wir zeigen dies am Beispiel aus dem Beitrag »m:n-Beziehung mit Listenfeld und Datenblatt« (www.access-im-unternehmen.de/1510).

Im oben genannten Beitrag haben wir bereits eine effiziente Möglichkeit entwickelt, um eine m:n-Beziehung mithilfe eines Listenfeldes und eines Datenblattes in einem Unterformular abzubilden. Das Ziel war es, dass wir nicht wie in üblichen Bestellformularen einfach nur das gesuchte Produkt aus dem Auswahlfeld der hinzuzufügenden Bestellposition heraussuchen können.

Stattdessen wollten wir die Möglichkeit bieten, solche Einträge aus einem Listenfeld auszuwählen und diese schnell per Doppelklick zu dem Datensatz im übergeordneten Formular hinzuzufügen. Bei dem Beispiel dreht es sich um Fahrzeuge und ihre Ausstattungen (siehe Bild 1).

Es ist nun vielleicht bereits etwas effizienter, einen Eintrag aus dem Listenfeld auszuwählen und dieses per Doppelklick zum Fahrzeug hinzuzufügen – gerade, wenn die ausgewählten Einträge danach nicht mehr über das Listenfeld angeboten werden. Allerdings kann es sein, dass es über 200 Einträge in der Liste der Sonderausstattungen gibt. In diesem Fall ist es immer noch langwierig, wenn man sich durch die ganze Liste wühlen muss, um zum gesuchten Eintrag zu gelangen. Auch eine alphabetische Sortierung ist nicht unbedingt zielführend, weil manche Ausstattungen so benannt sind, dass der gesuchte Begriff noch nicht einmal vorn in der Bezeichnung der Ausstattung steht.

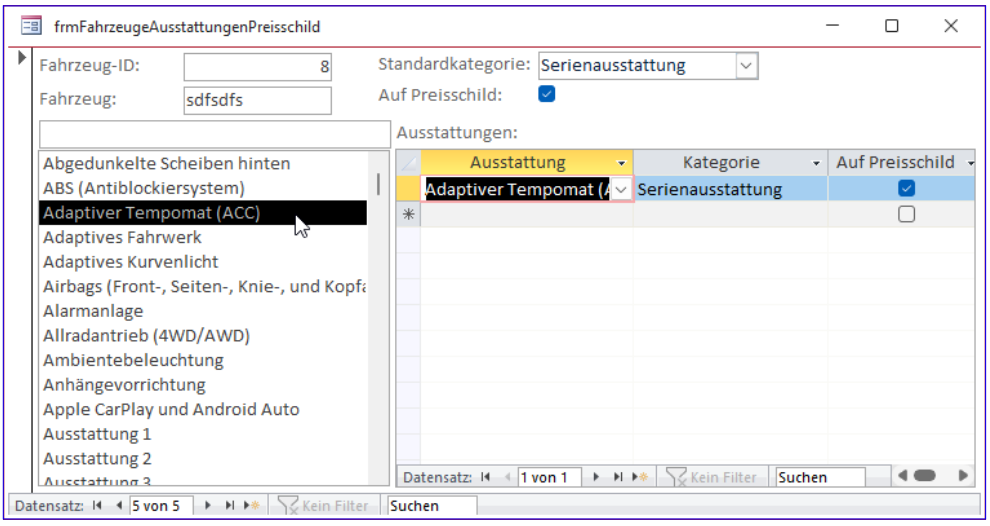


Bild 1: Beispiel für die Listenfeldsuche

Suchfunktion zum Listenfeld hinzufügen

Der nächste Schritt wäre also eine Such- beziehungsweise Filterfunktion zum Listenfeld hinzuzufügen. Eine solche haben wir schon des Öfteren gebaut und wir haben immer ein Textfeld zum Listenfeld hinzugefügt, in das der Benutzer zeichenweise den Suchbegriff eingeben kann.

Der Inhalt des Listenfeldes wurde nach jeder Änderung aktualisiert, damit der Benutzer schnell sehen konnte, wenn der gesuchte Eintrag im sichtbaren Bereich auftauchte oder sogar ganz oben in der Liste stand.

Diesem Textfeld, das wir wie auch in diesem Fall meist **txtSuche** nennen, fügen wir für die Ereigniseigenschaft **Bei Änderung** den Wert **[Ereignisprozedur]** hinzu und hinterlegen für diese eine Ereignisprozedur wie die folgende:

```
Private Sub txtSuche_Change()
    AusstattungslisteFuellen Me!txtSuche.Text
End Sub
```

Diese ruft die Prozedur **AusstattungslisteFuellen** auf und übergibt dieser den aktuellen Inhalt des Textfeldes **txtSuche** als Parameter.

Füllen des Listenfeldes auf Basis des Suchkriteriums

Die Prozedur **AusstattungslisteFuellen** erwartet mit dem Parameter **strSuche** den Suchbegriff, also die Zeichenket-

```
Public Sub AusstattungslisteFuellen(Optional strSuche As String)
    Dim strSQL As String
    Dim lngFahrzeugID As Long
    If Me.NewRecord = False Then
        lngFahrzeugID = Nz(Me!FahrzeugID)
        strSQL = "SELECT AusstattungID, Ausstattung " _
            & "FROM tblAusstattungen " _
            & "WHERE [Suche]AusstattungID " _
            & "Not In (" _
            & "SELECT AusstattungID " _
            & "FROM tblFahrzeugeAusstattungen " _
            & "WHERE FahrzeugID = " & lngFahrzeugID _
            & ") " _
            & "ORDER BY Ausstattung;"
        If Len(strSuche) = 0 Then
            strSQL = Replace(strSQL, "[Suche]", "")
        Else
            strSQL = Replace(strSQL, "[Suche]", "Ausstattung LIKE '*" & strSuche & "*' AND ")
        End If
    Else
        strSQL = "SELECT AusstattungID, Ausstattung FROM tblAusstattungen ORDER BY Ausstattung"
    End If
    Me!lstAusstattungen.RowSource = strSQL
    Me!lstAusstattungen.Value = Null
End Sub
```

Listing 1: Prozedur zum Aktualisieren der Einträge im Listenfeld

te, die in den anzuzeigenden Listenfeld-Einträgen enthalten sein muss (siehe Listing 1).

Sie prüft zuerst, ob das Hauptformular aktuell einen neuen, leeren Datensatz enthält. In diesem Fall wird der zweite Teil der **If...Then**-Bedingung ausgeführt, der dem Listenfeld eine Datensatzherkunft zuweist, die alle Einträge der Tabelle **tblAusstattungen** liefert.

Anderenfalls ermittelt die Prozedur den Primärschlüsselwert des im Hauptformular angezeigten Fahrzeugs und speichert diesen in der Variablen **lngFahrzeugID**.

Anschließend setzt sie in der Variablen **strSQL** eine SQL-Anweisung zusammen, die alle Datensätze der Tabelle **tblAusstattungen** liefert, die noch nicht mit dem aktuell angezeigten Fahrzeug verknüpft sind.

Diese Abfrage verwendet eine Unterabfrage, die alle Einträge der Tabelle **tblFahrzeugeAusstattungen** liefert, deren **FahrzeugID** mit dem Wert des aktuellen Fahrzeugs übereinstimmt.

Hier finden wir einen Platzhalter namens **[Suche]**, der anschließend ersetzt wird. Befindet sich im Textfeld **txtSuche** eine Zeichenkette mit einer Länge von **0** Zeichen, wird **[Suche]** einfach durch eine leere Zeichenkette ersetzt.

Anderenfalls fügen wir für **[Suche]** ein vollständiges Kriterium wie **Ausstattung LIKE '*Suchbegriff*' AND** ein.

Schließlich wird der Inhalt aus **strSQL** der Eigenschaft **RowSource** (**Datensatzherkunft**) des Listenfeldes zugewiesen und der Wert des Listenfeldes wird auf **Null** eingestellt.

Auf diese Weise funktioniert die herkömmliche Suche über das Textfeld (siehe Bild 2) – das Listenfeld liefert nur noch solche Einträge, die zum angegebenen Suchbegriff passen.

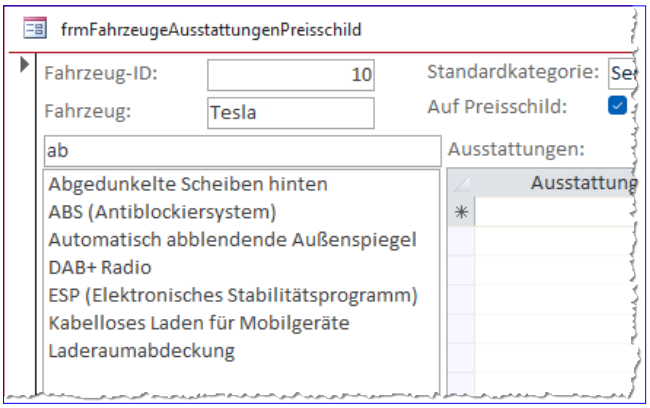


Bild 2: Beispiel für eine Suche über das Textfeld

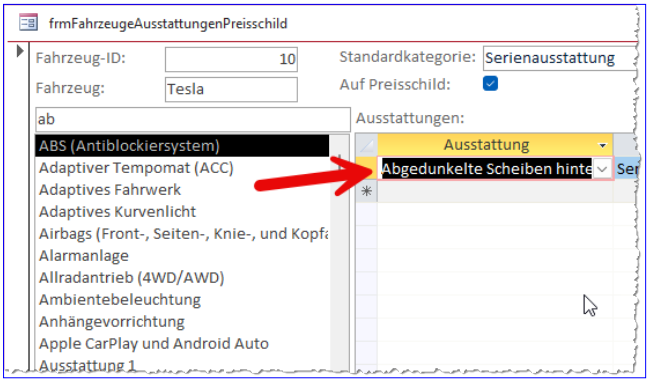


Bild 3: Ein per Doppelklick hinzugefügter Eintrag

Upgrade der Schnellsuche

Nun stellen wir uns vor, wie das Arbeiten in diesem Formular funktioniert: Der Benutzer gibt einen Suchbegriff ein, bis er den gewünschten Eintrag sieht, und klickt diesen dann mit der Maus doppelt an, damit dieser zum Fahrzeug hinzugefügt wird (siehe Bild 3). Alternativ kann er auch den Fokus zum Listenfeld wechseln, den gewünschten Eintrag markieren und diesen mit der Eingabetaste zu den Ausstattungen des Fahrzeugs hinzufügen. Zusätzlich können wir auch noch dafür sorgen, dass das Suchfeld geleert wird und das Listenfeld wieder alle noch verfügbaren Einträge anzeigt – das dürfte in den meisten Settings sinnvoll sein.

Wenn der Benutzer nun jedoch das nächste Ausstattungsmerkmal hinzufügen und dieses dazu suchen möchte, muss er den Fokus zunächst manuell wieder zum Suchen-Textfeld zurückversetzen.

Letztes geöffnetes Ribbon-Tab merken

Wer das Ribbon ausgiebig nutzt, hat schnell einige Tabs zusammen, die wiederum mehrere Gruppen mit den jeweiligen Steuerelementen enthalten. Diese Tabs sind in der Regel so ausgelegt, dass das zuerst angezeigte Tab aktiviert wird, wenn der Benutzer die Anwendung startet. Wenn der Benutzer aber regelmäßig eher mit den Aufgaben einsteigt, die sich in einem anderen Tab befinden, muss er jedes Mal erst noch zu diesem Tab wechseln. Um dies zu vereinfachen, stellen wir in diesem Beitrag eine Lösung vor, mit der sich die Anwendung das zuletzt verwendete Tab-Element merken kann und dieses beim nächsten Start wiederherstellt.

Umfangreiche Anwendungen enthalten manchmal ebenso reichhaltige Ribbon-Definitionen mit vielen Tabs, Groups und Steuerelementen. Benutzer wählen ein Tab aus, um Funktionen mit den darin enthaltenen Steuerelementen aufzurufen.

Gegebenenfalls ist es für den Benutzer hilfreich, wenn er öfter Befehle eines speziellen **tab**-Elements nutzt, dass dieses Tab beim Schließen und erneuten Starten einer Anwendung direkt wieder angezeigt wird.

Die damit verbundenen Techniken wollen wir in diesem Beitrag einmal vorstellen. Dazu haben wir eine kleine Beispielanpassung des Ribbons vorbereitet, die zunächst einmal nur drei Tabs mit den jeweiligen Gruppen und je einem Steuerelement anzeigt.

Diese definieren wir wie in Listing 1.

```
<?xml version="1.0"?>
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
  <ribbon>
    <tabs>
      <tab id="tab1" label="Tab 1">
        <group id="grp11" label="Gruppe 1-1">
          <button idMso="Copy" size="large"/>
          <button idMso="Cut" size="large"/>
          <button idMso="Paste" size="large"/>
        </group>
      </tab>
      <tab id="tab2" label="Tab 2">
        <group id="grp21" label="Gruppe 2-1">
          <button idMso="FindDialog" size="large"/>
        </group>
      </tab>
      <tab id="tab3" label="Tab 3">
        <group id="grp31" label="Gruppe 3-1">
          <button idMso="CreateTable" size="large"/>
          <button idMso="CreateTableInDesignView" size="large"/>
        </group>
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

Listing 1: Ausgangsposition für das Speichern des zuletzt verwendeten Tab-Elements

Wir sehen das **customUI**-Element mit dem **ribbon**- und dem **tabs**-Element, dem wiederum drei **tab**-Elemente untergeordnet sind.

Diese enthalten jeweils eine Gruppe mit ein paar Schaltflächen auf Basis eingebauter Steuerelemente. Damit dieses Ribbon in der Beispielanwendung wie in Bild 1 erscheint, sind folgende Schritte zu erledigen:

- Anlegen einer Tabelle namens **USysRibbons** wie in Bild 2, der wir die Daten wie abgebildet hinzufügen.

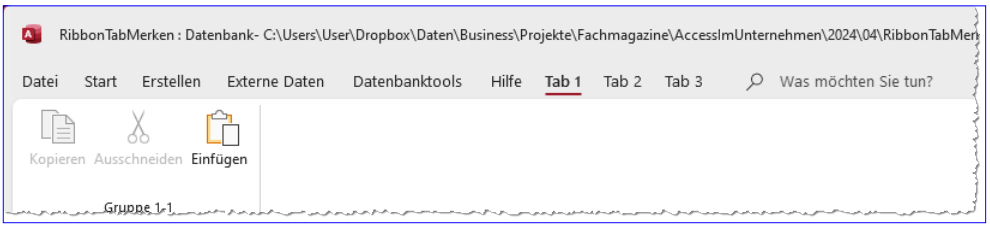


Bild 1: Beispielribbon, dessen Tabs wir uns merken wollen

- Komprimieren und Reparieren der Datenbank (schnellste Version, um eine Datenbankdatei zu schließen und wieder zu öffnen)
- Öffnen der Access-Optionen und dort unter **Aktuelle Datenbank** im Bereich **Menüband- und Symbolleistenoptionen** die Option **Name des Menübands** auf **Main** einstellen (siehe Bild 3).
- Erneutes Komprimieren und Reparieren.

Danach sollte das Ribbon wie in der Abbildung erscheinen. Wenn Sie nur die benutzerdefinierten Ribbon-Einträge hinzufügen wollen, ändern Sie die Zeile mit dem **Ribbon**-Element wie folgt:

```
<ribbon startFrom-
Scratch="true">
```

Danach noch einmal Komprimieren und Reparieren und das Ribbon erscheint wie in Bild 4.

Letztes Ribbontab speichern

Wie aber bekommen wir es nun hin, dass sich die Anwendung das beim Schließen geöffnete Ribbontab merkt? Dazu sind grob die folgenden Schritte nötig:

- Beim Schließen der Anwendung das aktuell geöffnete **tab**-Element ermitteln.
- Eine Information über dieses **tab**-Element an einer beliebigen Stelle speichern.
- Beim Öffnen der Anwendung das gespeicherte Tab wiederherstellen.

Den letzten Punkt können wir abdecken, wenn wir wissen, dass es eine **ActivateTab**-Methode gibt, der wir den Na-

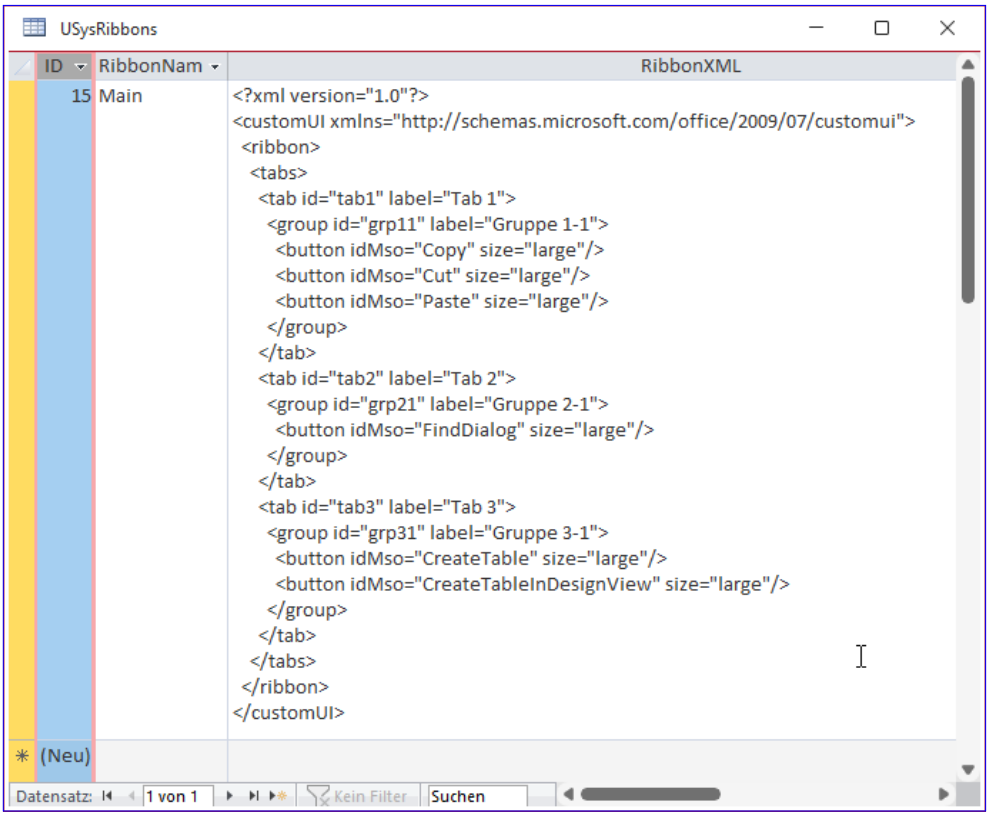


Bild 2: Tabelle **USysRibbons** mit unserer Ribbondefition

men des zu aktivierenden **tab**-Elements übergeben.

Der zweite Punkt ist reine Fleißarbeit. Wir können uns überlegen, ob diese Information in einer Tabelle der Datenbank, einer Textdatei, der Registry oder auch einer Datenbankeigenschaft gespeichert wird. Da die Information je Benutzer gespeichert werden soll, ist es sinnvoll, diese in der Registry im Bereich für das aktuelle Benutzerkonto zu hinterlegen.

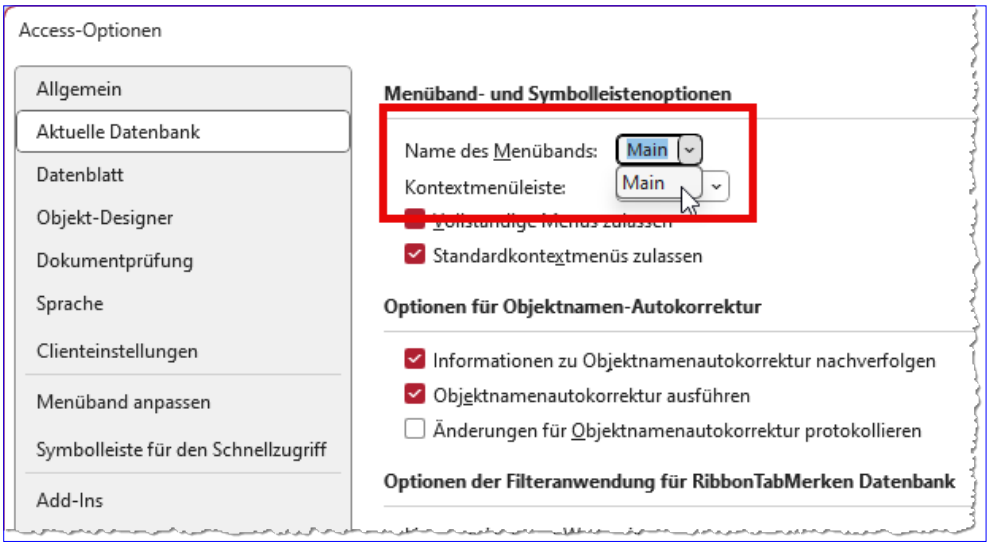


Bild 3: Einstellen der Ribbondefinition als Anwendungsribbon

Der erste Punkt ist etwas schwieriger. Zunächst einmal gibt es keine Möglichkeit, zu irgendeinem Zeitpunkt auszulesen, welches **tab**-Element gerade markiert ist. Es gibt keine Auflistung der Tabs im Objektmodell von Access oder Office, und es gibt auch keine Eigenschaft, die das aktuell aktivierte **tab**-Element referenziert.

Wir müssen also all unser Ribbon-Know-how nutzen, um diese Anforderung zu lösen.

Wir wissen: Es gibt keine Methode, mit der wir explizit auf das Wechseln des **tab**-Elements reagieren können. Aber es gibt einige Methoden, die beim Anzeigen von Steuerelementen wie beispielsweise bei **button**-Elementen ausgelöst werden.

So liefert Das **button**-Element einige **get...**-Callback-Funktionen, die immer ausgelöst werden, wenn ein Element erstmals sichtbar gemacht wird – also beispielsweise, wenn sich das **button**-Element auf dem ersten Tab befindet und dieses beim Starten der Anwendung angezeigt wird. Wenn sich ein **button**-Element auf einem anderen, zunächst nicht aktivierten **tab**-Element befindet, werden seine **get...**-Callbacks erst einmal nicht aufgerufen. Dies geschieht erst, wenn das zweite Tab dann eingeblendet wird.

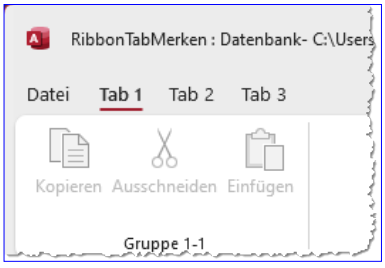


Bild 4: Ribbon ohne eingebaute Elemente

Wir können also auf jedem Tab ein **button**-Element unterbringen, das jeweils beim Aktivieren des **tab**-Elements die **get...**-Callbackfunktion aufruft.

Das **button**-Element muss noch nicht einmal sichtbar sein. Genau genommen soll es das auch gar nicht – wir wollen ein **button**-Element nur für diesen einen Zweck hinzufügen.

Und wenn es ohnehin nicht sichtbar sein soll, können wir auch gleich sein **getVisible**-Attribut nutzen und damit die **getVisible**-Callbackfunktion aufrufen, wenn das **tab**-Element aktiviert wird. In dieser Funktion können wir dann einfach speichern, welches **tab**-Element gerade aktiviert wurde.

Wir fügen also jedem **tab**-Element in einem **group**-Element ein neues **button**-Element hinzu, welches das Attribut **getVisible** enthält und beispielsweise wie folgt aussieht:

Registry-Einträge für VBA-Anwendungen

Die Registry von Windows ist ein undurchschaubarer Dschungel von Einstellungen. Wenn man nicht weiß, was man tut, können Änderungen an diesen Einstellungen zu Problemen beim Einsatz von Windows oder Anwendungen führen. Es gibt einige API-Funktionen, mit denen man auch per VBA Änderungen an der Registry vornehmen kann. Die API-Programmierung ist aber grundsätzlich ein Profithema, und erst recht sollten Einsteiger mit API-Funktionen nicht die Registry manipulieren. Zum Glück hat Microsoft jedoch einen Satz von VBA-Funktionen bereitgestellt, mit denen man halbwegs sicher Informationen in die Registry schreiben und diese auch wieder auslesen kann. Der Clou ist, dass man damit einen speziell für VB- und VBA-Anwendungen vorgesehenen Teil der Registry nutzen kann. In diesem Beitrag stellen wir diese Funktionen vor und zeigen, in welchem Bereich der Registry sie eingesetzt werden können.

Die Windows-Registry speichert eine Vielzahl von Daten, die für den Betrieb des Windows-Betriebssystems und installierter Anwendungen notwendig sind. Dazu gehören Konfigurationsinformationen und Einstellungen für Hardware, Software, Benutzerprofile und Systemressourcen.

Konkret umfasst dies Treiberinformationen, Benutzerkonten, installierte Programme, Systemdienste, Dateizuordnungen, System- und Anwendungsoptionen sowie Netzwerkeinstellungen. Diese Daten sind in einer hierarchischen Struktur organisiert, die als Registry bezeichnet wird.

Hier gibt es verschiedene Bereiche, von denen vor allem der Bereich **HKEY_LOCAL_MACHINE** und **HKEY_CURRENT_USER** interessant sind. Der Unterschied ist, dass **HKEY_LOCAL_MACHINE** Informationen speichert, die für alle Benutzer bereitstehen sollen, während **HKEY_CURRENT_USER** nur Informationen speichert, die mit dem aktuellen Benutzer in Zusammenhang stehen.

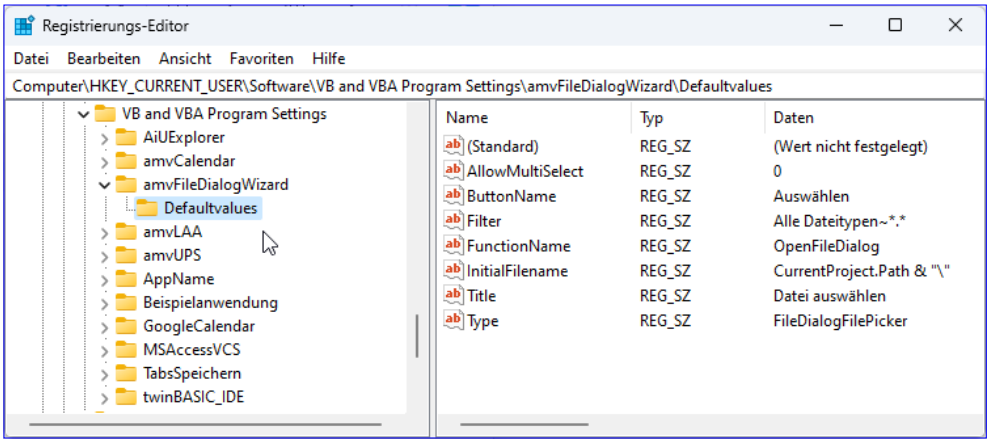


Bild 1: Bereich der Registry für VB- und VBA-Anwendungen

Wir schauen uns in diesem Beitrag einen Bereich der Registry an, der ein Teilbereich von **HKEY_CURRENT_USER** ist. Damit ist dieser Speicherort für Informationen in der Registry der perfekte Ort für Daten, die in Zusammenhang mit der Benutzung einer Anwendung durch den aktuellen Benutzer stehen.

Mit den nachfolgend vorgestellten VBA-Funktionen können wir also prima Daten in der Registry speichern,

die bei der Arbeit mit einer Anwendung anfallen und aus verschiedenen Gründen dauerhaft gespeichert werden sollen – beispielsweise, um diese beim nächsten Start der Anwendung wieder abzurufen.

Bereich für VB und VBA

Als Erstes schauen wir uns den Bereich genauer an, in dem wir unsere Anwendungsdaten speichern können.

Dazu öffnen wir die Registry, was zum Beispiel gelingt, indem wir den Text **Reg** in die Windows-Suche eingeben und dann das Ergebnis **Registrierungs-Editor** auswählen. Dies öffnet das Fenster **Registrierungs-Editor**. Hier können wir direkt zum folgenden Pfad navigieren:

```
Computer\HKEY_CURRENT_USER\Software\VB and VBA Program Settings
```

Darunter befinden sich Einträge in einer fest vorgegebenen Hierarchie. Die erste Ebene, in der wir im Screenshot die Namen von Anwendungen sehen, ist für die einzelnen Anwendungen vorgesehen.

In der zweiten Ebene können wir für jede Anwendung unterschiedliche Sektionen einrichten, sofern dies nötig ist (siehe Bild 1).

Wir müssen aber zumindest eine Sektion anlegen, um mit den gleich vorgestellten VBA-Funktionen auf die Daten zugreifen zu können.

Diese Daten befinden sich als Name-Wert-Paare in den einzelnen Sektionen. Neben dem Namen legen wir einen Wert fest, wobei wir mit den VBA-Funktionen aus diesem Beitrag ausschließlich Text-Einträge anlegen können.

Registry-Eintrag anlegen

Schreiten wir direkt zur Tat und legen einen Registryeintrag per VBA an. Nehmen wir an, unsere Anwendung heißt **aiuRegistrybeispiele**. Die Sektion soll **Optionen** heißen. Damit haben wir schon zwei der vier Parameter der Anweisung **SaveSetting**. Hier sind die Optionen in der Übersicht:

- **AppName**: Name der Anwendung
- **Section**: Sektion
- **Key**: Name des Schlüssels
- **Setting**: Wert der Einstellung

Mit der folgenden Anweisung legen wir einen ersten Eintrag an:

```
SaveSetting "aiuRegistrybeispiele", "Optionen", _
    "Beispielkey", "Beispielsetting"
```

Dieser Eintrag sieht anschließend wie in Bild 2 aus.

Registry-Eintrag auslesen

Um diesen Eintrag wieder auszulesen, verwenden wir die Funktion **GetSetting**. Diese erwartet ebenfalls vier Parameter, von denen der letzte allerdings optional ist.

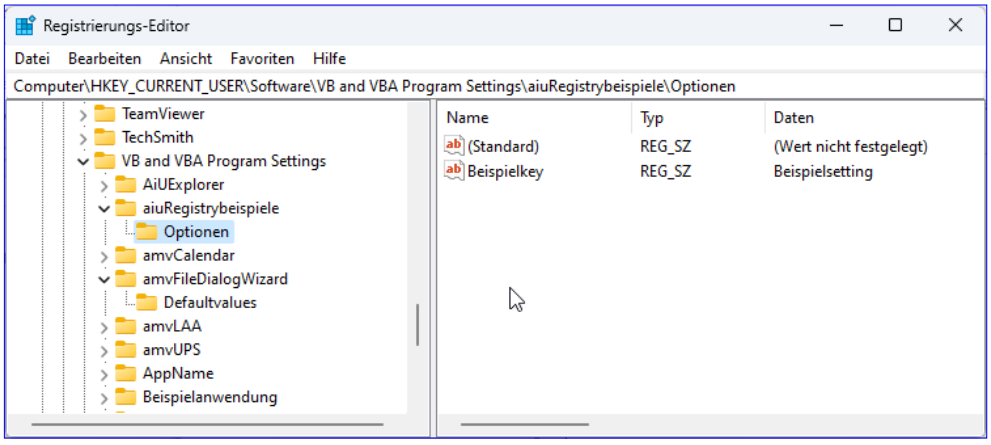


Bild 2: Selbst angelegter Eintrag in der Registry

Access und SQL Server: Der Migrations-Wizard

Wer schnell die Tabellen einer Access-Datenbank zum SQL Server migrieren möchte, kann dazu ein kostenloses Tool von Microsoft nutzen: Den SQL Server Migration Assistant, in diesem Fall in der Ausführung für Access. Wie wir die Tabelle einer Access-Datenbank zum SQL Server übertragen und gleichzeitig auch noch die passenden Verknüpfungen für den Zugriff auf diese Datenbank erstellen, zeigen wir in diesem Beitrag. Dabei nutzen wir die schnellste zur Verfügung stehende Methode, nämlich den Migration Wizard, ein Assistent im Assistenten. Im besten Fall kann man danach direkt von der Access-Datenbank auf die Daten im SQL Server-Backend zugreifen. Gegebenenfalls sind noch Vor- und Nacharbeiten erforderlich, um die kümmern wir uns jedoch in eigenen Beiträgen.

Voraussetzungen

Wenn Sie die Beispiele dieses Beitrags ausprobieren möchten, benötigen Sie eine Installation des SQL Servers mit der Möglichkeit, dort eine neue Datenbank anzulegen, das SQL Server Management Studio, um das Ergebnis zu betrachten, eine eigene Access-Datenbank oder die Beispieldatenbank aus dem Download zu diesem Artikel als Testmaterial.

Außerdem wollen wir den SQL Server Migration Assistant für Access herunterladen, den wir beispielsweise über Google mit den Suchbegriffen **sql server migration assistant for access** finden sollten.

Damit landen wir schnell auf der Microsoft-Seite mit dem Download dieses kostenlosen Tools, das wir hier direkt herunterladen können.

Beim Download taucht gegebenenfalls ein Dialog auf, der nach der gewünschten Version fragt. Hier stehen die 32-Bit- und die 64-Bit-Variante zur Verfügung. Welche Sie wählen, hängt von der Office-Version ab, die auf dem Zielsystem installiert ist, nicht von der Windows-Version. Wir haben aktuell noch Office in der 32-Bit-Version installiert, also verwenden wir den Download **SSMAforAccess_9.5.0_x86.msi** (siehe Bild 1).

Nachdem Download führen wir die **.msi**-Datei aus und installieren so den **SQL Server Migration Assistant for Access**, von nun an kurz **SSMA** genannt.

Erster Start des SSMA

Nach dem Start erscheint der SSMA bildschirmfüllend und bietet einen **Migration Wizard** zur Unterstützung an (siehe Bild 2). Diesen können wir per Klick auf die Option **Launch this Wizard at Startup** für zukünftige Starts de-

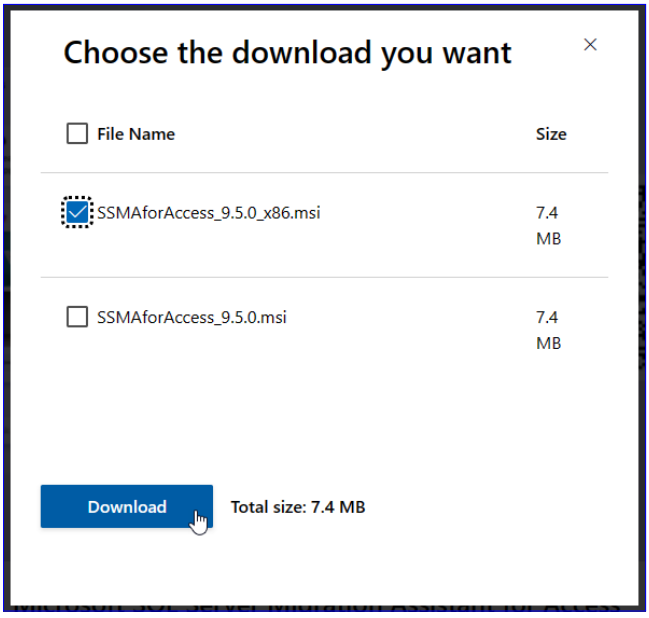


Bild 1: Auswahl der gewünschten Version des SSMA

aktivieren, wenn wir diesen nicht mehr nutzen wollen.

Vorerst wollen wir diesen Wizard jedoch nutzen, um eine schnelle, erste Migration durchzuführen.

Für einen Überblick über die anstehenden Aufgaben bietet es sich an, sich den Ablauf einmal anzusehen, der wie folgt aussieht:

- Erstellen eines neuen SSMA-Projekts
- Hinzufügen von Access-Datenbankdateien zum Migrationsprojekt
- Auswahl der zu migrierenden Objekte
- Verbinden zum SQL Server oder zur Azure Datenbank
- Optional: Verknüpfen der migrierten Tabellen
- Übertragen der Objekte und Migrieren der Daten

Wir führen die Schritte einmal anhand einer kleinen Beispieldatenbank durch.

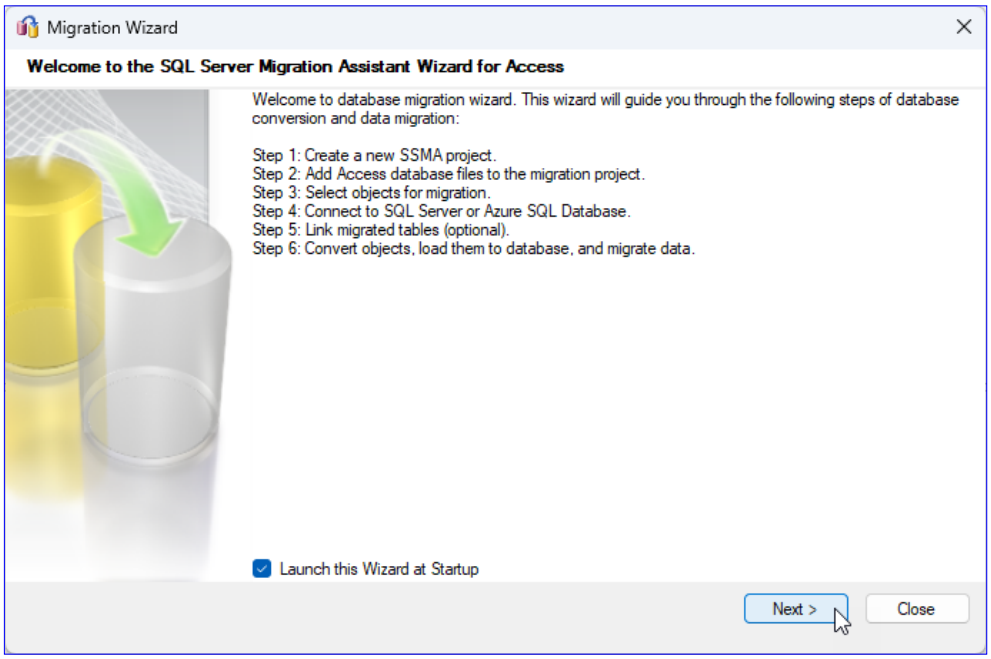


Bild 2: Der Migrations-Assistent

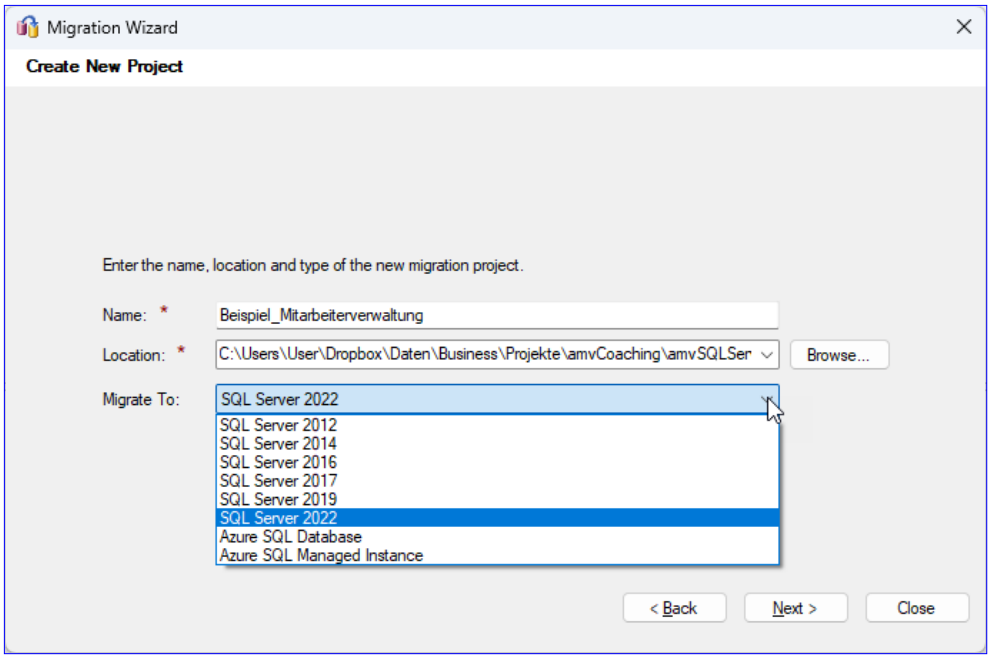


Bild 3: Schritt 1 des Migration Wizard: Erstellen eines neuen Projekts

Diese Beispieldatenbank stellen wir im Beitrag **Datenmodell Mitarbeiterverwaltung** (**www.access-im-unternehmen.de/1499**) vor.

Erstellen des Migrationsprojekts

Im ersten Schritt des **Migration Wizard** finden wir die Eingabemöglichkeiten aus Bild 3 vor. Dieser erwartet die Eingabe von drei Informationen. Die erste ist der Name, unter dem wir die neue Migration speichern wollen. Hier

geben wir beispielsweise **Migration_Mitarbeiterverwaltung** an. Unter **Location** legen wir fest, in welchem Verzeichnis die Projektdatei gespeichert wird.

Der Dateiname ergibt sich aus dem Namen für die Migration und die Dateiendung **.ssma**. Schließlich wählen wir noch die Version des SQL Server aus, der als Ziel der Migration dienen soll. Damit können wir mit einem Klick auf die Schaltfläche **Next** den ersten Schritt abschließen.

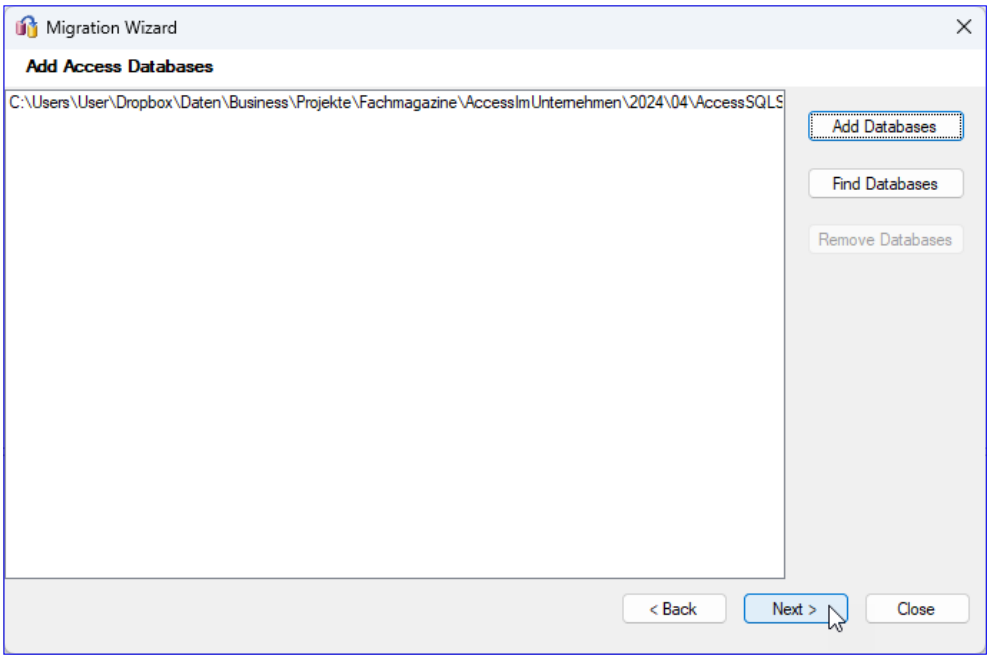


Bild 4: Schritt 2 des Migration Wizard: Hinzufügen der Datenbank, deren Tabellen migriert werden sollen

Hinzufügen von Access-Datenbankdateien zum Migrationsprojekt

Dieser Schritt ist in unserem Fall schnell zu erledigen, denn wir haben lediglich eine einzige Datenbankdatei, in der sich die zu migrierenden Tabellen befinden.

Wir klicken also auf **Add Databases** und wählen im folgenden **Dateiauswahldialog** die Datenbank **Mitarbeiterverwaltung.accdb** aus. Damit können wir in diesem Fall direkt zum nächsten Schritt fortfahren (siehe Bild 4).

Auswahl der Datenbankdatei bei aufgeteilten Datenbanken

Etwas komplizierter wird es jedoch, wenn wir nicht mit einer einzigen Datenbankdatei arbeiten, sondern wenn diese bereits aufgeteilt sind – in ein Frontend und ein Backend oder sogar mehrere Frontends, die auf das Backend zugreifen. Es gibt auch noch kompliziertere Konstellationen mit mehreren Frontends und mehreren Backends, aber auf diesen Fall wollen wir an dieser Stelle nicht eingehen.

Wenn wir eine Anwendung mit Datenbankfrontend und -backend verwenden, enthält das Frontend in der Regel die Anwendungslogik (also Abfragen, Formulare, Berichte und VBA-Code) sowie Tabellenverknüpfungen, über die man auf die Tabellen im Backend zugreifen kann.

Im Backend hingegen befinden sich ausschließlich die Tabellen der Anwendung. Es kann für spezielle Aufgaben auch noch Tabellen im Frontend geben, aber diese würde man, wenn man sie im Frontend-Backend-Szenario nicht in das Backend überführt hat, vermutlich auch nicht in die SQL Server-Datenbank geben.

Intuitiv würde man nun möglicherweise die Backend-Datenbank als Quelle für die Migration der Datenbank zum SQL Server angeben.

Wenn wir allerdings im gleichen Schritt die Funktion des **Migration Wizard** nutzen wollen, um die Tabellenverknüpfungen hinzuzufügen, ergibt dies keinen Sinn: Die Tabellenverknüpfungen für den Zugriff auf die Tabellen im SQL Server würden dann nämlich im Backend der Access-Datenbank landen.

Die zweite Option ist, dennoch das Backend als Quelle für die Migration zu verwenden und anschließend die Tabellenverknüpfungen zum SQL Server auf eine andere Weise als mit dem Migration Wizard zu erledigen.

Bleiben noch zwei Möglichkeiten: Wir geben beide Datenbankdateien an oder wir geben nur die Frontend-datenbank als Quelle für die Migration an.

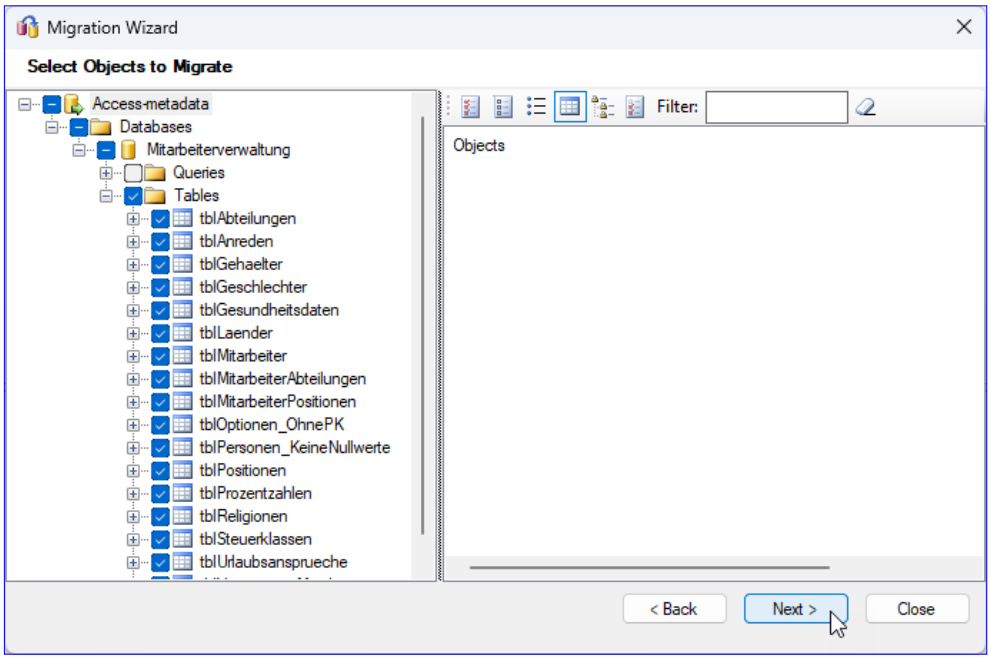


Bild 5: Schritt 3 des Migration Wizard: Auswahl der zu migrierenden Objekte

Wir betrachten erst den letzteren Fall: Hier würde der Migration Wizard über die Tabellenverknüpfungen im Frontend auf die Tabellen im Backend zugreifen, diese migrieren, und auf Wunsch die Tabellenverknüpfungen auf den SQL Server zum Frontend hinzufügen.

Dabei würden die zuvor auf die Tabellen im Access-Backend verweisenden Tabellenverknüpfungen umbenannt werden und die neuen Tabellenverknüpfungen erhalten deren Originalnamen. Dies ist die von uns empfohlene Vorgehensweise bei vorliegendem Frontend und Backend auf Access-Basis.

Schauen wir uns noch kurz an, was passiert, wenn wir beide Datenbanken einer Frontend-Backend-Konstellation als Quelle für die Migration hinzufügen: Dann haben wir später auch die Möglichkeit, sowohl die Tabellenverknüpfungen aus dem Frontend auszuwählen als auch die Originaltabellen aus dem Backend.

Wenn wir beide auswählen, geschieht logischerweise Folgendes: Die Tabellen aus der zuerst angegebenen Quelldatenbank werden in den SQL Server geschrieben. Für die gleichnamigen Tabellen der zweiten Datenbank fragt der Migration Wizard, ob die bereits vorhandenen Tabellen (die von der anderen Quelle) überschrieben werden sollen.

Es landet also ohnehin nur eine Kopie der jeweiligen Tabelle in der SQL Server-Datenbank.

Dafür erhalten wir in beiden Quelldatenbanken, also im Frontend und im Backend, jeweils die Verknüpfungen auf diese Tabellen.

Wie gesagt: In diesem Kontext macht es keinen Sinn, Frontend und Backend als Datenquelle anzugeben.

Zusammengefasst:

- Wenn es ohnehin nur eine Access-Datenbankdatei gibt, wird diese als Quelle angegeben.
- Wenn die Datenbank auf Frontend- und Backenddatei aufgeteilt ist, geben wir die Frontend-Datenbank als Quelle für den Migration Wizard an.

Auswahl der zu migrierenden Objekte

Im folgenden Schritt zeigt der Migration-Wizard auf der linken Seite die gewählten Datenbanken und die enthaltenen Objekte an. Dort werden nicht nur Tabellen, sondern, falls vorhanden, auch Abfragen angezeigt (siehe Bild 5).

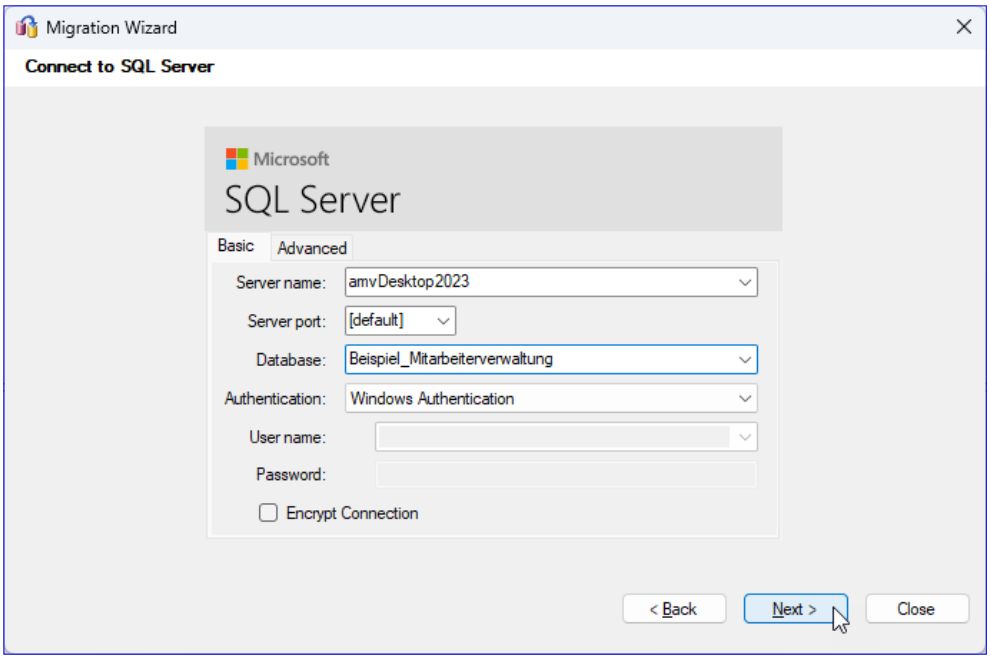


Bild 6: Schritt 4 des Migration Wizard: Eingeben von Informationen zur Zieldatenbank

In diesem Fall wollen wir nur die Tabellen migrieren. Abfragen kann man zwar auch direkt in entsprechende SQL Server-Objekte migrieren, in der Regel enthalten Access-Abfragen jedoch Elemente, die nicht SQL Server-kompatibel sind und viel Nacharbeit erfordern.

Wir wollen uns daher an dieser Stelle auf die Migration der Tabellen beschränken.

Hier haben wir die Gelegenheit, entweder einfach alle Tabellen zu migrieren oder diese noch manuell aus- oder abzuwählen. Gegebenenfalls befinden sich unter den Tabellen solche, die nur temporär und lokal genutzt werden sollen und die wir von der Migration ausklammern sollten – dann gibt es an dieser Stelle die Gelegenheit dazu.

In diesem Fall übernehmen wir einfach alle Tabellen und springen mit einem Klick auf die **Next**-Schaltfläche zum nächsten Schritt.

Verbinden zum SQL Server oder zur Azure Datenbank

Im nächsten Schritt legen wir verschiedene Informationen fest, mit denen wir den Zugriff auf die Zieldatenbank einstellen (siehe Bild 6):

- **Server name**: Gibt die IP-Adresse oder den Namen des SQL Servers an, gegebenenfalls gefolgt vom Namen einer benannten Instanz.
- **Server port**: Gibt den Server-Port an, hier kann man in der Regel den Wert **[default]** beibehalten.
- **Database**: Gibt den Namen der Datenbank an. Bei der erstmaligen Migration können wir den Namen einer be-

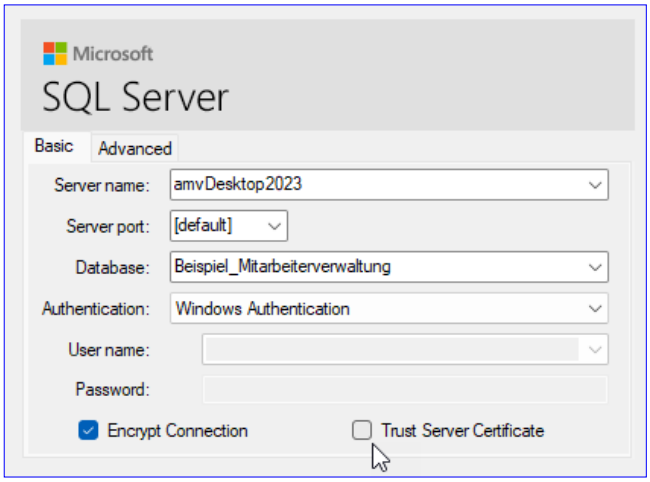


Bild 7: Einstellungen für Verschlüsselung

reits vorhandenen Datenbank als Ziel angeben oder wir legen den Namen der zu erstellenden Datenbank fest.

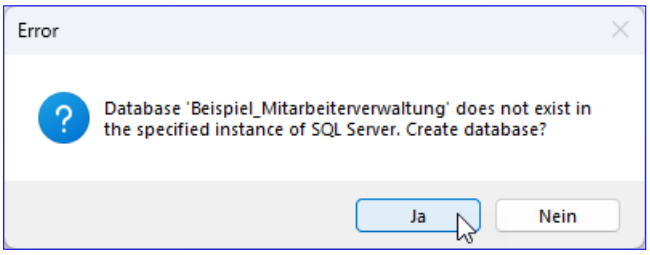


Bild 8: Frage, ob die Datenbank neu erstellt werden soll

- **Authentication**: Gibt die Authentifizierungsmethode an. **Windows Authentication** wird verwendet, wenn man sich auf dem gleichen Rechner oder in der gleichen Domäne wie der SQL Server befindet, ansonsten wird meist **SQL Server Authentication** genutzt. Die beiden folgenden Textfelder werden nur freigeschaltet, wenn man SQL Server Authentication wählt, denn sonst erfolgt die Anmeldung über das Windows-Konto des aktuellen Benutzers.

- **User name**: Nimmt bei SQL Server-Authentifizierung den Benutzernamen entgegen, der im SQL Server definiert sein muss.

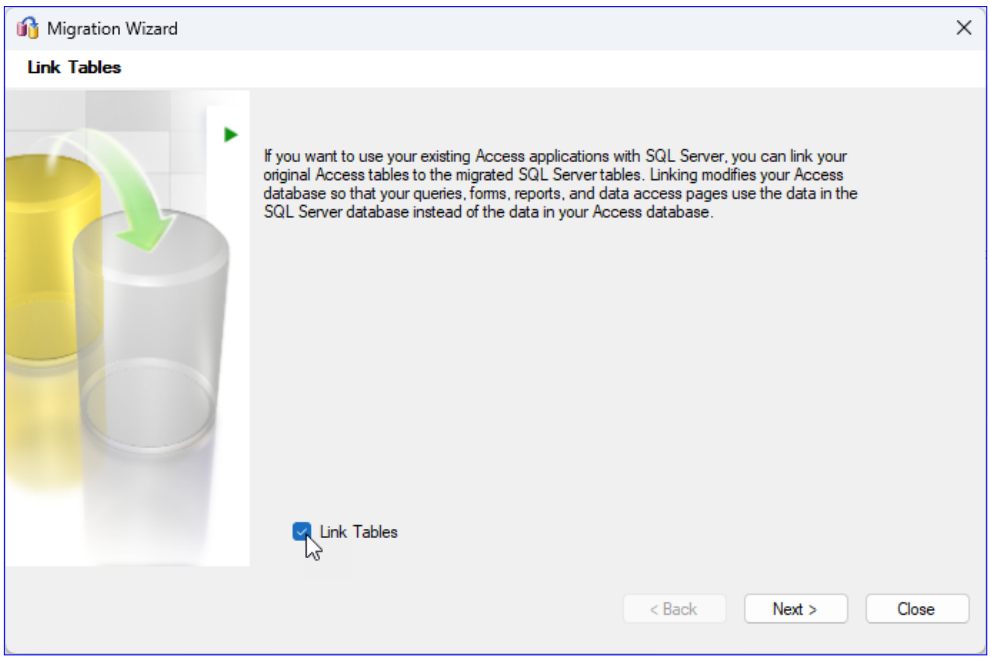


Bild 9: Schritt 5 des Migration Wizard: Angabe, ob die Tabellen direkt verknüpft werden sollen

- **Password**: Nimmt bei Verwendung der SQL Server-Authentifizierung das Kennwort entgegen.

- **Encrypt Connection**: Gibt an, ob die Verbindung verschlüsselt werden soll. Wird diese Option aktiviert, erscheint noch eine weitere Option namens **Trust Server Certificate** (siehe Bild 7).

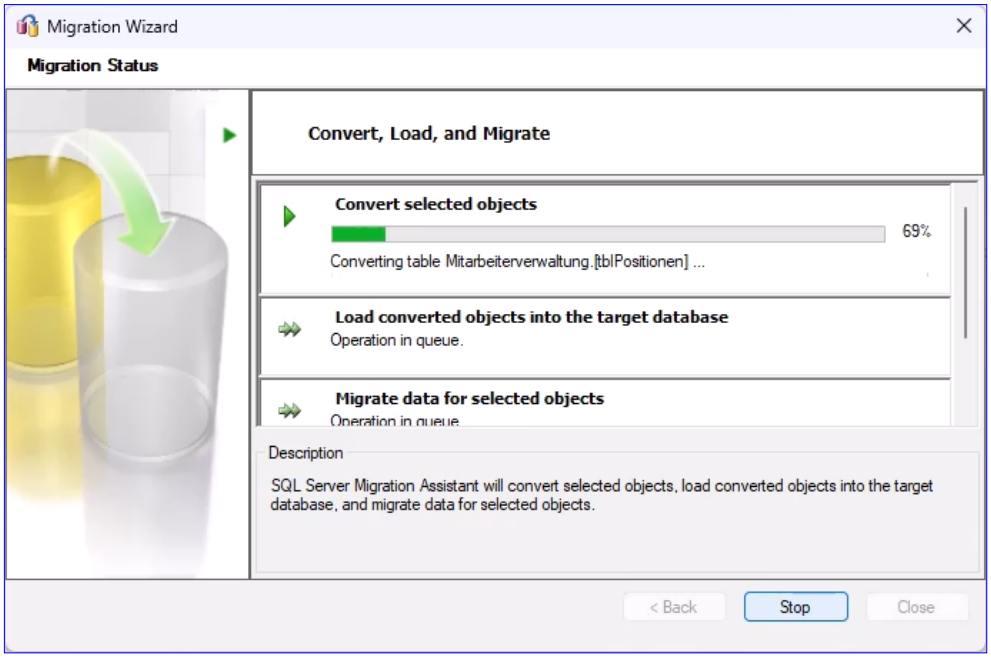


Bild 10: Migration: Konvertieren der zu migrierenden Objekte

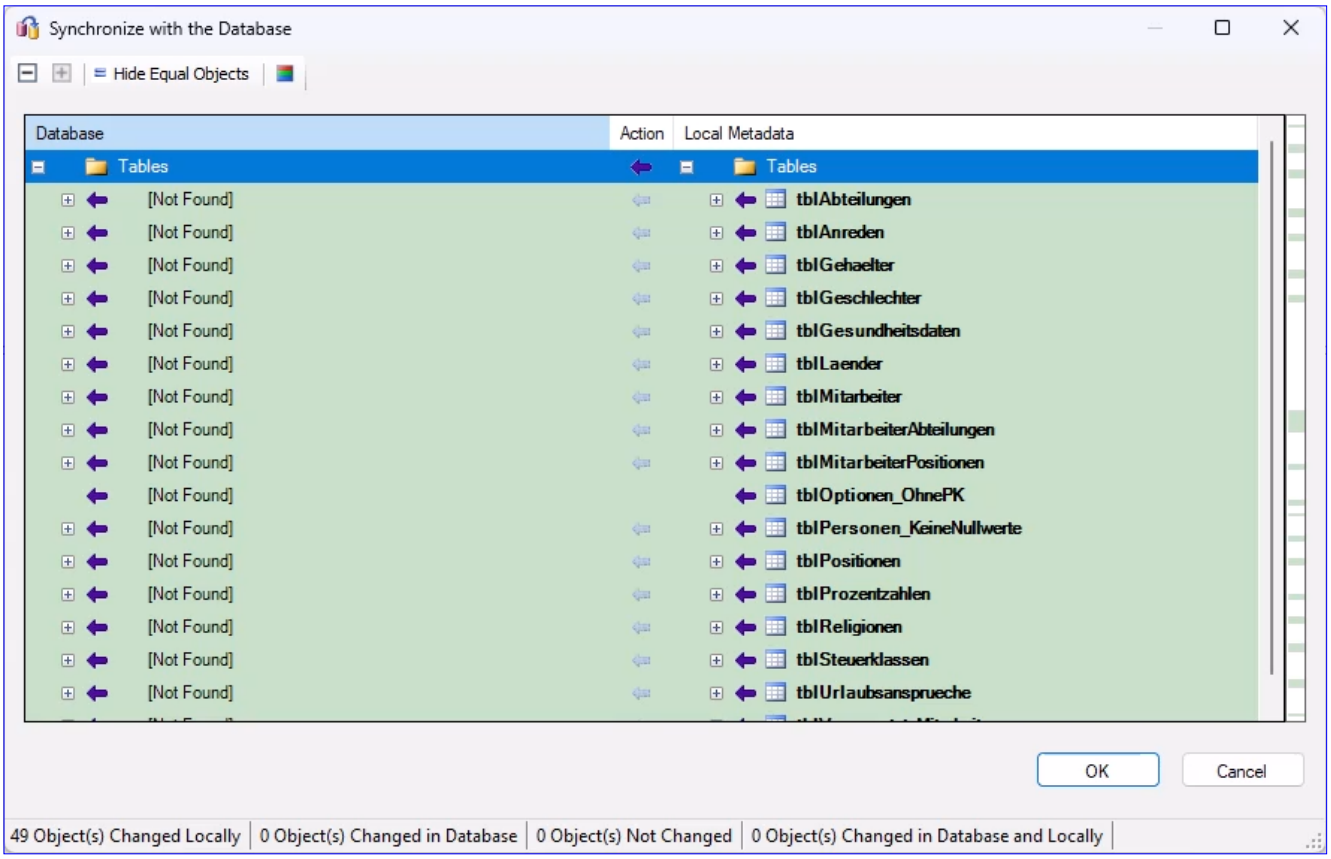


Bild 11: Zuordnung der zu erstellenden Tabellen festlegen

Beim Mausklick auf die Schaltfläche **Next** erscheint eine Meldung mit der Rückfrage, ob die angegebene Datenbank, die noch nicht im SQL Server enthalten ist, neu angelegt werden soll, was wir mit einem Klick auf die Schaltfläche **Ja** bestätigen (siehe Bild 8).

Optional: Verknüpfen der migrierten Tabellen

Einen Schritt später fragt der Migration Wizard noch, ob die Tabellen direkt in die Quelltabelle eingebunden werden sollen (siehe Bild 9). Da wir die Datenbank nach der Migration möglichst direkt mit den SQL Server-Tabellen verwenden wollen, aktivieren wir die entsprechende Option.

Übertragen der Objekte und Migrieren der Daten

Im nächsten Schritt startet die Migration. Dabei werden zuerst, wie im Bereich **Convert selected objects**, die zu migrierenden Objekte konvertiert (siehe Bild 10).

Beim Konvertieren werden meist direkt ein paar Fehler, Warnungen und Hinweise produziert.

Dann folgt der nächste Schritt, wo die konvertierten Elemente in die SQL Server-Datenbank geladen werden, was bedeutet, dass die entsprechenden Elemente in der

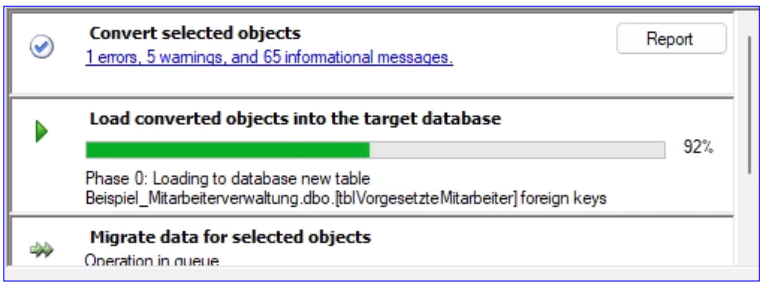


Bild 12: Anlegen der Datenbankobjekte in der Zieldatenbank