HAUFE.

Ausgabe 06/2017

ACCESS

IM UNTERNEHMEN

VEREINSVERWALTUNG

Lernen Sie das Datenmodell und den Export aus einer fiktiven Vereinsverwaltung auf Excel-Basis kennen (ab S. 2).

In diesem Heft:

DATEN ANONYMISIEREN

Stellen Sie Kunden ein Tool zur Verfügung, mit dem dieser seine Daten anonymisieren kann.

SEITE 57

UNTERDATENBLÄTTER

Mit Unterdatenblättern lassen sich in Formularen hierarchische Daten abbilden. Lernen Sie, wie es geht!

SEITE 23

NULL UND CO.

Lernen Sie den Unterschied zwischen Null, Nothing, Empty und der leeren Zeichenfolge kennen.

SEITE 41

www.access-im-unternehmen.de

Mat-Nr. H1583-5104

Vereinsarbeit

Im deutschsprachigen Raum gibt es Vereine wie Sand am Meer. Sportvereine, Kleingartenvereine, Hundeclubs und viele mehr. Was liegt da näher, als einmal die Entwicklung einer Vereinsverwaltung unter die Lupe zu nehmen? Das haben wir, auch aus aktuellen Anlass, in dieser Ausgabe in Angriff genommen. Der Anlass war die Bitte meines eigenen Vereins, doch einmal eine ordentliche Mitgliederverwaltung zu programmieren.

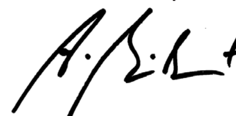
Kein Problem, denkt man sich da: Eine Tabelle für die Mitglieder, ein paar Lookup-Tabellen für Anreden und so weiter und fertig ist die Vereinsverwaltung. Das war allerdings etwas naiv: Auch, wenn die mir gelieferten Daten in einer einzigen Excel-Tabelle untergebracht waren, lieferten diese mir doch Material für wesentlich mehr Tabellen. So gibt es beispielsweise verschiedene Beitragsklassen von kleinen Kindern über Jugendliche bis hin zu aktiven und passiven erwachsenen Mitgliedern. Dann kommen noch diverse Rabatte hinzu: Für Mitglieder, die mit mehreren Familienmitgliedern zum Verein gehören, für verdiente Mitglieder oder auch für Studenten, Auszubildende und Schüler. Es war gar nicht so einfach, ein passendes Datenmodell zu entwerfen. Wie wir dies gelöst haben, lesen Sie im Beitrag **Vereinsverwaltung: Von Excel zum Datenmodell** ab S. 2. Noch komplizierter war es, die Daten aus der einen Excel-Tabelle dann in das neue Datenmodell zu übertragen. Dazu haben wir teilweise mit Abfragen arbeiten können, für manche Einsatzzwecke haben wir uns aber auch maßgeschneiderte VBA-Prozeduren gebaut. Wie wir die Daten aus Excel in die Access-Tabellen geschaufelt haben, erfahren Sie im Beitrag **Vereinsverwaltung: Migration** ab S. 15.

Da ich Ihnen natürlich die Lösung mit einigen Beispieldaten präsentieren wollte, damit Sie sich diese selbst ansehen können, musste ich die Daten noch etwas präparieren: Aus Gründen des Datenschutzes dürfen die Daten der Vereinsmitglieder natürlich nicht weitergegeben werden. Diese Anforderung haben Sie sicher auch gelegentlich, wenn ein Kunde Ihnen seine Datenbank schicken möchte, damit Sie diese bearbeiten können. In diesem Fall gibt es zwei Möglichkeiten: eine entsprechende Geheimhaltungsvereinbarung oder das Anonymisieren der Daten. Letzteres ist unser Vorschlag: Wir haben eine Lösung programmiert, mit der Sie die zu anonymisierenden Tabellen und Felder auswählen und die Inhalte etwa von Feldern wie Vorname oder Nachname durch zufällig gewählte Zeichenketten ersetzen können. Wie das gelingt, erfahren Sie im Beitrag **Daten anonymisieren** ab S. 57.

Für die Anzeige von Tabellen und Abfragen gibt es mit den Unterdatenblättern ein interessantes Feature, mit denen sich hierarchische Daten anzeigen lassen. Wie Sie Unterdatenblätter auch in Formularen einsetzen können, erfahren Sie unter dem Titel **Unterdatenblätter in Formularen** ab S. 23. Und unter **Undo in mehreren Unterformularen** erfahren Sie ab S. 29, wie Sie eine Undo-Funktion für ein Hauptformular mit mehreren Unterformularen implementieren. Dieser Beitrag ist übrigens aus einer Leseranfrage heraus entstanden – senden Sie uns also ruhig Ihre Fragen zu!

Schließlich liefern wir mit **Null, leere Zeichenkette, Nothing und Co.** (ab S. 41) und **Die CurrentDb-Funktion und das Database-Objekt** (ab S. 48) noch einige interessante Techniken rund um den Einsatz von VBA zur Automatisierung Ihrer Datenbank.

Und nun: Viel Spaß beim Lesen!



Ihr André Minhorst

Vereinsverwaltung: Von Excel zum Datenmodell

Die Tage war es soweit: Schwiegervater, seines Zeichens zweiter Vorsitzender eines Sportvereins, druckste herum: »Die Mitgliederdatei, also ... die Excel-Datei – kannst Du uns da nicht mal was Richtiges draus zaubern? Also ein Programm, mit dem ich Mitglieder eingeben und auch mal ein paar Auswertungen machen kann?« Na klar kann ich. Also hatte ich ein neues Projekt: Eine Excel-Tabelle mit allen relevanten Daten sollte in einer relationalen Datenbank landen, die nicht nur die Dateneingabe vereinfachte, sondern auch noch verschiedene andere Aufgaben übernehmen sollte. Ein perfektes Beispiel für eine Lösung in Access im Unternehmen! Dieser Teil beschreibt, wie wir die Daten aus der Excel-Tabelle in ein frisch erstelltes Datenmodell überführen.

Die Herausforderung bringt so Einiges mit sich, was man beim Überführen einer Excel-Tabelle in ein relationales Datenmodell an Herausforderungen erwarten kann. Alle Daten sind in einer einzigen Tabelle gespeichert, Lookup-Daten etwa zum Auswählen von Anreden, Altersklassen et cetera gibt es nicht, verschiedene Felder wie etwa der Nachname werden zum Hinzufügen zusätzlicher Informationen verwendet. Außerdem gibt es noch verschiedene farbige Markierungen, die einen bestimmten Status des Mitglieds belegen.

Verknüpfung erstellen

Als Erstes wollen wir uns die Daten der Excel-Datei in Form einer verknüpften Access-Tabelle für unsere gewohnten Werkzeuge verfügbar machen. Das ist schon mal nicht allzu einfach, da der bisherige Verwalter dieser Daten ja nicht nur Werte in die Spalten eingetragen hat, sondern auch noch Informationen in Form unterschiedlicher Formatierungen, in diesem Fall etwa farbiger Hintergründe oder fetter Schrift hinterlegt hat. Außerdem beginnen die Zeilen mit den Daten auch nicht gleich in der ersten oder zweiten Zeile, sondern im Kopf befindet sich zunächst noch eine Überschrift, die normalweise natürlich im Kopfbereich der Seite landen sollte – genauso wie die Legende im unteren Bereich, die im Fußbereich erscheinen sollte.

Die Lösung für die überflüssigen Zeilen über und unter den relevanten Daten ist relativ einfach: Wir legen dazu

Bild 1: Benennen eines Bereiches

einfach einen Bereich in der Excel-Tabelle fest, den wir dann beim Verknüpfen oder Importieren auswählen können. Wie geht das? Dazu markieren Sie den gewünschten Bereich und klicken dann in das Namenfeld links oben über der Tabelle und geben dort den Namen für den gewählten Bereich ein – in diesem Fall wollen wir den Bereich **Mitglieder** nennen (s. Bild 1).

Wenn Sie dann im Ribbon von Excel den Befehl **Formeln|Definierte Namen|Namensmanager** wählen, erscheint der Dialog aus Bild 2, mit dem Sie die bereits festgelegten Bereiche bearbeiten können. Hier können Sie auch eventuell falsch markierte Bereiche löschen oder bearbeiten.

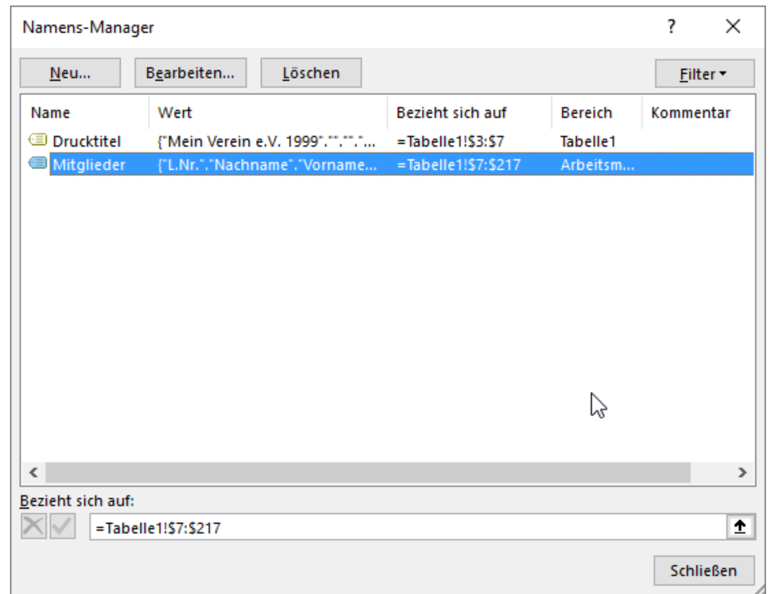


Bild 2: Verwalten der benannten Bereiche

Farbige Markierungen in Daten umwandeln

Damit haben wir die Voraussetzung geschaffen, dass wir die Daten gleich mit Access verknüpfen können. Nun ist allerdings noch ein weiteres Problem vorhanden: Wie können wir die farbigen Markierungen so darstellen, dass diese auch nach dem Verknüpfen von Access aus gelesen werden können?

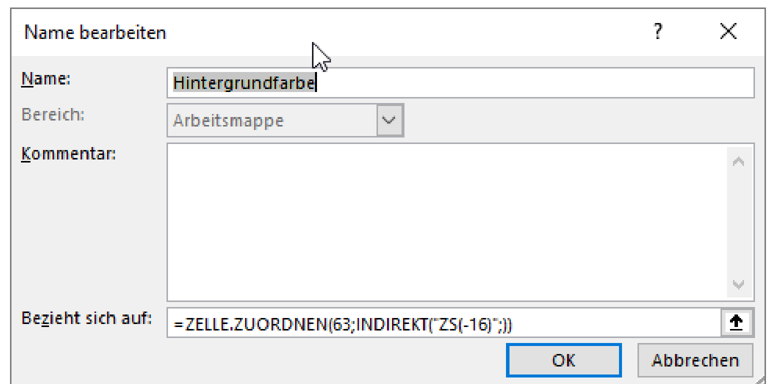


Bild 3: Name für eine Formel auf Basis einer Zelle

Im Detail sieht es es aus, dass wir in der ersten Spalte verschiedene farbige Markierungen vorfinden, die nicht etwa aus einer bedingten Formatierung stammen, sondern die vom Benutzer manuell erstellt wurden. In Access können wir diese Markierungen aber nach dem Importieren oder Verknüpfen der Excel-Tabelle nicht mehr erkennen, diese werden ignoriert. Also müssen wir die Markierungen irgendwie in einen Zahlencode oder eine andere Information übertragen.

Wer zu faul ist, kann natürlich einfach dem Auftraggeber sagen, dass er eine neue Spalte anlegen und die Informationen aus den farbigen Markierungen dort eintragen soll. Aber wir sind ja immer an kreativen Lösungen interessiert und nehmen dies daher selbst in die Hand. In diesem Fall wollen wir eine neue Spalte namens Farbcode anlegen und für die Zellen dieser Spalte eine Formel hinterlegen, welche die Hintergrundfarbe der ersten Spalte ausliest und einen entsprechenden Zahlenwert in die Felder der neuen Spalte einträgt.

Um die Hintergrundfarbe in Form eines Zahlencodes in einer neuen Spalte rechts von den eigentlichen Inhalten anzuzeigen, gehen Sie wie folgt vor:

- Wählen Sie den Ribbon-Eintrag **Formeln|Definierte Namen|Namen definieren**. Dies öffnet den Dialog aus Bild 3.

- Hier geben Sie in das Feld **Name** den Wert **Hintergrundfarbe** ein.
- In das Feld **Bezieht sich auf** geben Sie den folgenden Ausdruck ein: **=ZELLE.ZUORDNEN(63;INDIREKT("ZS(-15)";))**
- Nun schließen Sie den Dialog **Name bearbeiten**.
- Geben sie in die erste freie Spalte des Excel-Tabellenblatts die folgende Formel ein: **=Hintergrundfarbe**

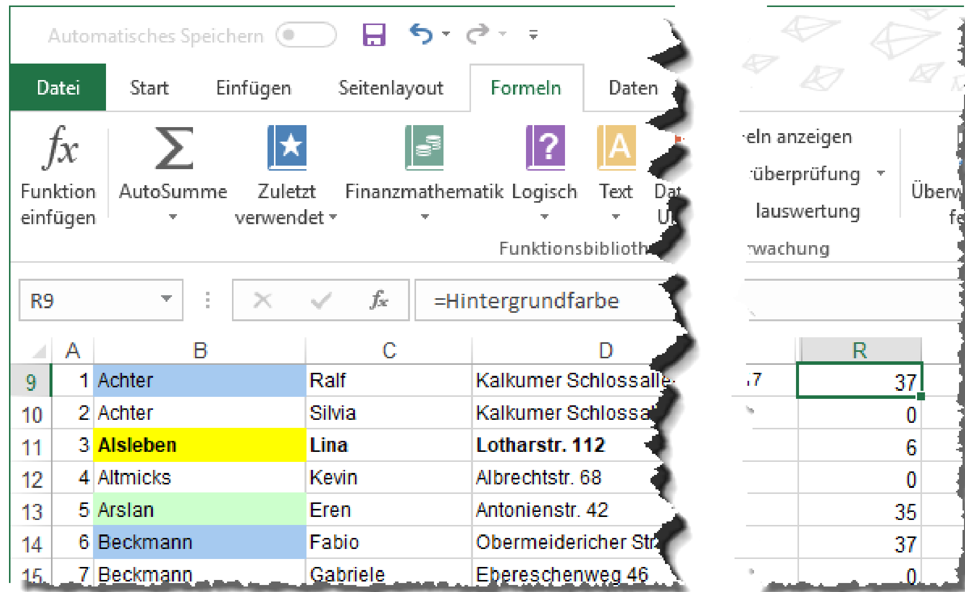


Bild 4: Hinzufügen des Namens mit der Referenz

Nun liefert die hintere Spalte die den Hintergrundfarben entsprechenden Farbwerte (s. Bild 4). Damit können wir beim Import von Access aus doch schon etwas anfangen!

Was aber haben wir hier gemacht – und wie funktioniert die angegebene Formel genau? Die **Zuordnen**-Funktion erwartet zwei Parameter: der erste erhält einen Zahlencode, der die Eigenschaft repräsentiert, die wir für die im zweiten Parameter angegebenen Bereich ermitteln wollen. In diesem Fall liefert der Zahlencode **63** einen Wert für die Eigenschaft **Hintergrundfarbe**. Mit dem zweiten Ausdruck geben wir die zu untersuchende Zelle an. In diesem Fall nutzen wir die indirekte Schreibweise: **INDIREKT("ZS(-15)")**. Hinter **Z** befindet sich keine Zahl, also wollen wir in der gleichen Zeile arbeiten, hinter **S** befindet sich der Wert **-15**, was bedeutet, dass wir uns auf die Zeile beziehen, die 15 Spalten weiter links liegt.

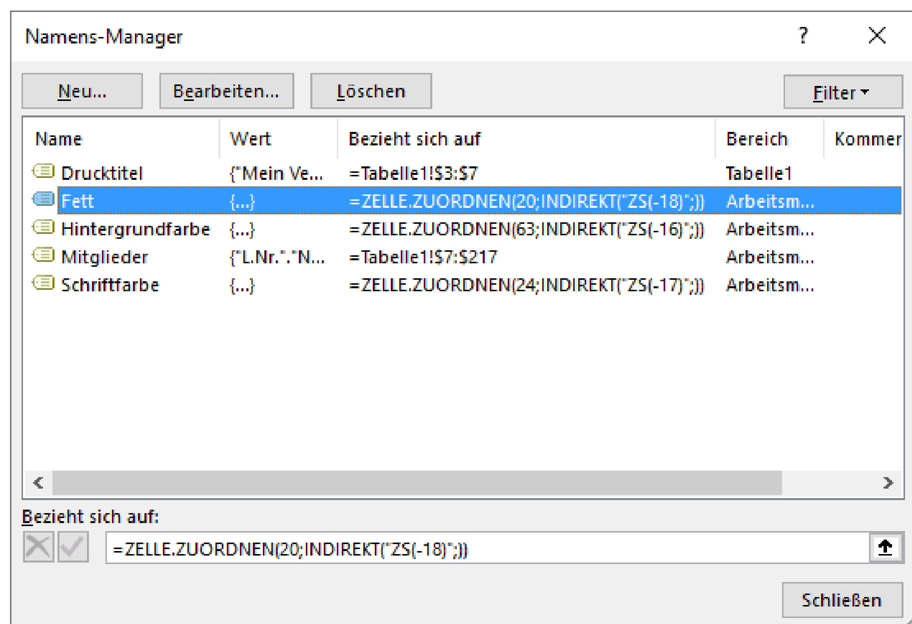


Bild 5: Anlegen zweier weiterer Benennungen

Dadurch dass wir die Formel **=Hintergrundfarbe** in die Zeile **R** schreiben, erhalten wir die Hintergrundfarbe für

die Zelle der gleichen Zeile in der Spalte **B**. Nun wollen wir auch noch zwei weitere Formatierungen in den folgenden beiden Spalten unterbringen, nämlich die Schriftart in der Spalte **B** und ob der Text in Spalte **B** fett formatiert ist.

Dazu legen wir zwei neue Einträge im Dialog **Namens-Manager** hinzu, sodass diese anschließend wie in Bild 5 aussieht. Die Formel für den Namen **Schriftfarbe** sieht so aus – hier verwenden wir also den Code **24**:

```
=ZELLE.ZUORDNEN(24;INDIREKT("ZS(-17)";))
```

Die Formel für den Namen **Fett** gestalten wir mit dem Code **20** wie folgt:

```
=ZELLE.ZUORDNEN(20;INDIREKT("ZS(-18)";))
```

Damit haben wir die drei möglichen Informationen, die durch die verschiedenen Formatierungen entstehen, und können diese auch beim Import in die Access-Datenbank berücksichtigen.

Den benannten Bereich für den Import müssen Sie übrigens nicht mehr um die drei neu angelegten Spalten erweitern, da wir dort die kompletten Zeilen als Bereich angegeben haben.

Import/Verknüpfung der Daten

Nun wollen wir die Daten aus der Excel-Tabelle in Access verfügbar machen. Dazu haben wir zwei Möglichkeiten: Entweder wir importieren die Daten oder wir erstellen eine Verknüpfung. In diesem Fall, wo nur eine einmalige Migration der Daten geplant ist, können wir die Daten direkt importieren. Das erledigen wir wie folgt:

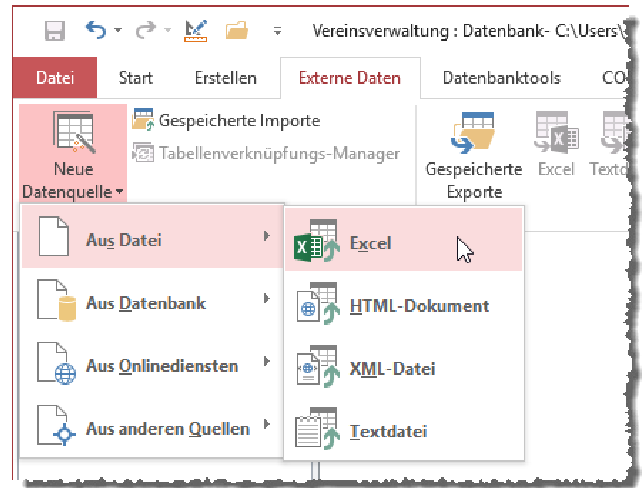


Bild 6: Starten des Import-Vorgangs

- Öffnen Sie eine neue Access-Datenbank.
- Wählen Sie den Ribbon-Befehl **Externe Daten|Importieren und Verknüpfen|Neue Datenquelle|Aus Datei|Excel** aus (s. Bild 6).

Bild 7: Auswahl von Quelldatei und Import

- Geben Sie im folgenden Dialog die Quelldatei an, behalten Sie die Einstellung **Importieren ...** bei und klicken Sie auf **OK** (s. Bild 7).

- Wechseln Sie im folgenden Dialog zur Option **Benannte Bereiche anzeigen**. Dies liefert unseren weiter oben angelegten Bereich **Mitglieder**, den wir nun aktivieren. Die Liste darunter zeigt nun den von uns ausgewählten Bereich an (s. Bild 8).

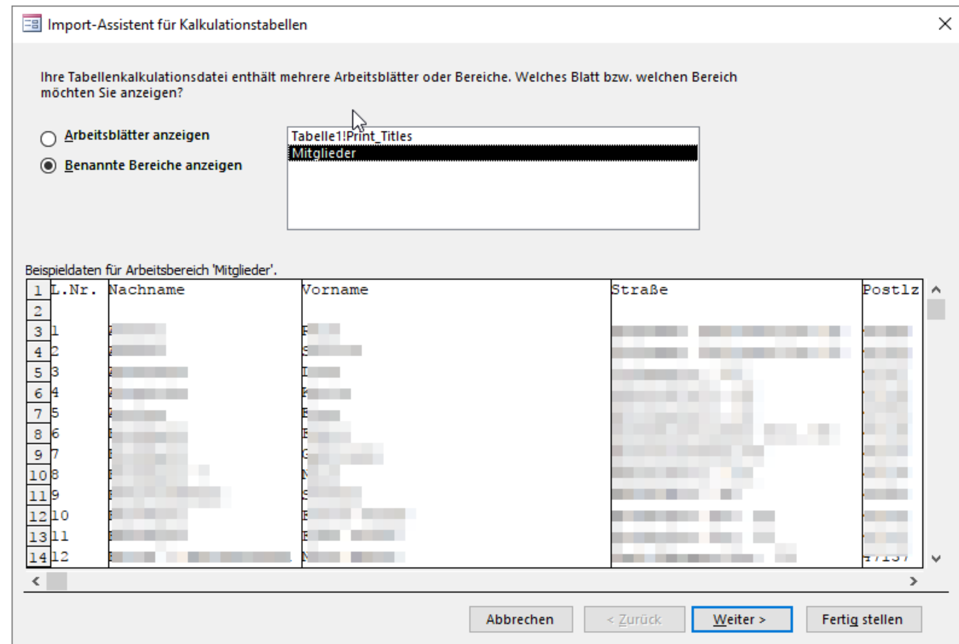


Bild 8: Festlegen des zu importierenden Bereichs

- Aktivieren Sie im folgenden Schritt die Option **Erste Zeile enthält Spaltenüberschriften**. Dies bringt zunächst die Meldung aus Bild 9 zutage. Dabei handelt es sich um die von uns hinzugefügten Spalten ganz rechts. Die fehlenden Spaltenüberschriften sind kein Problem, so lange wir die Daten nur einmal importieren müssen – Sie können sich also aussuchen, ob Sie diese in der Excel-Tabelle nachtragen und den Import erneut starten oder ob Sie die Feldnamen in Access anpassen.

- Einen Schritt weiter müssen Sie nun festlegen, welche Felder importiert werden sollen und können ein paar Eigenschaften für diese Felder festlegen. Dazu gehören der Feldname, der Datentyp und die Indizierung. Wir behalten alle Werte bei, da wir die Daten aus der Importtabelle ohnehin noch aus weitere Tabellen aufteilen müssen (s. Bild 10).

- Im vorletzten Schritt (s. Bild 11) geben wir noch an, dass wir keinen neuen Primärschlüssel anlegen wollen – dies aus dem gleichen Grund wie zuvor: Die Daten werden ohnehin noch in die Zieltabellen überführt.

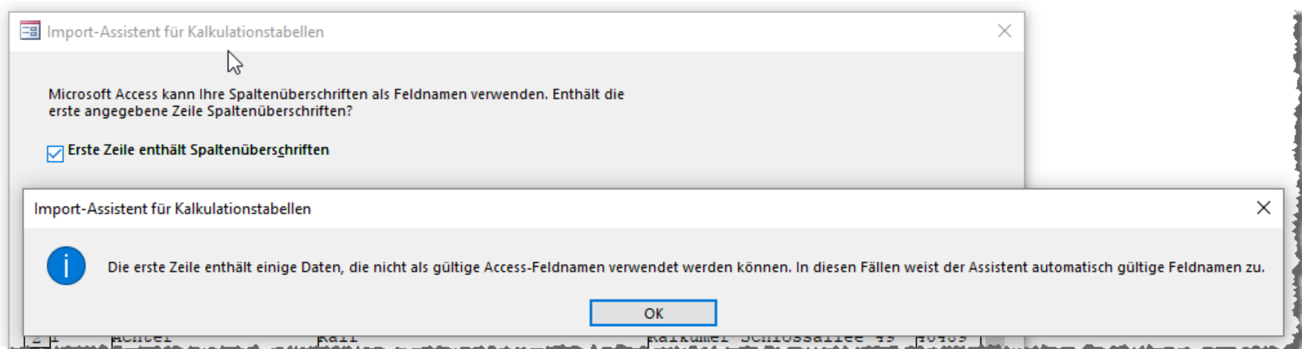


Bild 9: Es fehlen einige Spaltenüberschriften

- Im letzten Schritt können Sie noch den Namen der Zieltabelle festlegen, wobei wir den Namen **Mitglieder** beibehalten können.

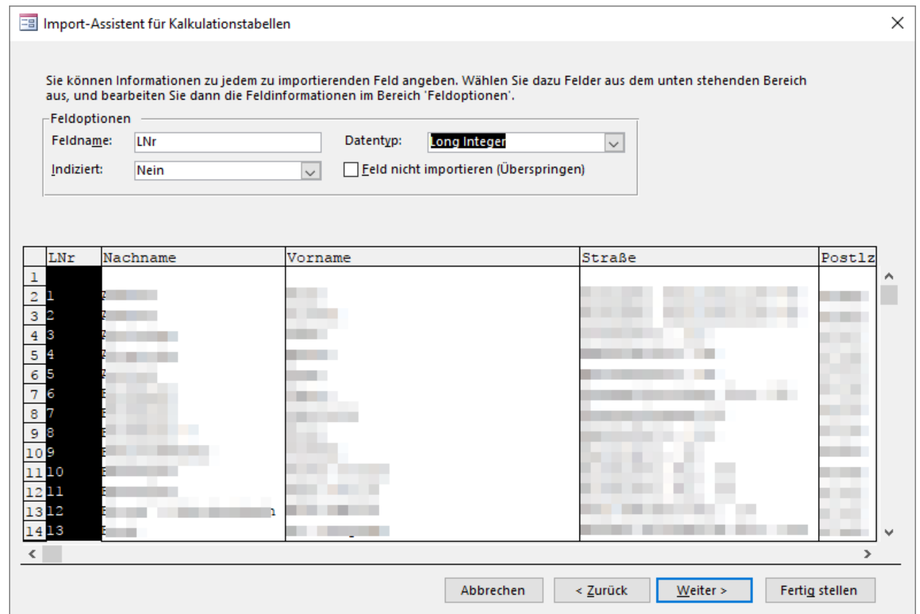


Bild 10: Einstellen der Feldeigenschaften

Im vorliegenden Beispiel traten beim Import ein paar Fehler auf, die der Import-Assistent in der Tabelle **Mitglieder_Importfehler** gespeichert hat. Die Fehler traten wohl in der Spalte mit der Postleitzahl auf, die entsprechenden Zeilennummern sind ebenfalls in der Tabelle angegeben (s. Bild 12). Ein Blick in die Zieltabelle **Mitglieder** zeigt, dass das Feld **Postlz** für die angegebenen Zeilen leer ist. Also werfen wir auch noch einen Blick auf die Daten in der Ausgangsdatei.

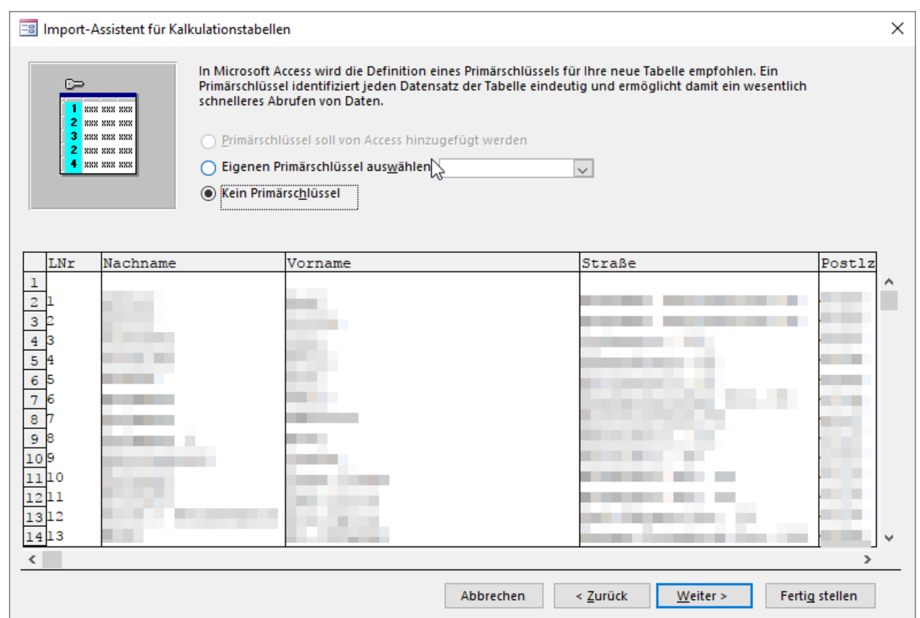


Bild 11: Festlegen eines Primärschlüssels

Hier zeigt sich dann ein gern gemacht er Fehler: Statt das Land in einer eigenen Spalte zu speichern, wird dieses als Kürzel vor die Postleitzahl geschrieben. In diesem Fall handelt es sich um die Schweiz und Einträge wie etwa **CH-8005**. Beim Import werden die Datentypen der Zielfelder auf Basis der oberen paar Werte ermittelt. Hier traten noch keine Postleitzahlen mit anderen Zeichen als mit Zahlen auf, daher wurde ein Zahlendatentyp für das Feld festgelegt. In ein solches Feld können wir natürlich keine Werte mit Buchstaben importieren. Auch hier haben Sie die Wahl: Ändern Sie die paar Einträge in der Zieltabelle von Hand oder wollen Sie dies in der Quelldatei erledigen und die Daten erneut importieren? Wir starten den Import erneut und stellen diesmal den Datentyp für

Mitglieder_Importfehler

Fehler	Feld	Zeile
Fehler bei Typumwandlung	Postlz#	112
Fehler bei Typumwandlung	Postlz#	113
Fehler bei Typumwandlung	Postlz#	155

Datensatz: 1 von 3 | Kein Filter | Suchen

Bild 12: Tabelle der Importfehler

die Spalte **PLZ** auf **Kurzer Text** ein – nun gelingt auch der Import der PLZ-Werte mit Länderkennzeichen.

Damit wäre der Import abgeschlossen und wir können uns dem Datenmodell und dem Übertragen der Daten aus der Tabelle **Mitglieder** in die noch zu erstellenden Tabellen der Anwendung konzentrieren. Wir können noch die meisten der in der Zieltabelle **Mitglieder** angelegten Felder mit den Bezeichnungen **F21**, **F22** und so weiter löschen sowie den drei von uns angelegten Feldern entsprechende Feldnamen zuweisen. Der Entwurf des Datenmodells sieht anschließend wie in Bild 13 aus.

Mitglieder

Feldname	Felddatentyp	Beschreibung (optional)
LNr	Zahl	Laufende Nummer
Nachname	Kurzer Text	Nachname
Vorname	Kurzer Text	Vorname
Straße	Kurzer Text	Straße
Postlz	Kurzer Text	PLZ
Ort	Kurzer Text	Ort
Geb- Datum	Kurzer Text	Geburtsdatum
Eintrittsdatum	Kurzer Text	Eintrittsdatum
Mitgl-Nr	Kurzer Text	Mitgliedsnummer
Ak/Pa	Kurzer Text	Altersklasse/Passiv
Beitrag / €	Zahl	Beitrag
FB	Kurzer Text	Familienbeitrag
B	Kurzer Text	***noch nachfragen
Bemerkung	Kurzer Text	Bemerkung
Alter	Zahl	Alter
Stichtag-Berechnung	Datum/Uhrzeit	Stichtag-Berechnung
Hintergrundfarbe	Zahl	Hintergrundfarbe
Schriftfarbe	Zahl	Schriftfarbe
Fett	Ja/Nein	Fett

Bild 13: Datenmodell der Tabelle nach dem Umbenennen und Entfernen unnötiger Felder

Erstellen der Zieltabellen

Nun erstellen wir die Zieltabellen für die Daten aus der temporären Tabelle **Mitglieder**. Als Erstes legen wir die Tabelle **tblMitglieder** an, welche die Basisdaten eines jeden Mitglieds enthält – also **Vorname**, **Nachname**, **Mitgliedsnummer** und so weiter. Des Weiteren werden wir einige Lookup-Tabellen brauchen – zum Beispiel für die Anrede. Wobei an dieser Stelle auffällt, dass die Excel-Tabelle gar keine Spalte für die Anrede enthält. Eine Spalte für das Geschlecht ist auch nicht vorhanden. Wie wurden denn dann die Serienbriefe mit der korrekten Anrede gefüllt? Vielleicht hat man diese einfach weggelassen. Wir werden auf jeden Fall eine Anrede unterbringen. Dazu benötigen wir die Lookup-Tabelle **tblAnreden**, deren Entwurf wie hier nicht extra vorstellen wollen – sie enthält lediglich die Felder **AnredeID**, **Anrede**, **Briefanrede** und **Adressanrede** (also für Werte wie **Herr**, **Sehr geehrter Herr** und **Herrn**).

Der Haken an der Tabelle mit den Anreden ist, dass das entsprechende Fremdschlüsselfeld in der Tabelle **tblMitglieder** manuell gefüllt werden muss, da die Excel-Tabelle ja noch gar keine Informationen bezüglich Anrede oder Geschlecht enthält. Das einzige Feld, dem man eine Information über das Geschlecht entnehmen kann, ist **Vorname**. Allerdings gibt es auch keinen Algorithmus, der zuverlässig das Geschlecht vom Vornamen ableitet. Wenn wir aber in der Excel-Tabelle etwas weiter nach rechts schauen, findet sich dort eine Spalte mit der Überschrift **AK/Pa**. Was **Pa** ist, ist aktuell nicht bekannt, aber **AK** heißt offensichtlich Altersklasse. Dort finden sich dann Werte wie **EPM**, **EPW**, **WJ** oder **EAM**. Die Gemeinsamkeit ist: Alle Einträge enthalten entweder ein **M** für **Männlich** oder ein **W** für **Weiblich**. Bingo!

tblGeschlecht

GeschlechtID	Geschlecht
1	männlich
2	weiblich
(Neu)	

Datensatz: 1 von 2 · Kein Filter · Suchen

Bild 14: Die Tabelle **tblGeschlechter**

Für das Geschlecht legen wir wie für die Anrede eine eigene Lookup-Tabelle namens **tblGe-**

Vereinsverwaltung: Migration

Im Beitrag »Vereinsverwaltung: Von Excel zum Datenmodell« haben wir gezeigt, wie Sie eine exemplarische Excel-Tabelle mit den Daten zur Mitgliedsverwaltung in ein Access-Datenmodell umwandeln. Dabei haben wir die Daten der Excel-Datei bereits in der Datenbank verfügbar gemacht. Neben der damit verbundenen konzeptionellen Arbeit kommt nun der interessante Teil: die Programmierung von VBA-Code und Abfragen, um die Daten aus der monolithischen Excel-Tabelle in das Datenmodell zu übertragen.

Dabei haben wir einige Brocken bereits aus dem Weg geräumt: Die Excel-Tabelle enthielt nämlich beispielsweise nicht nur reine Daten in Form von Zahlen und Buchstaben, sondern auch verschiedene Markierungen in Form von verschiedenen Hintergrundfarben, fett gedrucktem Text oder farbigem Text. Diese haben wir bereits in der Excel-Tabelle in entsprechende Werte umgewandelt, die wie allerdings auf den folgenden Seiten noch interpretieren müssen.

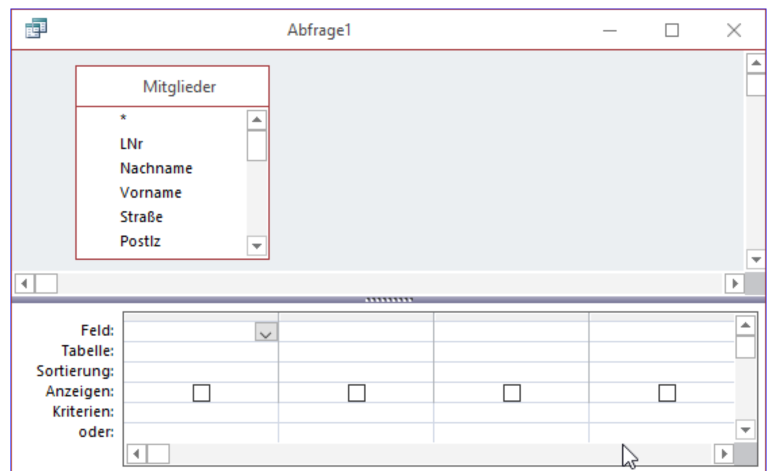


Bild 1: Anlegen einer Abfrage zum Übernehmen der Stammdaten

Schritt 1: Mitglieder übertragen

Die Basis der beiden Tabellen, also der Ausgangstabelle im Excel-Format als auch der Zieltabellen ist eine Tabelle zum Speichern der Personen, hier von Mitgliedern eines Sportvereins. Damit können wir auch beginnen. Die Tabelle Mitglieder enthält alle Daten, welche die Excel-Tabelle uns bereitgestellt hat. Daraus suchen wir uns nun die Felder heraus, die in die von uns entworfenen Tabelle **tblMitglieder** passen.

Der Automatismus zum Migrieren der Daten besteht aus ein paar Abfrage, welche uns die Daten in der gewünschten Form extrahieren, sowie einer VBA-Prozedur im Modul **mdlMigration**. Diese heißt **Migrieren** und steuert die Migration der Daten aus der Excel-Tabelle in die Tabellen unseres Datenmodells.

Für die Übernahme der Daten aus der Tabelle **Mitglieder** in die Tabelle **tblMitglieder** erstellen wir eine Abfrage auf Basis der Tabelle **Mitglieder**. Dieser fügen wir also nach dem Erstellen zunächst die Tabelle **Mitglieder** aus dem Dialog **Tabellen anzeigen** hinzu – mit dem Ergebnis aus Bild 1.

Wir wollen aus dieser Abfrage eine Anfügeabfrage machen, welche Daten aus der Tabelle **Mitglieder** an die Tabelle **tblMitglieder** anfügt. Dazu wählen Sie aus dem Ribbon den Befehl **Entwurf|Abfragetyp|Anfügen** aus.

Dies öffnet den Dialog **Anfügen**, mit dem Sie die Tabelle **tblMitglieder** als Zieltabelle festlegen (s. Bild 2).

Danach können Sie die Felder der Quelltabelle aus der Liste im oberen Bereich in das Entwurfsraster ziehen oder per Doppelklick dorthin befördern. Sofern Access automatisch ein gleichnamiges Feld in der Zieltabelle erkennt, fügt es direkt einen entsprechenden Eintrag in

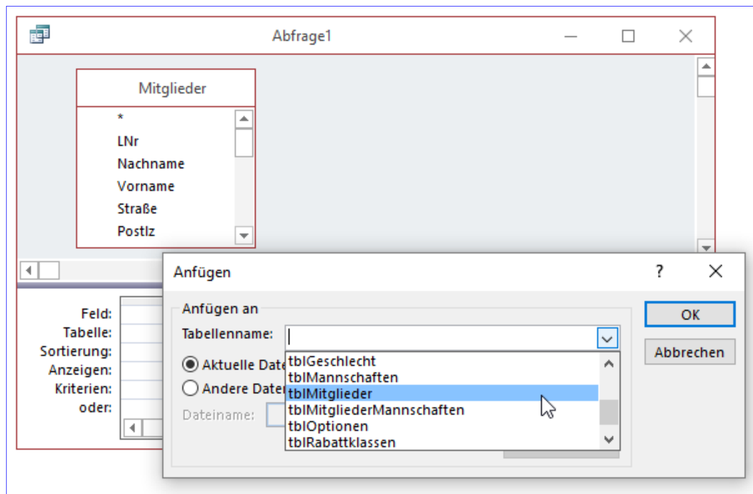


Bild 2: Angabe der Zieltabelle

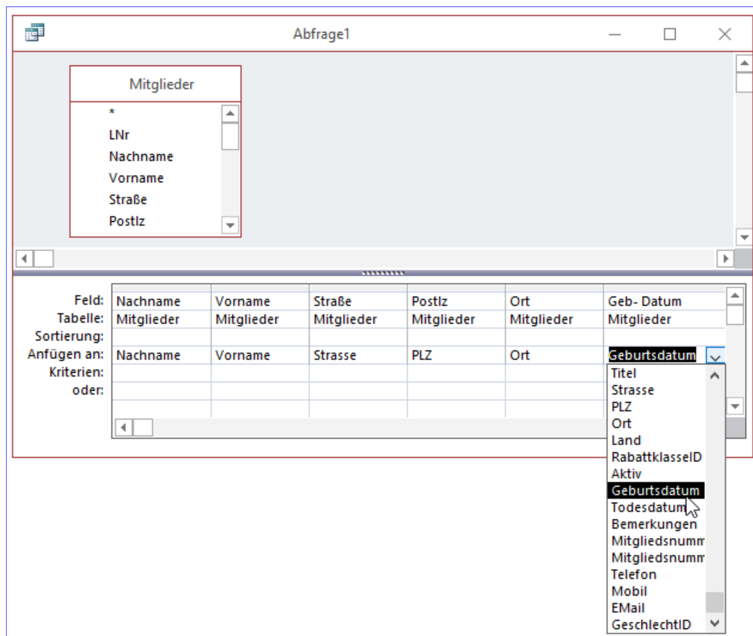


Bild 3: Auswahl der Zielfelder

die Zeile **Anfügename** ein. Sollte dies nicht automatisch funktionieren, was etwa beim Feld **Geb.-Datum** der Fall ist, wählen Sie das Zielfeld selbst aus – in diesem Fall **Geburtsdatum** (s. Bild 3).

Nun gibt es beim Feld **PLZ** das erste Hindernis: Wir haben ja schon beim Import der Tabelle gesehen, dass Access das zunächst vorgesehene Zahlenfeld nicht komplett füllen konnte, da einige Datensätze einen zusätzlichen Buchstaben enthielten – nämlich den führenden Ländercode etwa in **CH-8005**. Genau genommen liefert das Feld **PLZ** für die Schweizer Mitglieder das Länderkürzel, die Postleitzahl und auch noch den Ort – warum auch immer der Ersteller das so gemacht hat: Die Daten gehören natürlich auf drei Felder aufgeteilt. Also ändern wir den Ausdruck für das Feld **PLZ** ein wenig um:

```
PLZ: Wert(Ersetzen([Postlz];"CH-";""))
```

Dieser Ausdruck ersetzt zunächst im Feld **Postlz** die Zeichenfolge **CH-** durch eine leere Zeichenfolge. Aus **CH-8005 Zürich** wird dann schon einmal **8005 Zürich**. Wie erhalten wir davon nur die führenden Ziffern? Ganz einfach – mit der **Wert**-Funktion (unter VBA **Val**).

Der Ort könnte so einfach sein, wenn dieser nicht für die oben genannten Fälle komplett in der Zeile **Postlz** stehen würde und für alle anderen in der Spalte **Ort**. Doch mit ein wenig Geschick erhalten wir die gewünschten Werte. Dazu legen wir für das Feld **Ort** einen neuen Ausdruck namens **OrtX** an (damit es keinen Zirkelbezug gibt) und füllen diesen mit der folgenden Formel:

```
OrtX: Wenn(IstNull([Ort]);Teil([Postlz];InStrRev
([Postlz];" ")+1);[Ort])
```

Hier finden wir eine **Wenn**-Bedingung, die im ersten Abschnitt prüft, ob Ort den Wert **Null** enthält. In diesem Fall ermitteln wir aus dem Feld **Postlz** den Teil hinter dem letzten gefundenen Leerzeichen – bei **CH-8005 Zürich** liefert dies also **Zürich**. Ist das Feld **Ort** nicht leer, liefern wir einfach den Wert dieses Feldes zurück.

Komplizierter wird es mit dem Land, denn da wir ja nun die Länderkennzeichen aus der Postleitzahl entfernen wol-

len, soll das Land in das Feld **Land** eingetragen werden. Das ist auch interessant, weil wir in der Ursprungstabelle überhaupt kein Feld haben, welches das Land angibt. Wir gehen aber davon aus, dass alle Datensätze, die nicht das Länderkürzel **CH** in der Postleitzahl tragen, Deutschland als Land aufweisen. Also verwenden wir den folgenden Ausdruck:

```
Land: Wenn(Wenn(InStr([postlz];"-")>0;Links([postlz];InStr
([postlz];"-")-1);"DE")='CH';'Schweiz';'Deutschland')
```

Hier haben wir es mit ein paar Verschachtelungen zu tun. Die innere **Wenn**-Bedingung liefert das Länderkürzel, wenn **Postlz** ein Minuszeichen enthält, wenn kein Minuszeichen vorliegt, folgt **DE** als Länderkennzeichen.

Die äußere **Wenn**-Bedingung liefert das Land **Schweiz**, wenn das von der inneren Bedingung gelieferte Kürzel **CH** lautet, sonst **Deutschland**. Das ist zugegebenermaßen improvisiert und wird kompliziert, wenn es mehr solcher Sonderfälle gibt.

An einer solchen Stelle muss man dann entscheiden, ob man doch manuell die fehlerhaften Datensätze anpasst oder gegebenenfalls eine VBA-Routine baut, die etwas übersichtlicher daherkommt.

Diese könnte etwa wie folgt aussehen:

```
Public Function LandErmitteln(strPLZ As String)
    Dim strLaenderkuerzel As String
    Dim intPosStrich As Integer
    Dim strLand As String
    intPosStrich = InStr(strPLZ, "-")
    If intPosStrich = 0 Then
        strLand = "Deutschland"
    Else
        strLaenderkuerzel = Left(strPLZ, intPosStrich - 1)
        Select Case strLaenderkuerzel
            Case "CH"
                strLand = "Schweiz"
            Case "NL"
                strLand = "Niederlande"
        End Select
    End If
    LandErmitteln = strLand
End Function
```

In der Abfrage greifen Sie wie folgt auf diese Funktion zu:

```
Land: LandErmitteln([Postlz])
```

Die Funktion erhält den Wert für das Feld **Postlz** als Parameterwert. Sie ermittelt dann die Position des ersten Minuszeichens und speichert diese in der Variablen **intPosStrich**.

Ist kein Minuszeichen vorhanden, erhält die Variable den Wert **0**. In diesem Fall liefert die Funktion **Deutschland** zurück. Anderenfalls liest sie das Länderkürzel aus und weist in einer **Select Case**-Bedingung die Bezeichnung des entsprechenden Landes zu.

PLZ und Ort per VBA

Für den Fall, dass mehrere Länder hinzukommen, stellen wir auch die Ermittlung von **PLZ** und **Ort** noch auf VBA um. Für die PLZ sieht die Funktion wie folgt aus:

```
Public Function PLZErmitteln(strPLZ As String)
    Dim strLaenderkuerzel As String
    Dim intPosStrich As Integer
    intPosStrich = InStr(strPLZ, "-")
    If Not intPosStrich = 0 Then
        strPLZ = Mid(strPLZ, intPosStrich + 1)
        strPLZ = Val(strPLZ)
    End If
    PLZErmitteln = strPLZ
End Function
```

Die Funktion nimmt den Werte des Feldes **Postlz** entgegen und ermittelt die Position des Minuszeichens. Wenn der so ermittelte Wert größer als 0 ist, ist ein Minuszeichen

vorhanden. In diesem Fall entfernt die Funktion zuerst den Teil vor dem Minuszeichen inklusive Minuszeichen.

Im zweiten Schritt ermittelt sie mit der **Val**-Funktion alle Ziffern bis zum ersten Zeichen, das keine Ziffer ist, und gibt das Ergebnis zurück. Sollte der Parameter **strPLZ** eine einfache PLZ liefern, wird diese als Funktionsergebnis zurückgegeben.

Der Ausdruck im Feld der Abfrage lautet:

```
PLZ: PLZErmitteln([Postlz])
```

Den Ort ermitteln wir, indem wir die PLZ und den Ort untersuchen und diese Felder auch als Parameter übergeben. Das ist problemlos möglich, wie der folgende Ausdruck für das Feld **OrtX** zeigt (**X** angefügt, um keinen Zirkelbezug zu erzeugen):

```
OrtX: OrtErmitteln([Postlz];[Ort])
```

Die Funktion nimmt diese beiden Parameter wie folgt entgegen:

```
Public Function OrtErmitteln(strPLZ As String, _
                                          strOrt As String)
    Dim strLaenderkuerzel As String
    Dim intPosStrich As Integer
    Dim intLeerzeichen As Integer
    intPosStrich = InStr(strPLZ, "-")
    If Not intPosStrich = 0 Then
        intLeerzeichen = InStrRev(strPLZ, " ")
        If Not intLeerzeichen = 0 Then
            strOrt = Mid(strPLZ, InStrRev(strPLZ, " "))
        End If
    End If
    OrtErmitteln = strOrt
End Function
```

Dabei prüft sie wieder, an welcher Position sich gegebenenfalls das Minuszeichen befindet. Falls eines vorhanden ist, ermittelt die Funktion als Nächstes, wo sich die hinterste Leerstelle befindet.

Ist eine vorhanden, wie in **CH-8005 Zürich**, erhält die Variable **intLeerzeichen** einen Wert ungleich **0** und der Teil hinter dem letzten Leerzeichen wird in der Variablen **strOrt** gespeichert. Dieser Wert wird dann als Rückgabewert zurückgeliefert. Wenn kein Minuszeichen gefunden wird, liefert die Funktion den Inhalt des Parameters **strOrt** zurück.

Aktives oder passives Mitglied?

Ob es sich um ein aktives oder passives Mitglied handelt, finden wir über das Feld **Ak/Pa** der Originaltabelle heraus. In den Originaldaten finden wir dabei redundante Daten, denn diese enthält sowohl die Altersklasse als auch das Alter des Mitglieds.

Die Altersklasse soll aber aus dem Geburtsdatum und dem Stichtag für das aktuelle Jahr ermittelt werden. Also entnehmen wir diesem Feld nur die Angabe, ob es sich um ein aktives oder passives Mitglied handelt. Dafür legen wir direkt eine passende VBA-Routine an:

```
Public Function AktivPassiv(strAktivPassiv As String) _
                                             As Boolean
    If InStr(1, strAktivPassiv, "P") = 0 Then
        AktivPassiv = True
    End If
End Function
```

Der Ausdruck für das Feld lautet:

```
Aktiv: AktivPassiv([Ak/Pa])
```

Geschlecht

Aus dem Feld **Ak/Pa** können wir auch das Geschlecht des Mitglieds ermitteln. Wie im Beitrag **Vereinsverwaltung: Von Excel zum Datenmodell** (**www.access-im-unternehmen.de/1106**) erwähnt, weisen jeweils die Buchstaben **M** und **W** auf das Geschlecht hin – etwa wie in **EPM**

Unterdatenblätter in Formularen

Unterdatenblätter in Tabellen oder Abfragen kennen Sie sicherlich bereits. Das sind die Bereiche einer Datenblattansicht, die aufklappen, wenn Sie das Plus-Zeichen vor einem Datensatz anklicken. Wenn Sie eine Tabelle, für die eine solche Unterdatenblatt-Funktion eingerichtet ist, jedoch in der Datenblattansicht in einem Formular oder Unterformular anzeigen wollen, verschwinden die praktischen Plus-Zeichen und Sie schauen in die Röhre. Doch das ist nicht das Ende der Fahnenstange: Access sieht durchaus den Einsatz von Unterdatenblättern in Formularen vor – und damit lässt sich eine Menge anstellen!

Als Beispiel verwenden wir die beiden Tabellen **tblTabellen** und **tblFelder** der Lösung aus dem Beitrag **Daten anonymisieren** (**www.access-im-unternehmen.de/1112**). Dabei handelt es sich um zwei Tabellen, von denen die erste alle Tabellennamen einer Datenbank speichern soll und die zweite alle in den Tabellen enthaltenen Felder. Dabei enthält die zweite Tabelle **tblFelder** ein Fremdschlüsselfeld, mit dem ein Datensatz der Tabelle **tblTabellen** ausgewählt werden kann, um die Zugehörigkeit des Feldes zu einer Tabelle zu definieren.

Bild 1: Tabelle mit Unterdatenblatt

Wenn Sie die dortige Tabelle **tblTabellen** mit einigen Beispieldaten wie hier aus der Südsturm-Datenbank öffnen, erscheinen direkt die Plus-Zeichen, mit denen sich die untergeordneten Datensätze der verknüpften Tabelle **tblFelder** einblenden lassen (s. Bild 1).

Warum benötigen wir diese Ansicht? In diesem Fall wollen wir es uns ersparen, ein TreeView-Steuerelement für die Anzeige der hierarchisch verknüpften Daten der beiden Tabellen **tblTabellen** und **tblFelder** hinzuzuziehen. Stattdessen wollen wir einmal das oft verschmähte Feature Unterdatenblatt nutzen. In der Lösung, die diese Ansicht nutzen soll, wollen wir die Möglichkeit haben, sowohl Einträge der übergeordneten Tabelle als auch die der untergeordneten Tabelle auszuwählen und abzuwählen. Wie die Abbildung zeigt, könnte man damit schon recht komfortabel die Daten mit den dazu eingerichteten **Ja/Nein**-Feldern zugreifen. Allerdings wollen wir dort noch einen Schritt weiter gehen: Wenn der Benutzer das **Ja/Nein**-Feld einer Tabelle selektiert, sollen automatisch alle Einträge der untergeordneten Tabelle mit den Feldern ausgewählt werden. Andersherum soll das **Ja/Nein**-Feld für eine Tabelle abgewählt werden, wenn der Benutzer nur eines der untergeordneten Felder abwählt. Und wenn der Benutzer das **Ja/Nein**-Feld für eine Tabelle abwählt, sollen auch alle zu der Tabelle gehörenden Felder abgewählt werden.

Unterdatenblatt im Formular

Wie aber bekommen wir diese Ansicht in das Unterformular eines Formulars? Der erste und einfachere Ansatz ist der folgende: Sie fügen dem Formular ein Unterformular-Steuerelement hinzu. Dann stellen Sie für die Eigenschaft **Herkunftsobjekt** des Unterformular-Steuerelements den Wert **Tabelle.tblTabellen** ein (s. Bild 2).

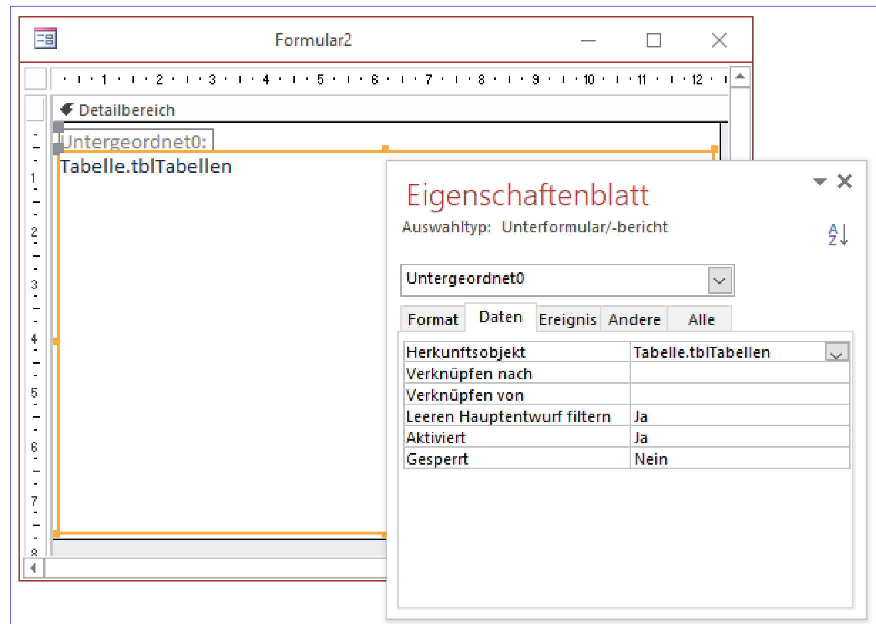


Bild 2: Einrichten eines Unterformulars mit Tabelle plus Unterdatenblatt

Wechseln Sie nun in die Formularansicht des Formulars, erhalten Sie die Ansicht aus Bild 3. Das ist ja schon fast genau das, was wir wollen! Aber Moment: Vielleicht bekommen wir noch irgendwie das Feld **TabelleID** der übergeordneten Tabelle und die Felder **FeldID** und **Felddatentyp** der untergeordneten Tabelle weg?

Auch das gelingt uns, indem wir einfach mit der rechten Maustaste auf die Spaltenköpfe der zu entfernenden Felder klicken und diese mit dem Befehl **Felder ausblenden** des Kontextmenüs verschwinden lassen. Das Ergebnis sehen sie in Bild 4. Nun müssen wir nur noch ein paar Ereignisprozeduren für die Ja/Nein-Felder namens Anonymisieren der Tabellen **tblTabellen** und **tblFelder** hinterlegen, mit denen wir die gewünschten und oben erwähnten Funktionen hinterlegen können. Und hier verlässt uns diese einfache Variante dann doch: Da wir als Herkunftsobjekt des Unterformulars eine Tabelle angegeben haben, können wir dafür keine Ereignisse hinterlegen. Dies funktioniert nur, wenn Sie ein Formular

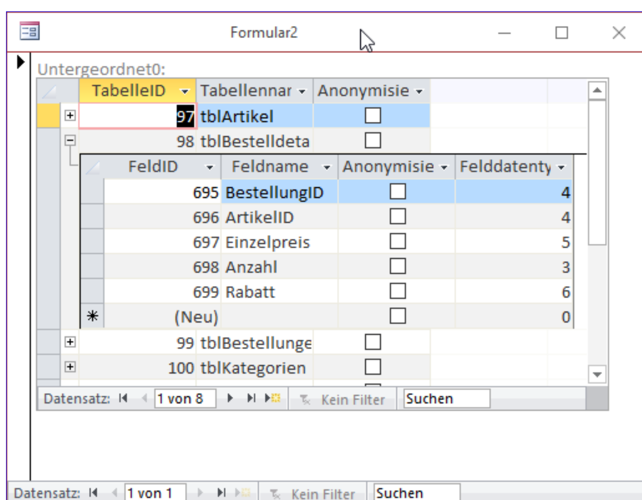


Bild 3: Unterformular mit Tabelle plus Unterdatenblatt

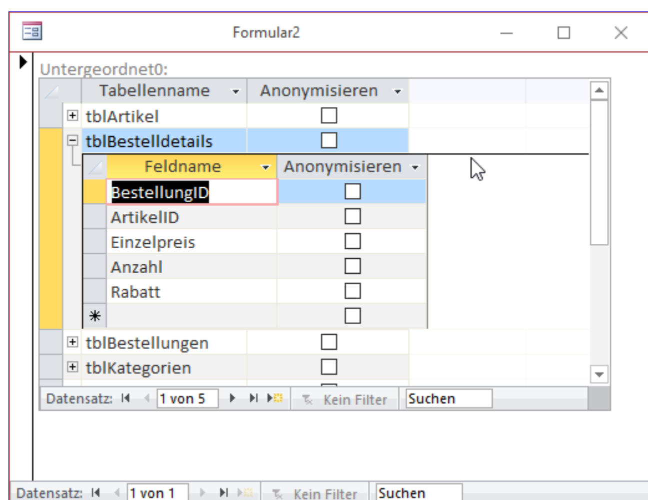


Bild 4: Unterdatenblatt mit ausgeblendeten Feldern

mit den entsprechenden Steuerelementen anlegen. Ist unser Plan, die Daten über die Unterdatenblätter anzuzeigen, gescheitert? Nein, wir haben noch eine Alternative!

Unterdatenblatt per Unterformular

Wir können die Daten nämlich auch als echtes Unterformular abbilden. Dazu erstellen Sie als Erstes ein Unterformular namens **sfmTabellen**. Diesem weisen Sie als Datenherkunft die Tabelle **tblTabellen** zu. Ziehen Sie dann alle benötigten Felder in den Formularentwurf. Wir wollen nur die beiden Felder **Anonymisieren** (das **Ja/Nein**-Feld) und das Feld **Tabellenname** verwenden. Wir ziehen zuerst das **Ja/Nein**-Feld in den Entwurf, da dieses in der ersten Spalte angezeigt werden soll (s. Bild 5).

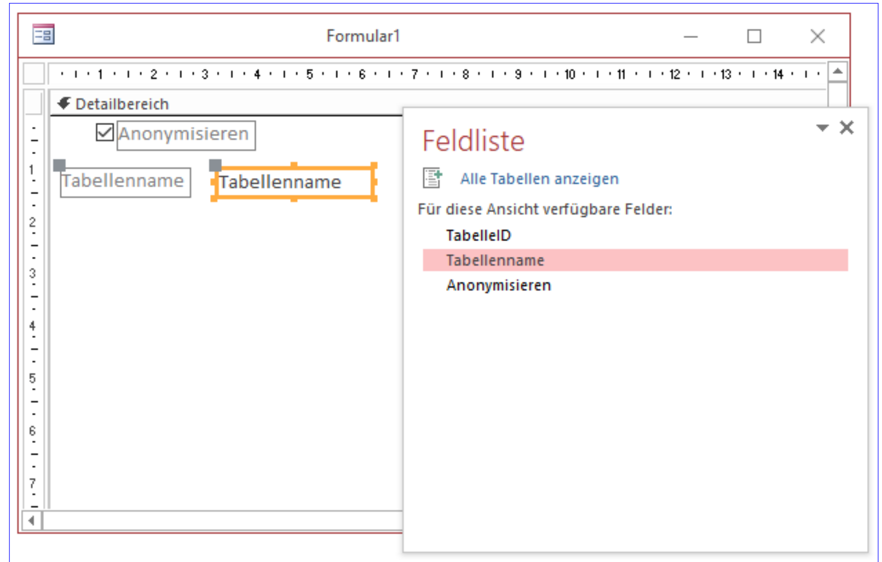


Bild 5: Erstellen des Unterformulars mit den Daten der Tabelle **tblTabellen**

Außerdem stellen wir die Eigenschaft **Standardansicht** auf den Wert **Datenblatt** ein. Ein Wechsel in die Entwurfsansicht liefert allerdings nicht das erwartete Ergebnis – es erscheinen zwar die Daten der Tabelle **tblTabellen**, aber keine Plus-Zeichen zum Aufklappen der Unterdatenblätter (s. Bild 6).

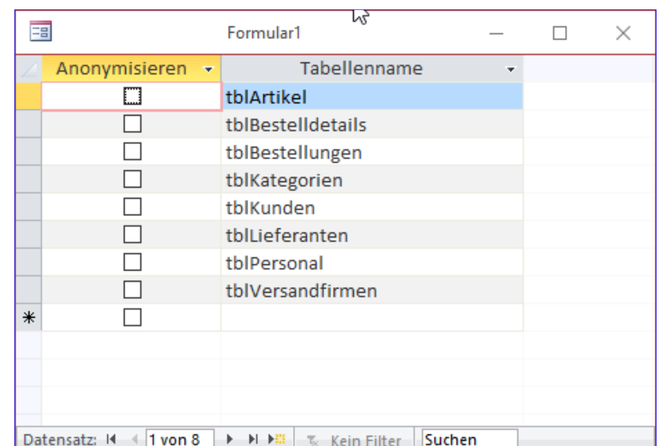


Bild 6: Kein Unterdatenblatt in Sicht!

Kein Wunder: Wir haben bisher auch den entscheidenden Schritt ausgelassen. Wir müssen nämlich auch noch für die Tabelle **tblFelder** ein Unterformular anlegen und dieses dann als Unterformular zur Datenblattansicht des Unterformulars mit den Tabellen hinzufügen! Speichern wir also zunächst das soeben erstellte Formular unter dem Namen **sfmTabellen**.

Danach erstellen Sie ein weiteres Unterformular, welches diesmal die Tabelle **tblFelder** als Datenherkunft verwendet.

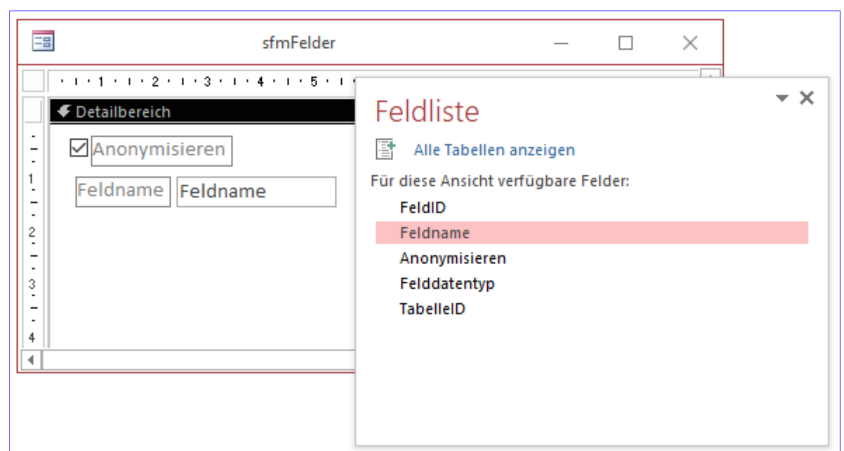


Bild 7: Unterformular für das Unterdatenblatt

Undo in mehreren Unterformularen

In den Beiträgen »Undo in Haupt- und Unterformular« und »Undo in Haupt- und Unterformular mit Klasse« haben wir gezeigt, wie Sie die Undo-Funktion etwa durch einen Abbrechen-Schaltfläche nicht nur auf das Hauptformular, sondern auch auf die Änderungen im Unterformular erstrecken. Nun hat ein Leser gefragt, ob man dies auch für mehrere Unterformulare erledigen kann. Klar kann man – die angepasste Lösung stellt der vorliegende Beitrag vor.

Dabei setzen wir auf der Lösung aus dem Beitrag **Undo in Haupt- und Unterformular mit Klasse** (**www.access-im-unternehmen.de/912**) auf. Diese enthält im Vergleich zu dem vorherigen Beitrag eine Klasse mit der vollständigen Funktionalität für das Implementieren des Undo in Haupt- und Unterformular, die man nur noch im Ereignis **Beim Laden** des Hauptformulars initialisieren und mit einigen Eigenschaften versehen muss. Der dazu anzulegende Code war sehr überschaubar gemessen an der Funktion, die er hinzufügte.

Die Einschränkung dieser Klasse ist, dass man in dieser lediglich das Hauptformular und das Unterformular angeben konnte, auf die sich die Undo-Funktion auswirken sollte. Wenn man ein zweites Unterformular hinzufügen wollte, war das nicht möglich. Also schauen wir uns im vorliegenden Beitrag an, wie wir dieses Feature nachrüsten.

Aber wie viele Formular dürfen es denn sein? Reichen zwei aus – oder benötigt man drei oder vier Unterformulare? Und macht man sich denn mit so vielen Unterformularen nicht die Übersicht im Formular kaputt? Der Leser, der mit seiner Anfrage an uns herangetreten ist, lieferte gleich einen Screenshot seines Formulars mit, auf dem ersichtlich wurde, dass er ein Register-Steuerelement verwendet, um die verschiedenen Unterformulare im Formular unterzubringen. So gelingt das natürlich bei Erhaltung guter Übersicht.

Damit war klar: Wir sollten uns nicht unbedingt einer Einschränkung hingeben und eine Lösung programmieren, die beliebig viele Unterformulare berücksichtigt.

Erster Ansatz: Zwei Unterformulare

Dennoch haben wir, um uns ein wenig in den VBA-Code der anzupassenden Lösung einzuarbeiten, zunächst die bestehende Lösung auf den Einsatz mit zwei Unterformularen erweitert. Dazu haben wir lediglich im Code überall dort, wo bisher das Unterformular erwähnt wurde, den Code für ein zweites Unterformular untergebracht. Das klappte recht leicht – die Hauptarbeit war Copy and Paste und ein wenig Achtsamkeit an den richtigen Stellen.

Zweiter Ansatz: So viele Unterformulare wie nötig

Klar: Die Anzahl der Unterformular je Formulare ist begrenzt. Dennoch wollen Sie den Code der Klasse ja nicht jedes Mal erweitern, wenn ein neues Unterformular zum Formular hinzukommt. Wir wollen dann maximal ein paar Zeilen Code im **Beim Laden**-Ereignis hinzufügen, die das neu hinzugekommene Formular und dessen Datenherkunft sowie das Fremdschlüsselfeld der zugrunde liegenden Tabelle an die Klasse übergeben.

Dafür ist dann zum Umwandeln des Codes ein wenig mehr Aufwand erforderlich. Aber es lohnt sich!

Aus eins mach zwei

Wenn Sie bereits andere Lösungen von uns gesehen haben, die Funktionen in Klassen auslagern und bei denen sich die Funktion auf ein oder mehrere Steuerelemente bezieht, dann wissen Sie bereits: Es wird eine Collection geben, welche die Informationen für die in beliebiger Anzahl auftretenden Unterformulare aufnehmen soll. Dazu legen wir für jedes Unterformular eine neue Klasse an, die

dann in der Collection gespeichert wird. Das heißt weiter: Wir benötigen nicht mehr nur eine Hauptklasse, sondern noch eine weitere Klasse für die untergeordneten Steuerelemente.

Bild 1 zeigt schon einmal, wie das Endergebnis aussehen soll. Zu diesem Zweck haben wir drei Tabellen namens **tblHaupt**, **tblUnter1** und **tblUnter2** erstellt, von denen die letzten beiden per Fremdschlüssel die erste Tabelle referenzieren.

Das Hauptformular ist an die Tabelle **tblHaupt** gebunden, die beiden Unterformulare an die Tabellen **tblUnter1** und **tblUnter2**.

Bild 1: Formular mit zwei Unterformularen

Undo-Funktion implementieren

Damit die Daten in diesen drei Formularen auch nach dem Ändern von Daten im Hauptformular und in den beiden Unterformularen noch wieder hergestellt werden können, ist nur wenig Code notwendig, wenn die beiden Klassen **clsUndoMultiMain** und **clsUndoMultiSub** einmal zur Datenbank hinzugefügt wurden.

Diese sieht dann nämlich wie in Listing 1 aus. Hier deklarieren wir im allgemeinen Teil der Code behind-Klasse des Formulars die Variable **objUndoMultiMain** vom Typ **clsUndoMultiMain**. Diese erstellen wir dann in der **Beim Laden**-Ereignisprozedur, wo wir zuerst die Datenherkunft des Hauptformulars für **RecordsourceForm** angeben, dann das Primärschlüsselfeld der Datenherkunft des Hauptformulars und schließlich einen Verweis auf das Hauptformular selbst.

```
Dim objUndomultiMain As clsUndoMultiMain

Private Sub Form_Load()
    Set objUndomultiMain = New clsUndoMultiMain
    With objUndomultiMain
        .RecordsourceForm = "tblHaupt"
        .PKForm = "HauptID"
        Set .Form = Me
        .AddSubform Me!sfmUnter1.Form, "tblUnter1", "HauptID"
        .AddSubform Me!sfmUnter2.Form, "tblUnter2", "HauptID"
        Set .OKButton = Me!cmdOK
        Set .CancelButton = Me!cmdAbbrechen
    End With
End Sub
```

Listing 1: Code zum Einbinden der Funktion zum Undo in Haupt- und Unterformular

Dann folgen die neuen Elemente: Die Methode **AddSubform** kann nämlich so oft aufgerufen werden, wie es nötig ist, und damit ein oder mehrere Unterformulare in die **Undo**-Funktion einschließen.

Dazu übergeben Sie dieser Methode die folgenden drei Parameter:

- **frm**: Verweis auf das Unterformular (nicht auf das Unterformular-Steuerelement, sondern auf das darin enthaltene Element – zu referenzieren mit der **Form**-Eigenschaft)

- **strRecordsource**: Bezeichnung der Datenherkunft (Tabelle oder Abfrage)

- **strFKSubform**: Fremdschlüsselfeld der Datenherkunft des Unterformulars, über welches die Beziehung zum Datensatz im Hauptformular hergestellt wird

Grundlegende Technik

Die grundlegende Idee ist es, die gesamte Bearbeitung des aktuellen Datensatzes des Hauptformulars und der mit diesem Datensatz verknüpften Datensätze im Unterformular in eine Transaktion einzuarbeiten. Dazu gibt es einen wichtigen Unterschied zur herkömmlichen Arbeit mit an Formulare gebundene Datenherkünften: Diese dürfen nämlich nicht wie sonst einfach etwa über die Eigenschaft **Recordsource** beziehungsweise **Datenherkunft** an die Formulare und Unterformulare gebunden werden. Stattdessen erstellen wir diese als zunächst unabhängiges Recordset und weisen dieses dann der **Recordset**-Eigenschaft des Hauptformulars und der Unterformulare zu.

Die Recordsets erstellen wir im Kontext eines **Database**-Objekts, das direkt dem **Workspace**-Objekt der aktuellen Sitzung untergeordnet ist. Auf diese Weise können wir in den Recordsets Änderungen vornehmen und diese dann über die **Transaction**-Methoden **BeginTrans**, **CommitTrans** und **Rollback** des **Workspace**-Objekts nach den Wünschen des Benutzers entweder zusammen speichern oder verwerfen.

Zusammenhang zwischen den beiden Klassen

Die Hauptklasse **clsUndoMultiMain** nimmt die wichtigsten Elemente wie das **Database**- und das **Workspace**-Objekt auf und enthält die Ereignisprozeduren, die für das Hauptformular angelegt werden. Die Klasse **clsUndoMultiSub**, die für jedes Unterformular einmal als Objekt angelegt wird, erhält den Verweis auf das jeweilige Unterformular und implementiert die Ereignisprozeduren für das Unterformular.

Wichtig: Hierbei ist zu beachten, dass das Unterformular ein VBA-Klassenmodul enthält! Sie können das am einfachsten erreichen, indem Sie die Eigenschaft **Enthält Modul** auf **Ja** einstellen. Anderenfalls werden die Ereignisprozeduren, die in der Klasse **clsUndoMultiSub** angelegt sind, nicht für das betroffene Unterformular ausgelöst. Beim Hauptformular sollte dies standardmäßig der Fall sein, da wir dort ja auch die Ereignisprozedur **Form_Load** unterbringen, in der wie die **Undo**-Klassen initialisieren und einstellen.

Die ersten beiden Einstellungen, die der Benutzer vornehmen muss, sind der Name des Primärschlüssels der Datenherkunft des Formulars sowie der Name der Datenherkunft. Diese landen in den beiden folgenden Variablen:

```
Private m_PKForm As String
Private m_RecordsourceForm As String
```

Damit diese Variablen über die Eigenschaft **PKForm** gefüllt und ausgelesen werden kann, legen wir die beiden folgenden **Property Let**/**Get**-Prozeduren an:

```
Public Property Let PKForm(str As String)
    m_PKForm = str
End Property

Public Property Get PKForm() As String
    PKForm = m_PKForm
End Property
```

Für **m_RecordsourceForm** benötigen wir nur eine öffentliche Eigenschaft zum Zuweisen, ausgelesen wird diese Eigenschaft von außen nicht:

```
Public Property Let RecordsourceForm(str As String)
    m_RecordsourceForm = str
End Property
```

Die Klasse **clsUndoMultiMain** enthält außerdem die folgenden zwei Elemente, für welche Ereignisprozeduren implementiert werden:

```
Private WithEvents m_OKButton As CommandButton
Private WithEvents m_CancelButton As CommandButton
```

Über die folgenden beiden **Property Set**-Methoden können die Schaltflächen des Formulars den Variablen zugewiesen werden.

Dabei stellen wir auch gleich ein, dass das Ereignis **Beim Klicken** in dieser Klasse implementiert werden soll:

```
Public Property Set OKButton(cmd As CommandButton)
    Set m_OKButton = cmd
    With m_OKButton
        .OnClick = "[Event Procedure]"
    End With
End Property

Public Property Set CancelButton(cmd As CommandButton)
    Set m_CancelButton = cmd
    With m_CancelButton
        .OnClick = "[Event Procedure]"
    End With
End Property
```

Angabe des Hauptformulars

Der Verweis auf das Hauptformular soll in der Variablen **m_Form** gespeichert werden, für das wir ebenfalls Ereignisprozeduren implementieren wollen können:

```
Private WithEvents m_Form As Form
```

Die beiden folgenden Variablen nehmen die Verweise auf das **Database**- und das **Workspace**-Objekt auf:

```
Private m_db As DAO.Database
Private m_wrk As DAO.Workspace
```

Die Variable **m_Form** wird über die folgende **Property Set**-Methode gefüllt, die auch gleich einstellt, für welche Ereignisse im aktuellen Modul Ereignisprozeduren hinterlegt werden sollen. Außerdem werden hier die beiden Variablen **m_db** und **m_wrk** gefüllt.

Schließlich füllt sie noch das Recordset des Hauptformulars mit der in **m_RecordsourceForm** angegebenen Datenherkunft (zuerst als Recordset **rst** als Element des aktuelle **Database**-Objekts und dann durch Zuweisen dieser Variablen an die **Recordset**-Eigenschaft des Formulars):

```
Public Property Set Form(frm As Form)
   Dim rst As DAO.Recordset
    Set m_Form = frm
    With m_Form
        .AfterUpdate = "[Event Procedure]"
        .BeforeDelConfirm = "[Event Procedure]"
        .OnCurrent = "[Event Procedure]"
        .OnDelete = "[Event Procedure]"
        .OnDirty = "[Event Procedure]"
        .OnError = "[Event Procedure]"
        .OnOpen = "[Event Procedure]"
        .OnUnload = "[Event Procedure]"
        .OnUndo = "[Event Procedure]"
    End With
    Set m_db = DBEngine(0)(0)
    Set m_wrk = DBEngine.Workspaces(0)
    Set rst = m_db.OpenRecordset(m_RecordsourceForm, _
        dbOpenDynaset)
    Set m_Form.Recordset = rst
End Property
```

Wir müssen auch von der Klasse **clsUndoMultiSub** auf das in **clsUndoMultiMain** enthaltene Formular zugreifen können, also legen wir auch eine **Property Get**-Prozedur an:

```
Public Property Get Form() As Form
    Set Form = m_Form
End Property
```

Auch die Variable **m_wrk** mit dem Workspace-Objekt wollen wir von den Unterklassen aus nutzen. Dazu legen wir die folgende **Property Get**-Methode an:

```
Public Property Get wrk() As DAO.Workspace
    Set wrk = m_wrk
End Property
```

Es gibt noch drei Eigenschaften, mit denen wir den aktuellen Zustand der Daten im Formular speichern müssen. Diese sollen in der Hauptklasse **clsUndoMultiMain** gespeichert werden, aber auch von den Unterklassen zugreifbar sein. Daher legen wir auch diese als private Variablen an:

```
Private m_DirtyForm As Boolean
Private m_SavedForm As Boolean
Private m_DeletedForm As Boolean
```

m_DirtyForm müssen wir lesen und schreiben können:

```
Public Property Get DirtyForm() As Boolean
    DirtyForm = m_DirtyForm
End Property

Public Property Let DirtyForm(bol As Boolean)
    m_DirtyForm = bol
End Property
```

Gleiches gilt für **m_SavedForm**:

```
Public Property Get SavedForm() As Boolean
    SavedForm = m_SavedForm
End Property

Public Property Let SavedForm(bol As Boolean)
    m_SavedForm = bol
End Property
```

m_DeletedForm verwenden wir nur innerhalb der Klasse **clsUndoMultiMain**, also brauchen wir keine **Property**-Eigenschaften. Schließlich fehlt noch eine **Collection**-Variable, in der wir die durch die weiter unten beschriebene **AddSubform**-Methode hinzugefügten Instanzen der Klasse **clsUndoMultiSub** speichern:

```
Private col As Collection
```

Diese instanzieren wir in dem folgenden Ereignis, das gleich beim Erstellen der Klasse **clsUndoMultiMain** ausgelöst wird:

```
Private Sub Class_Initialize()
    Set col = New Collection
End Sub
```

Unterklassen für Unterformular hinzufügen

Damit kommen wir auch gleich zur Methode **AddSubform.** Diese sieht wie folgt aus und erwartet die weiter oben beschriebenen Parameter:

```
Public Sub AddSubform(frm As Form, _
        strRecordsourceSubform As String, _
        strFKSubform As String)
    Dim objUndoMultiSub As clsUndoMultiSub
    Set objUndoMultiSub = New clsUndoMultiSub
    With objUndoMultiSub
        Set .Main = Me
        Set .Subform = frm
        .RecordsourceSubform = strRecordsourceSubform
        .FKSubform = strFKSubform
    End With
    col.Add objUndoMultiSub
End Sub
```

Sie erstellt mit jedem Aufruf ein neues Objekt des Typs **clsUndoMultiSub** und speichert dieses in **objUndoMultiSub**. Dann stellt sie die mit den Parametern übergebenen

Null, leere Zeichenkette, Nothing und Co.

In Datenbanken wie Microsoft Access gibt es einen entscheidenden Unterschied zwischen einem leeren Feld und einem Feld mit einer leeren Zeichenkette. Und es gibt noch mehr interessante Dinge rund um dieses Thema, zum Beispiel die Funktion IsNull, die Funktion Nz und weitere VBA-Elemente, die sich mit ähnlichen Dingen befassen – wie etwa Nothing, Empty oder Missing. Dieser Beitrag erläutert diese VBA-Elemente und zeigt die wichtigsten Unterschiede und Einsatzzwecke auf.

Wer hat sich beim Einstieg in die Access-Welt nicht mindestens einmal in die Nesseln gesetzt, als er versucht hat, ein leeres Textfeld mit dem Wert "" (für eine leere Zeichenkette) zu vergleichen?

Schauen wir uns ein einfaches Szenario an: Das Formular aus Bild 1 ist an eine Tabelle mit einem Primärschlüsselfeld und den beiden Textfeldern **Vorname** und **Nachname** gebunden. Wenn der Benutzer auf die Schaltfläche **Speichern** klickt, sollen die Felder validiert werden. Verläuft die Validierung erfolgreich, wird der Datensatz gespeichert.

Dazu legen wir zunächst die Prozedur für die Schaltfläche **cmdSpeichern** an, welche schlicht eventuell vorhandene Änderungen am Datensatz speichert:

```
Private Sub cmdSpeichern_Click()
    On Error Resume Next
    RunCommand acCmdSaveRecord
    Select Case Err.Number
        Case 0, 2501
        Case Else
            MsgBox "Fehler " & Err.Number & " " _
                & Err.Description
    End Select
    On Error GoTo 0
End Sub
```

Die Fehlerbehandlung haben wir hinzufügt, weil der Versuch, den Datensatz zu speichern, beim Abbrechen

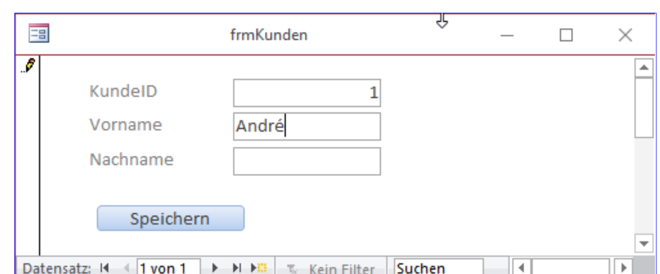


Bild 1: Ein einfaches Formular mit einem Textfeld

mit der nachfolgend vorgestellten Prozedur einen Fehler auslösen würde. Die folgende Ereignisprozedur wird durch das Ereignis **Vor Aktualisierung** ausgelöst. Es prüft, ob das Feld **Nachname** eine leere Zeichenkette enthält. Ist das nicht der Fall, soll eine entsprechende Meldung erscheinen, das Textfeld den Fokus erhalten und die Aktualisierung durch Setzen des Parameters **Cancel** auf den Wert **True** abgebrochen werden:

```
Private Sub Form_BeforeUpdate(Cancel As Integer)
    If Me!Nachname = "" Then
        MsgBox "Der Nachname ist eine leere Zeichenkette.", 
            vbOKOnly
        Me!Nachname.SetFocus
        Cancel = True
    End If
End Sub
```

Wenn wir dies ausprobieren, geschieht – nichts! Obwohl doch das Textfeld leer ist! Warum? Weil das Textfeld tatsächlich leer ist und der Inhalt genau nicht einer leeren Zeichenkette entspricht.

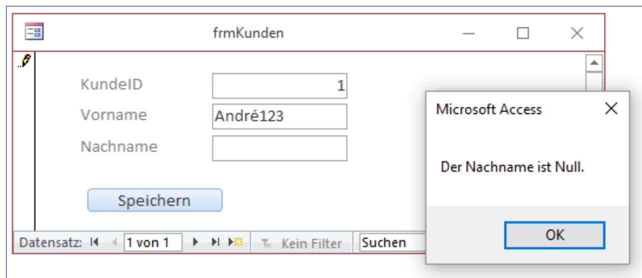


Bild 2: Es wurde ein Textfeld mit dem Inhalt **Null** erkannt.

Die IsNull()-Funktion

Diesen Sachverhalt prüfen wir nicht mit **=""**, und auch nicht mit **= Null**. Und auch das **Is**-Schlüsselwort kommt hier nicht zum Einsatz, **Is Null** hilft also auch nicht. Hier gibt es vielmehr zwei Möglichkeiten. Die erste ist, zusätzlich zum Vergleich mit der leeren Zeichenkette auch noch die Funktion **IsNull** einzusetzen:

```
...
If IsNull(Me!Nachname) Then
    MsgBox "Der Nachname ist Null.", vbOKOnly
    Me!Nachname.SetFocus
    Cancel = True
End If
...
```

Wenn wir den Datensatz nun speichern wollen, erscheint die gewünschte Meldung (s. Bild 2). Die Bedingung **Me!Nachname = ""** wird übrigens mit den Standardeinstellungen nicht eintreten.

Dies erreichen Sie erst, wenn zwei bestimmte Eigenschaften des Feldes im Tabellenentwurf spezielle Werte aufweisen. Die Eigenschaft **Eingabe erforderlich** muss dabei den Wert **Ja** aufweisen, die Eigenschaft **Leere Zeichenfolge** ebenfalls den Wert **Ja** (s. Bild 3).

Erst dann können Sie tatsächlich eine leere Zeichenfolge in das Feld **Nachname** eingeben – siehe Bild 4. Um auf Nummer Sicher zu gehen, was die zugrunde liegende Tabelle angeht,

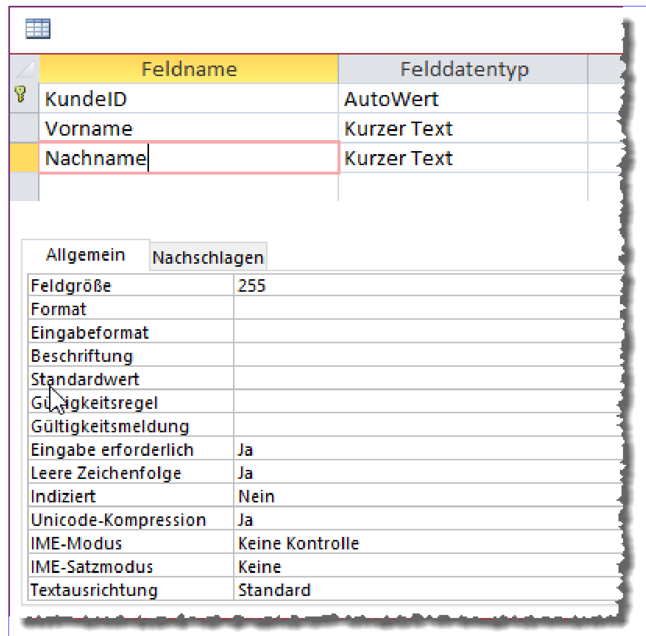


Bild 3: Einstellung für leere Zeichenketten

können Sie auch einfach eine Kombination der beiden Bedingungen nutzen:

```
If Me!Nachname = "" Or IsNull(Me!Nachname) Then
    MsgBox "Der Nachname ist Null oder eine leere 
        Zeichenkette.", vbOKOnly
    Me!Nachname.SetFocus
    Cancel = True
End If
```

Die Nz()-Funktion

Und es geht noch eine Nummer eleganter: Die Funktion **Nz()** erwartet den zu prüfenden Ausdruck, also beispiels-

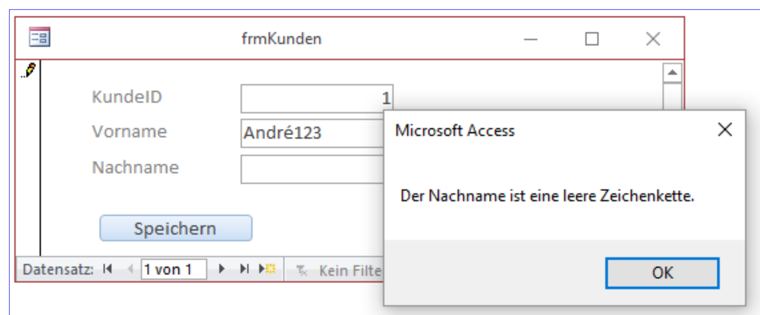


Bild 4: Eingabe einer leeren Zeichenkette

weise ein Textfeld wie in unserem Beispiel. Wenn der Inhalt des Textfeldes **Null** ist, liefert die Funktion den Wert zurück, der standardmäßig für diesen Datentyp verwendet wird. Bei Textfeldern ist das eine leere Zeichenkette.

Sie können allerdings auch explizit angeben, welcher Wert zurückgegeben werden soll, wenn der geprüfte Ausdruck den Wert **Null** liefert. In diesem Fall geben Sie als zweiten Parameter etwa die leere Zeichenkette an und prüfen dann nur noch, ob die Funktion eine leere Zeichenkette zurückgeliefert hat:

```
If Nz(Me!Nachname, "") = "" Then
    MsgBox "Der Nachname ist Null oder eine leere ⤶
                              Zeichenkette.", vbOKOnly
    Me!Nachname.SetFocus
    Cancel = True
End If
```

Bei einem Feld, das entweder den Wert **Null** oder den Wert **0** aufweisen darf, wird es interessanter. Dazu fügen wir der Tabelle ein Fremdschlüsselfeld namens **AnredeID** hinzu, mit der der Benutzer einen der Datensätze der Tabelle **tblAnreden** auswählen können soll. Für das Kombinationsfeld, mit dem wir dies erledigen wollen, hinterlegen wir die folgende Datensatzherkunft:

```
SELECT 0, "<Auswählen>"
FROM tblAnreden
UNION
SELECT [tblAnreden].[AnredeID], [tblAnreden].[Anrede]
FROM tblAnreden;
```

Dies liefert, wenn wir als Standardwert den Wert **0** einstellen, die Auswahl aus Bild 5 für einen neuen Datensatz. Wenn wir die beiden Textfelder ausfüllen und das Kombinationsfeld so beibehalten, gibt es Fehler **3201** (**Der Datensatz kann nicht hinzugefügt oder geändert werden, da ein Datensatz in der Tabelle 'tblAnreden' mit diesem Datensatz in Beziehung stehen muss.**). Diesen wollen wir natürlich verhindern, und zwar mit einer passenden Validierung. Damit der Fehler nicht ausgelöst wird, nehmen wir diesen zunächst in die Ereignisprozedur der Schaltfläche **cmdSpeichern** hinzu:

```
Private Sub cmdSpeichern_Click()
    ...
    Select Case Err.Number
        Case 2501, 3201
    ...
End Sub
```

Danach legen wir die folgende **If...Then**-Bedingung in der Methode **Form_BeforeUpdate** an:

```
  If IsNull(Me.AnredeID) Then
      MsgBox "Wählen Sie eine Anrede aus.", vbOKOnly
      Me!AnredeID.SetFocus
      Cancel = True
  End If
```

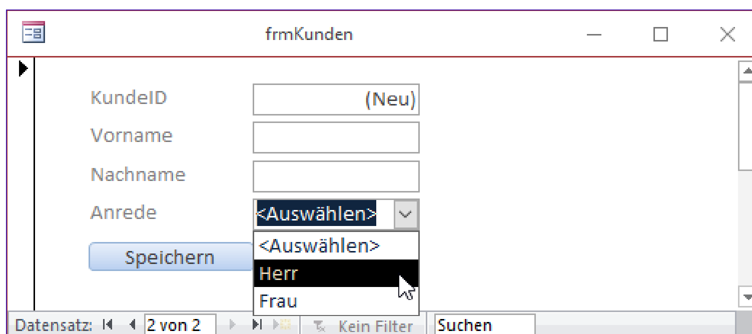


Bild 5: Ein Kombinationsfeld, das nicht den Wert **0** oder **Null** liefern soll

Wenn wir nun allerdings einen neuen Datensatz anlegen, die beiden Textfelder ausfüllen, den Standardwert des Kombinationsfeldes beibehalten und auf die Schaltfläche **cmdSpeichern** klicken, geschieht nichts! Der Datensatz wird nicht gespeichert, aber es erscheint auch keine Fehlermeldung. Der Grund ist einfach: Sobald Sie den Datensatz in den Bearbeitungsmodus versetzen, was geschieht, wenn Sie eines der Felder ändern, wird der ursprüngliche Wert,

Die CurrentDb-Funktion und das Database-Objekt

Der Zugriff auf die Objekte der aktuellen Datenbank erfolgt in vielen Fällen über eine Objektvariable mit dem Datentyp Database. Diese wird dann mit der CurrentDb-Funktion gefüllt. Der Zugriff auf die aktuelle Datenbank kann dabei auch direkt über CurrentDb erfolgen. Dieser Beitrag liefert Informationen über die Eigenschaften und Methoden des Database-Objekts.

Die folgenden beiden Zeilen kommen wohl in jeder Datenbank vor, die in irgendeiner Form mit VBA-Code ausgestattet ist:

```
Dim db As DAO.Database
Set db = CurrentDb
```

Die erste deklariert eine Objektvariable namens **db** auf Basis des Datentyps **Database**, die zweite weist dieser Objektvariablen mit der **CurrentDB**-Funktion einen Verweis auf das **Database**-Objekt der aktuellen Access-Instanz zu.

Aktuelles Abbild

In der Praxis können Sie das **Database**-Objekt mit der **CurrentDb**-Funktion füllen oder auch direkt über die **CurrentDb**-Funktion die Eigenschaften und Methoden des **Database**-Objekts der aktuellen Datenbank nutzen.

Es gibt allerdings einen entscheidenden Unterschied: Wenn Sie das aktuelle **Database**-Objekt einmal mit **CurrentDb** ermittelt und in einer Variablen wie **db** gespeichert haben, dann bekommen Sie Änderungen, die direkt über die Methoden von **CurrentDb** oder einer anderen Objektvariablen des Typs **Database** vorgenommen wurden, nicht mehr mit. Das folgende Beispiel zeigt, wie das gemeint ist:

```
Dim db As DAO.Database
Dim tdf As DAO.TableDef
Dim fld As DAO.Field
Set db = CurrentDb
On Error Resume Next
CurrentDb.TableDefs.Delete ("tblTest")
On Error GoTo 0
Debug.Print "Tabellen vorher: " & db.TableDefs.Count
Set tdf = CurrentDb.CreateTableDef("tblTest")
Set fld = tdf.CreateField("ID", dbLong)
tdf.Fields.Append fld
Set fld = tdf.CreateField("Test", dbText)
tdf.Fields.Append fld
CurrentDb.TableDefs.Append tdf
Debug.Print "Tabellen nachher CurrentDb: " ⁊
                         & CurrentDb.TableDefs.Count
Debug.Print "Tabellen nachher db: " ⁊
                         & db.TableDefs.Count
```

Hier füllen wir die Variable **db** über **CurrentDb** mit dem aktuellen **Database**-Objekt. Dann geben wir die Anzahl der Tabellendefinitionen des **Database**-Objekts aus **db** über **db.Table.Count** im Direktfenster aus – bei einer frischen **.accdb**-Datenbank im Format von Access 2016 sind das beispielsweise 11.

Anschließend erstellen wir über die **CreateTableDef**-Methode direkt auf Basis der von **CurrentDb** gelieferten Database ein neues **TableDef**-Objekt, also eine neue Tabelle, und fügen dieser zwei Felder hinzu, bevor wir sie mit der **Append**-Methode der **TableDefs**-Auflistung hinzufügen. Schließlich geben wir erneut die Anzahl der **TableDef**-Objekte der in **db** gespeicherten Database und der mit **CurrentDb** ermittelten aktuellen Version im Direktfenster aus. Es zeigt sich, dass die in **db** gespeicherte Version der Datenbank die neue Tabellendefinition nicht enthält und somit tatsächlich nur einen Snapshot der Objekte liefert.

Daten anonymisieren

Wenn ein potenzieller Kunde Sie um Unterstützung beim Programmieren oder Anpassen einer bestehenden Datenbank bittet, ist es am einfachsten, wenn diese Ihnen die Datenbank zum Analysieren zur Verfügung stellt. Das scheitert aber oft daran, dass der Kunde die Datenbank nicht herausgeben darf, weil die enthaltenen Daten nicht weitergegeben werden dürfen. Oft handelt es sich dabei um Adressdaten. Dieser Beitrag zeigt, wie Sie dem Kunden das Werkzeug bereitstellen, die enthaltenen Daten zu anonymisieren.

Warum anonymisieren?

Aber warum sollte man sich überhaupt die Mühe machen, die Daten zu anonymisieren? Es wäre doch viel einfacher, einfach die Datenbank zu kopieren und die enthaltenen Daten vor der Weitergabe zu löschen. Klar, das geht viel schneller: Allerdings enthält die Datenbank dann auch keine Daten und der Entwickler – in diesem Fall Sie – oder der Kunde müssen manuell einige Testdatensätze eingeben. Und manchmal sollen ja gerade Performance-Probleme behoben werden. In diesem Fall ist es natürlich komplett abwegig, eine leere Datenbank an den Entwickler zu übergeben.

Beides sind gute Anlässe, ein Tool zu programmieren, das den Kunden dabei unterstützt, die Datenbank in eine Form zu bringen, die keine nachvollziehbaren personenbezogenen Daten mehr enthält.

Konkreter Anlass

Im konkreten Fall geht es viel weniger um die oben beschriebene Konstellation, als um die Änderung der Personendaten zwecks Veröffentlichung der programmierten Lösung samt Beispieldaten zum Ausprobieren der Anwendung.

Der erste Vorsitzende eines befreundeten Sportvereins ist nämlich mit einer Excel-Datei mit Mitgliedsdaten an mich herangetreten und hat mich gebeten, einmal eine vernünftige Mitgliederverwaltung zu programmieren, mit der die Daten nicht mehr in einer einzigen Excel-Tabelle verwaltet werden müssen, sondern die es auch noch ermöglicht, zusätzliche Auswertungen zu produzieren.

Wenn ich Ihnen als Leser von Access im Unternehmen meine Umsetzung dieser Lösung darlegen möchte, muss ich also die Personendaten in dieser Excel-Tabelle vorher anonymisieren. Die Lösung mit der Umwandlung der Mitgliedsdatei in eine richtige Anwendung finden Sie übrigens unter dem Titel **Vereinsverwaltung: Von Excel zum Datenmodell** (**www.access-im-unternehmen.de/1106**) in diesem Heft.

Konzept für das Tool

Wie aber gestalten wir nun das Tool, mit dem wir die Personendaten anonymisieren wollen? Normalerweise würde ich es als Add-In programmieren. Allerdings müsste der Kunde dann erst das Add-In installieren, was diesen möglicherweise überfordern könnte. Also gehen wir diesmal einen anderen Weg und programmieren eine eigene Access-Lösung, mit welcher der Kunde dann die betroffene Access-Anwendung auswählen soll. Im Formular dieser Lösung sollen dann die Tabellen der Ausgangsanwendung zur Auswahl angeboten werden. Für die Felder der gewählten Tabelle soll der Benutzer dann angeben, mit welchen Daten diese gefüllt werden sollen – also mit Vornamen, Nachnamen, Straßen, PLZ und Ort, Land und so weiter. Dann soll er per Mausklick den Anonymisierungsvorgang starten können, welcher dann eine Kopie der Datenbank erstellt und die enthaltenen Daten ändert.

Das Tool erstellen

Als Tool legen wir also eine ganz normale Access-Datenbank an. Diese sollte beim Öffnen möglichst gleich die enthaltene Funktion offenbaren, damit der Benutzer sich

gleich zurechtfindet. In diesem Fall wäre dies das Öffnen des Formulars, welches die Funktionen zum Auswählen der zu kopierenden und zu manipulierenden Datenbank enthält.

Das Formular soll keine Datensätze anzeigen, also stellen wir die Eigenschaf-

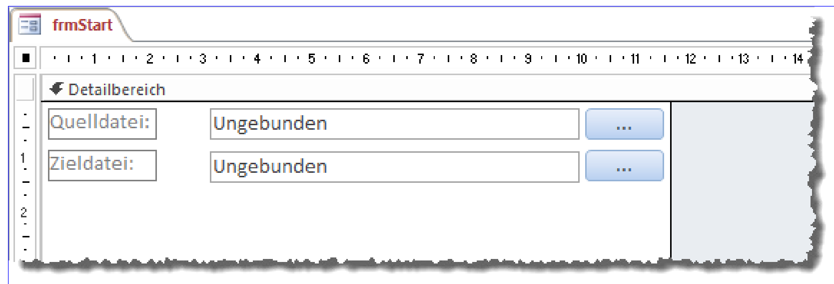


Bild 1: Steuerelemente zum Auswählen von Quell- und Zieldatei

```
Private Sub cmdDateiauswahl_Click()
    Dim strQuelldatei As String
    strQuelldatei = Openfilename(CurrentProject.Path, "Quelldatei auswählen", "Access-DB (*.mdb;*.accdb)|Alle Dateien (*.*)")
    Me!txtQuelldatei = strQuelldatei
    Me!txtZieldatei = VerzeichnisAusPfad(strQuelldatei) & "\" & ZieldateiAusPfad(strQuelldatei)
End Sub

Private Sub cmdZieldateiFestlegen_Click()
    Dim strZieldatei As String
    Dim strQuelldatei As String
    Dim strZielverzeichnis As String
    strQuelldatei = Me!txtQuelldatei
    If Len(Dir(strQuelldatei, vbDirectory)) > 0 Then
        strZieldatei = ZieldateiAusPfad(strQuelldatei)
        strZielverzeichnis = VerzeichnisAusPfad(strQuelldatei)
    End If
    strZieldatei = GetSaveFile(strZielverzeichnis, strZieldatei, "Access-DB (*.mdb;*.accdb)|Alle Dateien (*.*)", _
        "Zieldatei auswählen")
    Me!txtZieldatei = strZieldatei
End Sub

Private Function VerzeichnisAusPfad(strPfad As String) As String
    Dim strVerzeichnis As String
    strVerzeichnis = Left(strPfad, InStrRev(strPfad, "\") - 1)
    VerzeichnisAusPfad = strVerzeichnis
End Function

Private Function ZieldateiAusPfad(strPfad As String) As String
    Dim strZieldatei As String
    Dim intPunkt As String
    strZieldatei = Mid(strPfad, InStrRev(strPfad, "\") + 1)
    intPunkt = InStrRev(strZieldatei, ".")
    strZieldatei = Left(strZieldatei, intPunkt - 1) & "_Anonymisiert" & Mid(strZieldatei, intPunkt)
    ZieldateiAusPfad = strZieldatei
End Function
```

Listing 1: Prozeduren zum Auswählen von Quell- und Zieldatei

ten **Datensatzmarkierer**, **Navigationsschaltflächen**, **Trennlinien** und **Bildlaufleisten** auf **Nein** ein.

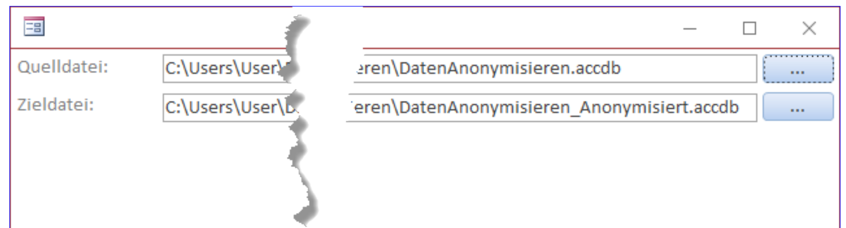


Bild 2: Auswählen von Quell- und Zieldatei

Fügen Sie im oberen Bereich zwei Textfelder namens **txtQuelldatei** und **txtZieldatei** ein sowie zwei Schaltflächen namens **cmdDateiauswahl** und **cmdZieldateiFestlegen**. Diese ordnen Sie wie in Bild 1 an.

Für die **Beim Klicken**-Ereigniseigenschaft hinterlegen wir jeweils eine entsprechende Ereignisprozedur. Ein Klick auf die Schaltfläche **cmdDateiauswahl** löst die Prozedur **cmdDateiauswahl_Click** aus Listing 1 aus. Diese verwendet die Funktion **Openfilename**, die Sie im Modul **mdlDateidialoge** finden. Sie übergibt der Funktion den aktuellen Datenbankpfad als Parameter und stellt die Access-Dateiendungen **.mdb** und **.accdb** als Filter ein.

Das Ergebnis dieses Dialogs speichert sie in der Variablen, deren Inhalt direkt an das Textfeld **txtQuelldatei** weitergegeben wird. Außerdem ruft die Prozedur zwei weitere Hilfsfunktionen namens **VerzeichnisAusPfad** und **ZieldateiAusPfad** auf, welche sich ebenfalls im Klassenmodul des Formulars befinden und im Listing abgebildet sind.

VerzeichnisAusPfad erwartet einen kompletten Pfad, also beispielsweise die Angabe aus Bild 2 im oberen Textfeld. Es ermittelt die Position des hintersten Backslash und liefert die Zeichenkette bis zu dieser Position minus eins, damit der Backslash wegfällt.

Die Funktion **ZieldateiAusPfad** arbeitet etwas spezifischer und liefert nicht nur einfach den Dateinamen des hineingegebenen Pfades, sondern ergänzt diesen so, dass der Dateiname gut als die anonymisierte Version zu erkennen ist. Die Funktion ermittelt ebenfalls zunächst in einer Anweisung den Teil des mit **strPfad** übergebenen Pfades, der sich hinter dem letzten Backslash in **strPfad** befindet. Das Verzeichnis wird also vorn abgeschnitten. Dann ermittelt sie die Position des Punktes, also des Zeichens, das sich zwischen dem eigentlichen Dateinamen und der Dateiendung befindet. Der in der Variablen **strZieldatei** zwischengespeicherte Ausdruck besteht dann aus dem Teil vor dem Punkt, dem Ausdruck **_anonymisiert.** und der Dateiendung. Dieses automatische Füllen des Feldes **txtZieldatei** ist eine Möglichkeit, dem Benutzer weitere Schritte abzunehmen, denn er braucht in den meisten Fällen nicht noch die Zieldatei auszuwählen.

Möchte er dies dennoch tun, hat er natürlich die Möglichkeit dazu. Er braucht dann nur auf die Schaltfläche **cmdZieldateiFestlegen** zu klicken, welche die Prozedur **cmdZieldateiFestlegen_Click** auslöst. Diese liest den Wert des Feldes **txtQuelldatei** in die Variable **strQuelldatei** ein und prüft, ob dieser Ausdruck ein gültiges Verzeichnis enthält.

In diesem Fall ermittelt die Prozedur über die Funktion **ZieldateiAusPfad** einen Vorschlag für den Namen der Zieldatei und speichert diesen in der Variablen **strZieldatei**. Das Zielverzeichnis wird über die Funktion **VerzeichnisAusPfad** ermittelt und landet in der Variablen **strZielverzeichnis**.

Danach ruft die Prozedur über die Funktion **GetSaveFile** einen Dialog auf, über den der Benutzer die Zieldatei festlegen soll. Hier wird auch klar, warum wir zuvor Zielverzeichnis und Zieldatei in zwei verschiedene Variablen geschrieben haben: Beim Aufruf dieser Funktion wird das Zielverzeichnis als beim Öffnen anzuzeigendes Verzeichnis gewählt und der Name der Zieldatei wird unten im Dialog voreingestellt (s. Bild 3). Schließlich landet die hier ausgewählte Datei im Textfeld **txtZieldatei**.

Datei kopieren

Wir haben ja avisiert, dass wir nicht in der vom Benutzer ausgewählten Datei herumspielen – es kann immerhin sein, dass der Benutzer zuvor keine Kopie anlegt und direkt eine Originaldatei zum anonymisieren auswählt. Also legen wir auf jeden Fall zuvor eine Kopie der Datenbank an, wozu wir die zuvor ermittelten Daten zur Quell- und Zieldatei nutzen.

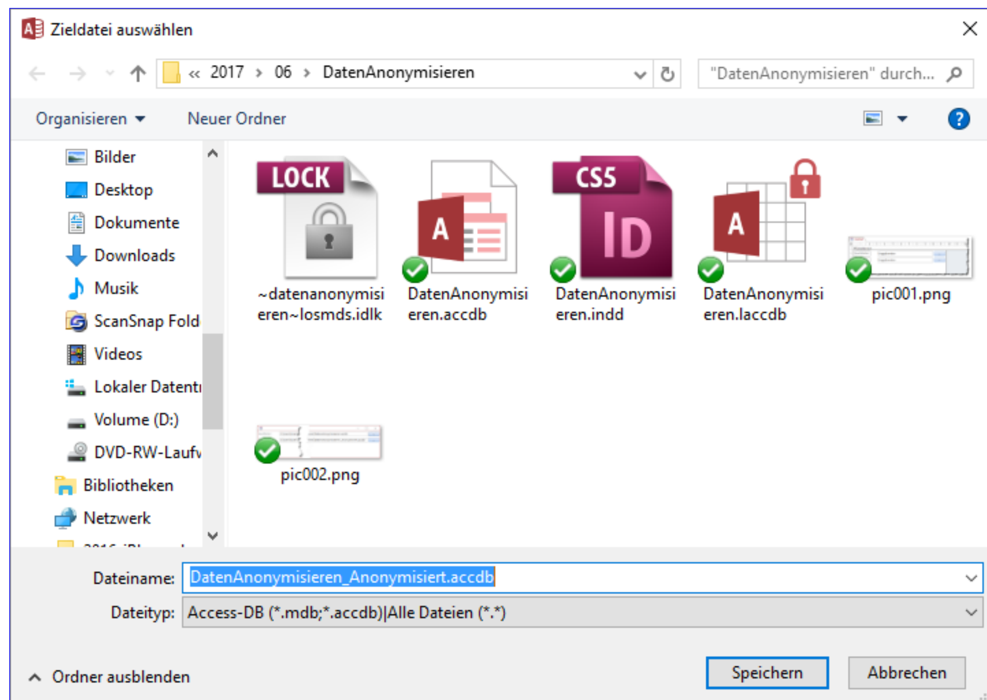


Bild 3: Auswählen der Zieldatei

Wir legen direkt einmal eine Schaltfläche namens **cmdAnonymisieren** im Formular an, auch wenn wir noch keine Steuerelemente zum Einstellen der Anonymisierungsdetails hinzugefügt habe und diese Funktionalität noch fehlt. Allerdings ist das Kopieren der Datei der Grundstein, daher beginnen wir damit (s. Bild 4). Diese Schaltfläche löst nun die folgende Prozedur aus, welche wiederum eine Funktion zum Kopieren der Quelldatenbank in die Zieldatenbank aufruft. Sollte diese Funktion namens **DateiKopieren** den Wert **True** zurückliefern, soll der Inhalt der **If...Then**-Bedingung ausgeführt werden:

```
Private Sub cmdAnonymisieren_Click()
    If DateiKopieren(Me!txtQuelldatei, Me!txtZieldatei) Then
        '...anonymisieren
    End If
End Sub
```

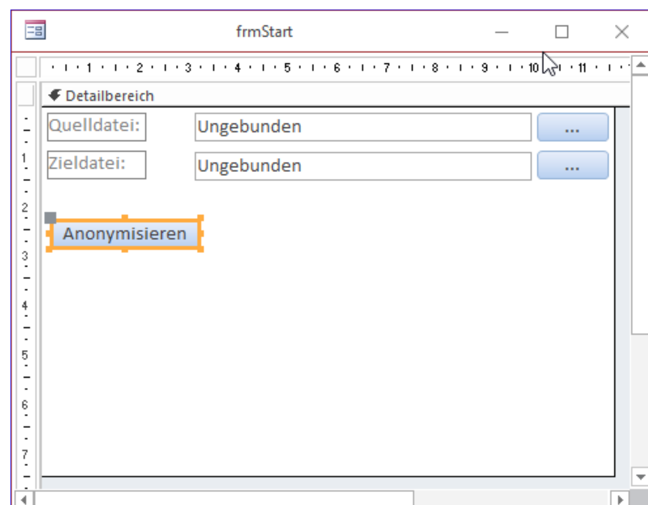


Bild 4: Schaltfläche zum Start des Vorgangs

Funktion zum Kopieren der Datenbank

Die dadurch aufgerufene Funktion **DateiKopieren** finden sie in Listing 2. Die Funktion erwartet den Pfad der Quell- und der Zieldatenbank als Parameter. Das Kopieren gelingt nicht, wenn die Quelldatei geöffnet ist. Ob dies der Fall ist, ermitteln wir, indem wir nach einer Datei suchen, deren Dateiname auf **.ldb** oder **.laccdb** endet. Dies ist die Dateiendung der Datei, die beim Öffnen einer Access-Datei angelegt wird. Ist eine solche Datei vorhanden, dann ist die Quelldatei offensichtlich geöffnet und kann nicht kopiert werden. Der Versuch, dies mit der **FileCopy**-Funktion zu erledigen, würde sonst zum Auslösen des Fehlers mit

```
Public Function DateiKopieren(strQuelle As String, strZiel As String)
    Dim strQuelleTemp As String
    Dim strZielTemp As String
    strQuelleTemp = Replace(strQuelle, ".mdb", ".ldb")
    strQuelleTemp = Replace(strQuelleTemp, ".accdb", ".laccdb")
    If Len(Dir(strQuelleTemp)) > 0 Then
        MsgBox "Die Quelldatei muss geschlossen sein."
        Exit Function
    End If
    strZielTemp = Replace(strZiel, ".mdb", ".ldb")
    strZielTemp = Replace(strZielTemp, ".accdb", ".laccdb")
    If Len(Dir(strZielTemp)) > 0 Then
        MsgBox "Die Zieldatei muss geschlossen sein."
        Exit Function
    End If
    FileCopy strQuelle, strZiel
    If Len(Dir(strZiel)) > 0 Then
        DateiKopieren = True
    End If
End Function
```

Listing 2: Die Funktion zum Kopieren der Zieldatei in die Quelldatei

der Nummer **70** führen (**Zugriff verweigert**). In diesem Fall gibt die Funktion eine entsprechende Meldung aus. Auch für die Zieldatei erfolgt eine entsprechende Überprüfung. Ist die Quelldatenbank hingegen nicht geöffnet und die Zieldatenbank entweder nicht geöffnet oder nicht vorhanden, kopiert die Prozedur diese mit der **FileCopy**-Methode, der Sie den Pfad zur Quell- und zur Zieldatei übergibt.

Ist die Zieldatei anschließend im Dateisystem vorhanden, stellt die Funktion den Rückgabewert auf **True** ein.

Tabellen und Felder speichern

Wir wollen es dem Benutzer ermöglichen, alle zu ändernden Tabellen und Felder auf einen Streich zu definieren. Dazu muss er diese zunächst einlesen und wir müssen mit einem geeigneten Formular dafür sorgen, dass diese auch vom Benutzer bearbeitet werden können.

Wir wollen also Informationen über die Tabellen und die darin enthaltenen Felder speichern. Dazu benötigen wir zwei Tabellen namens **tblTabellen** und **tblFelder**. Die erste sieht im Entwurf wie in Bild 5 aus.

Sie enthält ein Primärschlüsselfeld, ein Feld mit dem Namen der Tabelle sowie ein **Ja/Nein**-Feld namens **Anonymisieren**, mit der Benutzer festlegen kann, ob diese Tabelle komplett anonymisiert werden soll. Die zweite Tabelle namens **tblFelder** enthält einige Felder mehr. Das liegt daran, dass hier die Details zum Anonymisieren der Inhalte gespeichert werden. Den Entwurf der

tblTabellen

Feldname	Felddatentyp	Beschreibung (optional)
TabelleID	AutoWert	Primärschlüsselfeld der Tabelle
Tabellenname	Kurzer Text	Name der Tabelle
Anonymisieren	Ja/Nein	Ja/Nein-Feld, ob die Tabelle anonymisiert werden soll

Feldeigenschaften

Allgemein | Nachschlagen

Feldgröße	255
Format	
Eingabeformat	
Beschriftung	
Standardwert	
Gültigkeitsregel	
Gültigkeitsmeldung	
Eingabe erforderlich	Nein
Leere Zeichenfolge	Ja
Indiziert	Ja (Ohne Duplikate)
Unicode-Kompression	Ja
IME-Modus	Keine Kontrolle
IME-Satzmodus	Keine
Textausrichtung	Standard

Ein Feldname kann bis zu 64 Zeichen lang sein, einschließlich Leerzeichen. Drücken Sie F1, um Hilfe zu Feldnamen zu erhalten.

Bild 5: Entwurf der Tabelle **tblTabellen**

Tabelle können Sie Bild 6 entnehmen. Neben den bereits aus der Tabelle **tblTabellen** bekannten Feldern nutzen wir hier die folgenden Felder:

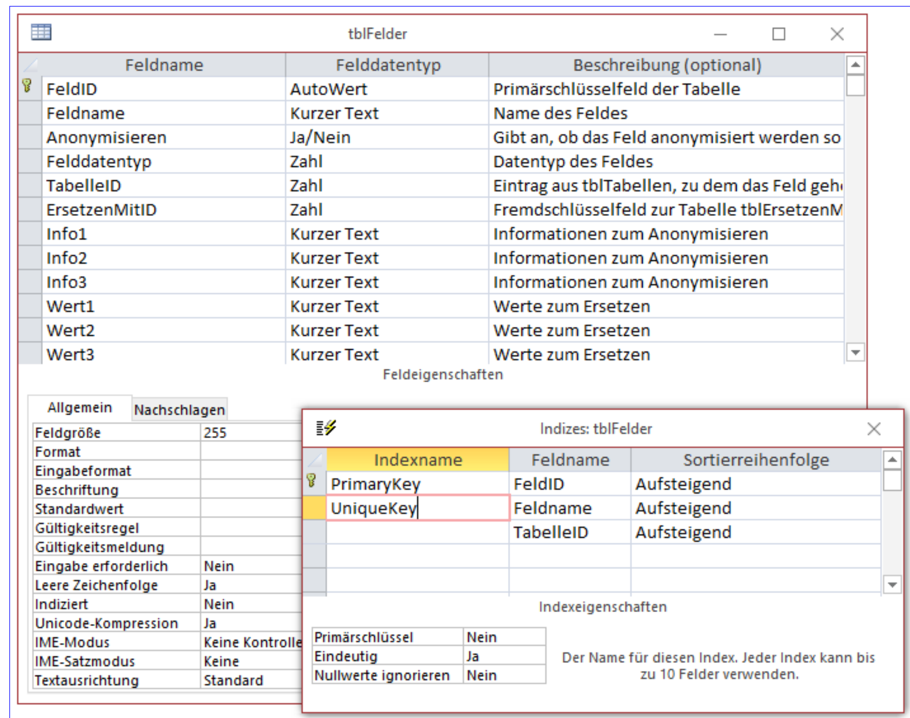


Bild 6: Entwurf der Tabelle **tblFelder**

- **Felddatentyp**: Speichert den Zahlenwert, der den Felddatentyp des Feldes repräsentiert
- **TabelleID**: Fremdschlüsselfeld, mit dem der Datensatz der Tabelle **tblTabellen** festgelegt wird, zu dem das Feld gehört
- **ErsetzenMitID**: Fremdschlüsselfeld zu einer Tabelle namens **tblErsetzenMit**. Hier werden verschiedene Möglichkeiten zum Ersetzen abgebildet, zum Beispiel Vorname, Nachname, Straße et cetera. Damit wird beispielsweise festgelegt, aus welchen Hilfstabellen die zu ersetzenden Daten kommen.
- **Info1**, **Info2**, **Info3**: Diese Felder enthalten Hinweistexte für weitere Informationen, die in den Feldern **Wert1**, **Wert2** und **Wert3** gespeichert werden. Damit können Sie zum Beispiel für Felder, die mit Datumswerten gefüllt werden sollen, den Datumsbereich festlegen.
- **Wert1**, **Wert2**, **Wert3**: Diese Felder nehmen die Werte auf, die zum Füllen der Felder als Referenz benutzt werden sollen.

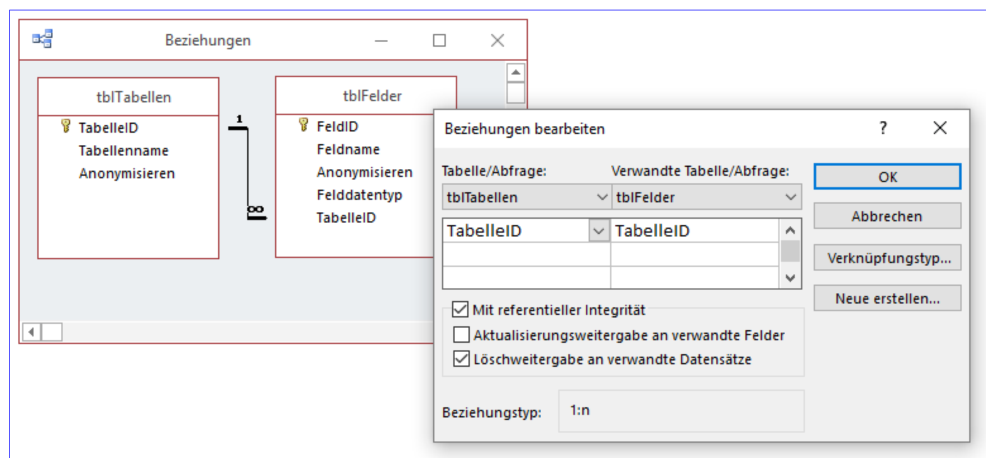


Bild 7: Beziehung zwischen den beiden Tabellen **tblTabellen** und **tblFelder**

Wie Sie der Abbildung entnehmen können, legen wir für die Tabelle außerdem einen zusammengesetzten, eindeutigen Index an, der dafür sorgt, dass jeder Feldname nur einmal je Tabelle eingegeben werden darf. Dazu fügen wir diesem die beiden Felder **Feldname** und **TabelleID** hinzu und legen für die Eigenschaft **Eindeutig** den Wert **Ja** fest.

tblErsetzenMit

Feldname	Felddatentyp	Beschreibung (optional)
ErsetzenMitID	AutoWert	Primärschlüsselfeld der Tabelle
ErsetzenMit	Kurzer Text	Bezeichnung der Ersetzungsart
Info1	Kurzer Text	Informationen zum Ersetzen
Info2	Kurzer Text	Informationen zum Ersetzen
Info3	Kurzer Text	Informationen zum Ersetzen
Wert1	Kurzer Text	Werte zum Ersetzen
Wert2	Kurzer Text	Werte zum Ersetzen
Wert3	Kurzer Text	Werte zum Ersetzen

Feldeigenschaften

Allgemein | Nachschlagen

Eigenschaft	Wert
Feldgröße	255
Format	
Eingabeformat	
Beschriftung	
Standardwert	
Gültigkeitsregel	
Gültigkeitsmeldung	
Eingabe erforderlich	Nein
Leere Zeichenfolge	Ja
Indiziert	Ja (Ohne Duplikate)
Unicode-Kompression	Ja
IME-Modus	Keine Kontrolle
IME-Satzmodus	Keine
Textausrichtung	Standard

Ein Feldname kann bis zu 64 Zeichen lang sein, einschließlich Leerzeichen. Drücken Sie F1, um Hilfe zu Feldnamen zu erhalten.

Bild 8: Entwurf der Tabelle **tblErsetzenMit**

tblKonfigurationen

Feldname	Felddatentyp	Beschreibung (optional)
KonfigurationID	AutoWert	Primärschlüsselfeld der Tabelle
Konfiguration	Kurzer Text	Bezeichnung der Konfiguration
Quelldatenbank	Kurzer Text	Pfad zur Quelldatenbank
Zieldatenbank	Kurzer Text	Pfad zur Zieldatenbank

Feldeigenschaften

Allgemein | Nachschlagen

Eigenschaft	Wert
Feldgröße	255
Format	
Eingabeformat	
Beschriftung	
Standardwert	
Gültigkeitsregel	
Gültigkeitsmeldung	
Eingabe erforderlich	Nein
Leere Zeichenfolge	Ja
Indiziert	Ja (Ohne Duplikate)
Unicode-Kompression	Ja
IME-Modus	Keine Kontrolle
IME-Satzmodus	Keine
Textausrichtung	Standard

Ein Feldname kann bis zu 64 Zeichen lang sein, einschließlich Leerzeichen. Drücken Sie F1, um Hilfe zu Feldnamen zu erhalten.

Bild 9: Entwurf der Tabelle **tblKonfigurationen**

Beziehung zwischen den Tabellen tblTabellen und tblFelder

Die Tabelle **tblTabellen** und **tblFelder** sollen in einer 1:n-Beziehung stehen. Dazu haben wir per Nachschlagefeld bereits die notwendige Verknüpfung hinzugefügt. Die Verknüpfung können Sie in Bild 7 einsehen. Hier definieren wir noch referenzielle Integrität und aktivieren die **Löschweitergabe an verwandte Datensätze**. Dadurch werden beim Löschen eines der Datensätze der Tabelle **tblTabellen** auch alle mit diesem Datensatz verknüpften Datensätze der Tabelle **tblFelder** gelöscht.

Die Tabelle tblErsetzenMit

Die Tabelle, die Informationen darüber liefert, wie die Daten eines Feldes ersetzt werden sollen, heißt **tblErsetzenMit** (s. Bild 8). Diese enthält neben dem Primärschlüsselfeld ein Feld mit der Bezeichnung der Ersetzungsart sowie, genau wie die Tabelle **tblFelder**, die drei Felder **Info1**, **Info2** und **Info3**. Diese werden mit Standardwerten befüllt, die dann nach der Auswahl eines der Einträge für das Fremdschlüsselfeld **ErsetzenMitID** der Tabelle **tblFelder** in die entsprechenden Felder **Info1**, **Info2** und **Info3** übertragen werden – gleiches gilt für die Felder **Wert1**, **Wert2** und **Wert3**. Der Benutzer kann die Felder dann nach eigenen Vorstellungen anpassen.

Konfigurationen speichern

Wenn wir schon darüber reden, dass der Benutzer die Einstellungen für das Anonymisieren der Daten in den beiden Tabellen **tblTabellen** und **tblFelder** speichert, dann sollten wir uns auch Gedanken machen, was geschieht, wenn der Vorgang abgeschlossen ist. Erst hatten wir geplant, die Daten einfach zu löschen. Allerdings ist dann klar geworden, dass es ja durchaus sein kann, dass der Benutzer nicht nur einmalig seine aktuelle Version der Datenbank anonymisieren und an einen Entwickler übergeben möchte. Also speichern wir die Konfigurationen in einer Tabelle namens **tblKonfigurationen** (s. Bild 9). Damit die gespeicherten Daten über die Tabellen und Felder auch der entsprechenden Konfiguration zugeordnet werden können, fügen wir der Tabelle **tblTabellen** noch ein Fremdschlüsselfeld namens **KonfigurationID** auf die Tabelle **tblKonfigurationen** hinzu.

Die Tabelle **tblKonfigurationen** enthält neben Primärschlüsselfeld und Bezeichnung noch die beiden Felder **Quelldatenbank** und **Zieldatenbank**. Gleich werden wir noch das Formular **frmStart** so anpassen, dass dieses das Anlegen und Abrufen von Konfigurationen ermöglicht.

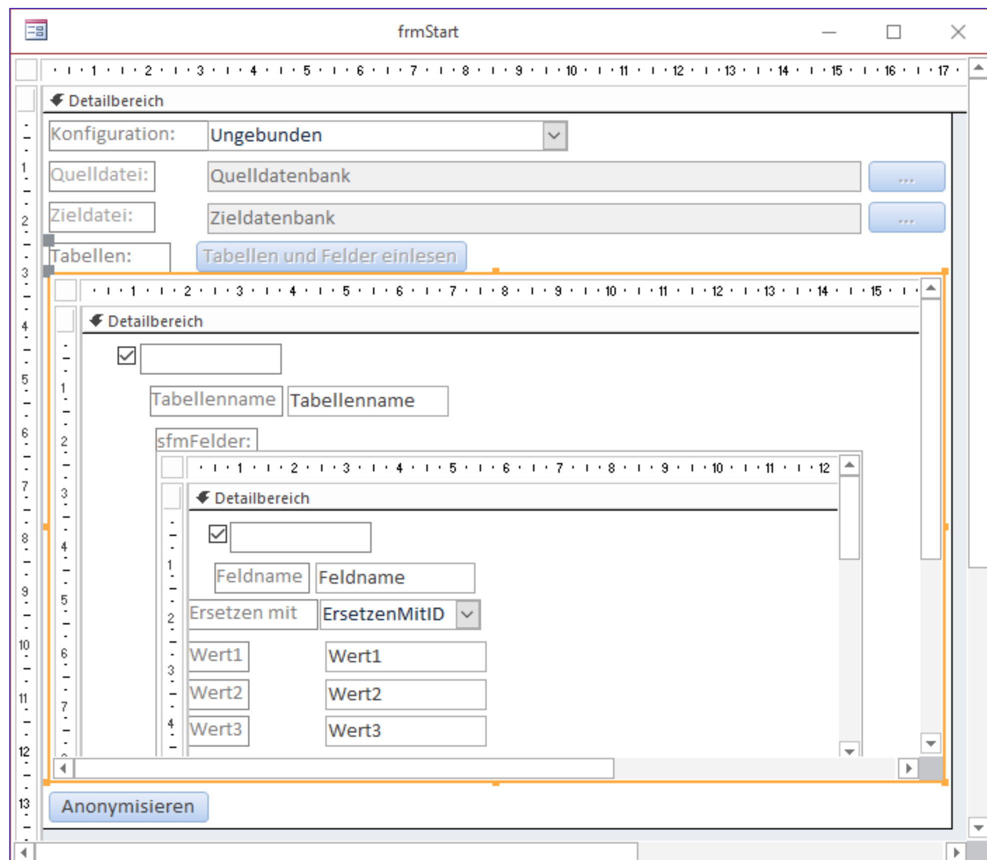


Bild 10: Das Formular **frmStart** nach dem Umbau

Formulare für die Festlegung von Tabellen und Feldern

Zuvor wollen wir jedoch noch die Unterformulare anlegen, die später im Formular **frmStart** landen und die Tabellen und Felder der zu anonymisierenden Datenbank zur Konfiguration durch den Benutzer anzeige sollen. Die Tabellen und Felder sollen in einer hierarchischen Ansicht erscheinen, wobei wir diesmal kein TreeView verwenden wollen, sondern die Unterdatenblatt-Ansicht von Access.

Wie Sie diese einrichten, erfahren Sie im Beitrag **Unterdatenblätter in Formularen** (**www.access-im-unternehmen.de/1108**) in dieser Ausgabe. Dort finden Sie sowohl die Beschreibung der Prozeduren, um die beiden Tabellen **tblTabellen** und **tblFelder** zu füllen als auch die Anleitung, wie Sie die Daten der beiden Tabellen in Form eines Datenblatts mit Unterdatenblatt anzeigen. Schließlich zeigen wir dort auch noch, wie Sie die Kontrollkästchen für die **Ja/Nein**-Felder **Anonymisieren** der beiden Tabellen so programmieren, dass die jeweils über- beziehungsweise untergeordneten Elemente aktiviert oder deaktiviert werden (mehr zu diesen Abhängigkeiten im genannten Beitrag).

Sobald sich nicht nur das Formular **frmStart**, sondern auch die beiden Unterformulare **sfmTabellen** und **sfmFelder** in Ihrer Datenbank befinden, können wir diese zusammenführen. Das Unterformular **sfmFelder** wurde ja bereits im Beitrag **Unterdatenblätter in Formularen** in das Formular **sfmTabellen** integriert.

Damit machen wir uns nun ans Werk, um den Entwurf des Formulars **frmStart** in den Zustand aus Bild 10 zu bringen.

Dazu sind die folgenden Schritte nötig:

- Einfügen des Unterformulars **sfmTabellen** aus dem Navigationsbereich in den Entwurf des Formulars **frmStart**
- Einstellen der Eigenschaften **Horizontaler Anker** und **Vertikaler Anker** für das entstandene Unterformular-

Steuerelement auf **Beide** einstellen. Passend dazu für das Bezeichnungsfeld des Unterformulars die eigenschaften auf **Links** beziehungsweise **Oben** einstellen und für die Schaltfläche **cmdAnonymisieren** auf **Links** und **Unten**.

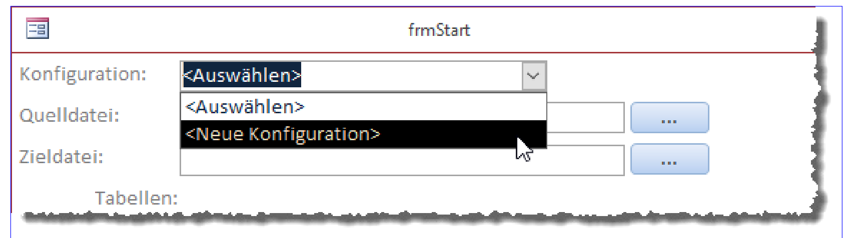


Bild 11: Auswahl einer neuen Konfiguration

- Einstellen der Datenherkunft des Formulars **frmStart** auf die Tabelle **tblKonfigurationen**.
- Die beiden Textfelder **txtQuelldatenbank** und **txtZieldatenbank** können Sie nun an die entsprechenden Felder der Datenherkunft binden, indem Sie die Eigenschaft **Steuerelementinhalt** auf die Werte **Quelldatenbank** und **Zieldatenbank** einstellen.
- Außerdem fügen wir oben im Formular noch ein Kombinationsfeld namens **cboKonfigurationen** hinzu, das wir gleich im Anschluss anpassen.

Für die Eigenschaft **Datensatzherkunft** des Kombinationsfeldes geben wir die folgende SQL-Abfrage ein:

```
SELECT KonfigurationID, Konfiguration
FROM tblKonfigurationen
ORDER BYKonfiguration;
```

Damit nur der Wert des Feldes **Konfiguration** angezeigt wird und nicht der Wert des Primärschlüsselfeldes **KonfigurationID**, stellen wir die Eigenschaften **Spaltenanzahl** auf **2** und **Spaltenbreiten** auf **0cm** ein.

Neue Konfiguration anlegen

Nun wollen wir eine neue Konfiguration anlegen. Dies wollen wir über das Kombinationsfeld **cboKonfigurationen** erledigen, indem wir diesem einen Eintrag mit dem Text **<Auswählen>** und einen weiteren mit dem Text **<Neue Konfiguration>** hinzufügen.

Dazu erweitern wir die Datensatzherkunft wie folgt:

```
SELECT -1, "<Auswählen>" AS Konfiguration
FROM MSysObjects
UNION
SELECT 0 , "<Neue Konfiguration>" AS Konfiguration
FROM MSysObjects
UNION
SELECT KonfigurationID, Konfiguration
FROM tblKonfigurationen
ORDER BY Konfiguration;
```

Da die Tabelle **tblKonfigurationen** zu Beginn gegebenenfalls keine Daten enthält, müssen wir als Datenquelle der beiden erstgenannten Einträge eine Tabelle angeben, welche mindestens einen Datensatz liefert. Dies ist bei der Systemtabelle **MSysObjects** immer der Fall.

Damit das Kombinationsfeld beim Öffnen des Formulars gleich den Eintrag **<Auswählen>** anzeigt, legen wir eine Prozedur an, die durch das Ereignis **Beim Laden** des Formulars ausgelöst wird. Diese sieht wie folgt aus und stellt den Wert des Kombinationsfeldes auf **-1** ein – also den Wert des ersten Eintrags der Datensatzherkunft:

```
Private Sub Form_Load()
    Me!cboKonfigurationen = -1
End Sub
```

Dadurch zeigt das Kombinationsfeld wie in Bild 11 direkt den Eintrag **<Auswählen>** an. Wenn der Benutzer nun den Eintrag **<Neue Konfiguration>** auswählt, soll eine **InputBox** erscheinen, in die der Benutzer den Namen der neuen Konfiguration eintragen kann. Dazu hinterlegen wir für die Ereignisprozedur, die durch das Ereignis **Nach**