HAUFE.
Ausgabe 06/2023

ACCESS
IM UNTERNEHMEN

STANDARDWERTE VON A-Z

Erfahren Sie alles über benutzerdefinierte Standardwerte und Standardzuordnungen in 1:n-Beziehungen (ab Seite 23).

In diesem Heft:

DATENMODELLE FÜR DIE RECHNUNGSVERWALTUNG

Entwickeln Sie das perfekte Datenmodell für die Rechnungsverwaltung – mit vielen Varianten für alle Zwecke.

SEITE 8

DATENEINGABE IN HAUPT- UND UNTERFORMULAR

Lernen Sie die Schwachstelle von verknüpften Daten in Haupt- und Unterformular kennen und beheben Sie diese.

SEITE 61

SELECT CASE FÜR TEXTE OPTIMIEREN

Entdecken Sie eine etwas andere Variante für die Programmierung von Select Case-Bedingungen zum Untersuchen von Texten.

SEITE 74

Einen Standard setzen

Ein stark unterschätztes Thema bei der Datenbankentwicklung sind Standardwerte. Es kann sehr viel Zeit sparen, wenn Steuerelemente beim Anlegen von Daten bereits vorab eingetragen sind und man diese nur noch bestätigen muss. Dabei gibt es verschiedene Möglichkeiten: Einerseits können wir statisch Standardwerte für Felder direkt im Tabellenentwurf festlegen, die dann in Formular übernommen werden. Oder wir nutzen dynamische Funktionen, die Informationen wie das aktuelle Datum, den zuletzt verwendeten Wert oder auch den am meisten verwendeten Wert liefern.

Standardwerte im Tabellenentwurf haben jedoch einen Nachteil: Um sie zu ändern, müsste man immer den Tabellenentwurf ändern, und das ist ein No-Go in Datenbankanwendungen. Also zeigen wir im Beitrag **Benutzerdefinierte Standardwerte** (ab Seite 23), wie wir dem Benutzer die Möglichkeit bieten, selbst Standardwerte für die einzelnen Felder der Tabelle zu definieren. So kann er selbst entscheiden, welche Werte er beim Anlegen neuer Datensätze präsentiert bekommen möchte.

Eine andere Art von Standardwerten liegt vor, wenn wir Daten in einer 1:n-Beziehung verwalten – wie bei Personen und E-Mail-Adressen, wobei wir jeder Person mehrere E-Mail-Adressen zuweisen können. Auch hier macht es Sinn, eine E-Mail-Adresse festzulegen, die bei neuen E-Mails vorbelegt wird. Allerdings müssen wir hier ein paar extra Schritte gehen, um sicherzustellen, für jeden Benutzer genau eine E-Mail-Adresse als Standard markiert ist. Wie das durch die Programmierung von Formularereignissen gelingt, zeigen wir im Beitrag **1:n-Beziehung mit Standardzuordnung verwalten** (ab Seite 40).

Die Lösung funktioniert, allerdings müsste man sie für jedes Formular, das auf die 1:n-Beziehung zugreift, erneut programmieren. Da hört es sich attraktiver an, die Absicherung der Datenkonsistenz über das Datenmodell selbst zu erledigen, in diesem Fall durch die Formulierung entsprechender Datenmakros. Wie das geht, zeigt der Beitrag **Standardzuordnung per Datenmakro** (ab Seite 53).

Wir haben einen neuen Bug entdeckt, der zwar nur in bestimmten Situationen auftaucht, aber dennoch zu Problemen führen kann, die man besser gleich erkennt. Alles dazu lesen Sie im Beitrag **Bug: Unterformular entlädt bei Bereichshöhe gleich 0** (ab Seite 2).

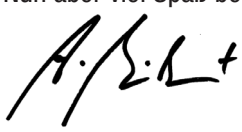
Unter dem Titel **Datenmodelle für die Rechnungsverwaltung** (ab Seite 8) schauen wir uns nochmal genau an, wie das Datenmodell für eine Rechnungsverwaltung aufgebaut sein kann – inklusive mehrerer Adressen pro Kunde für Lieferung und Rechnung.

Wer Daten aus 1:n- oder m:n-Beziehungen wie Bestellungen, Rechnungen und Co. eingibt, benötigt Haupt- und Unterformulare. Hier gibt es ein paar Stolpersteine, die wir im Beitrag **Dateneingabe in Haupt- und Unterformular** (ab Seite 61) ausräumen.

Gegebenenfalls möchte man in diesem Szenario die Daten im Hauptformular vor dem Speichern validieren, und wie das gelingt, zeigen wir unter dem Titel **Unterformular erst nach Validierung aktivieren** (ab Seite 69).

Schließlich werfen wir noch einen Blick auf die Select Case-Anweisung und wie wir diese für den Einsatz mit Texten optimal nutzen können – und zwar unter **Select Case-Bedingung für Texte optimieren** (ab Seite 74).

Nun aber viel Spaß beim Stöbern in der neuen Ausgabe!



Ihr André Minhorst

Bug: Unterformular entlädt bei Bereichshöhe gleich 0

Es gibt den einen oder anderen Bug in Microsoft Access, der nicht als solcher identifiziert werden kann, weil man einfach nicht herausfindet, wie man ein Fehlverhalten reproduzierbar auslöst. Im vorliegenden Fall ist es einem unserer findigen Kollegen gelungen, einen Bug zu erkennen, der bisher nach unserer Recherche noch nicht dokumentiert wurde. Es handelt sich um einen Bug, der je nach Konstellation mal gar keine Auswirkungen hat und mal schwere Folgen mit sich bringen kann. Den Auslöser zu identifizieren ist auch alles andere als trivial – aber wir haben ihn gleich im Titel präsentiert: Wenn die Höhe eines Unterformulars gleich 0 wird, entlädt Access das Formular. Welche Schritte zum Nachvollziehen nötig sind, welche Folgen dies haben kann und wie sich das Problem beheben lässt, erläutern wir in diesem Beitrag.

Eigentlich kann das gar nicht sein, dachte sich der Access-Entwickler Stefan M. (Name von der Redaktion geändert). Irgendwie spielt das Formular verrückt: Der Timer im Unterformular funktioniert nicht mehr, Variablen werden geleert, obwohl es keinen unbehandelten Laufzeitfehler gab (Stefan M. achtet sehr auf die Behandlung von Fehlern). Zumindest hat sein Kunde dieses Verhalten geschildert – er selbst hatte keinen Schimmer, wie er das Verhalten reproduzieren sollte.

Schließlich setzte er sich selbst mit dem Kunden zusammen und schaute sich an, was dieser in der Praxis mit dem Formular veranstaltet. Schließlich zeigte sich, dass wann immer das merkwürdige Verhalten auftrat, der Kunde die Höhe des Formulars verkleinert hatte, damit er zwar noch den Kopfbereich des Formulars sehen konnte, gleichzeitig aber genug Platz für andere Formulare verfügbar war. Es musste also irgendetwas mit der Höhe des Formulars zu tun haben.

Zurück am eigenen Rechner setzte Stefan M. sich hin und testete die halbe Nacht. Irgendwann hatte er den Fehler identifiziert: Wann immer er das Formular so klein machte, dass der Detailbereich nicht mehr zu sehen war, wurden die Variablen geleert und die aktuelle Markierung des Datensatzes im Unterformular wurde zurückgesetzt. Die erste Erkenntnis war: Das Unterformular musste auf irgendeine Art entladen worden sein. Einige Experimente später hatte er die

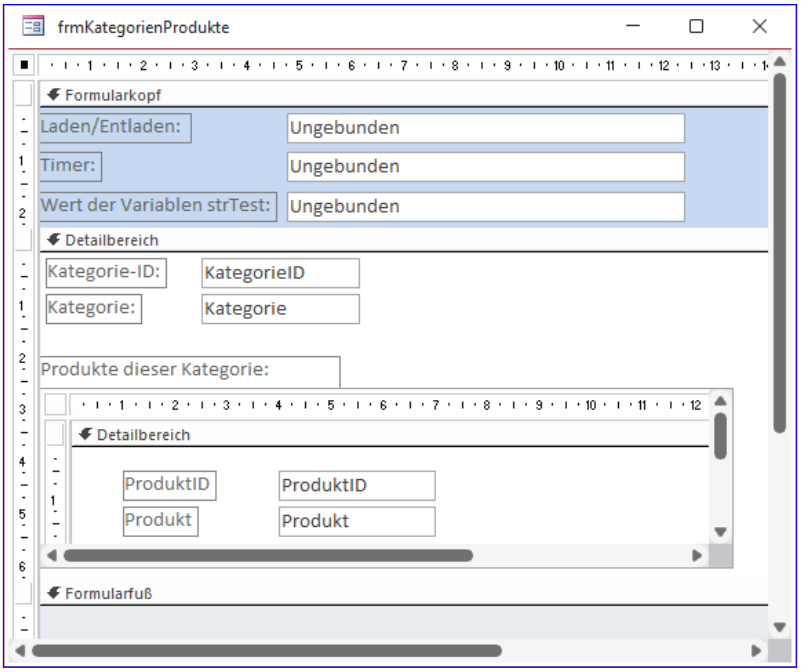


Bild 1: Entwurf des problembehafteten Formulars

Konstellation ermittelt, die zum Fehler führt – wir werden diese nun nachstellen.

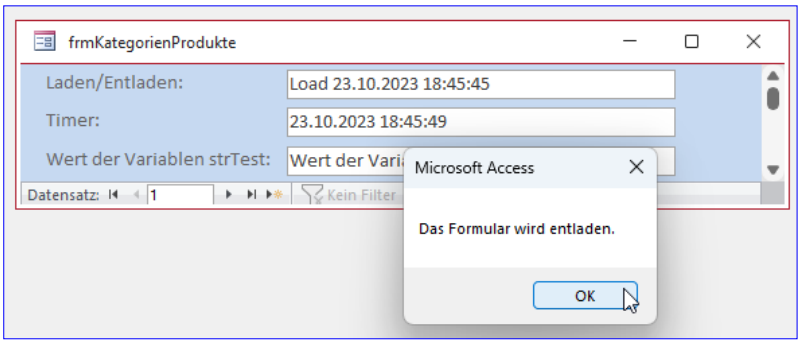


Bild 2: Der Bug tritt genau beim Erreichen der Höhe **0** des Detailbereichs auf.

Reproduzieren des Problems

Ausgangspunkt ist die folgende Situation: Wir verwenden ein Haupt- und ein Unterformular. Das Unterformular befindet sich im Detailbereich des Hauptformulars. Im Hauptformular sind außerdem Formularkopf und -fuß aktiviert. Diese Bereiche müssen nicht sichtbar sein, können also die Höhe **0** aufweisen – es reicht, dass sie aktiviert sind.

In unserem Beispiel zeigen wir im Detailbereich und im Unterformular noch Kategorien und die darin enthaltenen Produkte an – das ist kein dekoratives Beiwerk, sondern nötig für das Nachvollziehen des Beispiels (siehe Bild 1).

Um zu zeigen, dass das Formular zu einem bestimmten Zeitpunkt entladen wird, haben wir zunächst das Ereignis **Beim Entladen** des Unterformulars in Form der folgenden Ereignisprozedur implementiert:

```
Private Sub Form_Unload(Cancel As Integer)
    MsgBox "Das Formular wird entladen."
End Sub
```

Sobald das Formular entladen wird, sollte also diese Meldung erscheinen. Damit ließ sich schnell ermitteln, zu welchem Zeitpunkt das Formular entladen wurde – nämlich genau dann, wenn wir die Höhe des Formulars so verkleinert haben, dass der Detailbereich nicht mehr sichtbar war (siehe Bild 2).

Unterformular entladen – die Folgen

Nachdem die Ursache gefunden war, galt es, die Folgen zu untersuchen. Hier gibt es zumindest die folgenden Probleme:

- Der Datensatzzeiger im Unterformular wird wieder auf den ersten Datensatz gesetzt.
- Filter im Unterformular werden gelöscht.
- Variablen, die im Klassenmodul des Unterformulars deklariert sind, werden geleert.

Das haben wir wie folgt belegt. Der erste Teil ist der Timer. Wir haben die Eigenschaft **Zeitgeberintervall** auf **1.000** eingestellt (die Einheit ist Millisekunden, also wird das **Timer**-Ereignis einmal pro Sekunde ausgelöst).

Außerdem haben wir für die Ereigniseigenschaft **Bei Zeitgeber** die folgende Ereignisprozedur hinterlegt:

```
Private Sub Form_Timer()
    Me.Parent.txtTimer = Now
    Me.Parent.txtTest = strTest
End Sub
```

Diese schreibt bei jedem Aufruf des **Timer**-Ereignisses, also einmal pro Sekunde, das aktuelle Datum und die aktuelle Zeit in das Textfeld **txtTimer**. Außerdem schreibt sie den Wert einer Variablen namens **strTest** in das Textfeld **txtTest**. Diese Variable deklarieren wir wie folgt im Klassenmodul des Unterformulars:

```
Dim strTest As String
```

Beim Laden des Unterformulars wollen wir die Variable **strTest** erstmalig füllen und außerdem in ein weiteres Textfeld namens **txtLoadUnload** den Text **Load** und Datum und Zeit des Ladens eintragen.

```
Private Sub Form_Load()
    strTest = "Wert der Variablen strTest"
    Me.Parent.txtLoadUnload = "Load " & Now
    Me.Parent.txtTest = strTest
End Sub
```

Damit sieht der Inhalt des Formulars wie in Bild 3 aus. Das erste Textfeld enthält den Text **Load ...**, das zweite liefert sekündlich die aktuelle Zeit und das dritte wird regelmäßig mit dem Inhalt der Variablen **strTest** gefüllt.

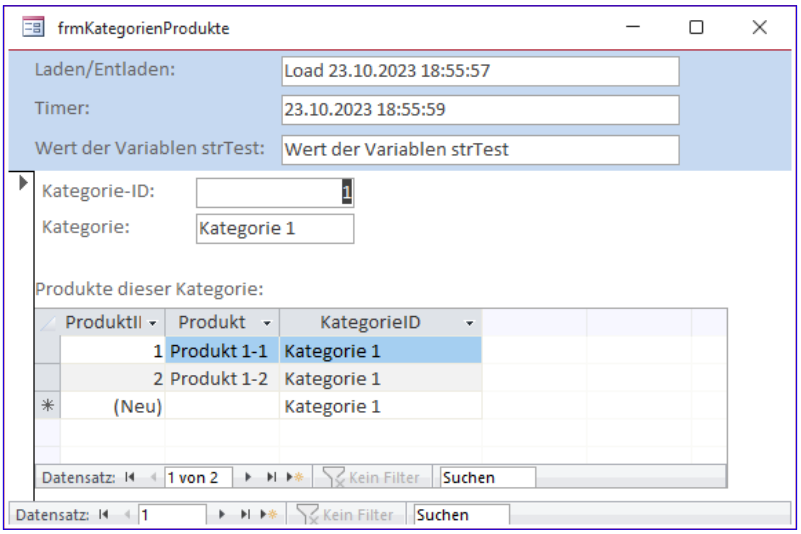


Bild 3: Das Formular direkt nach dem Öffnen

Um nun auch die Auswirkungen des Entladens des Formulars zu demonstrieren, kommentieren wir die **MsgBox** in der Prozedur für das Ereignis **Beim Entladen** aus und fügen eine Anweisung hinzu, die den Text im obersten Textfeld auf **Unload ...** ändert:

```
Private Sub Form_Unload(Cancel As Integer)
    'MsgBox "Das Formular wird entladen."
    Me.Parent.txtLoadUnload = "UnLoad " & Now
End Sub
```

Wenn wir nun die Höhe des Detailbereichs minimieren, zeigt das oberste Textfeld sofort den Text **Unload ...** an, was zeigt, dass die **Form_Unload**-Prozedur ausgelöst wurde.

Weitere Experimente

Wir sehen nun, dass das **Unload**-Ereignis ausgelöst wurde und somit alle Variablen geleert wurden. Weitere Ergebnisse sind:

- Wenn wir im Unterformular den zweiten Datensatz markieren, die Höhe des Detailbereichs minimieren und diesen dann wieder vergrößern, ist wieder der erste Datensatz im Unterformular markiert.

- Wenn wir im Unterformular einen Filter anwenden, der beispielsweise nur den ersten Datensatz anzeigt, und den Detailbereich des Hauptformulars verkleinern und wieder vergrößern, ist der Filter wieder entfernt.

- Andererseits bleibt eine einmal getätigte Sortierung im Unterformular erhalten.

Lösungen des Problems

Eine Lösung dieses Problems sahen wir zunächst nicht. Selbst das Setzen des Parameters **Cancel** auf den Wert **True** im **Unload**-Ereignis des Unterformulars hat keinerlei Auswirkung – das Formular wird dennoch entladen.

Die einzige Möglichkeit schien, die offensichtlichen Fehler und Probleme zu identifizieren und diese per Workaround zu beheben – also zum Beispiel Variablen in der Code behind-Klasse des Hauptformulars zu deponieren, den aktuellen Filter in einer Variablen zu speichern und wiederherzustellen, wenn das Formular entladen wurde oder auch die Position des Datensatzzeigers zu speichern und wiederherzustellen, wenn das Unterformular wieder sichtbar wird.

Ungewöhnliche, aber unpraktische Lösung

Dann erschien doch noch eine Lösung, die schlicht und einfach mit intuitiv bezeichnet werden kann – wer auf

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP

Datenmodelle für die Rechnungsverwaltung

Beim Schreiben von Anwendungen, mit denen Rechnungen erstellt werden sollen, stellt sich immer wieder die Frage nach dem korrekten Datenmodell. Davon ausgehend, dass es nicht das perfekte Datenmodell für alle Anwendungsfälle gibt, wollen wir in diesem Beitrag einmal unterschiedliche Ansätze betrachten und diskutieren. Diese haben eines gemein: Die Anwendung mit diesem Datenmodell soll die Möglichkeit bieten, sowohl Bestellungen zu erfassen als auch Rechnungen und Lieferscheine zu erstellen. Wie welche Daten gespeichert werden und welche Möglichkeiten es gibt, schauen wir uns nun an.

Bestellungen erfassen

Über bestimmte grundlegende Anforderungen herrscht vermutlich eine einstimmige Meinung: Wenn man Bestellungen erfassen möchte und die Regeln der Normalisierung von Datenmodellen befolgen will, benötigt man mindestens drei Tabellen. Die erste Tabelle enthält die Kundendaten mitsamt einer Kunden-ID und den Daten wie Firma, Anrede, Vorname, Nachname, Straße, PLZ, Ort, Land, Telefon, E-Mail und so weiter.

In den meisten Fällen liegen die für das Speichern der Anschrift vorgesehenen Felder in zweifacher Ausführung vor, nämlich für die Lieferadresse und die Rechnungsadresse, denn diese können sich unterscheiden. Normalerweise finden wir hier für Liefer- und Rechnungsadresse meist auch noch ein Fremdschlüsselfeld zur Auswahl eines Eintrags einer Anrede-Tabelle – die zählen wir hier aber nicht mit.

Die zweite Tabelle enthält die grundlegenden Daten einer Bestellung wie die Bestell-ID, das Bestelldatum und ein Fremdschlüsselfeld, mit dem der Kunde ausgewählt werden kann, der diese Bestellung getätigt hat. Auf diese Weise können wir zu jedem Kunden keine, eine oder mehrere Bestellungen anlegen.

Die dritte Tabelle enthält die Positionen der Bestellung. Diese umfassen eine Bezeichnung der Position, den Einzelpreis, die Menge, die Mehrwertsteuer und gegebenenfalls noch einen Rabatt. Außerdem benötigen wir hier ein Fremdschlüsselfeld, das mit der Bestellungen-Tabelle verknüpft ist, damit wir die einzelnen Bestellpositionen der Bestellung zuordnen können.

Normalerweise gibt es mindestens noch eine vierte Tabelle, aus der die grundlegenden Daten für die Bestellpositionen stammen. Diese nennen wir üblicherweise Produkt- oder Artikeltabelle. Sie enthält die Produkt- oder Artikelbezeichnung, den Einzelpreis sowie den Mehrwertsteuersatz und weitere Detailinformationen, die für andere Vorgänge wie Warenwirtschaft erforderlich sind – wie die Verwaltung von Beständen et cetera. Wir wollen im Folgenden von Produkten statt von Artikeln sprechen.

Um die Bestellposition mit dem jeweiligen Produkt zu verknüpfen, enthält die Tabelle der Bestellpositionen meist ein Fremdschlüsselfeld zu der Tabelle der Produkte. Auf diese Weise kann man über die Benutzeroberfläche leicht das zu einer Bestellposition hinzuzufügende Produkt auswählen.

Der erste Entwurf sieht im Datenbankfenster von Access wie in Bild 1 aus.

Produktdaten mit den Bestellpositionen speichern?

Hier gibt es direkt die erste spannende Frage: Wenn ich eine Tabelle mit Bestellpositionen verwende und eine mit den Produkten und in der Tabelle der Bestellpositionen ein

Feld ist, mit dem ich das mit dieser Position bestellte Produkt selektieren kann, benötige ich dann überhaupt die ganzen bereits erwähnten Felder? Wenn bereits in der Produkte-Tabelle die Bezeichnung, der Einzelpreis und der Mehrwertsteuersatz gespeichert sind, reicht es dann nicht aus, wenn ich mit einem Fremdschlüsselfeld aus der Bestellpositionen-Tabelle auf den Eintrag aus der Produkte-Tabelle verweise? Und in der Bestellpositionen-Tabelle nur die Felder Menge und Rabatt fülle?

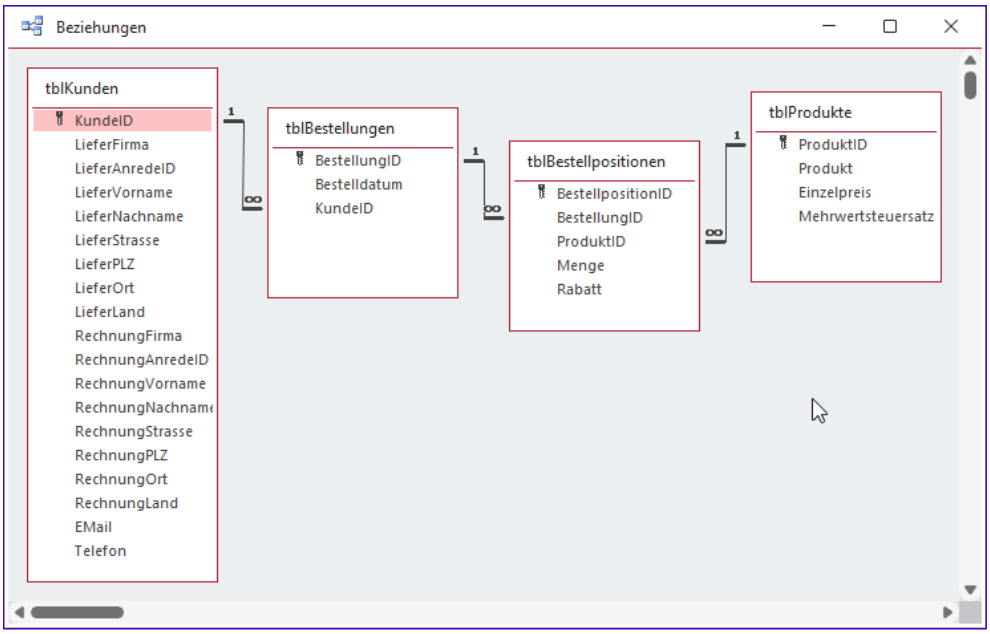


Bild 1: Erster Entwurf mit vielen Schwachstellen (V1)

Unter dem Gesichtspunkt, möglichst wenige Daten zu speichern, scheint diese Idee verlockend. Es gibt jedoch ein entscheidendes Problem: Einzelpreise und Mehrwertsteuersätze ändern sich von Zeit zu Zeit. Und es kann sogar vorkommen, dass wir den Namen eines Produktes ändern wollen (auch wenn es sich dann anbieten würde, einen neuen Produktdatensatz zu diesem Zweck anzulegen). Wenn es Bestellungen für ein Produkt gibt, und der Einzelpreis und der Mehrwertsteuersatz nur in der Produkttabelle vorliegen, dann können wir später nicht mehr nachvollziehen, zu welchem Preis und zu welchem Mehrwertsteuersatz der Kunde zuvor bestellt hat. Stattdessen werden wir alle Bestellpositionen immer mit den aktuellen Einzelpreisen und Mehrwertsteuersätzen vorfinden.

Wir können so also nach der Änderung auch nur eines Einzelpreises oder Mehrwertsteuersatzes für ein Produkt in einer Bestellposition einer Bestellung nicht mehr die tatsächlichen Umsätze nachvollziehen.

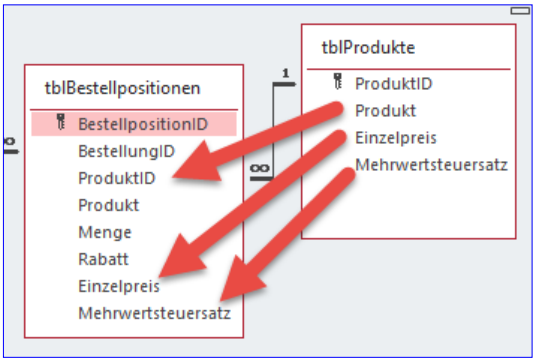


Bild 2: Diese Felder sollen dauerhaft mit der Bestellposition gespeichert werden (V2).

Den aktuellen Stand des Datenmodells finden Sie in der Beispieldatenbank **GrundlagenDatenmodellRechnungsverwaltung_V1.accdb**.

Änderungen von Einzelpreis und Mehrwertsteuersatz behandeln

Was tun? Damit die Änderungen am Produktnamen und an den Einzelpreisen und Mehrwertsteuersätzen eines einzelnen Produkts in der Produkttabelle keine Auswirkungen mehr auf die zum Zeitpunkt der Bestellung gültigen Werte haben, müssen wir diese Daten direkt in der Tabelle mit den Bestellpositionen speichern.

Dazu sehen wir einfach drei Felder mit den gleichen Namen wie in der Produkte-Tabelle vor, also beispielsweise **Produkt**, **Einzelpreis** und **Mehrwertsteuersatz** (siehe Bild 2).

Fügen wir in der Benutzeroberfläche einer Bestellposition ein Produkt beispielsweise durch Auswahl eines Nachschlagefeldes hinzu, kopieren wir direkt die Werte der Felder **Produkt**, **Einzelpreis** und **Mehrwertsteuersatz** in die Tabelle der Bestellpositionen.

Diese und die folgende Änderung haben wir in der Beispieldatenbank **GrundlagenDatenmodellRechnungsverwaltung_V2.accdb** hinterlegt.

Auf das Thema **Mehrwertsteuersätze** gehen wir gegen Ende des Beitrags im Detail ein.

Änderungen von Adressdaten behandeln

Das sind allerdings nicht die einzigen Daten, die Änderungen unterliegen. Auch die Adressen von Kunden ändern sich von Zeit zu Zeit. Das ist auch kein Problem: Wir ändern einfach die Adressdaten in der Kundentabelle und haben jederzeit den aktuellen Stand, um neue Bestellungen korrekt erfassen zu können.

Die Sache hat allerdings einen Haken: Im aktuellen Stand unseres Datenmodells ist die Kundentabelle noch über ein Fremdschlüsselfeld in der Tabelle **tblBestellungen** mit der Bestellung verknüpft. Wenn wir die Daten in der Kunden-Tabelle ändern, dann ändern wir automatisch auch die Adressdaten des Kunden einer Bestellung.

Das bringt wieder Nachteile mit sich, wenn wir einmal von der Region abhängige Auswertungen über unsere Bestelldaten durchführen möchten. Zieht ein Kunde von Duisburg nach München, ändert sich auch die Auswertung, denn nun hat Bayern ein paar Bestellungen und Umsätze mehr auf der Habenseite.

Auch das wollen wir tunlichst vermeiden. Wie können wir das tun? Wir wählen einen ähnlichen Ansatz wie bei den Bestellpositionen und Produkten: Wir legen in der Bestellungen-Tabelle weitere Felder an, in denen wir die Kundenadressen zum Zeitpunkt der Bestellung speichern. Das Fremdschlüsselfeld zum Kunden in der Tabelle

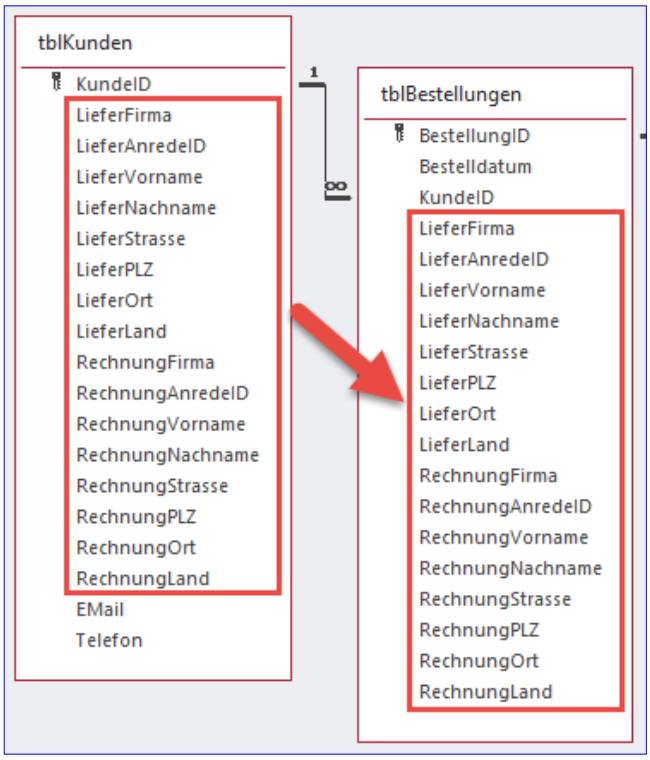


Bild 3: Diese Felder sollen dauerhaft mit der Bestellung gespeichert werden (V2).

tblBestellungen sollten wir jedoch beibehalten, denn wir wollen ja gegebenenfalls auch einmal einen Überblick über die Bestellungen eines Kunden und die darin enthaltenen Positionen haben.

Wie in Bild 3 zu sehen, haben wir eine Reihe von Feldern auch in der Tabelle der Bestellungen angelegt. In der Praxis gelingt das übrigens ganz einfach – Sie müssen einfach nur die zu kopierenden Felder im Entwurf der Quelltabelle kopieren und dann in der Zieltabelle einfügen.

Auch hier müssen wir in der Benutzeroberfläche beim Auswählen des Kunden zu einer Bestellung Code auslösen, der dafür sorgt, dass die Felder der Bestellungen-Tabelle mit den Daten aus der Kundentabelle gefüllt werden.

Theoretisch könnten wir auch noch die beiden Felder **EMail** und **Telefon** zur Tabelle **tblBestellungen** hinzufügen. Dies hätte den Sinn, dass wir später noch reproduzieren könnten, wohin wir E-Mails beispielsweise mit

Bestellbestätigungen oder Rechnungen gesendet haben. Allerdings gehen wir davon aus, dass die E-Mails auch archiviert werden und im späteren Verlauf gegebenenfalls eingesehen werden können.

Adressen separat speichern

In vielen Datenbanken sehen wir, dass Adressen separat in eigenen Tabellen gespeichert werden. Das könnte man angehen, indem man eine Tabelle für Adressen anlegt und dort Adressen speichert, die man dann über zwei Fremdschlüsselfelder namens **LieferadresseID** und **RechnungsadresseID** referenziert.

Danach würde dieser Teil des Datenmodells wie in Bild 4 aussehen. Wir haben hier im gleichen Zuge noch eine Frage gelöst, nämlich die, ob wir E-Mail und Telefon nur einmal je Kunde benötigen oder ob wir für die Liefer- und die Rechnungsadresse jeweils eigene Felder namens **EMail** und **Telefon** vorsehen sollen.

Wir haben die beiden Felder einfach in die Tabelle **tblAdressen** verschoben, wo man diese Daten nun für alle Adressen hinterlegen kann.

Wenn wir entschieden hätten, dass wir nur eine E-Mail-Adresse und eine Telefonnummer benötigen, hätten wir diese Felder in der Tabelle **tblKunden** belassen können.

Es macht jedoch mehr Sinn, hier auf Flexibilität zu setzen, denn es kann beispielsweise passieren, dass ein Kunde ein digitales Produkt ordert, zu dem Daten per E-Mail versendet werden sollen und der für die Rechnungen eine eigene Rechnungs-E-Mail-Adresse verwendet.

Beliebige Anzahl an Adressen

Mit dieser Version des Datenmodells sind wir allerdings auf je zwei Adressen pro Kunde beschränkt. Wer zum Beispiel schon einmal bei Amazon bestellt hat, weiß, dass man dort beliebig viele Lieferadressen hinterlegen kann. Wir wollen uns einmal ansehen, wie wir unser Datenmodell verändern müssten, um dies zu implementieren.

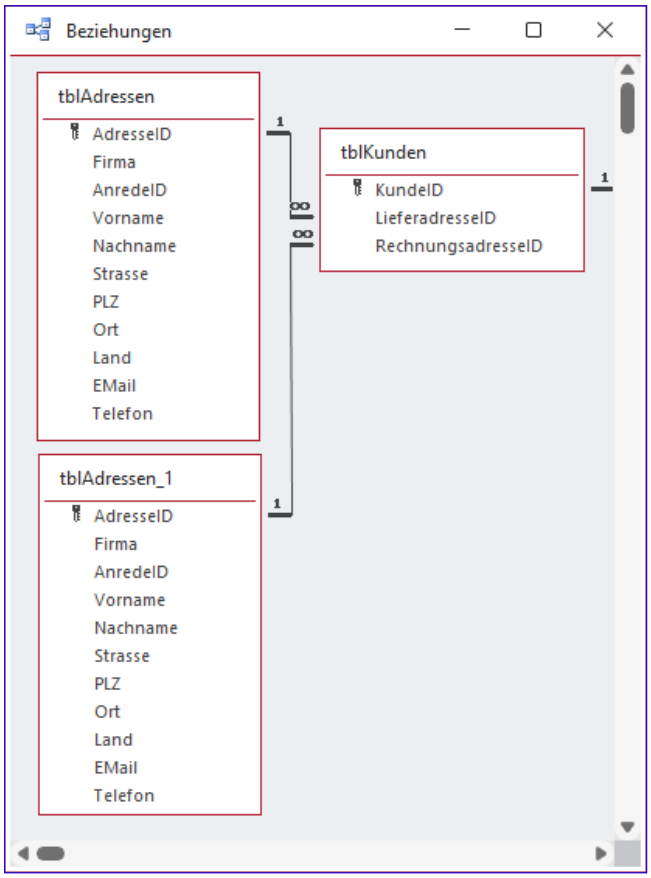


Bild 4: Ausgliederung der Liefer- und Rechnungsadressen in eine weitere Tabelle (V3).

Voraussetzung soll jedoch sein, dass wir eine der Adressen als Standardlieferadresse festlegen können und eine als Standardrechnungsadresse. Dazu gibt es zwei Möglichkeiten:

- Wir belassen die Struktur weitgehend so, wie sie ist, und erlauben damit die Zuordnung der aktuellen Standardadresse für die Lieferung und die Rechnungen über die beiden Fremdschlüsselfelder **LieferadresseID** und **RechnungsadresseID** der Tabelle **tblKunden**. Damit wir mehr als diese beiden Adressen für einen Kunden anlegen und diese auch zuordnen können wollen, müssen wir der Tabelle **tblAdressen** ein Fremdschlüsselfeld hinzufügen, mit dem wir den Kunden festlegen können, zu dem diese Adresse gehört. Erst dann können wir in der Benutzeroberfläche für diesen Kunden die passenden Adressen zur Auswahl für die

beiden Felder **LieferadresseID** und **RechnungsadresseID** anbieten (siehe Bild 5).

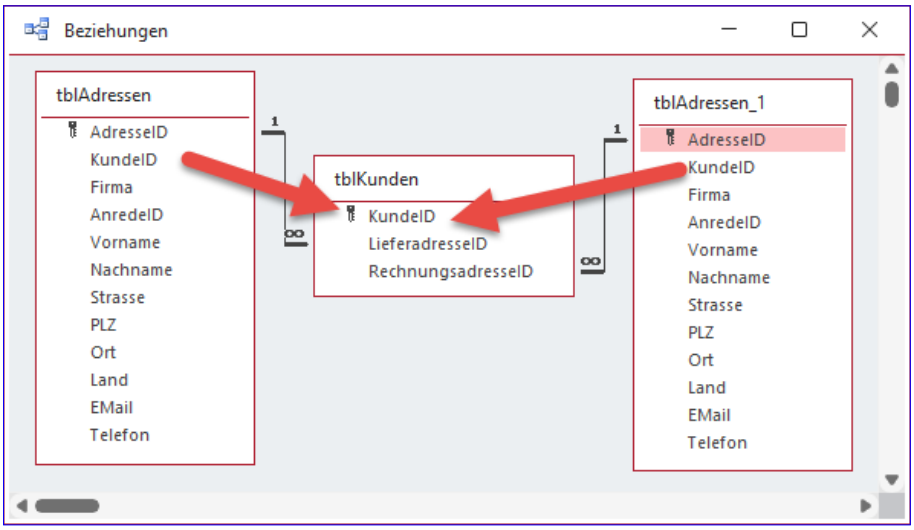


Bild 5: Das Fremdschlüsselfeld **KundeID** ermöglicht die Zuordnung der Adressdatensätze zu einem Kundendatensatz (V4).

- Die zweite Möglichkeit ist, die Zuweisung der Adressen gar nicht mehr über Fremdschlüsselfelder in der Kundentabelle zu erledigen. Stattdessen entfernen wir die Fremdschlüsselfelder wie in Bild 6 und stellen die Verknüpfung nur noch über das Fremdschüsselfeld **KundeID** der Adresstabelle her.

Das heißt, wir können keine, eine oder mehrere Tabellen zu jedem Kunden in der Adresstabelle speichern. Allerdings wissen wir bisher noch nicht, welche wir dann als Liefer- und Rechnungsadresse verwenden sollen. Dies regeln wir über zwei Ja/Nein-Felder namens **IstStandardLieferadresse** und **IstStandardRechnungsadresse**. Damit können wir zum Beispiel festlegen, das eine einzige Adresse als Liefer- und Rechnungsadresse fungiert. Der Nachteil ist allerdings, dass wir in Access per Code die Integrität der Daten sicherstellen müssen – also beispielsweise, dass für jeden Kunden immer mindestens eine Adresse gespeichert ist und das jeweils eine Adresse als Standardlieferadresse und eine als Standardrechnungsadresse markiert ist.

Da scheint es zumindest aus Sicht der Datenintegrität zuverlässiger, Variante **V4** zu wählen.

Eigene Adresstabellen in Bezug auf Bestellungen

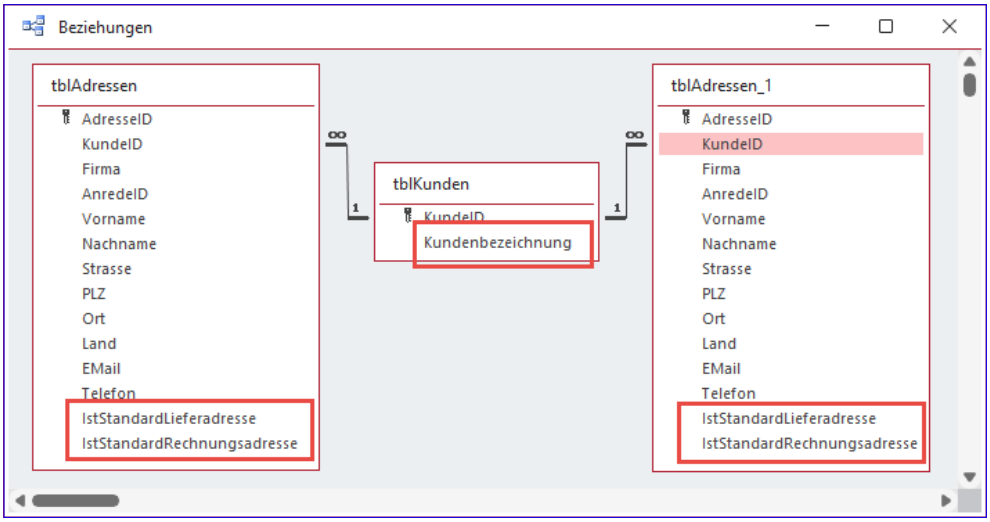


Bild 6: Wir können die Beziehung auch komplett umdrehen. Dann müssen wir angeben, welche Adresse die Standardliefer- und Standardrechnungsadresse ist (V5).

Wenn wir wie in **V3**, **V4** oder **V5** eine eigene Tabelle zum Speichern der Kundenadressen verwenden, können wir nach wie vor wie in **V2** die bei Bestellung gültige Liefer- und Rechnungsadresse in die Felder der Tabelle **tblBestellungen** übernehmen.

Wir könnten aber auch hier die Adressen in einer oder mehreren Tabellen speichern. Hier haben wir einen etwas einfacheren Fall, als

dass wir zu jeder Bestellung nur maximal eine Liefer- und eine Rechnungsadresse speichern müssen. Wir könnten also die Felder der Tabelle **tblBestellungen** wie in Bild 7 in weiteren Tabellen organisieren (siehe **GrundlagenDatenmodellRechnungsverwaltung_V6.accdb**).

Hier haben wir einmal einen anderen Ansatz gewählt als in den vorherigen Beispielen, wo wir dem Kunden die Adressen auf verschiedene Arten zugeordnet haben.

In diesem Fall verwenden wir nicht eine Tabelle für alle Adressen, sondern wir haben einmal zwei Tabellen genutzt – eine für die Liefer- und eine für die Rechnungsadressen. Diese haben wir **tblBestellungenLieferadressen** und **tblBestellungenRechnungsadressen** genannt. Beide sind jeweils über das Fremdschlüsselfeld **BestellungID** mit der jeweiligen Bestellung aus der Tabelle **tblBestellungen** verknüpft.

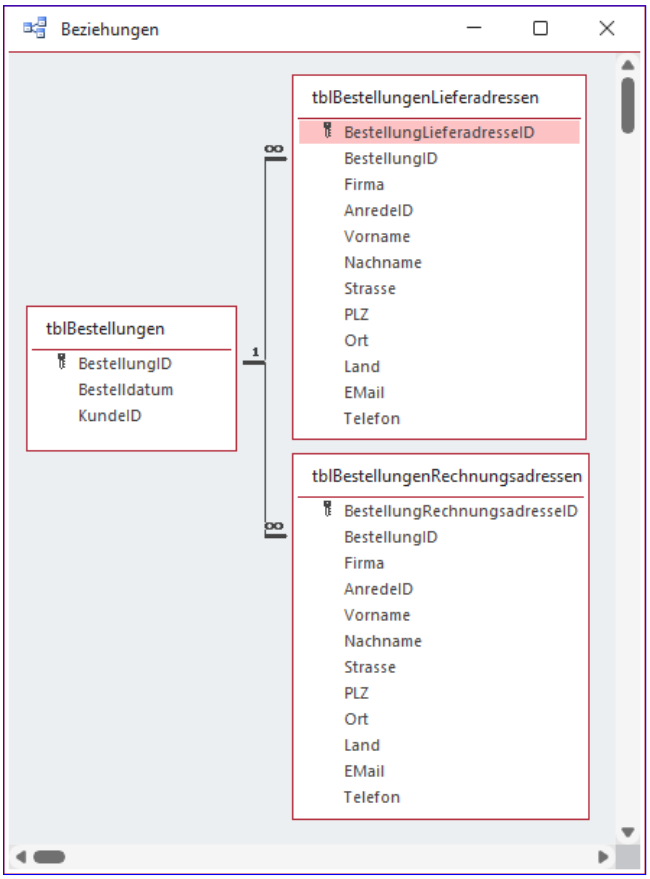


Bild 7: Auch beiden Bestellungen könnte man die Adressen wie hier in eigenen Tabellen speichern – hier einmal mit jeweils einer Tabelle für Liefer- und Rechnungsadressen (V6).

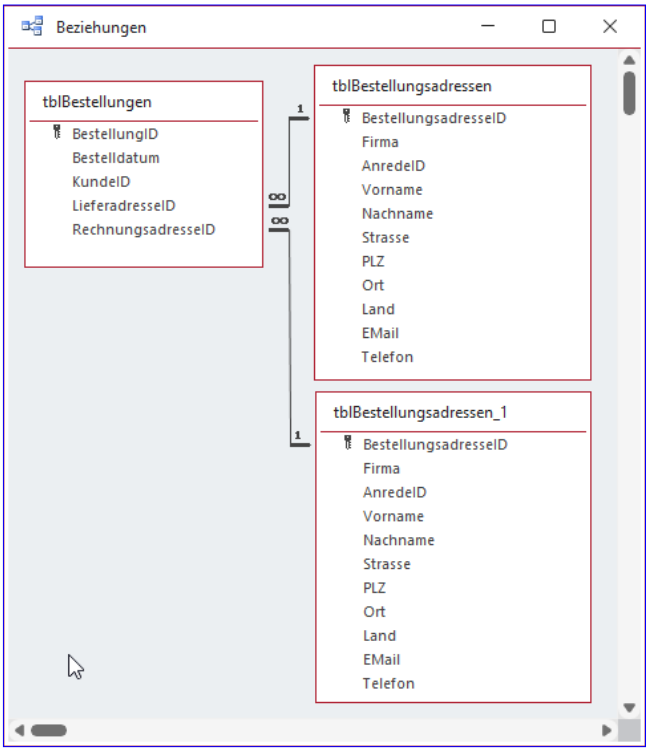


Bild 8: Man könnte auch nur eine Tabelle mit Bestelladressen verwenden, die dann mit zwei Fremdschlüsselfeldern der Tabelle **tblBestellungen** verknüpft wird (V7).

Wir könnten auch hier mit nur einer Adresstabelle arbeiten. Dazu müssten wir dann wieder zwei Fremdschlüsselfelder zur Tabelle **tblBestellungen** hinzufügen, mit denen wir jeweils die Lieferadresse und die Rechnungsadresse für die jeweilige Bestellung hinterlegen können.

Im Gegensatz zur Version **V6** können wir hier, wenn die Liefer- und die Rechnungsadresse gleich sind, einen Datensatz einsparen.

In **V6** benötigen wir immer zwei Datensätze, je einen in der Tabelle **tblBestellungenLieferadressen** und einen in der Tabelle **tblBestellungenRechnungsadressen**. Die Version mit nur einer Adresstabelle ist in Bild 8 dargestellt (siehe **GrundlagenDatenmodellRechnungsverwaltung_V7.accdb**).

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP

Benutzerdefinierte Standardwerte

Standardwerte für Felder legt in der Regel der Entwickler beim Definieren des Datenmodells in der Entwurfsansicht der Tabellen fest. Während man in bestimmten Maßen dynamische Werte nutzen kann wie beispielsweise mit der Datum()-Funktion für Datumsfelder, ist man ansonsten recht unflexibel. Leider können nur einige eingebaute Funktionen genutzt werden, benutzerdefinierte VBA-Funktionen sind nicht möglich. Aber welche benutzerdefinierten Standardwerte wollen wir überhaupt nutzen – und wo wollen wir diese festlegen? Die Einsatzzwecke sind vielfältig und prinzipiell können wir jedes Feld mit einem benutzerdefinierten Standardwert nutzen. Wichtig ist der Gedanke dahinter: Sinnvoll gewählte Standardwerte sparen dem Benutzer wertvolle Zeit bei der Dateneingabe. Wenn wir zum Beispiel eine Datenbank planen, die überwiegend weibliche Kontakte verwaltet, ist es sinnvoll, als Anrede »Frau« vorzugeben. Oder wir haben eine Autowerkstatt, die hauptsächlich Fahrzeuge einer bestimmten Marke repariert – dann macht es Sinn, diese vorzubelegen. Wir können auch dynamisch den zuletzt verwendeten Eintrag als Standardwert angeben oder auch den meistgenutzen. Wie wir dies realisieren, zeigen wir im vorliegenden Beitrag.

Herkömmliche Standardwerte

Das Festlegen von Standardwerten ist recht einfach – man wechselt in den Entwurf der betroffenen Tabelle, klickt das Feld an, dem man einen Standardwert zuweisen will und stellt die Eigenschaft **Standardwert** auf den gewünschten Wert ein.

Bei Zahlenfeldern legt man so vielleicht den Wert **0** fest, bei Datumsfeldern mit dem Ausdruck **=Datum()** das aktuelle Datum oder bei **Ja/Nein**-Feldern den Wert **Ja**. Das ist schnell gemacht, aber relativ unflexibel – es gibt ein paar eingebaute Funktionen, die man hier nutzen kann, aber ansonsten ist man auf statische Werte angewiesen.

Dynamische Standardwerte

Wollen wir dies flexibel erledigen, haben wir im Tabellen- und Abfrageentwurf keine Möglichkeiten. Die nächste Ebene sind die Formulare: Hier können wir für Steuerelemente, die an Tabellenfelder gebunden sind, per VBA einen Standardwert definieren. Aber welche Werte wollen wir überhaupt als Standardwerte nutzen? Es gibt zum Beispiel die folgenden Möglichkeiten:

- Den beim zuletzt angelegten oder geänderten Datensatz verwendeten Wert,
- den Wert, der am meisten verwendet wurde oder
- Werte, die durch den Benutzer als Standardwert vordefiniert wurden.

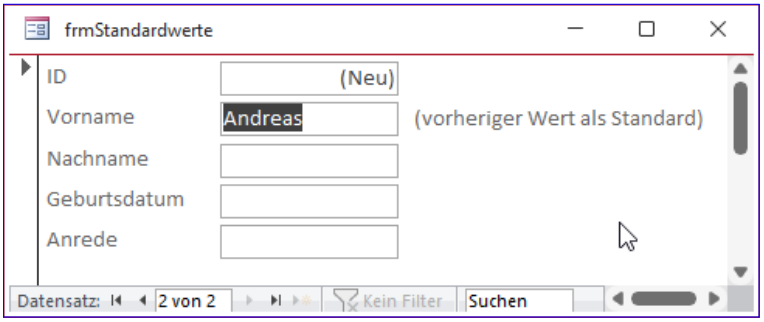


Bild 1: Dieser Wert wurde soeben als Standardwert übernommen

Die ersten beiden Vorschläge erfordern keine aktive Mitarbeit des Benutzers, der letzte hingegen schon.

Zuletzt verwendeten Wert als Standardwert nutzen

Wenn wir wie in Bild 1 den Namen **Andreas** in das Textfeld **txtVorname** eingeben und wollen, dass dieser fortan als Standardwert verwendet wird, benötigen wir eine VBA-Prozedur, die entweder nach dem Aktualisieren des Steuerelements oder des Formulars ausgelöst wird und die Eigenschaft **DefaultValue** des Textfeldes auf den soeben eingegebenen Wert einstellt.

In diesem Fall wollen wir dies direkt nach dem Eingeben des Wertes und dem Aktualisieren des Steuerelements erledigen. Dazu hinterlegen wir die folgende Ereignisprozedur:

```
Private Sub txtVorname_AfterUpdate()
    Me!txtVorname.DefaultValue = _
        Chr(34) & Me!txtVorname & Chr(34)
End Sub
```

Wir weisen hier allerdings nicht einfach den Wert des Textfeldes zu, sondern müssen diesen noch in Anführungszeichen einfassen, die wir wie hier durch die entsprechende **Chr**-Funktion angeben oder alternativ beispielsweise mit vier Anführungszeichen:

```
Me!txtVorname.DefaultValue = """" & Me!txtVorname & """"
```

Auch wenn wir eine feste Zeichenkette angeben wollen, müssen wir diese übrigens in zusätzliche Anführungszeichen einfassen:

```
Me!txtVorname.DefaultValue = """Beispiel"""
```

Standardwert für weitere Datentypen

Bei Datumsangaben müssen wir das Datum zuvor in einen Zahlenwert konvertieren. Bei reinen Datumsangaben ohne Uhrzeit reicht dies aus:

```
Private Sub txtGeburtsdatum_AfterUpdate()
    Me!txtGeburtsdatum.DefaultValue = CLng(Me!txtGeburts-
datum)
End Sub
```

Wenn wir ein Feld verwenden, das auch eine Uhrzeit enthält, benötigen wir die Funktion **CDbl** – die Nachkommastellen nehmen in diesem Fall die Uhrzeit auf:

```
Me!txtGeburtsdatum.DefaultValue = CDbl(Me!txtGeburtsdatum)
```

Bei **Ja/Nein**-Feldern greifen wir direkt auf den jeweiligen Wert zu:

```
Private Sub chkAktiv_AfterUpdate()
    Me!chkAktiv.DefaultValue = Me!chkAktiv
End Sub
```

Bei Zahlenwerten und Währungsfeldern müssen wir das Komma durch den Punkt ersetzen:

```
Private  Sub txtTaschengeld_AfterUpdate()
    Me!txtTaschengeld.DefaultValue = _
        Replace(Me!txtTaschengeld, ",", ".")
End Sub

Private Sub txtGewicht_AfterUpdate()
    Me!txtGewicht.DefaultValue = _
        Replace(Me!txtGewicht, ",", ".")
End Sub
```

Nachteil: Kein Speichern des Standardwerts

Der Nachteil dieser Methode ist, dass der zuletzt verwendete Wert erst als Standardwert genutzt wird, wenn wir den Inhalt des Steuerelements nach dem Öffnen des Formulars erstmals bearbeitet haben.

Wir müssten uns also eine Möglichkeit überlegen, diesen Wert alternativ zu speichern und ihn wiederherzustellen. Dazu kommen wir weiter unten – erst schauen wir uns die anderen Varianten an.

```
Private Sub NachnameStandardwert()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Dim strNachname As String
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT TOP 1 Nachname FROM tblPersonen GROUP BY Nachname ORDER BY COUNT(*) DESC")
    If Not rst.EOF Then
        strNachname = rst.Fields(0)
    End If
    Me!txtNachname.DefaultValue = """" & strNachname & """"
End Sub
```

Listing 1: Einstellen des häufigsten Nachnamens als Standardwert

Am meisten verwendeten Wert als Standardwert nutzen

Wenn wir schon eine VBA-Prozedur verwenden, können wir auch gleich noch andere Methoden zur Ermittlung des Standardwertes nutzen. So können wir beispielsweise ermitteln, welcher Wert in einem Feld am meisten verwendet wurde und diesen als Standardwert in das Feld eintragen.

Dazu verwenden wir die Prozedur aus Listing 1. Diese ermittelt in einem Recordset den Wert des Feldes **Nachname**, der den höchsten Wert für die Anzahl der Werte der Gruppierungen über das Feld **Nachname** enthält. Sprich: Die Abfrage gruppiert die Datensätze nach dem Wert im Feld **Nachname**, zählt diese und verwendet die Anzahl gleich noch als Sortierkriterium.

Davon holt sie mit **TOP 1** den ersten Eintrag. Danach prüft sie, ob die Abfrage einen Datensatz gefunden hat und stellt den Wert der Variablen **strNachname** auf den gefundenen Nachnamen ein. Dieser wird dann als Standardwert des Feldes **txtNachname** festgelegt.

Damit wir sowohl beim Öffnen des Formulars als auch nach dem Aktualisieren eines der Nachnamen jeweils den korrekten Standardwert für dieses Feld angeben können, rufen wir diese Prozedur gleich zwei Mal auf – zuerst in der Prozedur, die durch das Ereignis **Beim Öffnen** des Formulars ausgelöst wird:

```
Private Sub Form_Open(Cancel As Integer)
    Call NachnameStandardwert
End Sub
```

Zweitens nach dem Ändern des Wertes für einen Datensatz im Feld **Nachname**:

```
Private Sub txtNachname_AfterUpdate()
    Call NachnameStandardwert
End Sub
```

Standardwert beim Öffnen einstellen

Den Standardwert können wir also auch gleich beim Öffnen des Formulars zu setzen – und zwar in den Prozeduren für die Ereigniseigenschaften **Beim Laden** und **Beim Öffnen**.

Wert aus dem zuletzt hinzugefügten Datensatz ermitteln

Nun fehlt noch eine Möglichkeit, diese Standardwerte zu speichern, denn wie wir oben beschrieben haben, sind diese nach dem erneuten Öffnen des Formulars wieder geleert. Wir haben zwar gesehen, dass wir in den Ereignisprozeduren **Form_Load** oder **Form_Open** die Werte neu einstellen können. Hier können wir aber beispielsweise nicht den zuletzt verwendeten Wert einstellen, da dieser nicht gespeichert wird. Dies würde höchstes gelingen, wenn wir die Anforderung so definieren, dass der Wert des zuletzt angelegten Datensatzes als Standardwert verwen-

det wird. Dann würden wir beispielsweise diese Prozedur nutzen, die durch das Ereignis **Beim Anzeigen** ausgelöst wird – also beim Wechseln zu einem anderen Datensatz und somit auch zu einem neuen, leeren Datensatz:

```
Private Sub Form_Current()
    Dim strAnrede As String
    strAnrede = Nz(DLookup("Anrede", "tblPersonen", _
        "ID = " & Nz(DMax("ID", "tblPersonen"), 0)), "")
    Me!txtAnrede.DefaultValue = """" & strAnrede & """"
End Sub
```

Hier ermitteln wir mit **DMax** die höchste Zahl im Feld **ID**, was in der Regel dem zuletzt angelegten Datensatz entspricht. Diese verwenden wir als Parameter für einen Aufruf der **DLookup**-Funktion, um die passende Anrede dazu zu finden. Das Ergebnis schreiben wir anschließend als Standardwert in das Textfeld **txtAnrede**.

Standardwerte speichern

Damit kommen wir endgültig zum Speichern der Standardwerte. Wenn dies gewünscht ist, benötigen wir einen Ort, wo wir die Werte speichern können.

Da wir ohnehin in einer Access-Datenbank arbeiten, legen wir dazu logischerweise eine eigene Tabelle an, die wir beispielsweise **tblStandardwerte** nennen. Hier wollen wir zunächst die folgenden Informationen speichern:

- Tabellenname
- Feldname

Danach folgt der eigentlich interessante Wert, nämlich der Standardwert. Hier stellt sich die Frage, ob wir für jeden Datentyp ein eigenes Feld anlegen oder einfach alle Werte in ein Feld mit dem Datentyp speichern, der alles speichern kann – nämlich in einem Textfeld.

Darüber hinaus ließen sich je nach Einsatzzweck noch weitere Felder einfügen:

- **BenutzerID**: Für den Fall, dass mehrere Benutzer mit der Anwendung arbeiten und jeder seine eigenen Standardwerte speichern möchte.
- **Geloescht**: Ermöglicht es, Datensätze als gelöscht zu markieren, die man gegebenenfalls nochmal wiederherstellen möchte.
- **Aktiv**: Erlaubt das Deaktivieren von Datensätzen für Standardwerte.
- **Gruppe**: Gegebenenfalls möchte man die Standardwerte nach Gruppen sortiert darstellen – das würde diese Einstellung ermöglichen.
- **Datentyp**: Erlaubt es, je nach Datentyp gezielt auf den gespeicherten Wert zuzugreifen. Noch wichtiger: Wenn wir ein eigenes Formular zur Verwaltung der Standardwerte anbieten, sollten wir validieren können, ob der Wert gültig ist.

Mehrbenutzer

Bevor wir an die Umsetzung gehen, schauen wir uns die Möglichkeiten für Mehrbenutzer an. Hier gibt es zwei Varianten:

- Die Tabelle mit den Standardwerten ist im Backend gespeichert. In diesem Fall müssten wir ein Feld wie **BenutzerID** oder **MitarbeiterID** verwenden, um festlegen zu können, welchem Benutzer oder Mitarbeiter der Standardwert gehört.
- Die Tabelle mit den Standardwerten ist im Frontend gespeichert und jeder Benutzer hat sein Frontend in seinen Benutzerdokumenten, auf die er nach der Anmeldung zugreifen kann. Dann können wir auf ein Feld wie **BenutzerID** oder **MitarbeiterID** verzichten.

Wir gehen der Einfachheit halber davon aus, dass wir entweder eine Einzelbenutzer-Anwendung haben oder dass jeder Benutzer eine Kopie des Frontends mit der Tabelle

tblStandardwerte in seinem Benutzerverzeichnis hat.

Tabellen zum Verwalten der Standardwerte

Die Benutzeroberfläche gestalten wir je nach den möglichen Elementen des Datenmodells, die wir oben vorgestellt haben. Um den Rahmen nicht zu sprengen, wollen wir die folgenden Daten darstellen:

- Tabellenname
- Feldname
- Standardwert
- Gruppe
- Datentyp

Die Tabelle **tblStandardwerte** sieht in der Entwurfsansicht schließlich wie in Bild 2 aus. Für die Kombination der beiden Felder **Tabellenname** und **Feldname** haben wir einen eindeutigen Schlüssel definiert, damit kein Feldname doppelt zu einer Tabelle zugeordnet werden kann.

Für das Feld **DatentypID** haben wir ein Nachschlagefeld eingerichtet, das mit der Tabelle **tblDatentypen** verknüpft ist. Auf diese Weise brauchen wir dort nur die dem Datentyp entsprechende Zahl zu speichern. Die damit verknüpfte Tabelle heißt **tblDatentypen** und enthält im Feld **ID** den Zahlenwert, den wir anschließend per VBA für das jeweilige Feld ermitteln, im Feld **Datentyp** die VBA-Konstante für den Datentyp sowie eine **Beschreibung** (siehe Bild 3).

Entwurf der Benutzeroberfläche

Um die Standardwerte zu verwalten, erstellen wir zwei Formulare. Das Unterformular **sfmStandardwerte** soll die Daten der Tabelle **tblStandardwerte** in der Datenblatt-

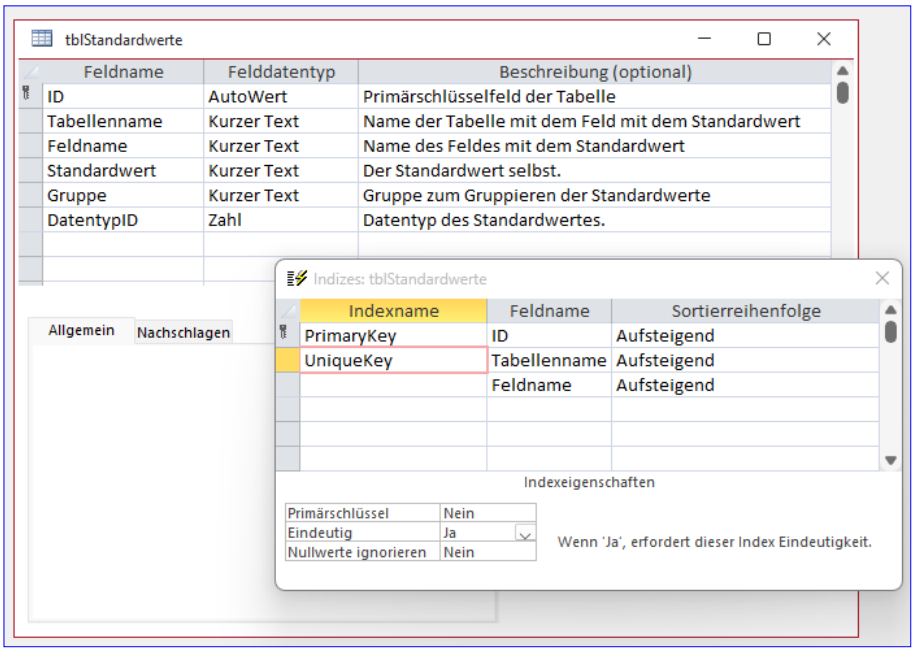


Bild 2: Festlegen eines eindeutigen, zusammengesetzten Indexes

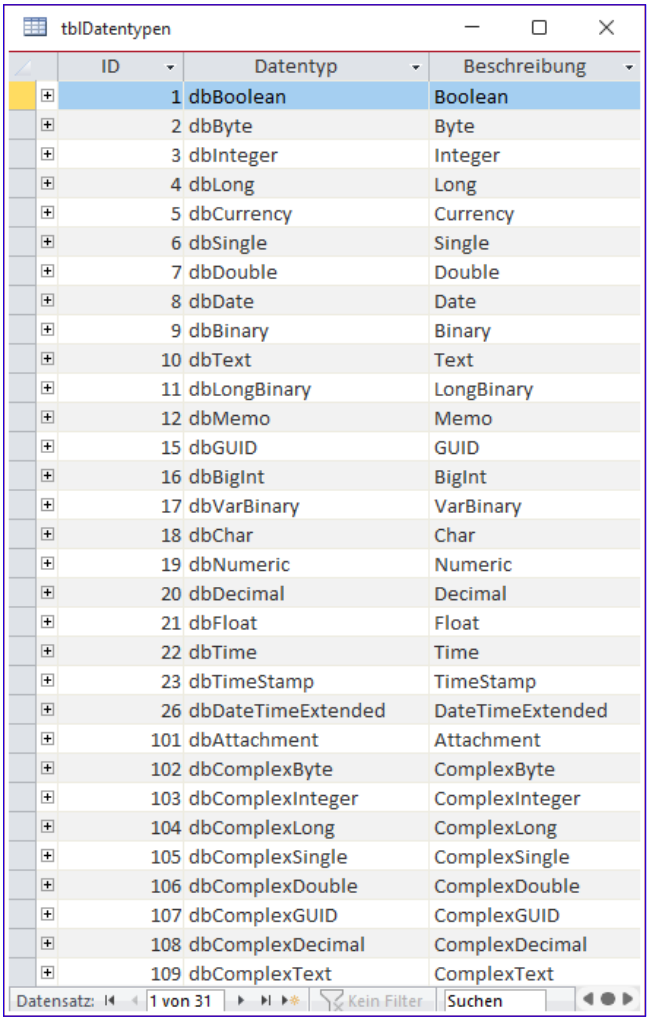


Bild 3: Datentypen für die Standardwerte

ansicht anzeigen. Dazu weisen wir dem Formular diese Tabelle für die Eigenschaft **Datensatzquelle** zu und ziehen die drei Felder **Tabellenname**, **Feldname** und **Standardwert** in den Detailbereich des Entwurfs.

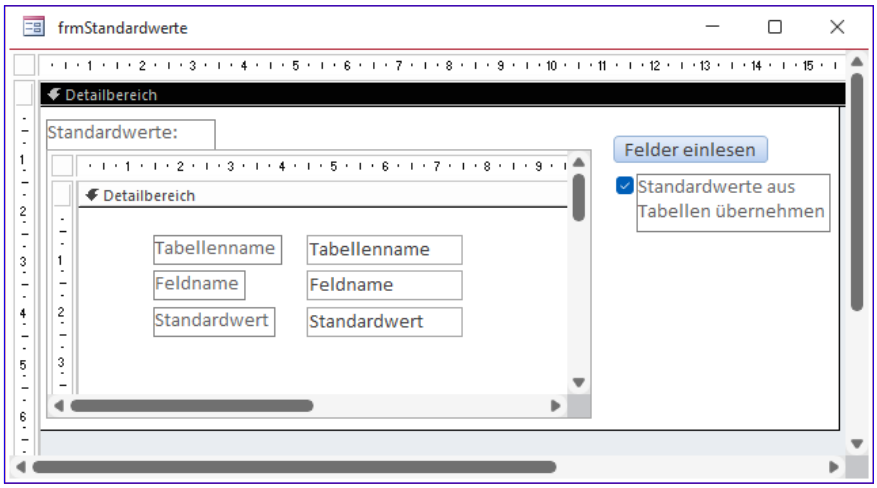


Bild 4: Entwurf von Haupt- und Unterformular

Anschließend speichern und schließen wir das Unterformular und ziehen es aus dem Navigationsbereich in den Entwurf des Hauptformulars **frmStandardwerte**. Hier fügen wir noch eine Schaltfläche hinzu, mit der wir die Felder der Tabellen der aktuellen Datenbank in die Tabelle **tblStandardwerte** einlesen können.

Außerdem wollen wir mit dem Kontrollkästchen **chkStandardwerteUebernehmen** festlegen, ob die Standardwerte aus den Felddefinitionen übernommen werden sollen, was nur beim ersten Einlesen empfehlenswert ist (siehe Bild 4).

Felder und Standardwerte einlesen

Die Schaltfläche **cmdFelderEinlesen** löst die folgende Prozedur aus. Diese ruft zwei weitere Routinen auf, von denen die erste die im Datenmodell vorhandenen Felder einliest und die zweite prüft, ob alle in der Tabelle **tblStandardwerte** angegebenen Felder noch vorhanden sind. Danach aktualisiert sie das Unterformular **sfmStandardwerte** mit der **Requery**-Methode:

```
Private Sub cmdFelderEinlesen_Click()
    Call FelderEinlesen(Me!chkStandardwerteUebernehmen)
    Call FelderEntfernen
    Me!sfmStandardwerte.Requery
End Sub
```

Die Prozedur **FelderEinlesen** nimmt den Wert des Kontrollkästchens **chkStandardwerteUebernehmen** als optionalen Parameter entgegen, der standardmäßig auf **False** eingestellt ist (siehe Listing 2).

Sie durchläuft in einer **For Each**-Schleife alle Tabellendefinitionen der aktuellen Datenbank und prüft in einer **Select Case**-Bedingung, ob der Tabellenname bestimmten Namen wie **MSys***, **USys***, **tblStandardwerte** oder **tblDatentypen** entspricht. Diese Tabellen sollen beim Einlesen nicht berücksichtigt werden.

Für alle anderen durchläuft die Prozedur die **Fields**-Auflistung mit allen Feldern. Nach der Deaktivierung der eingebauten Fehlerbehandlung prüft die Prozedur, ob die Standardwerte übernommen werden sollen. Falls ja, versucht sie, für die aktuelle Kombination aus Tabellenname und Feldname einen neuen Datensatz inklusive Standardwert und Felddatentyp in der Tabelle **tblStandardwerte** anzulegen. Das könnte einen Fehler auslösen, wenn diese Kombination bereits vorhanden ist. In diesem Fall wird in der folgenden **Select Case**-Bedingung geprüft, ob die Fehlernummer **3022** lautet. Dieser Fehler wird immer ausgelöst, wenn man versucht, einen Datensatz anzulegen, dessen Kombination aus Schlüsselwerten bereits in einem anderen Datensatz vorliegt. In diesem Fall wollen wir die Werte für diesen Datensatz überschreiben, was wir mit einer UPDATE-SQL-Anweisung erledigen, die genau die Daten für den Datensatz mit dem Tabellennamen und dem Feldnamen aktualisiert. Die Aktualisierung bezieht sich auf den Felddatentyp und gegebenenfalls auf den im Tabellenentwurf gespeicherten Standardwert. Auch haben wir zwei

Varianten – eine mit Übernahme der Standardwerte aus dem Tabellenentwurf und eine ohne.

Wenn wir diese Prozedur ausführen, erhalten wir bereits die gewünschten Daten im Formular (siehe Bild 5).

```
Public Sub FelderEinlesen(Optional bolStandardwerteUebernehmen As Boolean = False)
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    Dim fld As DAO.Field
    Set db = CurrentDb
    For Each tdf In db.TableDefs
        Select Case True
            Case tdf.Name Like "MSys*"
            Case tdf.Name Like "USys*"
            Case tdf.Name = "tblStandardwerte"
            Case tdf.Name = "tblDatentypen"
            Case Else
                For Each fld In tdf.Fields
                    On Error Resume Next
                    If bolStandardwerteUebernehmen Then
                        db.Execute "INSERT INTO tblStandardwerte(Tabellenname, Feldname, Standardwert, DatentypID) " _
                            & "VALUES('" & tdf.Name & "', '" & fld.Name & "', '" & fld.DefaultValue & "', " _
                            & fld.Type & ")", dbFailOnError
                    Else
                        db.Execute "INSERT INTO tblStandardwerte(Tabellenname, Feldname, DatentypID) VALUES('" _
                            & tdf.Name & "', '" & fld.Name & "', " & fld.Type & ")", dbFailOnError
                    End If
                    Select Case Err.Number
                        Case 3022
                            If bolStandardwerteUebernehmen Then
                                db.Execute "UPDATE tblStandardwerte(Tabellenname, Feldname, Standardwert, " _
                                    & "DatentypID) VALUES('" & tdf.Name & "', '" & fld.Name & "', '" _
                                    & fld.DefaultValue & "', " & fld.Type & ") WHERE Tabellenname = '" & tdf.Name _
                                    & "' AND Feldname = '" & fld.Name & "'", dbFailOnError
                            Else
                                db.Execute "UPDATE tblStandardwerte(Tabellenname, Feldname, DatentypID) VALUES('" _
                                    & tdf.Name & "', '" & fld.Name & "', " & fld.Type & ") WHERE Tabellenname = '" _
                                    & tdf.Name & "' AND Feldname = '" & fld.Name & "'", dbFailOnError
                            End If
                        Case 0
                        Case Else
                            MsgBox "Fehler " & Err.Number & ":" & vbCrLf & vbCrLf & Err.Description
                    End Select
                Next fld
        End Select
    Next tdf
End Sub
```

Listing 2: Speichern der Felder in der Tabelle **tblStandardwerte**

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP

1:n-Beziehung mit Standardzuordnung verwalten

Im Beitrag »Rechnungsverwaltung: Kundenadressen ausgliedern« (www.access-im-unternehmen.de/1470) verwalten wir zu jedem Kunden mehrere Adressen, die dem Kunden über ein Fremdschlüsselfeld in der Adresstabelle zugewiesen sind. Für jeden Kunden legen wir in dieser Tabelle eine Standardadresse fest. Damit ist einiger Pflegeaufwand verbunden, denn wir müssen sicherstellen, dass immer genau eine Adresse je Kunde als Standardadresse markiert ist. In diesem Beitrag schauen wir uns an, wie wir sicherstellen können, dass die Daten immer konsistent sind und wir nicht plötzlich mehrere Standardadressen je Kunde vorfinden – oder gar keine.

Elemente der hier behandelten Konstellation

Die Konstellation, die wir in diesem Beitrag vorstellen, enthält folgende Elemente:

- Hauptentität: Die Hauptentität repräsentiert den übergeordneten Datensatz oder die Hauptkomponente, zu der die Beziehung besteht. Im einführenden Beispiel ist die Hauptentität der Kunde.
- Unterentitäten: Die Unterentitäten sind mit der Hauptentität verknüpft und repräsentieren zusätzliche Informationen oder Komponenten, die zu jeder Hauptentität gehören. Im Beispiel sind die Unterentitäten die Adressen.
- Fremdschlüssel: Um die Beziehung zwischen der Hauptentität und den Unterentitäten herzustellen, wird ein Fremdschlüsselfeld in den Unterentitäten verwendet, das auf die Hauptentität verweist. In unserem Beispiel ist dies das **KundeID**-Feld in der Adresstabelle.
- Standardentität: Es gibt eine Möglichkeit, eine der Unterentitäten als Standard oder Hauptentität für jede Hauptentität festzulegen. Dies wird durch ein zusätzliches Feld wie **Standardadresse** erreicht.

Weitere Beispiele für diese Konstellation

Neben dem in der Einleitung vorgestellten Beispiel mit den Kunden und den zugeordneten Adressen (siehe Bild 1) gibt es noch weitere Beispiele:

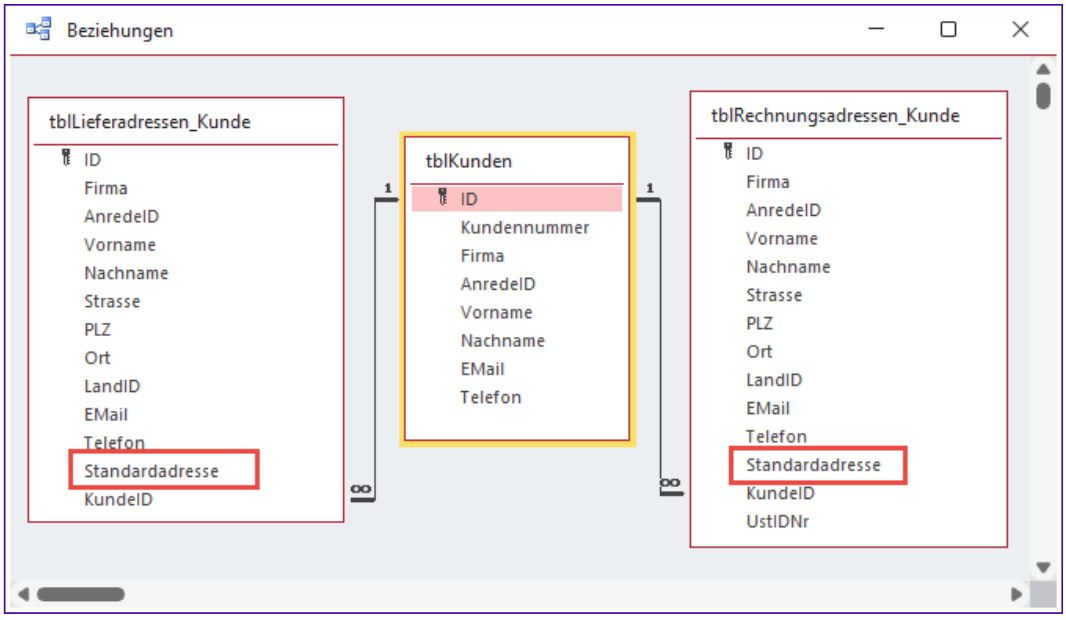


Bild 1: Beispiel für eine 1:n-Beziehung mit Standardzuordnung

- Personen und verschiedene Telefonnummern oder E-Mail-Adressen, die in einer separaten Tabelle gespeichert werden und von denen eine als Standard-E-Mail-Adresse oder Standardtelefonnummer markiert wird

- Produkte und verschiedene Lieferanten, von denen einer als Standardlieferant festgelegt wird

- Produkte und Variante: Für jedes Produkt gibt es verschiedene Varianten, von denen eine als Standardvariante vorgesehen wird.

Pflege von 1:n-Beziehungen mit Standardzuordnung

Übliche 1:n-Beziehungen, deren Pflege durch die Definition der Beziehung und der referenziellen Integrität mit oder ohne Löschweitergabe oder Aktualisierungsweitergabe geregelt ist, benötigen keinen zusätzlichen Aufwand – man legt einen Datensatz in der Detailtabelle an, beispielsweise **tblAnreden**, und wenn man einen Datensatz in der Mastertabelle wie **tblKunden** hinzufügt, wählt man einen der Datensätze aus der Detailtabelle aus.

Bei unserem Beispiel ist es im Prinzip genauso: Wir wählen für jeden Datensatz der Tabelle **tblAdressen** über das Fremdschlüsselfeld **KundeID** einen Datensatz aus der Tabelle **tblKunden** aus.

Nun wollen wir jedoch angeben, welche der Adressen aus der Tabelle **tblAdressen** die Standardadresse ist, die zum Beispiel beim Anlegen neuer Rechnungen oder Bestellungen auf Basis dieses Kunden herangezogen werden soll (siehe Bild 2).

Wenn wir einem Kunden eine erste Adresse hinzufügen, soll diese direkt als Standardadresse markiert werden.

Das allein ist schon nicht einfach: Wir können nicht einfach den Standardwert des Feldes **Standardadresse** auf **Wahr** einstellen, denn wenn der Benutzer anschließend eine neue Adresse für diesen Kunden hinzufügt, würde das Feld **Standardadresse** für diesen Kunden ebenfalls auf **Wahr** eingestellt werden und wir hätten bereits zwei als Standardadresse markierte Adressen für diesen Kunden. Das Einstellen des Feldes **Standardadresse** auf den Wert **Wahr** entfällt also bereits.

Daraus lassen sich bereits erste Regeln für unser Vorhaben ableiten:

- Wenn eine neue Adresse angelegt wird und dies ist die erste Adresse für diesen Kunden, soll das Feld **Standardadresse** den Wert **Wahr** erhalten.

- Wenn eine neue Adresse angelegt wird und es ist nicht die erste Adresse für diesen Kunden, müssen wir prüfen, ob es bereits eine Standardadresse gibt. Aber was dann – welche Adresse legen wir dann als Standardadresse fest? Wir schlagen vor, dass in diesem Fall

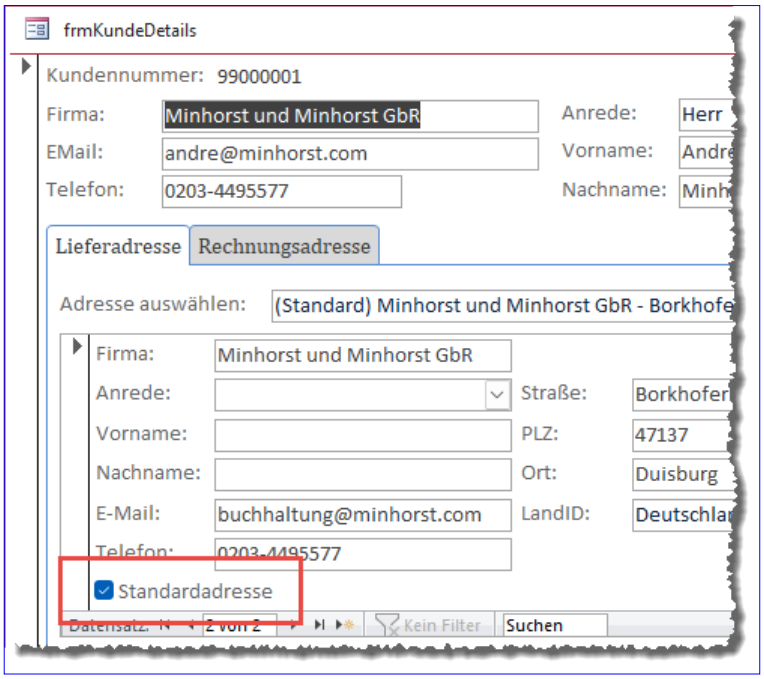


Bild 2: Die 1:n-Beziehung mit Standardzuordnung im Formular

die zuletzt hinzugefügte Adresse als Standardadresse festgelegt werden soll.

Davon abgesehen müssten wir eine Vorgehensweise etablieren, die beim Löschen einer der Adressen in Aktion tritt. Mit dieser müssen wir prüfen, ob einer der verbleibenden Adressdatensätze die Standardadresse für diesen Kunden ist und falls nicht, müssten wir eine der verbleibenden Adressen zur neuen Standardadresse machen.

Man könnte hier einfach die zuerst angelegte Adresse zur Standardadresse machen oder auch die zuletzt angelegte – oder auch eine ganz andere, beispielsweise eine, die besonders oft für Bestellungen oder Rechnungen verwendet wurde. Wir können auch den Benutzer fragen, welche Adresse die neue Standardadresse werden soll. Und wenn wir sie einfach festlegen, sollten wir dem Benutzer darüber informieren.

Entfernen des Status als Standardadresse

Der Benutzer könnte auf die Idee kommen, den Haken für das Feld **Standardadresse** zu entfernen, weil er diese Adresse nicht mehr als Standardadresse verwenden möchte. Das wollen wir verhindern – der Benutzer soll die aktuelle Standardadresse nur durch Setzen einer neuen Standardadresse entfernen können.

Setzen des Status als Standardadresse

Wenn der Benutzer eine neue Standardadresse einstellen möchte, soll das durch einfaches Aktivieren des Kontrollkästchens für das Feld **Standardadresse** möglich sein. Wenn dies geschieht, müssen wir allerdings dafür sorgen, dass das Feld **Standardadresse** für alle anderen Datensätze der Adresstabelle, die diesem Kunden zugeordnet sind, entfernt wird.

Ermitteln der aktuellen Standardadresse

Schließlich wird es Situationen geben, in denen wir, beispielsweise zum Steuern der Darstellung in Formularen, die aktuelle Standardadresse für einen Kunden ermitteln müssen.

VBA-Funktionen zum einfachen Umsetzen der Anforderungen

Damit wir die oben genannten Anforderungen einfach umsetzen können, wollen wir dazu einen flexibel einsetzbaren Satz an VBA-Funktionen bereitstellen, mit denen wir nicht nur die Standardadresse aus den Adressen eines Kunden, sondern auch die Standarddatensätze für andere Datenmodelle ermitteln können.

Welche Funktionen benötigen wir also dazu?

- Ermitteln des aktuellen Standardwertes. Wenn diese Funktion den Wert **0** zurückliefert, wenn also kein Standardwert vorhanden ist, haben wir gleichzeitig bereits eine Funktion, um zu prüfen, ob ein Standardwert vorhanden ist.

- Setzen des aktuellen Standardwertes. Setzt voraus, dass zuvor bisherige Standardwerte entfernt wurden.

- Entfernen bisheriger Standardwerte. Setzt voraus, dass anschließend ein neuer Standardwert gesetzt wird.

- Ermitteln des neuen Standardwertes, falls keiner festgelegt ist.

Diese Funktionen müssen wir nun noch so verknüpfen, dass sie sinnvoll in einer Anwendung einsetzbar sind. Außerdem müssen wir noch festlegen, zu welchen Zeitpunkten die Integrität dieser Daten geprüft werden soll – also beispielsweise, ob der aktuell angezeigte Kundendatensatz eine Standardadresse aufweist. Also schauen wir uns nun die Zeitpunkte an, zu denen wir auf die Daten zugreifen und diese ändern. Dies ist dann die Grundlage für die tatsächlich zu definierenden Funktionen zur Pflege der 1:n-Beziehung mit Standardzuordnung.

Zeitpunkte des Zugriffs auf die 1:n-Beziehung mit Standardzuordnung

Wann also greifen wir tatsächlich auf die Standardzuordnungen zu? Wir sehen die folgenden Zeitpunkte:

```
Public Function HoleStandardzuordnung(strTabelle As String, strStandardfeld As String, strPKFeld As String, _
        strFKFeld As String, lngFKID As Long) As Long
    Dim lngPKID As Long
    lngPKID = Nz(DLookup(strPKFeld, strTabelle, strFKFeld & " = " & lngFKID & " AND " & strStandardfeld & " = -1"), 0)
    HoleStandardzuordnung = lngPKID
End Function
```

Listing 1: Funktion zum Ermitteln der Standardzuordnung

- Öffnen eines Formulars, das die Daten des Masterdatensatzes und des Detaildatensatzes anzeigt: Hier sollte geprüft werden, ob eine Standardzuordnung besteht und gegebenenfalls eingerichtet werden.
- Wechsel zu einem anderen Datensatz im gleichen Formular
- Anlegen einer neuen Adresse für einen Kunden: Abfrage, ob eine Standardzuordnung besteht und gegebenenfalls einrichten
- Löschen einer Adresse eines Kunden: Abfragen, ob nach dem Löschen noch eine Standardzuordnung besteht und gegebenenfalls einrichten
- Ändern der Standardzuordnung durch den Benutzer: Löschen aller vorherigen Zuordnungen und anlegen der neuen Zuordnung

Funktionen zum Verwalten der Standardzuordnungen

Basierend auf diesen Anforderungen haben wir die folgenden Funktionen abgeleitet, die wir anschließend im Detail betrachten:

- **HoleStandardzuordnung**: Liest die ID des Datensatzes mit der Standardzuordnung für die angegebene Tabelle, Standardfeld, Fremdschlüsselfeld und Fremdschlüsselwert ein.
- **SetzeStandardzuordnung**: Setzt den Wert des mit **Standardfeld** übergebenen Feldes in der angegebenen Tabelle für den angegebenen Fremdschlüsselwert im Feld des Parameters **Fremdschluesselfeld** auf **True**.
- **EntferneStandardzuordnungen**: Stellt für alle Datensätze der mit **strTabelle** angegebenen Tabelle mit dem Wert aus **lngFKID** im Feld aus **strFKFeld** auf **False** ein.
- **HoleErstezuordnung**: Liest den Primärschlüsselwert des ersten Datensatzes der mit **strTabelle** angegebenen Tabelle ein, deren Feld aus **strFKFeld** den Wert aus **lngFKID** aufweist.
- **HoleOderSetzeStandardzuordnung**: Kombination aus mehreren der vorgenannten Funktionen, die versucht, die Standardadresse zu holen und falls dies nicht gelingt, eine neue Standardadresse zu setzen.
- **MehrfacheStandardzuordnungenEntfernen**: Entfernt für die mit **strTabelle** angegebene Tabelle alle doppelten Standardzuordnungen mit Ausnahme der Ersten.

Standardzuordnung holen

Die Funktion **HoleStandardzuordnung** ermittelt mit der **DLookup**-Funktion den Primärschlüsselwert des Datensatzes der Tabelle aus **strTabelle**, dessen Fremdschlüsselfeld den Fremdschlüsselwert aus **lngFKID** enthält, und gibt diesen als Funktionsergebnis zurück (siehe Listing 1).

Standardzuordnung setzen

Die Funktion **SetzeStandardzuordnung** aktualisiert die mit **strTabelle** übergebene Tabelle, indem Sie mit einer

```
Public Sub SetzeStandardzuordnung(strTabelle As String, strStandardfeld As String, strPKFeld As String, lngPKID As Long)
    Dim db As DAO.Database
    Set db = CurrentDb
    db.Execute "UPDATE " & strTabelle & " SET " & strStandardfeld & " = -1 WHERE " & strPKFeld & " = " & lngPKID, _
        dbFailOnError
End Sub
```

Listing 2: Funktion zum Setzen einer Standardzuordnung

```
Public Sub EntferneStandardzuordnungen(strTabelle As String, strStandardfeld As String, strFKFeld As String, _
        lngFKID As Long)
    Dim db As DAO.Database
    Set db = CurrentDb
    db.Execute "UPDATE " & strTabelle & " SET " & strStandardfeld & " = 0 WHERE " & strFKFeld & " = " & lngFKID, _
        dbFailOnError
End Sub
```

Listing 3: Funktion zum Entfernen von Standardzuordnungen

```
Public Function HoleErsteZuordnung(strTabelle As String, strPKFeld As String, strFKFeld As String, _
        lngFKID As Long) As Long
    HoleErsteZuordnung = Nz(DLookup(strPKFeld, strTabelle, strFKFeld & " = " & lngFKID), 0)
End Function
```

Listing 4: Funktion zum Ermitteln der ersten Standardzuordnung

UPDATE-Anweisung das Feld aus **strStandardfeld** für den Datensatz auf den Wert **-1** aktualisiert, der im Feld aus **strPKFeld** den Wert aus **lngPKID** aufweist (siehe Listing 2).

Standardzuordnungen entfernen

Die Funktion **EntferneStandardzuordnungen** stellt das Feld aus **strStandardfeld** mit einer **UPDATE**-Anweisung für alle Datensätze der Tabelle aus **strTabelle** auf den Wert **0** ein, deren Feld aus **strFKFeld** den Wert aus **lngFKID** aufweist (siehe Listing 3).

Erste Zuordnung holen

Die Funktion **HoleErsteZuordnung** verwendet den Aufruf der **DLookup**-Funktion, um den Wert des Feldes aus **strPKFeld** aus dem ersten Datensatz der Tabelle aus **strTabelle** zu holen, dessen Feld aus **strFKFeld** den Wert aus **lngFKID** enthält (siehe Listing 4).

Standardadresse holen oder setzen, falls nicht vorhanden

Die Funktion **HoleOderSetzeStandardzuordnung** verwendet einige der zuvor bereits beschriebenen Funktionen (siehe Listing 5). Als Erstes versucht sie mit der Funktion **HoleStandardzuordnung**, den Wert des Feldes **strPKFeld** aus der Tabelle aus **strTabelle** zu holen, die im Feld aus **strFKFeld** den Wert aus **lngFKID** enthält.

Liefert dies den Wert **0**, liest sie durch einen Aufruf der Funktion **HoleErsteStandardzuordnung** den Wert des Feldes aus **strPKFeld** des ersten Datensatzes aus der Tabelle aus **strTabelle** ein, dessen Feld aus **strFKFeld** den Wert aus **lngFKID** enthält.

Liefert dies ebenfalls den Wert **0**, ist noch kein zugeordneter Datensatz in der Tabelle aus **strTabelle** vorhanden. Dann liefert die Funktion den Wert **0** zurück. Anderenfalls

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP

Standardzuordnung per Datenmakro

Während wir den Beitrag »1:n-Beziehung mit Standardzuordnung verwalten« (www.access-im-unternehmen.de/1469) schrieben, haben wir uns gefragt, ob man die Einhaltung der Konsistenz bei 1:n-Beziehungen mit Standardzuordnung nicht auch auf andere Weise sicherstellen kann als per VBA-Code. Zumal wir diesen auch noch in allen Formularen erneut programmieren müssten, welche die Daten aus dieser 1:n-Beziehung darstellen. Uns fielen zwei Lösungsansätze ein: Der erste ist die Verwendung der mit Access 2010 eingeführten Datenmakros, also eine Art Trigger für Access-Datenbanken. Der zweite Ansatz wären die echten Trigger, und zwar die vom SQL Server. In diesem Beitrag schauen wir uns nun die Variante mit den Datenmakros an. Kann man unser Vorhaben damit abbilden? Und wenn ja, wie?

Die Datenmakros liefern gleich fünf verschiedene Ereignisse, zu denen wir Makrobefehle ausführen können:

- **Vor Änderung**
- **Vor Löschung**
- **Nach Einfügung**
- **Nach Aktualisierung**
- **Nach Löschung**

Warum es kein Ereignis namens **Vor Einfügung** gibt? Weil wir dazu das Ereignis **Vor Änderung** nutzen können. Aber welche dieser Ereignisse benötigen wir, um die im oben genannten Beitrag abgebildeten Regeln zur Einhaltung der Konsistenz sicherzustellen?

Noch mal zur Wiederholung: Wir wollen in einer 1:n-Beziehung, beispielsweise zwischen den beiden Tabellen **tblPersonen** und **tblMailAdressen** aus Bild 1, Folgendes sicherstellen: Genau einer der Datensätze aus **tblEMailAdressen**, der mit einem der Datensätze aus **tblPersonen** verbunden ist, muss für das Feld **StandardEMail** den Wert **True** aufweisen. Dazu ist zu berücksichtigen:

- Wenn der erste Datensatz zu einer Person angelegt wird, muss dieser als Standard-E-Mail gekennzeichnet werden.
- Wenn ein Datensatz als **StandardEMail** gekennzeichnet wird, muss die Markierung beim bisher als Standard gekennzeichneten Datensatz entfernt werden.
- Wenn der aktuelle Standarddatensatz entfernt wird, muss ein anderer markiert werden, sofern noch einer vorhanden ist.

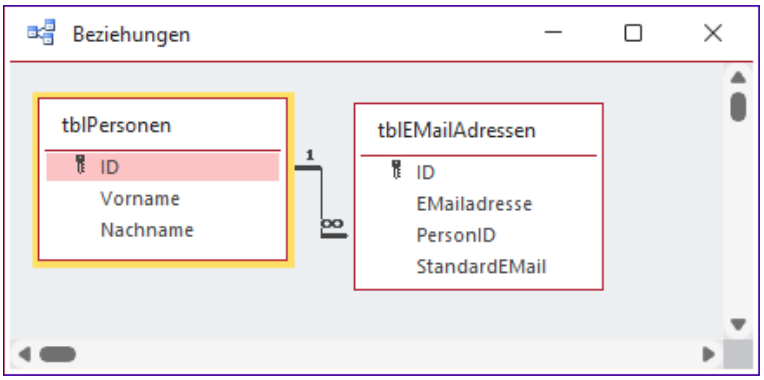


Bild 1: 1:n-Beziehung mit Standardzuordnung

- Wenn der Benutzer versucht, den aktuellen Datensatz als Standard abzuwählen, muss dies verhindert werden. Dies kann nur durch die Auswahl eines anderen Datensatzes geschehen.

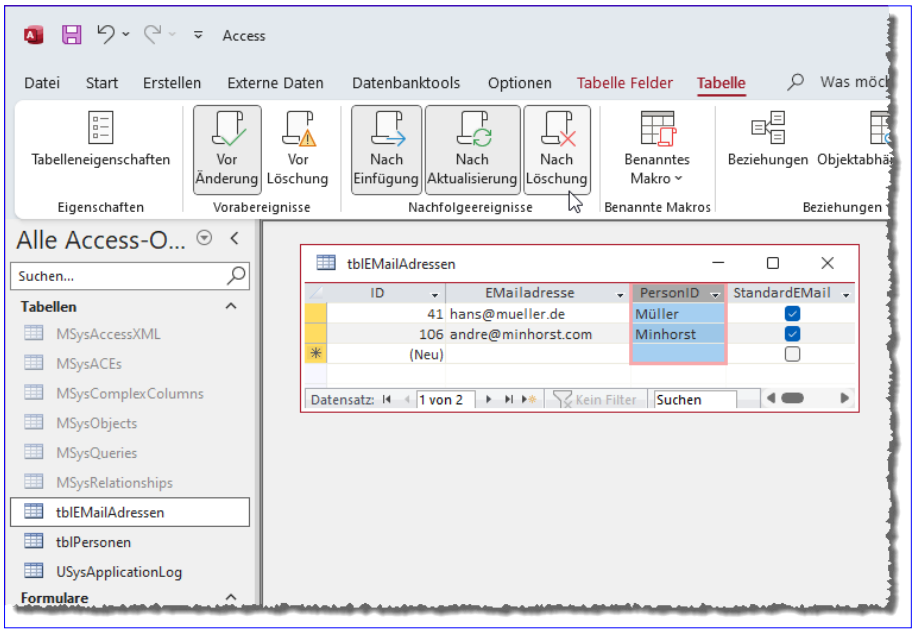


Bild 2: Aufrufen des Entwurfs der verschiedenen Makros

Datenmakro beim Löschen

Damit begeben wir uns an die Programmierung der Datenmakros. Der einfachste Fall ist das Löschen eines Datensatzes, der als Standarddatensatz markiert ist. Dazu nutzen wir das Datenmakro **Nach Löschung**. Dieses legen wir an, indem wir die Tabelle in der Datenblattansicht öffnen und im Ribbon unter Tabelle den Befehl **Nach Löschung** anklicken (siehe Bild 2). Wichtig ist, das wir die Option **Systemobjekte anzeigen** aktivieren, denn so können wir auf die Tabelle **USysApplicationLog** zugreifen, in die wir vom Makro aus Protokollmeldungen eintragen können.

Das Datenmakro legen wir wie in Bild 3 an. Wir starten hier mit einer **Wenn**-Bedingung. Die Bedingung lautet **[Alt].[StandardEMail] = -1**. Mit **[Alt]** greifen wir auf die Daten des zu löschenden Recordsets zu. Hat dessen Feld **StandardEMail** den Wert **-1**, ist er als Standardzuordnung für seine **PersonID** registriert. In diesem Fall müssen wir nach dem Löschen einen neuen Datensatz bestimmen, der den Wert **-1** im Feld **StandardEMail** erhalten soll. Dazu verwenden wir die Aktion **Datensatz nachschlagen in** und übergeben als Quelle die Tabelle **tblEMailAdressen**. Außerdem legen wir als Bedingung fest, dass der gesuchte Datensatz im Feld **PersonID** den Wert des Feldes **PersonID** des zu löschenden Datensatzes aufweist. Damit wir überhaupt zwischen dem Feld **PersonID** des aktuellen Datensatzes und der Tabelle **Alt** unterscheiden

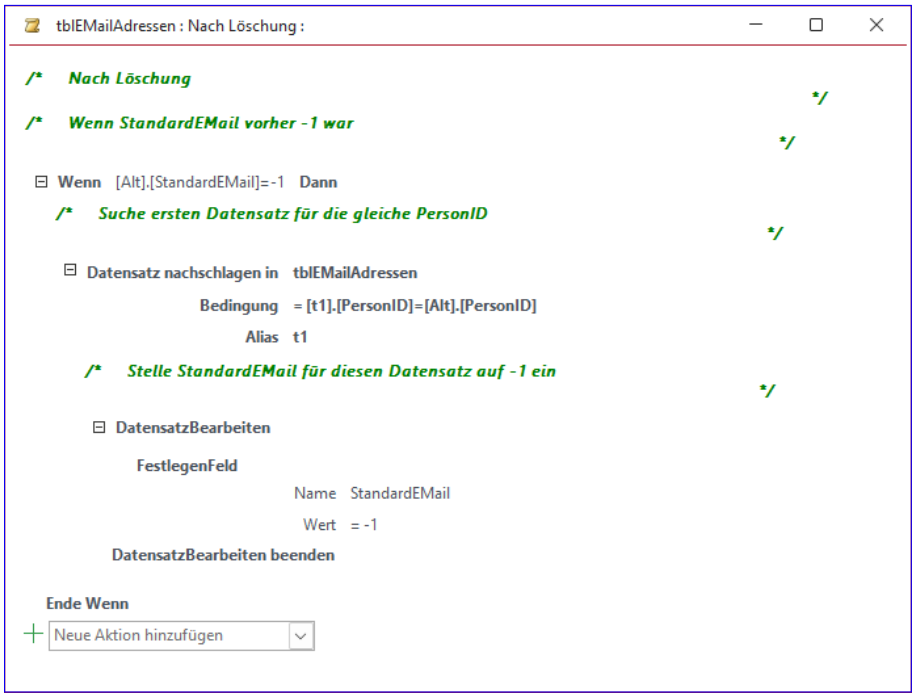


Bild 3: Datenmakro **Nach Löschung**

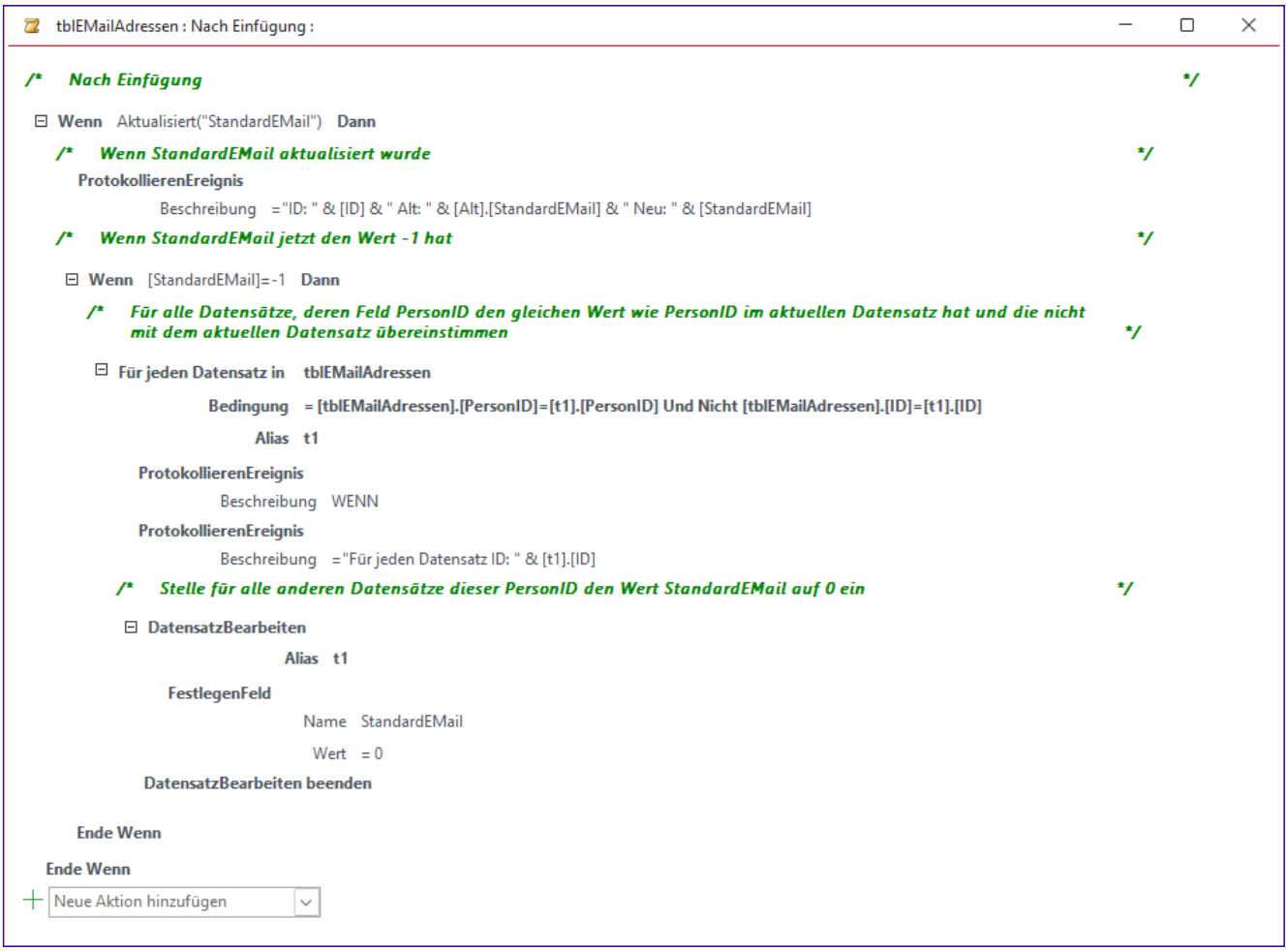


Bild 4: Makro zum Einstellen der Standardzuordnung beim Anlegen neuer E-Mail-Adressen

können, vergeben wir für das Recordset den Alias **t1**. Falls wir einen geeigneten Datensatz finden, was immer der Fall ist, wenn mindestens noch ein weiterer Datensatz für diese **PersonID** vorhanden ist, stellen wir dessen Wert **StandardEMail** auf **-1** ein. Damit haben wir eine neue Standardzuordnung erstellt. Wenn kein geeigneter Datensatz gefunden wird, haben wir die letzte E-Mail-Adresse für diese Person gelöscht und müssen keine neue Standardzuordnung vornehmen.

Anlegen neuer Datensätze

Wenn wir den ersten neuen Datensatz für eine **PersonID** in der Tabelle **tblEMailAdressen** eintragen, soll dieser direkt mit dem Wert **-1** im Feld **StandardEMail** versehen werden. Dazu hinterlegen wir das Makro aus Bild 4 für **Nach Einfügung**. Hier prüfen wir zuerst, ob das Feld **StandardEMail** aktualisiert wurde und protokollieren den vorherigen und den aktuellen Wert (dies hat bei der Entwicklung geholfen, zu verstehen, was passiert – daher belassen wir diese Protokollzeilen im Makro).

Wenn der Benutzer für den neuen Datensatz direkt den Wert **-1** für **StandardEMail** hinterlegt hat, müssen wir gegebenenfalls den Wert für die bereits vorhandene Datensätze auf **0** einstellen.

Dazu durchlaufen wir mit **Für jeden Datensatz** alle Datensätze, welche der gleichen **PersonID** zugeordnet sind und deren **ID** nicht der **ID** des angelegten Datensatzes entspricht.

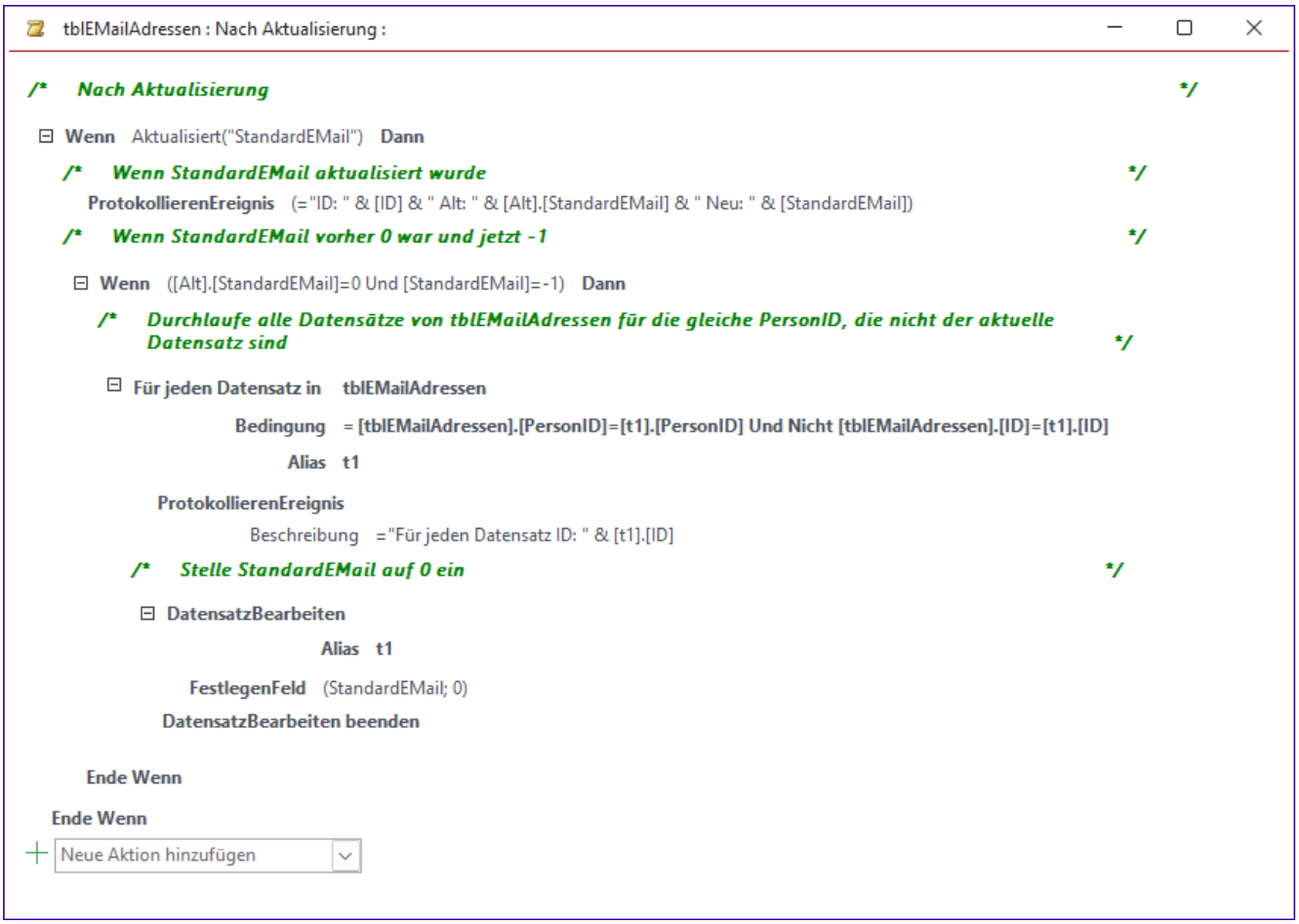


Bild 5: Datenmakro **Nach Aktualisierung**

In dieser Schleife stellen wir mit **DatensatzBearbeiten** und **FestlegenFeld** nach einem weiteren Kommentar den Wert des Feldes **StandardEMail** für die übrigen Datensätze auf **0** ein.

Anpassen nach dem Aktualisieren

Damit fehlt noch das Aktualisieren der übrigen Datensätze, wenn man einen anderen als den aktuellen Datensatz als neue Standardzuordnung festlegt. Das erledigen wir im Datenmakro **Nach Aktualisierung** (siehe Bild 5).

Hier prüfen wir mit der Funktion **Aktualisiert** wieder, ob **StandardEMail** überhaupt aktualisiert wurde. Falls, prüfen wir, ob das Feld **StandardEMail** vor dem Ändern den Wert **0** enthielt (aus dem Recordset **Alt**) und jetzt den Wert **-1**. Falls ja, durchlaufen wir jeden Datensatz eines Recordsets, das wir mit **Für jeden Datensatz** ermitteln, wobei die Tabelle **tblEMailAdressen** als Datenquelle dient und wir alle Datensätze suchen, die zur gleichen **PersonID** wie der geänderte Datensatz gehören und deren Wert aus dem Feld **ID** nicht der des geänderten Datensatzes entspricht.

Für all diese Datensätze stellen wir den Wert von **StandardEMail** auf **0** ein und entfernen somit den Status als Standardzuordnung.

Zwischenfazit

All dies funktioniert wunderbar. Wir legen eine erste E-Mail-Adresse an, diese wird direkt als Standardzuordnung definiert. Wie stellen eine andere E-Mail-Adresse als Standard ein und das Datenmakro erkennt der vorherigen diesen Status ab. Wir löschen die aktuelle Standard-E-Mail

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP

Dateneingabe in Haupt- und Unterformular

Wenn man Daten in die Tabellen einer 1:n-Beziehung eingeben möchte, nutzt man in der Regel ein Haupt- und ein Unterformular. Das Hauptformular nimmt die Daten der Mastertabelle auf, das Unterformular die Daten der Detailtabelle. Wenn man nun keine weiteren Vorsichtsmaßnahmen trifft, kann der Benutzer Daten in das Unterformular eingeben, ohne dass das Hauptformular einen Datensatz anzeigt. Damit erzeugt er Datensätze in der Detailtabelle, die mit keinem Datensatz der Mastertabelle verknüpft sind. Das ist aus mehreren Gründen schlecht – diese schauen wir uns an und zeigen, wie wir das Problem beheben können.

Beispieldatenbank

Wir wählen als einfaches Beispiel zwei Tabellen namens **tblKategorien** und **tblProdukte**. Die Tabelle **tblProdukte** ist über das als Nachschlagefeld ausgelegte Fremdschlüsselfeld **KategorieID** mit der Tabelle **tblKategorien** verknüpft, sodass wir für jedes Produkt eine Kategorie auswählen können. Im Datenmodell sehen die beiden Tabellen mit ihrer Verknüpfung wie in Bild 1 aus.

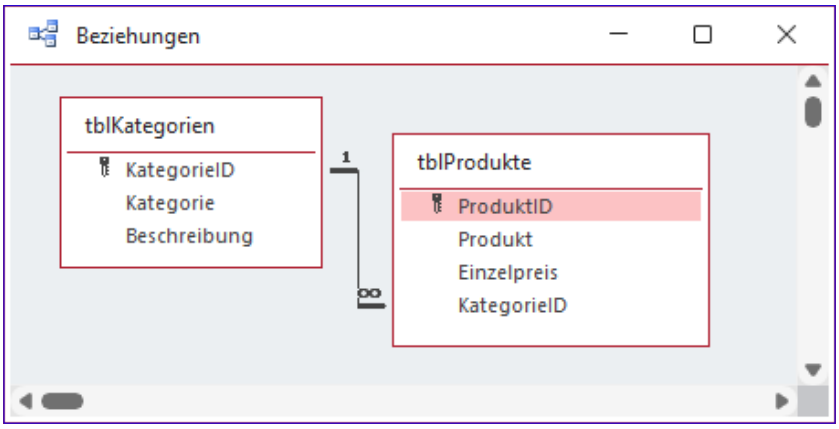


Bild 1: Die Tabellen der Beispielanwendung

Der Eigenschaft **Datensatzquelle** des Unterformulars **sfmKategorienProdukte** weisen wir die Tabelle **tblProdukte** hinzu. Aus der Feldliste ziehen wir alle Felder in den Detailbereich des Formularentwurfs und stellen die Eigenschaft **Standardansicht** auf **Datenblatt** ein (siehe Bild 2). Danach speichern und schließen wir dieses Formular.

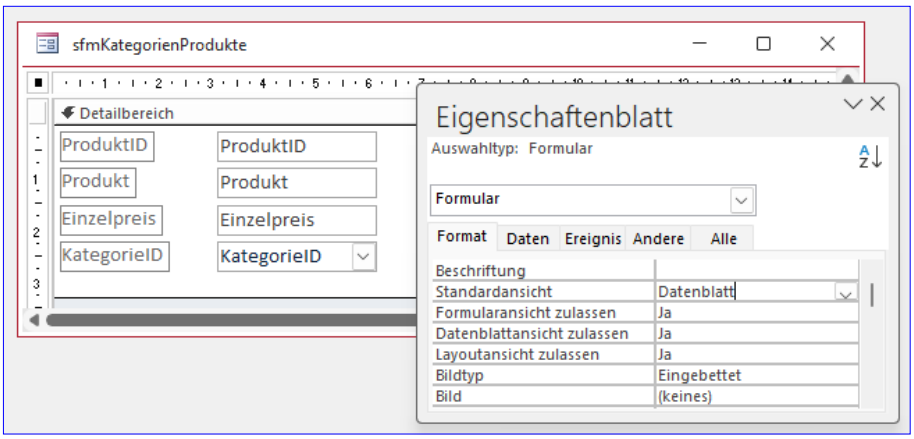


Bild 2: Das Unterformular der Anwendung

Für das Hauptformular **frmKategorienProdukte** legen wir die Tabelle **tblKategorien** als Datensatzquelle fest. Auch hier ziehen wir alle Felder aus der Feldliste in den Detailbereich. Außerdem ziehen wir das Formular **sfmKategorienProdukte** aus dem Navigationsbereich in das Hauptformular **frmKategorienProdukte** und platzieren es unterhalb der hinzugefügten Felder. Access erkennt dabei automatisch die Beziehung zwischen den Tabellen, die als Datensatzquelle für die

beiden Formulare dienen, und trägt die Verknüpfungsfelder, also das Fremdschlüsselfeld der Tabelle **tblProdukte** und das Primärschlüsselfeld der Tabelle **tblKategorien**, in die beiden Eigenschaften **Verknüpfen von** und **Verknüpfen nach** des Unterformular-Steuerelements ein (siehe Bild 3).

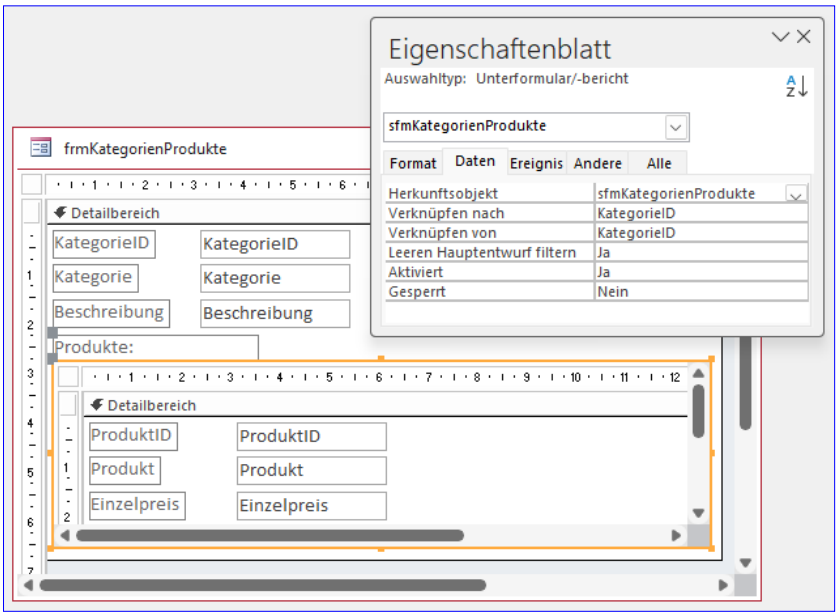


Bild 3: Einbau des Unterformulars in das Hauptformular

Problem: Produkte ohne Kategorie

Geht man in diesem Formular den üblichen Weg und legt zunächst eine Kategorie an, bevor man ein Produkt hinzufügt, werden die Daten miteinander verknüpft und das Fremdschlüsselfeld **KategorieID** der Tabelle **tblProdukte** weist den gewünschten Wert der Tabelle **tblKategorien** auf. Das ist deswegen der Fall, weil das Fremdschlüsselfeld **KategorieID** im Unterformular automatisch mit dem Wert des Primärschlüsselfeldes im Hauptformular als Standardwert gefüllt wird (siehe Bild 4).

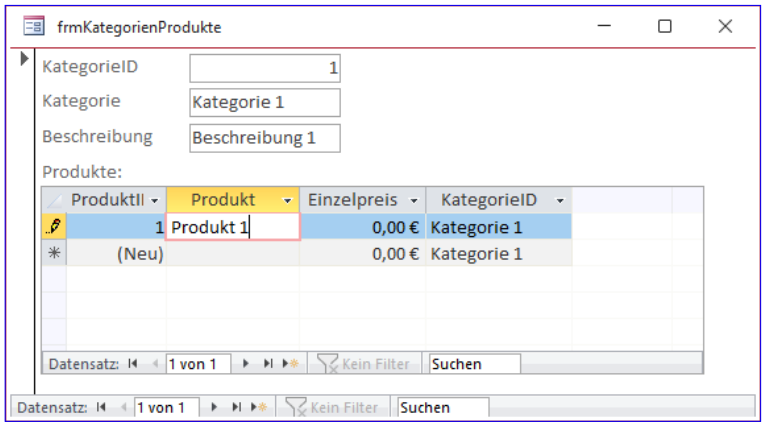


Bild 4: Eingabe von Daten auf dem geplanten Weg

Das ist insbesondere deshalb möglich, weil durch den Wechsel ins Unterformular der Datensatz im Hauptformular gespeichert wird und der Primärschlüsselwert der Kategorie bereits verfügbar ist.

Wir können in diesem Formular allerdings auch direkt einen Datensatz im Unterformular anlegen, ohne dass das Hauptformular einen Datensatz anzeigt – beziehungsweise während es einen neuen, leeren Datensatz anzeigt, für den es noch keinen automatisch vergebenen Primärschlüsselwert gibt. Wechseln wir ins Unterformular, ist das Fremdschlüsselfeld **KategorieID** noch nicht gefüllt (siehe Bild 5).

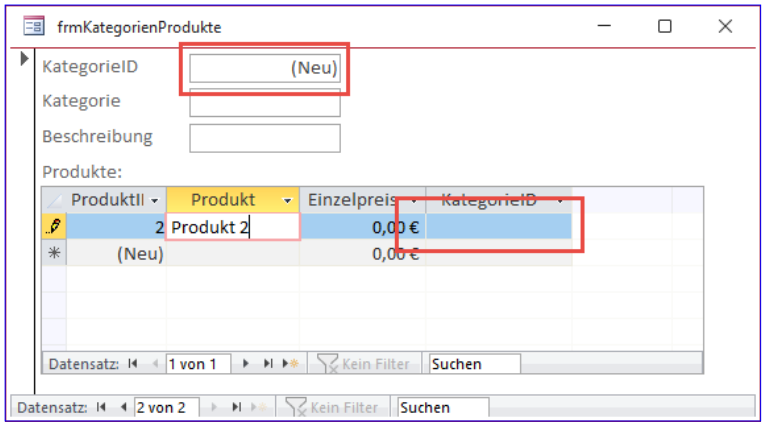


Bild 5: Ohne Kategorie wird auch das Fremdschlüsselfeld nicht vorbelegt

Wir können hier nun einen Datensatz anlegen, den wir **Produkt 2** nennen. Fällt uns dann auf,

dass wir noch die Kategorie anlegen wollten und wechseln wir dazu ins Hauptformular, ist der Datensatz im Unterformular noch sichtbar.

Sobald wir jedoch mit der Eingabe in die Felder des Hauptdatensatzes beginnen, wird das Primärschlüsselfeld gefüllt und somit passt der Datensatz im Unterformular nicht mehr zu diesem Datensatz – er hat den Wert **Null** im Fremdschlüsselfeld **KategorieID**, während das Primärschlüsselfeld **KategorieID** im Hauptformular nun den Wert **2** aufweist (siehe Bild 6). Speichern wir nun den Datensatz im Hauptformular, indem wir auf den Datensatzmarkierer klicken oder zum Unterformular wechseln, wird der Standardwert für das dortige Fremdschlüsselfeld **KategorieID** korrekt auf **2** eingestellt. Der dort angelegte Produkt-Datensatz bleibt jedoch verschwunden (siehe Bild 7).

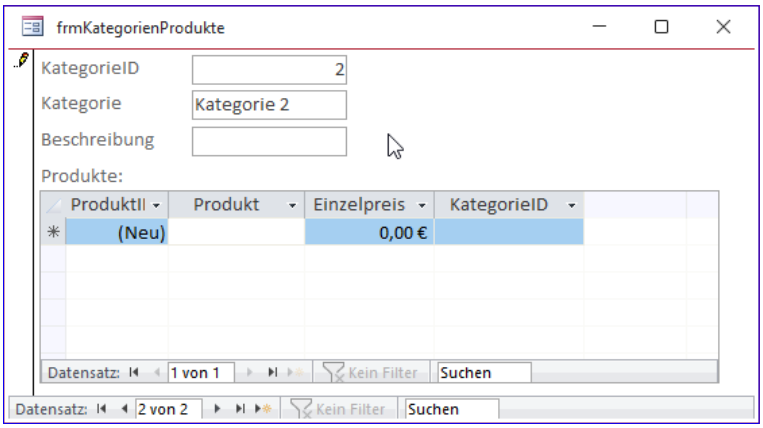


Bild 6: Nach dem Ergänzen der Kategorie verschwindet der Datensatz im Unterformular.

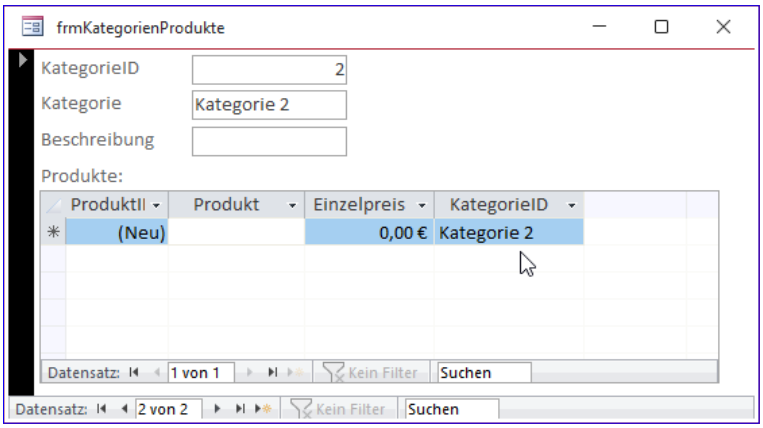


Bild 7: Nach dem Speichern im Hauptformular wird der Fremdschlüsselwert im Unterformular wieder auf den richtigen Wert eingestellt.

Werfen wir einen Blick in die Tabelle **tblProdukte**, finden wir den verlorengeglaubten Datensatz allerdings wieder – er hat lediglich keinen Wert im Feld **KategorieID**, weshalb er im Formular nicht im Kontext einer Kategorie erscheint (siehe Bild 8).

Der Datensatz erscheint erst wieder, wenn wir im Formular zu einer neuen, leeren Kategorie wechseln – dann passt der Fremdschlüsselwert **Null** wieder zu dem Primärschlüsselwert im Hauptformular, der für einen neuen, leeren Datensatz ebenfalls **Null** lautet. Den falsch erstellten Datensatz können wir nun löschen oder wir weisen ihm über das Nachschlagefeld in der Tabelle eine Kategorie zu.

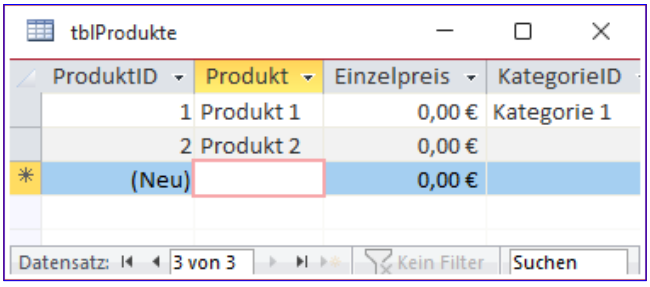


Bild 8: Der Produkt-Datensatz wurde angelegt – wenn auch ohne Kategorie.

Anlegen von Datensätzen im Unterformular ohne Datensatz im Hauptformular verhindern

Es gibt viele Wege, mit denen wir verhindern können, dass ein Datensatz im Unterformular angelegt wird, wenn das Hauptformular noch keinen Datensatz enthält.

Ein Weg ist, die Eigenschaft **Eingabe erforderlich** des Feldes **KategorieID** der Tabelle **tblProdukte** auf **Ja** einzustellen. Das erledigen wir wie in Bild 9 im Entwurf der Tabelle **tblProdukte**.

Nun speichern und schließen wir die Tabelle und öffnen das Formular **frmKategorienProdukte** erneut. Hier wechseln wir nun wieder direkt ins Unterformular, während das Hauptformular einen neuen, leeren Datensatz anzeigt.

Sobald wir hier das erste Zeichen in eines der Felder eintippen und somit den Datensatz im Unterformular in den Bearbeitungszustand versetzen, erscheint die Meldung aus Bild 10.

Wir können dieser Meldung zwar nun folgen und einen Eintrag im Feld **KategorieID** auswählen. Aber erstens zeigen wir diese Feld hier nur an, um zu sehen, wann und ob die Beziehung über das Fremdschlüsselfeld hergestellt wird und zweitens wollen wir dem Benutzer nicht eine solche Meldung präsentieren.

Wir könnten an dieser Stelle noch weitergehen und diesen Fehler abfangen. Das ginge über das Ereignis **Bei Fehler** des Unterformulars, wo wir prüfen können, welcher Fehler aufgetreten ist und diesen Fehler entsprechend behandeln. Allerdings ist dieser Weg relativ aufwendig, daher schauen wir uns eine einfachere Alternative an.

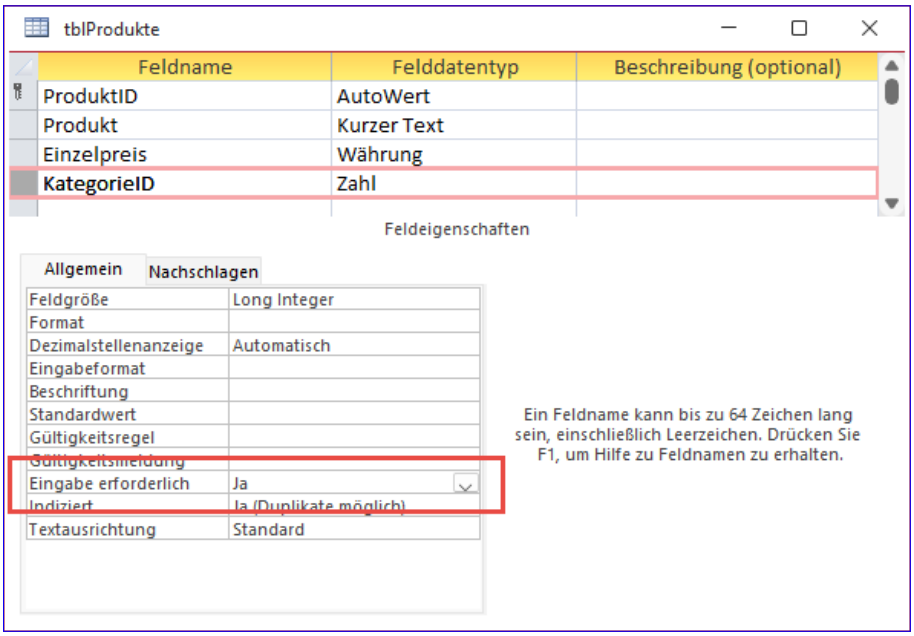


Bild 9: Verhindern von Datensätzen ohne Fremdschlüsselwert

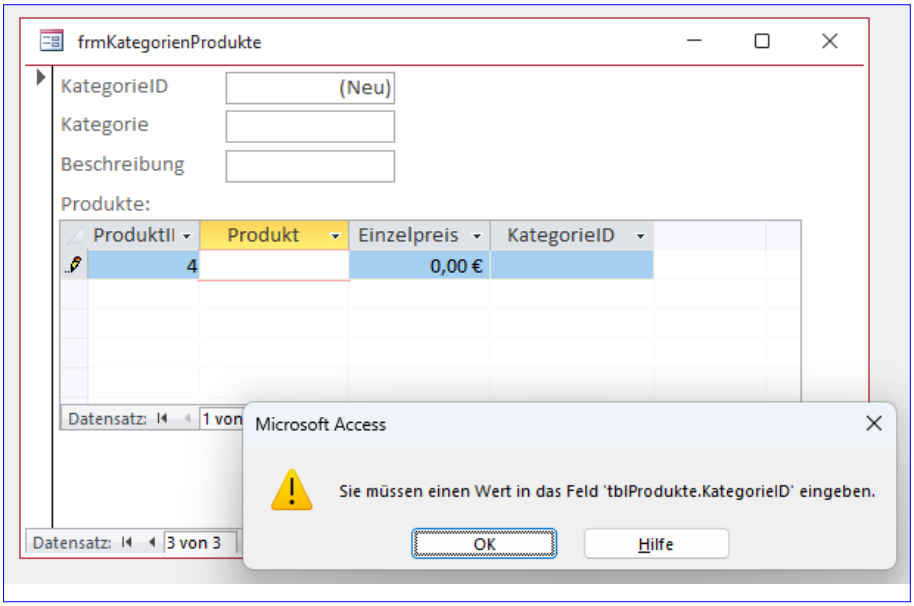


Bild 10: Fehler beim Anlegen eines Produkt-Datensatzes ohne erforderlichen Wert im Feld **KategorieID**

Zugriff auf das Unterformular ohne Datensatz im Hauptformular verwehren

Dabei wollen wir verhindern, dass der Benutzer überhaupt auf das Unterformular zugreifen kann, wenn das Hauptformular noch keinen Datensatz aufweist. Das können wir mit einer sehr einfachen Maßnahme erreichen.

Dazu hinterlegen wir eine Ereignisprozedur für die Ereigniseigenschaft **Beim Hingehen** des Unterformular-Steuerelements (nicht des Unterformulars selbst). Hier wählen

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP

Unterformular erst nach Validierung aktivieren

Im Beitrag Dateneingabe in Haupt- und Unterformular (www.access-im-unternehmen.de/1463) haben wir gezeigt, wir man bei der Eingabe neuer Daten verhindert, dass der Benutzer Daten in das Unterformular eingibt, bevor ein Datensatz im Hauptformular angelegt wurde. Das kann man noch vertiefen, wenn das Hauptformular Felder enthält, die validiert werden müssen, bevor der dortige Datensatz gespeichert werden kann. Wir schauen uns an einem einfachen Beispiel an, wie sich eine Validierung auf die Lösung aus dem oben genannten Beitrag auswirkt.

Angenommen, unser Formular enthält bereits eine Validierung – wie können wir dann die Techniken aus dem Artikel **Dateneingabe in Haupt- und Unterformular** (**www.access-im-unternehmen.de/1463**) damit kombinieren?

Einfache Validierungsfunktion

Dabei gehen wir davon aus, dass wir die Validierung durchführen, bevor der Benutzer den Datensatz speichert. Beim Speichern wird das Ereignis **Vor Aktualisierung** ausgelöst, in dem wir das Speichern durch Einstellen des Parameters **Cancel** auf den Wert **True** abbrechen können. Hier rufen wir wie folgt die Validierungsfunktion auf und werten das Ergebnis aus – wenn es **False** lautet, wird das Speichern abgebrochen:

```
Private Sub Form_BeforeUpdate(Cancel As Integer)
    If Validierung = False Then
        Cancel = True
    End If
End Sub
```

Die Validierungsfunktion haben wir zu Beispielzwecken recht einfach gehalten (siehe Listing 1). Sie prüft lediglich die vom Benutzer zu füllenden Felder und wenn eines leer ist, zeigt es eine Meldung an, die den Benutzer auffordert, das Feld zu füllen. Anschließend setzt sie den Fokus auf das entsprechende Steuerelement und verlässt die Funktion. Nur wenn alle Validierungen gelingen, wird die Funktion bis zum Ende ausgeführt und liefert den Wert **True** zurück.

```
Private Function Validierung() As Boolean
    If Len(Nz(Me!Kategorie, "")) = 0 Then
        MsgBox "Bitte geben Sie eine Kategorie ein.", vbOKOnly + vbExclamation
        Me!txtKategorie.SetFocus
        Exit Function
    End If
    If Len(Nz(Me!Beschreibung, "")) = 0 Then
        MsgBox "Bitte geben Sie eine Beschreibung ein.", vbOKOnly + vbExclamation
        Me!txtBeschreibung.SetFocus
        Exit Function
    End If
    Validierung = True
End Function
```

Listing 1: Validierungsfunktion

Validieren und zum Unterformular wechseln

Wie verträgt sich dies mit dem Unterformular, das ja den Fokus erhalten soll beziehungsweise nur aktiviert werden soll, wenn der Benutzer den Datensatz im Hauptformular in einen speicherfähigen Zustand gebracht hat?

Wir haben im oben genannten Beitrag zwei Varianten vorgestellt:

- die erste gibt eine Meldung aus, wenn der Benutzer versucht, das Unterformular zu nutzen, obwohl noch kein Datensatz im Hauptformular vorliegt und
- die zweite deaktiviert das Unterformular-Steuerelement solange, bis im Hauptformular ein Datensatz vorliegt.

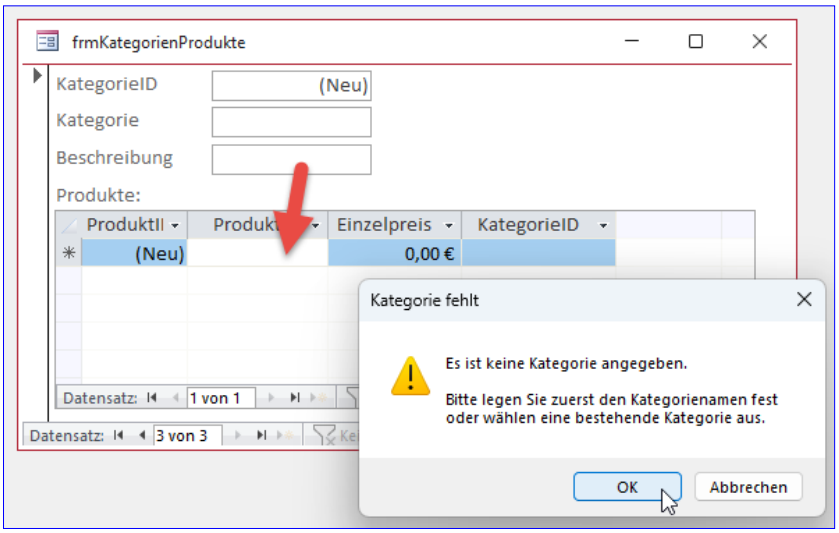


Bild 1: Meldung ohne Validierungsfunktion

Validieren und Meldung, wenn noch keine Daten ins Unterformular eingegeben werden können

Die erste Variante sehen wir im Hauptformular **frmKategorienProdukte**. Hier erhalten wir nach dem Hinzufügen der Validierungsfunktion folgendes Verhalten:

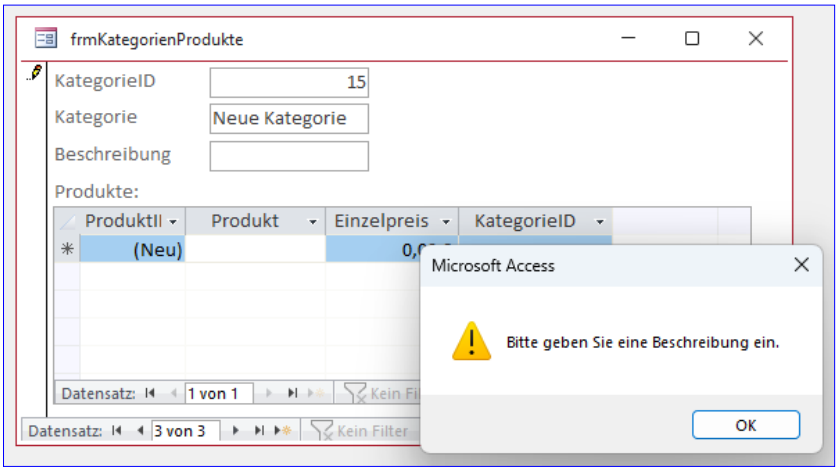


Bild 2: Die Validierung schlägt zuerst an.

Wenn wir direkt nach dem Anlegen eines neuen Datensatzes zum Unterformular wechseln, also ohne überhaupt Werte einzugeben, erscheint die Meldung, dass noch keine Kategorie angegeben ist. Die Validierung wird hier zunächst außen vor gelassen (siehe Bild 1). Dies wird durch das Ereignis **Beim Hingehen** des Unterformular-Steuerelements realisiert.

Geben wir hingegen Daten ein und versuchen dann, zum Unterformular zu wechseln, wird zuerst die Validierungsfunktion aufgerufen. Diese meldet eventuelle Probleme (siehe Bild 2) und bricht die Aktualisierung des Datensatzes im Hauptformular durch Setzen von **Cancel** auf **True** ab. Außerdem wird der Fokus zurück auf das nicht validierte Feld gesetzt, sodass dieser nicht im Unterformular landet und somit auch nicht die Meldung mit dem Titel **Kategorie fehlt** angezeigt wird.

Geben wir hingegen alle von der Validierung geforderten Daten an, können wir problemlos zum Unterformular wechseln, um dort die entsprechenden Datensätze einzugeben.

ACCESS
IM UNTERNEHMEN

Hier geht es weiter ...

Wenn Sie bis hierhin gelesen haben, könnte ich mir vorstellen, dass Sie der Rest auch noch interessieren würde. Das Beste kommt nämlich meistens erst am Ende!

Für weniger als 50 Cent am Tag* bekommen Sie nicht nur diesen Beitrag, sondern auch noch mehr als tausend weitere Artikel über unser Onlinearchiv auf www.access-im-unternehmen.de. Plus alle zwei Monate 76 Seiten neues Know-how rund um die Programmierung mit Microsoft Access.

Alle Informationen hier – jetzt bestellen und direkt loslesen!

ZUM SHOP

Select Case-Bedingung für Texte optimieren

Normalerweise verwenden wir die Select Case-Bedingung so, dass wir im Kopf einen einen Teil des Vergleichsausdrucks platzieren und in den einzelnen Case-Zweigen die Vergleichswerte. Genau genommen ist das der große Unterschied zur If...Then-Bedingung, die immer den kompletten Ausdruck in einem Zweig darstellt. Die If...Then-Bedingung scheint daher bei der Auswertung von Zeichenketten Vorteile zu haben. Wir können aber auch die Select Case-Bedingung prima für Zeichenketten nutzen.

Angenommen, wir untersuchen eine Zeichenkette, die wir in diesem Fall über eine **InputBox** erhalten. Mit einer **If... Then**-Bedingung können wir einfach bestimmte Auswertungen durchführen:

```
Public Sub IfThen()
    Dim strText As String
    strText = InputBox("Text:", "Text eingeben", "abc")
    If strText = "abc" Then
        MsgBox "Text ist 'abc'"
    ElseIf strText Like "a*" Then
        MsgBox "Text beginnt mit 'a'"
    ElseIf InStr(1, strText, "a") Then
        MsgBox "Text enthält 'a'"
    ElseIf Right(strText, 1) = "a" Then
        MsgBox "Text endet auf 'a'"
    End If
End Sub
```

Wenn wir das mit einer **Select Case**-Bedingung abbilden wollten, die manchmal übersichtlicher wirken, hätten wir außer bei der Abbildung der ersten Bedingung ein Problem:

```
Public Sub SelectCase()
    Dim strText As String
    strText = InputBox("Text:", "Text eingeben", "abc")
    Select Case strText
        Case "abc"
            MsgBox "Text ist 'abc'"
        Case ?
            MsgBox "Text beginnt mit 'a'"
        Case ?
            MsgBox "Text enthält 'a'"
        Case ?
            MsgBox "Text endet auf 'a'"
    End Select
End Sub
```

Der Trick ist dann, als ersten Vergleichswert **True** hinter **Select Case** zu schreiben und in den einzelnen **Case**-Zweigen einfach die kompletten Bedingungen als Bool'sche Ausdrücke einzufügen:

```
Public Sub SelectCaseMalAnders()
    Dim strText As Integer
    strText = InputBox("Text:", "Text eingeben", "abc")
    Select Case True
        Case strText = "abc"
            MsgBox "Text ist 'abc'"
        Case strText Like "a*"
            MsgBox "Text beginnt mit 'a'"
        Case InStr(1, strText, "a") > 0
            MsgBox "Text enthält 'a'"
        Case Right(strText, 1) = "a"
            MsgBox "Text endet auf 'a'"
    End Select
End Sub
```

So können wir in der **Select Case**-Bedingungen auch alle möglichen Zeichenkettenfunktionen nutzen und Vergleichsoperatoren wie **LIKE** mit dem Platzhalter ***.