HAUFE.
Ausgabe 06/2020

# ACCESS
## IM UNTERNEHMEN

# XRECHNUNG MIT ACCESS

Wir zeigen, wie Sie Rechnungen entsprechend der Anforderungen öffentlicher Auftraggeber erstellen können! (ab S. 59)

## In diesem Heft:

### TASTENKOMBINATIONEN MIT AUTOHOTKEYS

Fügen Sie anwendungsweit verfügbare Tastenkombinationen für mehr Komfort hinzu.

SEITE 19

### WEBCAM MIT ACCESS STEUERN

Starten Sie Ihre Webcam von Access aus und lesen Sie die Bilder direkt in die Datenbank ein.

SEITE 45

### NEUIGKEITEN FÜR ACCESS

Welche Funktionen plant Microsoft für die Datenbankanwendung und welche gibt es schon?

SEITE 2

# Einfacher Austausch

**Rechnungen haben den Nachteil, dass sie nicht standardisiert sind. Es gibt einige grundlegende Abmachungen, damit beispielsweise die Empfängeradresse im Fenster des Briefumschlages erscheint und Vorgaben, welche Informationen enthalten sein müssen. Aber bezüglich der Gestaltung gibt es so viel Freiraum, dass Rechnungen nicht einfach automatisiert weiterverarbeitet werden können. Für die Einreichung von Rechnungen an öffentliche Auftraggeber gibt es nun allerdings ein maschinenlesbares Format: die XRechnung. Wir zeigen, wie Sie damit kompatible Rechnungen erstellen.**

Die XRechnung wird im XML-Format erstellt. Wie Sie die wichtigsten Vorgaben für das Schreiben solcher Rechnungen erfüllen, zeigen wir im Beitrag **XRechnung, Teil 1: Rechnungen generieren** ab Seite 59. Dort verarbeiten wir die üblichen Informationen einer Rechnung, also den Sender, den Empfänger, die Positionen und die Rechnungssummen samt Mehrwertsteuerangaben in einem XML-Dokument.

Dieses schicken Sie an den öffentlichen Auftraggeber, der diese automatisch einlesen kann. Damit sollen Fehler vermieden werden, die beim manuellen Einlesen durch Mitarbeiter oder auch beim automatisierten Scannen passieren. Außerdem soll auch der Aufwand zum Einlesen minimiert werden. In den folgenden Ausgaben zeigen wir außerdem, wie Sie Rechnungen in diesem Format einlesen und diese für Menschen lesbar darstellen.

Access 365 erhält im Gegensatz zu Access 2016, Access 2019 und so weiter nicht mit jeder neuen Version neue Funktionen. Stattdessen werden diese mit mehr oder weniger regelmäßigen Updates installiert. Welche Neuerungen es in letzter Zeit gab, zeigt der Beitrag **Neues in Access** ab Seite 2. Hier erfahren Sie auch, wo Sie sich über anstehende neue Funktionen informieren können.

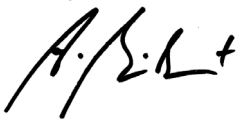
Tastenkombinationen erleichtern dem Benutzer die Bedienung von Anwendungen. Access liefert bereits eine Menge eingebauter Tastenkombinationen. Sie können allerdings auf verschiedene Arten eigene Tastenkombinationen hinzufügen. Eine Möglichkeit, mit der Sie einzelne Datenbanken mit neuen Tastenkombinationen ausstatten können, ist das **AutoKeys**-Makro. Damit legen Sie Makro-Aktionen fest, die beim Betätigen der angegebenen Tastenkombinationen ausgelöst werden sollen. Wie Sie das erledigen, zeigt der Beitrag **Das AutoKeys-Makro** ab Seite 15.

Der Nachteil dieses Makros ist, dass es immer nur in der Anwendung funktioniert, die das Makro enthält. Wollen Sie Tastenkombinationen festlegen, die immer funktionieren, können Sie das Tool **AutoHotKeys** nutzen. Damit definieren Sie systemweit wirksame Tastenkombinationen. Wie das gelingt, erfahren Sie im Beitrag **Tastenkombinationen mit AutoHotKeys** ab Seite 19.

In unserer Reihe zum Thema Sicherheit im SQL Server geht es mit dem vierten Teil weiter. Unter dem Titel **SQL Server-Security, Teil 4: Schutz mit Datenbankrollen** erläutert Bernd Jungbluth ab Seite 28, wie Sie Benutzern Datenbankrollen zuweisen können und darüber ihre Berechtigungen festlegen.

Nutzen Sie eine Webcam oder eine im Rechner oder Monitor eingebaute Kamera zum Aufnehmen von Fotos? Dann möchten Sie diese vielleicht auch direkt in einer Access-Datenbank weiterverarbeiten. Eine Lösung dazu finden Sie im Beitrag **Webcam-Bilder in Datenbank speichern** ab Seite 45.

Viel Spaß beim Lesen!



Ihr André Minhorst

# Neues in Access

**Mit der Access-Version aus Office 365 gibt es keine größeren Updates mehr, wie es früher der Fall war, sondern Microsoft schickt Neuerungen unbemerkt an den Endbenutzer. Dieser Beitrag zeigt die jüngsten Aktualisierungen, die sich unbemerkt von der Öffentlichkeit in Ihre Access-Installation geschlichen haben. Die Neuerungen betreffen vor allem das Beziehungen-Fenster, aber auch die SQL-Ansicht von Abfragen sowie den Abfrage-Entwurf.**

## Beziehungen-Fenster mit neuen Funktionen

Eine Neuerung betrifft die Befehle des Kontextmenüs im Beziehungen-Fenster, genauer gesagt das Kontextmenü für die Tabellen-Elemente (siehe Bild 1).

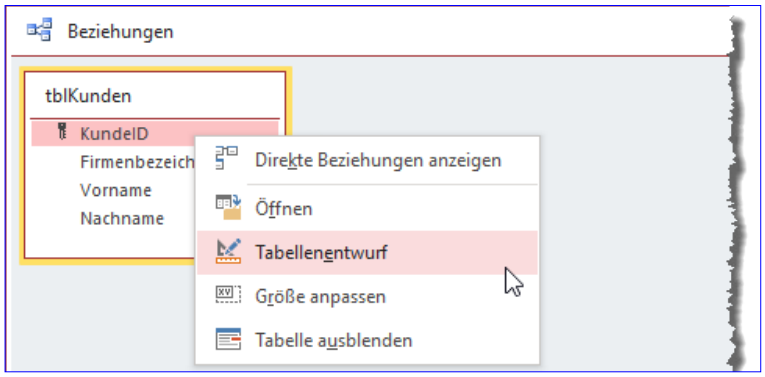



**Bild 1:** Kontextmenü im **Beziehungen**-Fenster

Hier finden Sie nun die folgenden Einträge:

- **Direkte Beziehungen anzeigen**: Blendet alle mit dieser Tabelle verknüpften Tabellen plus Beziehungspfeile ein.
- **Öffnen**: Öffnet die Tabelle in der Datenblattansicht (neu in 2020).
- **Tabellenentwurf**: Öffnet die Tabelle in der Entwurfsansicht.
- **Größe anpassen**: Stellt die Größe auf die optimale Größe ein, sodass alle Felder vollständig angezeigt werden (neu in 2020).
- **Tabelle ausblenden**: Blendet die Tabelle aus.

## Größe automatisch anpassen

Das Anpassen der Größe gelingt nicht nur über den Kontextmenü-Eintrag **Größe anpassen**, sondern Sie können auch doppelt auf den rechten oder unteren Rand einer Tabelle im Beziehungen-Fenster klicken. Dies optimiert allerdings nur die Breite beziehungsweise Höhe der angeklickten Tabelle.

## Scrollen im Beziehungen-Fenster mit der Maus

Wenn Ihre Datenbank sehr viele Tabellen enthält, müssen Sie im Beziehungen-Fenster scrollen, um alle Tabellen und Beziehungen betrachten zu können. Früher ging das nur mithilfe der Bildlaufleisten. Jetzt können Sie dazu auch das Scrollrad der Maus verwenden: Wenn Sie einfach nur das Scrollrad bedienen, scrollt der Inhalt des Beziehungen-Fensters nach oben oder unten. Wenn Sie gleichzeitig die Umschalttaste gedrückt halten, scrollen Sie nach rechts oder links.

## Markieren und Verschieben mehrerer Tabellen gleichzeitig

Bisher konnten Sie im Beziehungen-Fenster immer nur eine Tabelle gleichzeitig markieren. Das bedeutete auch, dass Sie alle Elemente einzeln verschieben mussten,

wenn Sie dem Beziehungen-Layout eine Tabelle links von den vorhandenen Tabellen hinzufügen wollten.

Jetzt können Sie mehrere Tabellen markieren und diese nicht nur gemeinsam verschieben, sondern auch noch für alle Tabellen gleichzeitig den Kontextmenü-Eintrag **Größe anpassen** verwenden.

Um mehrere Tabellen gleichzeitig zu markieren, klicken Sie diese entweder bei gedrückter **Strg**-Taste an oder ziehen Sie einen Rahmen um alle zu markierenden Tabellen auf (siehe Bild 2).

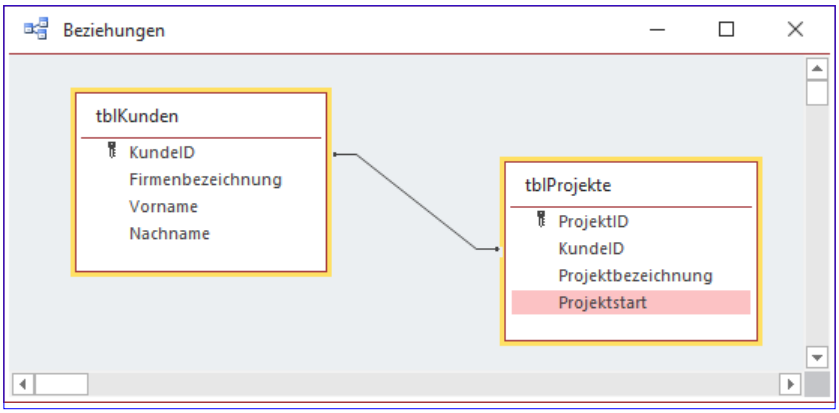



**Bild 2:** Markieren mehrerer Tabellen gleichzeitig

## Suchen und Ersetzen in der SQL-Ansicht von Abfragen

Abfragen können Sie zum Entwurf in der Entwurfsansicht oder in der SQL-Ansicht anzeigen. Letzteres liefert ein Fenster, das nur den SQL-Code der Abfrage darstellt.

Bisher konnten Sie hier nur Texte bearbeiten oder diese ausschneiden, kopieren und einfügen. Nun steht auch die **Suchen und Ersetzen**-Funktion bereit.

Diese aktivieren Sie mit der Tastenkombination **Strg + F**. Anschließend erscheint der **Suchen und Ersetzen**-Dialog aus Bild 3.

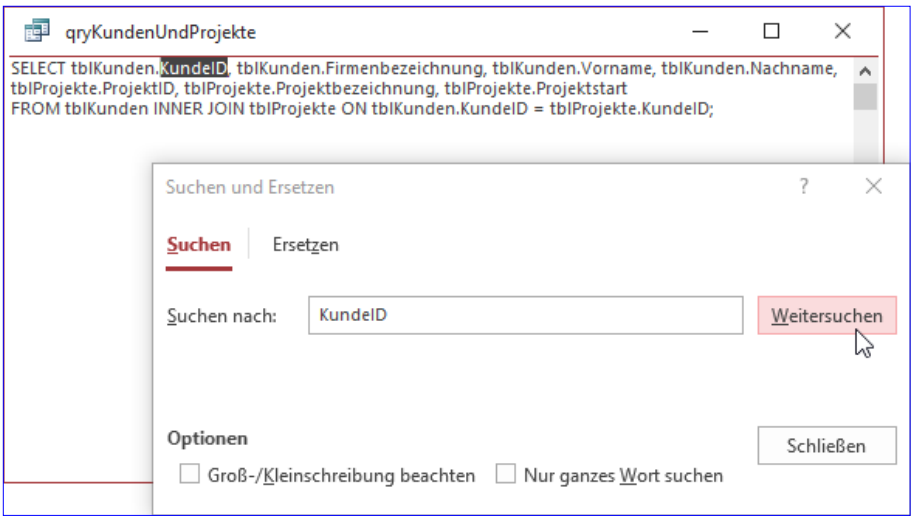



**Bild 3:** Suchen und Ersetzen in der SQL-Ansicht von Abfragen

## Neuerungen im Abfrageentwurf

Im Abfrageentwurf gibt es zunächst die gleichen Neuerungen wie im Beziehungen-Fenster:

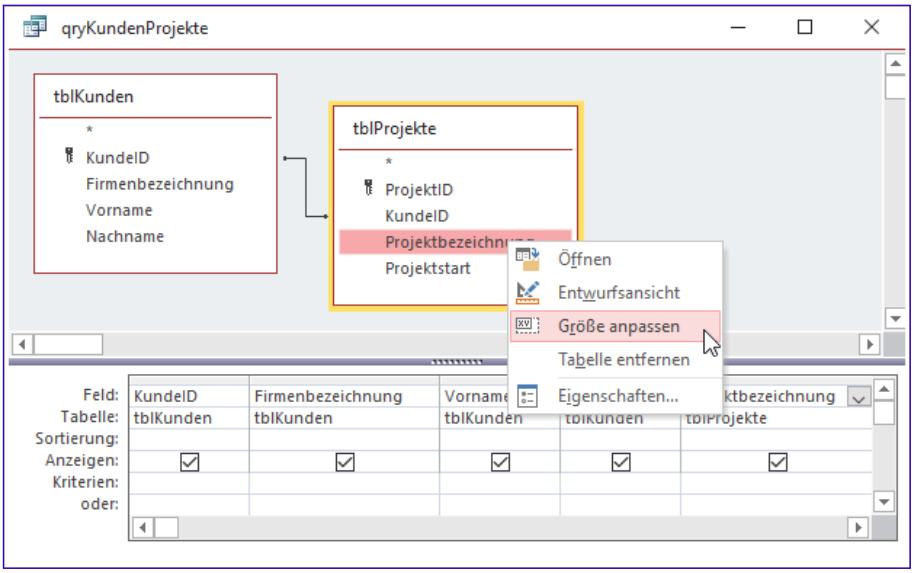



**Bild 4:** Neue Funktionen im Abfrage-Entwurf

# Office 365-Updates im Griff

**Wenn Sie Office 365 verwenden, kommen Updates automatisch zu Ihnen. Für Access bedeutet dies, dass auch einmal Neuerungen ohne »großes« Update wie früher auf dem System landen. Der Nachteil ist allerdings, dass Sie üblicherweise nichts von den Neuerungen erfahren. Dieser Beitrag zeigt, wie Sie beeinflussen können, ob neue Versionen auf dem Rechner landen, wo Sie Informationen über neue Features erhalten und wie Sie, wenn eine neue Version mal Fehler mit sich bringt, wieder zur vorherigen Version wechseln können.**

## Office-Version herausfinden

In vielen Fällen ist der erste Schritt, erst einmal die Version der aktuellen Office-Version auf Ihrem Rechner zu ermitteln. Dazu bieten alle Office-Anwendungen den gleichen Bereich an, den Sie wie folgt finden:

- Klicken Sie im Ribbon auf **Datei**.
- Wählen Sie im nun erscheinenden Backstage-Bereich den Reiter **Konto** aus.
- Hier klicken Sie auf die Schaltfläche **Info zu Access**.

Dies öffnet den Dialog aus Bild 1. Neben der Version, hier **16.0.13231.20110**, erfahren Sie auch noch, ob Sie die 32-Bit- oder die 64-Bit-Version verwenden.

## Office-Updates einstellen

Standardmäßig lädt Office Updates automatisch aus dem Internet und installiert diese anschließend – das geschieht weitgehend unbemerkt. Wenn Sie allerdings ständig mindestens eine Office-Anwendung geöffnet haben, wie es beispielsweise bei Outlook der Fall sein könnte, kann das Update nicht automatisch ausgeführt werden.

Für das Update müssen nämlich alle Office-Anwendungen geschlossen sein. In diesem Fall blendet Office in allen Office-Anwendungen oben eine Leiste mit einem Hinweis auf ein anstehendes Update ein. Sie können dann alle Office-Anwendungen schließen und das Update anstoßen.

## Office-Updates verwalten

Ob Updates überhaupt automatisch auf Ihrem Rechner landen, können Sie selbst beeinflussen. Dazu gibt es im gleichen Backstage-Bereich eine Schaltfläche mit der Beschriftung **Updateoptionen** (siehe Bild 2).

Die erste Option lautet **Jetzt aktualisieren** und weist darauf hin, dass es Updates für Microsoft Office gibt. Offensichtlich werden also nicht alle Updates direkt automa-


**Bild 1:** Versionsinformationen

tisch installiert. Sie können aber Zwischenupdates durch Betätigen dieser Option installieren.

Neben dem Text **Für dieses Produkt stehen Updates zum Download zur Verfügung** haben wir auch den Text **Für dieses Produkt sind Updates zur Installation bereit** gesehen. Vermutlich ist der Unterschied, dass im zweiten Fall bereits der Download des Updates erfolgt ist.

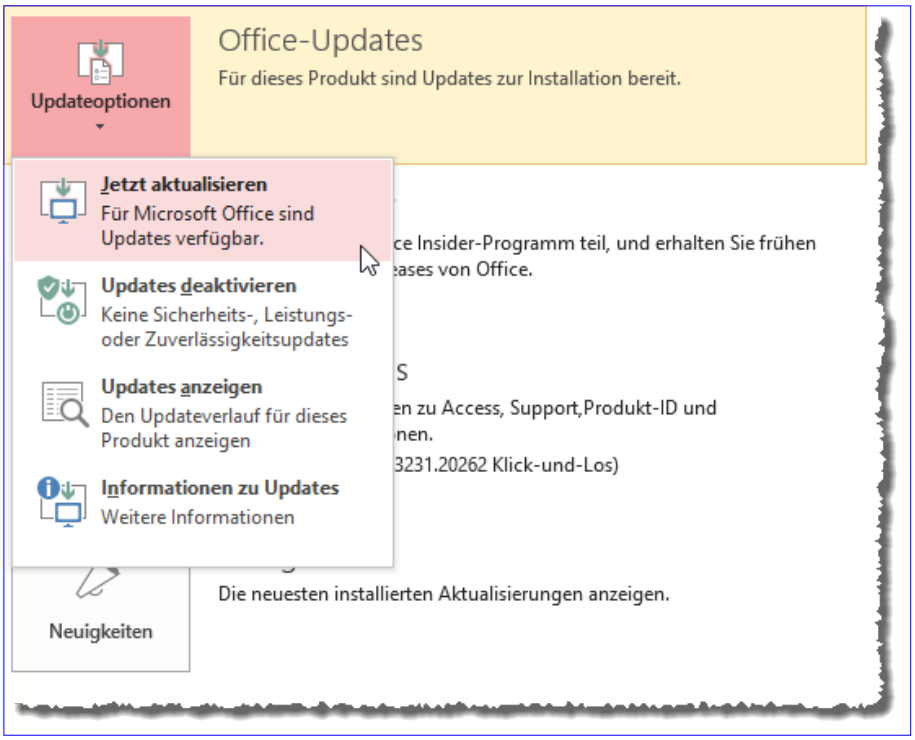



**Bild 2:** Update-Optionen

## Keine automatischen Updates

Mit der Option **Updates deaktivieren** legen Sie fest, dass Updates nicht mehr automatisch heruntergeladen und installiert werden sollen.

Vielmehr ist es dann Ihre Aufgabe, mit einem Klick auf **Jetzt aktualisieren** ein Update zu starten.

## Updates anzeigen

Ein Klick auf die Schaltfläche **Updates anzeigen** öffnet eine Webseite, welche die Neuerungen des zuletzt eingespielten Updates anzeigt.

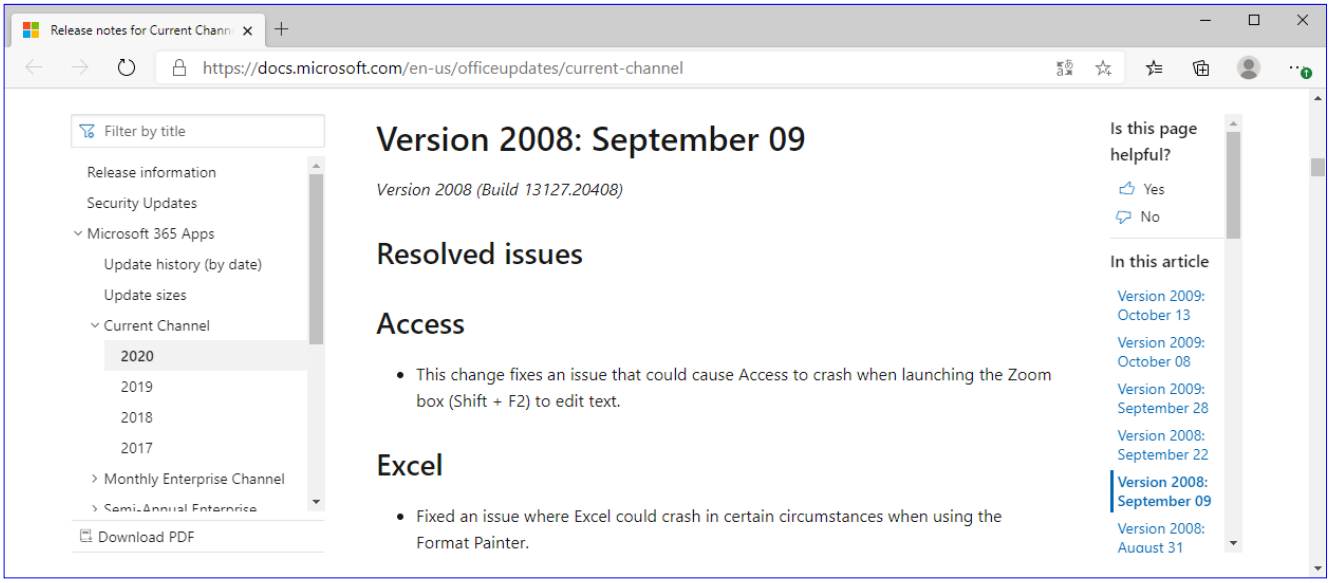



**Bild 3:** Versionsinformationen der letzten Updates

# Numerische Autowerte einstellen

**Autowertfelder verwendet man, um sich keine Sorgen mehr um die automatisch vergebenen Werte für Primärschlüsselfelder machen zu müssen. Manchmal geraten die Werte dieser Felder allerdings aus dem Takt oder Sie möchten selbst festlegen, welchen Wert der nächste Datensatz erhalten soll. Dieser Beitrag zeigt, wie Sie die Autowerte in Ihren Tabellen im Griff haben.**

Die meisten Tabellen in Access-Datenbanken (und auch in anderen Datenbanksystemen) verwenden einen automatisch hochzählenden Wert in den Primärschlüsselfeldern. Unter Access stellen Sie dazu einfach den Eintrag **Autowert** als **Felddatentyp** ein. Legen Sie dann einen neuen Datensatz an, ermittelt Access für das Autowertfeld den größten bisher vergebenen Wert in diesem Feld und addiert eins hinzu.

Der größte bisher vergebene Wert muss dabei aber nicht der größte Wert sein, den die Tabelle anzeigt. Wenn Sie einer Tabelle einen Datensatz hinzufügen, der im Autowertfeld den Wert **2** erhält, und Sie löschen diesen Wert anschließend wieder, wird der nächste neue Datensatz dennoch mit dem Wert **3** versehen (siehe Bild 1).

Wenn Sie möchten, dass nach dem Löschen des neuesten Datensatzes mit der ID **2** der nächste neue Datensatz wieder den Wert **2** erhält, können Sie die Datenbank komprimieren. Dann wird die Information, welche ID zuletzt vergeben wurde, zurückgesetzt.

Sie sollten das Design des Datenmodells jedoch so wählen, dass es nicht nötig ist, den Primärschlüsselwert mit bestimmten Werten zu füllen – dieser sollte nur zur eindeutigen Kennzeichnung eines Datensatzes dienen und um über Fremdschlüsselfelder Beziehungen zwischen Tabellen herzustellen.

### Datensatz mit bestimmtem Primärschlüsselwert unter DAO anlegen

Gelegentlich kann es jedoch dazu kommen, dass ein Datensatz einen bestimmten Primärschlüsselwert

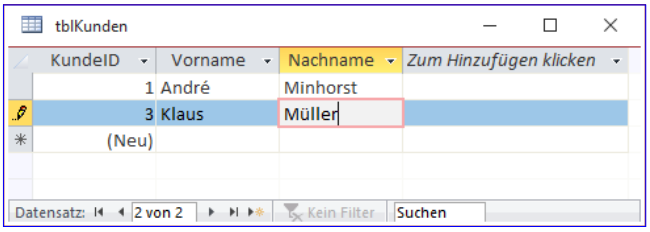



**Bild 1:** Wurde der neueste Datensatz gelöscht, wird die ID nicht nochmals vergeben.

erhalten soll. Beim Hinzufügen von Datensätzen über die Benutzeroberfläche können Sie dies nicht beeinflussen. Dazu benötigen Sie eine **INSERT INTO**-Abfrage, welche nicht nur die Werte der übrigen Felder, sondern auch den Wert für das Primärschlüsselfeld übergibt. Wenn Sie im Beispiel aus dem Screenshot einen Datensatz mit der Nummer 2 anlegen wollen, benötigen Sie die folgende Abfrage:

```
INSERT INTO tblKunden(KundeID, Vorname, Nachname)
VALUES(2, 'Hermann', 'Meier')
```

Diese Abfrage führen Sie beispielsweise als Parameter der **Execute**-Methode des **Database**-Objekts aus:

```
Public Sub BeispielAutowert()
    Dim db As DAO.Database
    Set db = CurrentDb
    db.Execute "INSERT INTO tblKunden(
                        KundeID, Vorname, Nachname)
            VALUES(2, 'Hermann', 'Meier')", dbFailOnError
End Sub
```

Hier ist jedoch zu beachten, dass Access sich nun den Wert für das Autowertfeld als den zuletzt vergebenen

# Autowertfelder: Long durch GUID ersetzen

**In den meisten Fällen ist die Wahl des Datentyps »Long« für ein Autowertfeld sinnvoll. Da es die Standardeinstellung ist, denkt man vielleicht gar nicht über alternative Datentypen nach. Manchmal kann es jedoch Vorteile haben, den Datentyp GUID zu wählen. Dann landen keine durchnummerierten Zahlen im Primärschlüsselfeld, sondern Zeichenketten im Format einer GUID. Die Nutzung der GUID ist vor allem dann anzuraten, wenn mehrere Datenbanken parallel existieren, deren Daten gelegentlich zusammengeführt werden. In diesem Fall kann man die Primärschlüsselwerte einfach weiterverwenden und muss sich nicht darum kümmern, dass der gleiche Primärschlüsselwert in mehreren Datenbanken vorkommt und angepasst werden muss.**

Wie oben beschrieben, wird der Datentyp **Long** standardmäßig für neue Autowert-Felder vorgeschlagen. Darüber macht man sich in der Regel auch keine Sorgen, bis man feststellt, dass die Daten über mehrere Datenbanken verteilt bearbeitet und zu bestimmten Zeitpunkten zusammengeführt werden sollen.

Im ungünstigsten Fall enthalten die Tabellen dann schon eine Menge Datensätze, was ein Ändern des Datentyps unmöglich macht.

Wie kann man in diesem Fall dennoch den Datentyp der Primärschlüsselfelder ändern, ohne die vorhandenen Daten zu verlieren – mit Ausnahme der vorhandenen **Long**-Werte in den Primärschlüsselfeldern, die ja durch GUID-Werte ersetzt werden sollen? Die Vorgehensweise hängt davon ab, ob es sich nur um eine oder mehrere Tabellen handelt, die nicht miteinander in Beziehung stehen, oder ob es Beziehungen zwischen den Tabellen der Datenbank gibt.

### Nicht verknüpfte Tabellen aktualisieren

Handelt es sich nur um nicht verknüpfte Tabellen, sieht die Vorgehensweise wie folgt aus:

- Sie fügen ein neues Feld hinzu, das als neues Primärschlüsselfeld im GUID-Format mit Autowert-Funktion dienen soll.
- Sie füllen das neue Feld mit GUID-Werten.
- Sie löschen das vorhandene Primärschlüsselfeld.
- Sie benennen gegebenenfalls das GUID-Feld so um, dass es den Namen des vorherigen Primärschlüsselfeldes erhält.
- Sie machen das neue GUID-Feld zum Primärschlüsselfeld.

Die Vorgehensweise ist recht einfach und lässt sich über die Benutzeroberfläche schnell abbilden. Zunächst schauen wir uns noch die Vorgehensweise für verknüpfte Tabellen an.

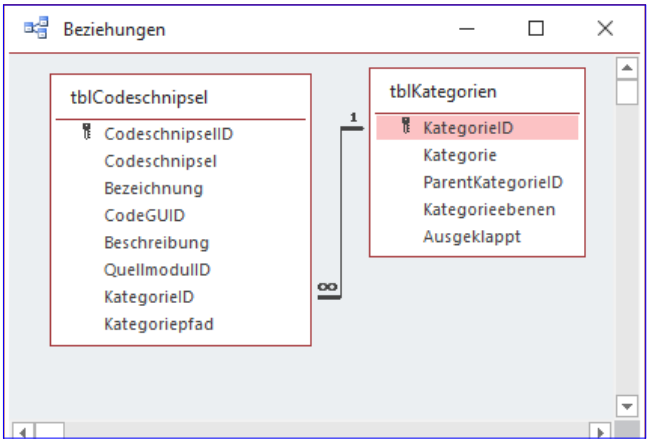



**Bild 1:** Beispieltabellen mit Beziehung

## Verknüpfte Tabellen aktualisieren

Der Aufwand für verknüpfte Tabellen ist ungleich höher. Angenommen, wir haben zwei verknüpfte Tabellen namens **tblCodeschnipsel** und **tblKategorien** (siehe Bild 1).

Hier ist die Tabelle **tblCodeschnipsel** über das Feld **KategorieID** mit dem Primärschlüsselfeld **KategorieID** der Tabelle **tblKategorien** verknüpft.

In diesem Fall sind die folgenden Schritte nötig, um die Datentypen der Primärschlüsselfelder der beiden Tabellen zu ändern:

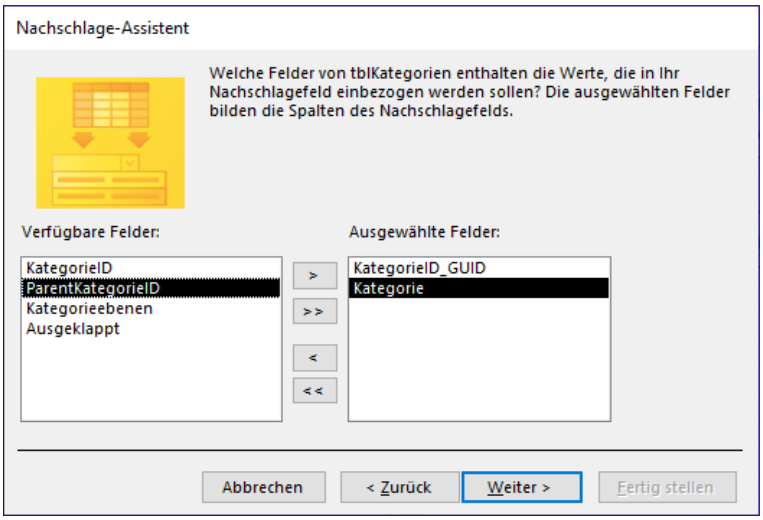



**Bild 2:** Einrichten eines neuen Nachschlagefeldes

- Hinzufügen jeweils eines neuen Feldes zu den beiden Tabellen, das als neues GUID-Primärschlüsselfeld dienen soll – beispielsweise mit den vorübergehenden Bezeichnungen **CodeschnipselID_GUID** und **KategorieID_GUID**.

- Hinzufügen eines neuen Fremdschlüsselfeldes namens **KategorieID_GUID** zur Tabelle **tblCodeschnipsel**.

- Einrichten des Feldes **KategorieID_GUID** als Feld mit dem Datentyp **Zahl** und der Feldgröße **Replikation-ID**. Man könnte zwar versuchen, ein Nachschlagefeld mit der Tabelle **tblKategorien** und den Feldern **KategorieID_GUID** und **Kategorie** als Ziel einrichten (siehe Bild 2). Allerdings erkennt der Nachschlage-Assistent automatisch, dass **KategorieID** das Primärschlüsselfeld der Tabelle ist und nicht **KategorieID_GUID** – und verwendet auch das Feld **KategorieID** als Primärschlüsselfeld der Beziehung. Also definieren wir das Nachschlagefeld später.

- Füllen der neuen, als Primärschlüssel vorgesehenen Felder mit GUID-Werten

- Füllen des Fremdschlüsselfeldes **KategorieID_GUID** der Tabelle **tblCodeschnipsel** mit den GUID-Werten, die dem GUID-Wert des Feldes der Tabelle **tblKategorien** entsprechen, das eigentlich mit dem Datensatz der Tabelle **tblCodeschnipsel** verknüpft ist.

- Löschen der alten Primärschlüsselfelder und Fremdschlüsselfelder

- Umbenennen der neuen Autowertfelder und Fremdschlüsselfelder von **CodeschnipselID_GUID** in **CodeschnipselID** und **KategorieID_GUID** in **KategorieID**.

- Einstellen der neuen Autowertfelder als Primärschlüsselfelder

- Einstellen des Fremdschlüsselfeldes **KategorieID** der Tabelle **tblCodeschnipsel** als Nachschlagefeld mit dem Feld **KategorieID** der Tabelle **tblKategorien** als Ziel.

Nachdem wir die Vorgehensweise theoretisch beleuchtet haben, schauen wir uns die Praxis an.

## Praktisches Umwandeln nicht verknüpfter Tabellen

Zum Umwandeln einer nicht verknüpften Tabelle, hier der Tabelle **tblTest**, fügen wir zunächst das neue Feld namens **TestID_GUID** hinzu.

# Das AutoKeys-Makro

**Makros sind unter Microsoft Access eine stiefmütterlich behandelte Objektart. In der Tat ist die Programmierung einer Datenbank mit VBA wesentlich flexibler und mächtiger. Allerdings gibt es Anwendungszwecke, die sich mit VBA nicht abdecken lassen. Hier kann man auf zwei spezielle Makros zugreifen: Das AutoExec-Makro wird beim Start einer Anwendung aufgerufen und kann zum Anstoßen von Aktionen verwendet werden, die beim Start benötigt werden. Das AutoKeys-Makro nimmt die Definition von Tastenkombinationen auf, die während des Betriebs der jeweiligen Access-Anwendung zur Verfügung stehen. Das letztere Makro schauen wir uns in diesem Beitrag im Detail an.**

Technisch gesehen ist das **AutoKeys**-Makro ein Makro wie jedes andere. Der wichtigste Unterschied ist, dass es unter dem Namen **AutoKeys** gespeichert wird. Nur dieses Makro liest Access beim Ausführen von Tastenkombinationen aus. Und auch hier gibt es eine Ausnahme – die schauen wir uns weiter unten an. Die im **AutoKeys**-Makro definierten Tastenkombinationen stehen in der ganzen Anwendung zur Verfügung. Sie stehen damit im Gegensatz zu den Ereignissen **Bei Taste ab** und **Bei Taste auf** von Formularen, die nur dann ausgelöst werden, wenn das jeweilige Formular den Fokus hat.

## AutoKeys-Makro anlegen

Zum Anlegen des **AutoKeys**-Makros betätigen Sie den Ribbon-Eintrag **ErstellenIMakros und CodeIMakro**.

Danach erscheint bereits der Entwurf eines neuen Makros. Wenn Sie bereits unter älteren Access-Versionen Makros programmiert haben und speziell das **AutoKeys**-Makro, dann vermissen Sie in der Entwurfsansicht vermutlich die **Name**-Spalte, in der üblicherweise die Tastenkombination angegeben wird, auf die das Makro reagieren soll (siehe Bild 1).

Unter Access 2016 beispielsweise fügen Sie daher ein Untermakro zum Makro hinzu (siehe Bild 2).

Der Name des Untermakros entspricht dann dem Wert in der Spalte **Name** im Makroentwurf früherer Access-Versionen.

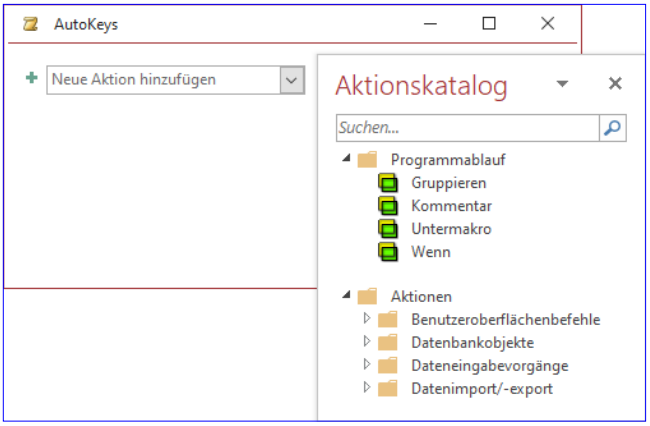



**Bild 1:** Ein Makro ohne sichtbare **Name**-Spalte

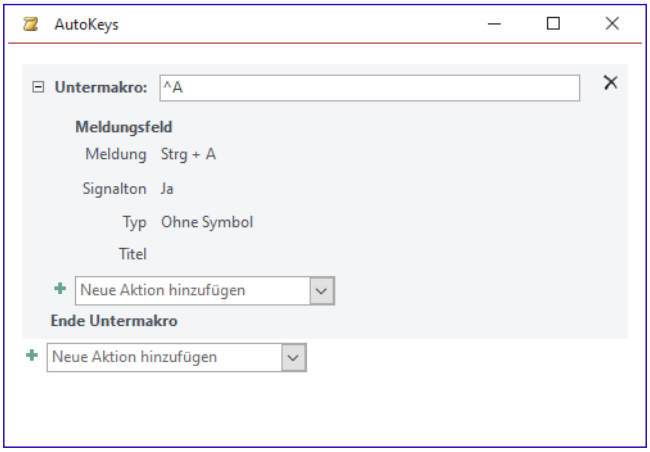



**Bild 2:** Das Makro mit Untermakro

### Festlegen der Tastenkombination

Hier legen wir bereits die Tastenkombination fest. Grundsätzlich kombiniert man immer mindestens eine der beiden Tasten **Strg** oder **Umschalt** mit einer anderen Taste. Die Verwendung der **Alt**-Taste ist nicht möglich. Die Buchstaben- und Zahlentasten können dabei nur mit einer dieser beiden Tasten kombiniert werden, die anderen Tasten auch ohne **Strg** oder **Umschalt**.

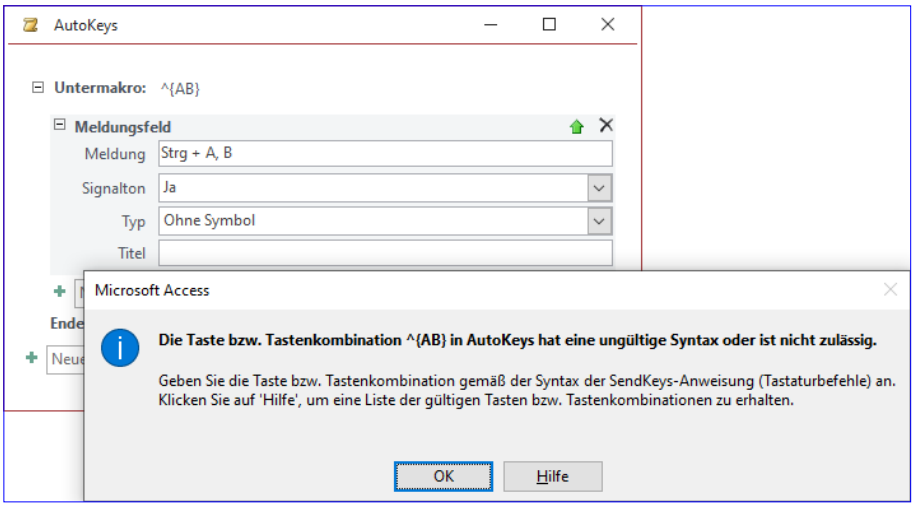



**Bild 3:** Fehlermeldung beim Versuch, eine ungültige Tastenkombination zu speichern

Für die Kombination mit **Strg** und **Umschalt** verwenden Sie zu Beginn eines der folgenden Kürzel:

- **^**: **Strg**
- **+**: **Umschalt**
- **^+**: **Strg + Umschalt**

Hier fügen Sie nun die gewünschte weitere Taste hinzu. Es stehen die folgenden Tasten zur Auswahl:

- **{F1}** bis **{F12}**: Funktionstasten **F1** bis **F12**
- **{INSERT}**: Einfügen-Taste
- **{DEL}**, **{DELETE}**: Entfernen-Taste
- **{A}** bis **{Z}**, **{0}** bis **{9}**: Tasten **A** bis **Z** und **0** bis **9**

Sie können nur jeweils eine dieser Tasten mit **Strg** und/oder **Umschalt** kombinieren.

Die Zeichen **A** bis **Z** und **0** bis **9** können Sie nur in Kombination mit **Strg** oder **Strg + Umschalt** verwenden, aber nicht nur mit der **Umschalt**-Taste – was sinnvoll ist, da Sie sonst in der Anwendung zum Beispiel keine großen Buchstaben eingeben könnten.

Diese Zeichen brauchen Sie nicht unbedingt in geschweifte Klammern zu setzen. Bei den anderen Tasten wie den Funktionstasten et cetera, deren Bezeichnung aus mehr als einem Zeichen besteht, müssen Sie hingegen die Bezeichnung in geschweifte Klammern einfassen. Praktischerweise gibt Access beim Versuch, eine nicht gültige Kombination mit **Strg + S** zu speichern, eine Fehlermeldung aus (siehe Bild 3).

### Beispiele für Tastenkombinationen

Nachfolgend einige Beispiele für Tastenkombinationen:

- **^{F1}**: **Strg + F1**
- **^+{F2}**: **Strg + Umschalt + F2**
- **+{F3}**: **Umschalt + F3**
- **{F4}**: **F4**
- **^{DEL}**: **Strg + Entfernen**
- **^q** oder **^{q}**: **Strg + Q**

# Tastenkombinationen mit AutoHotKeys

**Access bietet einige Tastenkombinationen, mit denen sich die wichtigsten Aufgaben schnell erledigen lassen. Manche Tastenkombination wurde aber über die Jahre geändert – zum Beispiel können Sie seit einigen Access-Versionen nicht mehr mit der Tastenkombination Strg + Tab zwischen den aktuell geöffneten Access-Objekten wechseln. Stattdessen erledigen Sie diese Aufgabe nun mit der Tastenkombination Strg + F6. Eine Kombination, die nur für Menschen mit halbwegs großen Händen mit einer Hand greifbar ist. Da man während der Entwicklung gern einmal von einem Objekt zum nächsten wechselt und Strg + Tab sehr gut dafür geeignet war, wollen wir diesen Zustand wiederherstellen. Wie das gelingt und warum die Bordmittel von Access dafür nicht ausreichen, zeigt dieser Beitrag.**

## Unsinniger Wechsel?

Was sich die Microsoft-Entwickler dabei gedacht haben, die Tastenkombination **Strg + Tab** zum Wechseln zwischen den geöffneten Access-Objekten durch die Kombination **Strg + F6** zu ersetzen, kann man nur mutmaßen. Mit **Strg + Tab** wechseln Sie nun jedenfalls von einem Steuerelement zum nächsten, was allerdings auch nur mit der Tabulator-Taste möglich ist. Möglicherweise hat **Strg + Tab** in einer anderen Office-Anwendung eine andere Bedeutung und man wollte Kompatibilität zwischen den Anwendungen herstellen.

## Ziel dieses Beitrags

In diesem Beitrag wollen wir die von Microsoft vorgenommene Änderung jedenfalls wieder rückgängig machen und dafür sorgen, dass Sie mit **Strg + Tab** wieder zwischen den geöffneten Access-Objekten wechseln können. Das probieren wir natürlich zunächst mit Bordmitteln. Uns schwebte vor, ein **AutoKeys**-Makro zu erstellen, das die Tastenkombination **Strg + Tab** abfängt und stattdessen die Tastenkombination **Strg + F6** abschickt.

## AutoKeys-Makro

Gesagt, getan. Also haben wir ein **AutoKeys**-Makro angelegt – was nichts anderes ist als ein Makro, das unter dem Namen **AutoKeys** abgespeichert wird und dadurch beim Start von Access nach der Definition von Tastenkombinationen durchsucht wird, die im Laufe dieser Session Gültigkeit haben sollen. Wie Sie ein **AutoKeys**-Makro anlegen, erfahren Sie übrigens im Beitrag **Das AutoKeys-Makro** (**www.access-im-unternehmen.de/1271**).

Hier haben wir allerdings schnell festgestellt, dass uns hier nur eine beschränkte Anzahl von Tastenkombinationen zur Verfügung steht. Davon abgesehen müssten wir das **AutoKeys**-Makro, das die Tastenkombination von **Strg + Tab** auf **Strg + F6** umleitet, auch erst einmal zu allen Access-Anwendungen hinzufügen, in denen wir diese Tastenkombination nutzen wollen.

Allerdings wäre es kein Problem gewesen, ein Add-In zu programmieren, mit dem sich dieses Makro per Menübefehl zu einer Access-Datenbank hinzufügen lässt.

## Alternative: AutoHotKeys

Es gibt jedoch ein weit verbreitetes Tool für die Bereitstellung von Tastenkombinationen namens **AutoHotKeys**. Dieses können Sie kostenlos unter folgender Internetadresse herunterladen:

```
autohotkey.com
```

Wir schauen uns in den nächsten Abschnitten zunächst an unserem konkreten Beispiel an, wie wir ein **AutoHotKey**-

Makro anlegen und werfen dann einen Blick darauf, was wir noch mit diesem Tool anstellen können.

## Installation

Die Installation von **AutoHotkey** erfolgt über das Setup. Hier können Sie einfach die Express-Installation auswählen.

Damit werden einige Vorannahmen automatisch getroffen – zum Beispiel, ob Sie die 32-Bit- oder die 64-Bit-Version benötigen. Außerdem ist in unserem Fall direkt die **Unicode**-Version eingestellt (siehe Bild 1).

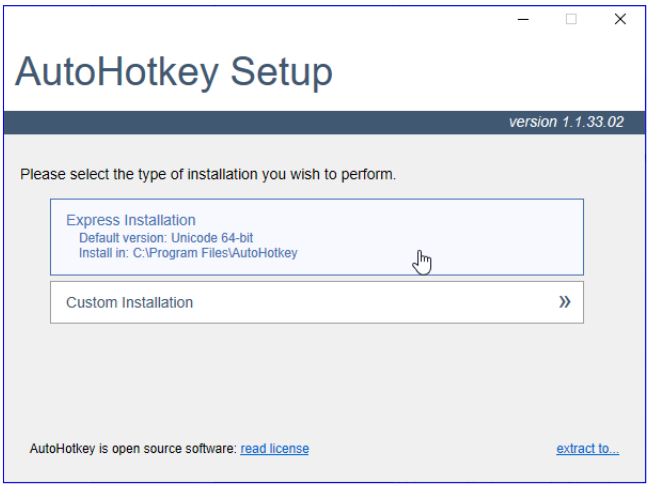



**Bild 1:** Das Setup von **AutoHotkey**

## Ein erstes Skript erstellen

Das erste Skript erstellen wir einfach auf dem Desktop. Hier klicken Sie mit der rechten Maustaste auf einen freien Bereich und wählen aus dem Kontextmenü den Eintrag **Neu|AutoHotkey Script** aus (siehe Bild 2).

Die neue Datei erscheint dann auf dem Desktop und Sie können den Namen anpassen – behalten Sie unbedingt die Dateiendung **.ahk** bei!

Danach öffnen Sie die Datei in einem beliebigen Texteditor. Die Datei enthält bereits ein paar Befehle, wie Bild 3 zeigt. Diese Befehle behalten wir zunächst bei, da es sich um empfohlene Anweisungen für einen performanten und zuverlässigen Betrieb handelt. Darunter können wir dann eigene Befehle hinzufügen.

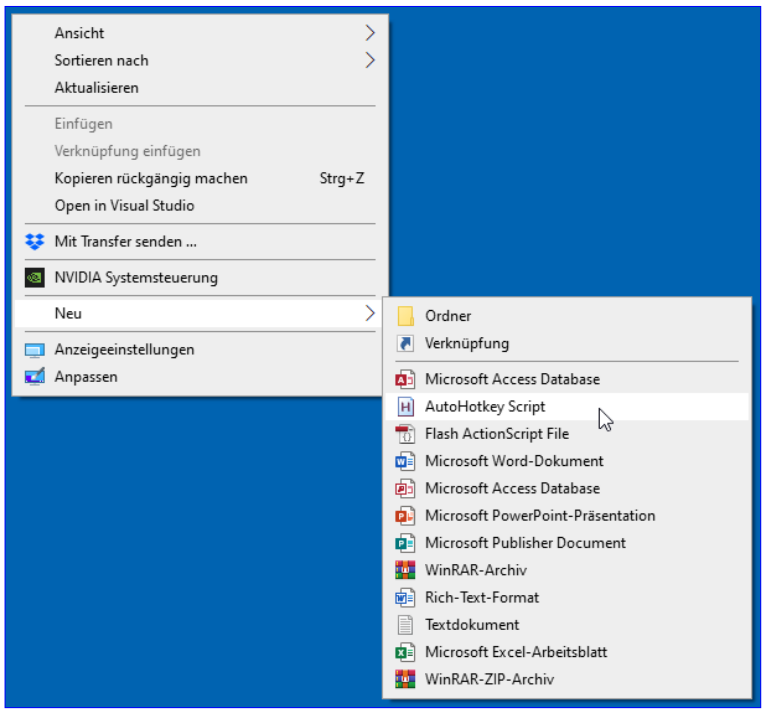



**Bild 2:** Anlegen einer Skriptdatei

## Von Strg + Tab zu Strg + F6

Wir wollen als erstes Beispiel ein Skript erstellen, das die Tastenkombination **Strg + Tab** abfängt und stattdessen die Tastenkombination **Strg + F6** ausführt – also das Verhalten von **Strg + Tab** abbilden, wie es in früheren Versionen von Access üblich war.

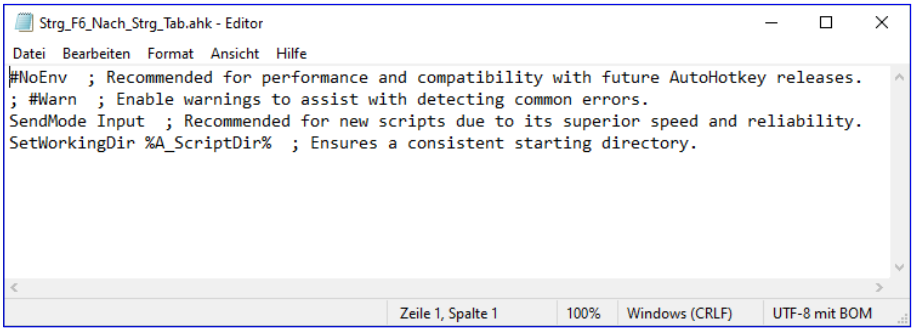



**Bild 3:** Anzeigen der neuen Skriptdatei

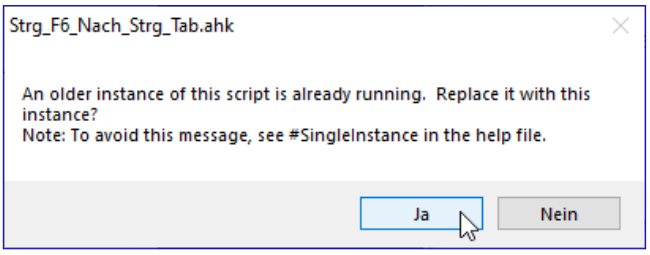



**Bild 4:** Meldung beim erneuten Starten eines laufenden Skripts

Dazu legen wir einfach die folgenden Zeilen in unserer Skriptdatei an:

```
^TAB::
Send ^{F6}
return
```

Die erste Zeile gibt an, auf welche Tastenkombination das Skript reagieren soll. In diesem Fall verwenden wir **^TAB**, wobei **^** für **Strg** und **TAB** für die Tabulator-Taste steht. Mit den beiden Doppelpunkten trennen wir die Tastenkombination, auf die das Skript lauschen soll, von den auszuführenden Aktionen.

In der zweiten Zeile folgt die **Send**-Anweisung gefolgt von **^{F6}**. Die **Send**-Anweisung erwartet die Zeichen beziehungsweise Tastenkombinationen als Parameter, die nun gesendet werden sollen. In diesem Fall verwenden wir wieder das Zeichen für **Strg** (**^**) und den Code für die Taste **F6**, nämlich **{F6}**.

Die letzte Zeile enthält den Befehl **return**, der lediglich dafür sorgt, dass alles, was hinter diesem Befehl steht, nicht mehr ausgeführt wird.

Speichern Sie dieses Skript und führen Sie es aus, indem Sie das Skript doppelt anklicken. Danach können Sie eine Access-Anwendung öffnen, mindestens zwei Access-Objekte öffnen und dann die Tastenkombination **Strg + Tab** ausprobieren. Das Ergebnis: Sie wechseln zwischen den geöffneten Access-Objekten hin und her! Das fühlt sich doch gleich viel ergonomischer an als der umständliche Griff zur Tastenkombination **Strg + F6**.

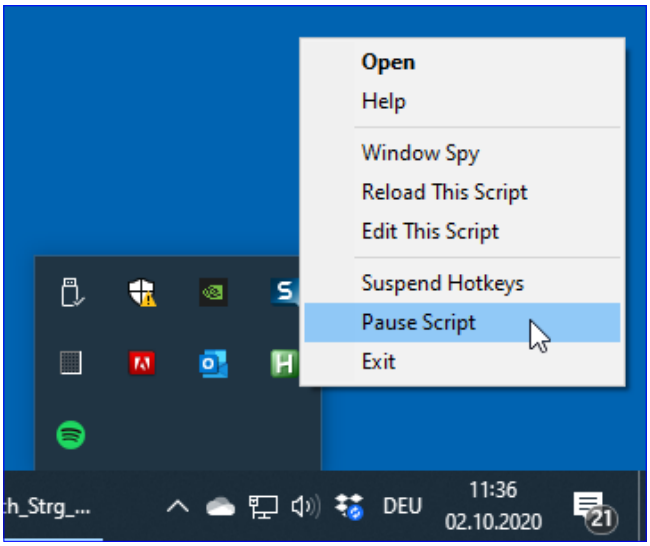



**Bild 5:** Beenden eines Skripts

## Beenden eines AutoHotkey-Skripts

Wenn Sie ein **AutoHotkey**-Skript beenden wollen, klicken Sie mit der rechten Maustaste auf das **AutoHotkey**-Symbol in der Taskleiste und wählen Sie dort den Eintrag **Pause Script** aus (siehe Bild 4).

## Erneutes Starten eines AutoHotkey-Skripts

Wenn Sie ein **AutoHotkey**-Skript nach dem Speichern einer Änderung erneut starten wollen und dazu doppelt auf die Skript-Datei klicken, erscheint die Meldung aus Bild 5.

Diese können Sie unterbinden, indem Sie den Befehl **#SingleInstance** mit einem der folgenden Parameter hinzufügen:

- **Prompt** (Standardeinstellung): Zeigt die genannte Meldung an.
- **Force**: Beendet die laufende Instanz und startet das Skript erneut.
- **Ignore**: Startet das Skript nicht neu, sondern behält das laufende Skript bei.
- **Off**: Startet das neue Skript und lässt das alte weiterlaufen.

# .accdb läuft, .accde aber nicht

**Haben Sie das schon einmal erlebt? Sie haben Ihre Anwendung fertig programmiert und alles läuft einwandfrei. Sie müssen nur noch die Kompilierung in eine .accde-Datenbank vornehmen, damit der Kunde keinen Zugriff auf den Quellcode hat. Auch das gelingt ohne Probleme. Kaum starten Sie die .accde-Datenbank, geht jedoch plötzlich nichts mehr: Es funktionieren einfach keine Ereignisprozeduren mehr, und falls vorhanden, meckert auch das Ribbon, dass es seine Callback-Funktionen nicht mehr ausführen kann. Was ist hier geschehen und wie lösen wir das Problem? Das zeigen wir in diesem Artikel.**

Manchmal liefert Access Phänomene, die man sich einfach nicht erklären kann. In meiner neuesten Anwendung, die das Ribbon nutzt, erscheint zum Beispiel nach der Umwandlung in eine **.accde**-Datenbank die Meldung aus Bild 1.

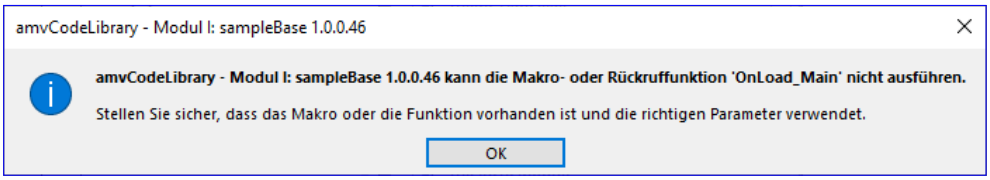



**Bild 1:** Fehlermeldung in einer neu erstellten **.accde**-Datenbank

Was genau passiert hier? Das konnten wir nicht exakt herausfinden. Uns war an dieser Stelle wichtig, überhaupt eine Lösung zu finden.

Und die gab es: Offensichtlich entstehen diese Probleme mit **.accde**-Datenbanken, wenn sich in irgendeinem Klassenmodul eines Formulars oder Berichts noch eine leere Ereignisprozedur befindet, die beispielsweise wie folgt aussieht – also so, wie direkt nach der Erstellung:

```
Private Sub Form_Load()

End Sub
```

Leider ist es nicht so einfach, das Problem so nachzustellen – es tritt nämlich nicht immer so auf. Offensichtlich gibt es noch andere Faktoren, die das Problem »aktivieren«. Wir können daher keine Beispieldatenbank anbieten, die dieses Problem aufzeigt. Dennoch ist es für Sie vermutlich hilfreich, den Inhalt dieses Artikels im Hinterkopf zu behalten ... wer weiß, ob und wann Sie einmal dieses Phänomen in einer Ihrer Datenbankanwendungen vorfinden.

## Problemlösung

Die Lösung ist ganz einfach: Sie brauchen lediglich die leeren Ereignisprozeduren aus Ihrer Anwendung zu entfernen. Es reicht auch aus, wenn Sie eine beliebige Zeile in die Prozedur einfügen – und wenn es eine Kommentarzeile ist.

Das hört sich allerdings leichter an, als es ist, denn Sie müssen ja zunächst einmal herausfinden, ob sich überhaupt eine leere Ereignisprozedur in der Anwendung befindet und falls ja, wo diese liegt.

Dazu haben wir eine Prozedur programmiert, die alle Prozedurnamen samt Modul ausgibt, die weder Code noch Kommentare enthalten.

Damit die Prozedur funktioniert, benötigen wir einen Verweis auf die Bibliothek **Microsoft Visual Basic for Applications Extensibility 5.3 Object Library** (siehe Bild 2).

# SQL Server-Security, Teil 4: Schutz mit Datenbankrollen

**Bernd Jungbluth, Horn**

**In Teil 4 dieser Beitragsreihe wurden die Zugriffsrechte der Anwender auf die Datenbank begrenzt. In der Datenbank jedoch gibt es für die Anwender keine Grenzen. Durch die Zuordnung zur Datenbankrolle db_owner besitzen sie dort administrative Rechte. Dies erlaubt ihnen u.a. das Anlegen und Löschen von Tabellen sowie das Lesen und Ändern aller Daten. Wie Sie die Rechte mit Hilfe der Datenbankrollen weiter eingrenzen, lesen Sie in diesem Beitrag.**

## Warnung

Die beschriebenen Aktionen haben Auswirkungen auf Ihre SQL Server-Installation. Führen Sie die Aktionen nur in einer Testumgebung aus. Verwenden Sie unter keinen Umständen Ihren produktiven SQL Server!

In der aktuellen Beispielumgebung verwendet der Anwender, sprich die Person am Rechner, zur Authentifizierung am SQL Server die Anmeldung **WaWiMa**. Durch die Zuordnung dieser Anmeldung zum gleichnamigen Benutzer in der Datenbank **WaWi_SQL** ist der Anwender für den Zugriff auf diese Datenbank autorisiert. Innerhalb der Datenbank erlaubt die Mitgliedschaft des Benutzers zur Datenbankrolle **db_owner** dem Anwender das Lesen und Ändern der Daten. Soweit in Kurzform die Beschreibung der derzeitigen Rechtevergabe.

Datenbank-Administratoren nutzen die Datenbankrolle **db_owner** gerne, um Anwendern innerhalb einer Datenbank administrative Tätigkeiten zu erlauben. Auf diese Weise lassen sich Aufgaben an einen oder mehrere Anwender übertragen. Zum Beispiel an die Mitarbeiter des Supports. Sie geben neuen Kollegen die entsprechenden Rechte für den Datenzugriff und kümmern sich um die Datenbanksicherung. Datenbankobjekte wie Tabellen, gespeicherte Prozeduren und Sichten müssen sie jedoch weder anlegen, noch ändern oder gar löschen. Das sind eher die Tätigkeiten der Datenbankentwickler. Diese findet man in der Praxis ebenfalls oft in der Datenbankrolle **db_owner**. Für die Installation neuer Datenbankversionen brauchen Sie die Rechte zum Anlegen, Ändern und Löschen von Datenbankobjekten. Allumfassende Lese- und Schreibzugriffe sind dafür allerdings nicht erforderlich.

Obwohl es sich in beiden Fällen um administrative Tätigkeiten handelt, haben die Anwender dennoch zu viele Rechte. Es gibt keinen Grund für eine Mitgliedschaft in der Datenbankrolle **db_owner**. Weder für den Support, noch für die Datenbankentwickler, noch für irgendeinen anderen Anwender. Schließlich stellt SQL Server noch neun weitere Datenbankrollen zur Verfügung. Diese reichen fürs Erste, um den Anwendern die Rechte zu geben, die sie für die Realisierung ihrer Aufgaben benötigen. Und falls diese wider Erwarten nicht genügen, können Sie eigene Datenbankrollen anlegen und dort die Rechte gezielt vergeben.

Die Datenbankrolle **db_owner** hingegen sollte nur ein einziges Mitglied enthalten: den Datenbankbesitzer, kurz den Benutzer **dbo** (siehe Bild 1). Dieser Benutzer ist standardmäßig in der Datenbankrolle eingetragen und lässt sich nicht entfernen. Die Unterschiede der Datenbankrolle **db_owner** zum Datenbankbesitzer **dbo** wurden bereits in Teil 3 dieser Beitragsserie beschrieben.

## Rechte für den Datenbankentwickler

Datenbankentwickler erstellen neue Versionen bestehender Datenbanken in ihrer Entwicklungsumgebung und installieren das Ergebnis im produktiven SQL Server.

Dies erfolgt über eigens dafür erzeugte SQL-Skripte oder über die Möglichkeiten der SQL Server Data Tools.

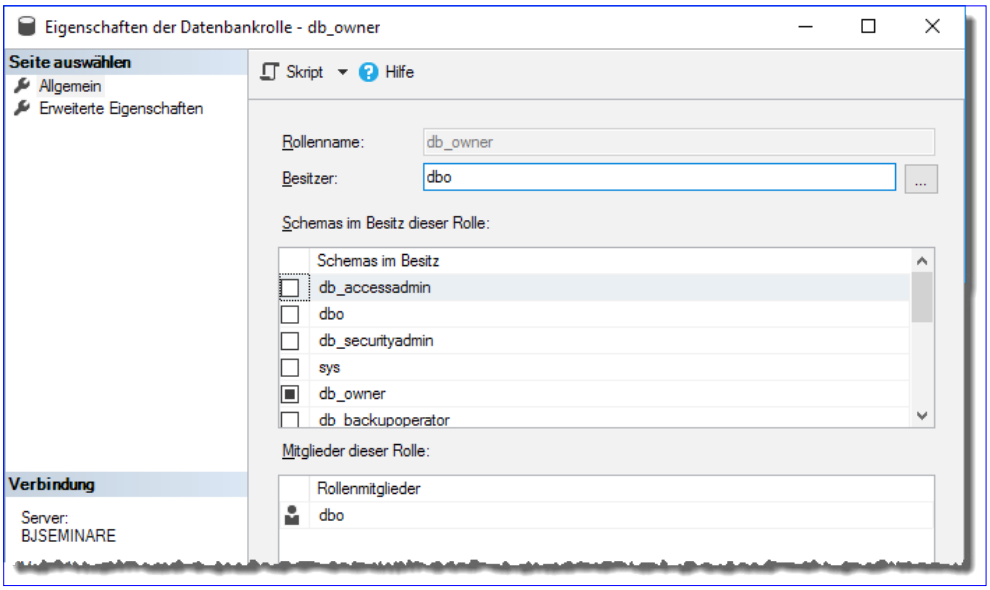



**Bild 1:** Die optimale Datenbankrolle **db_owner**

Für die Installation der Änderungen sind nur eine Handvoll Rechte notwendig. Neue Objekte werden mit **CREATE** angelegt, bestehende mit **ALTER** geändert und nicht mehr benötigte mit **DELETE** gelöscht. Bei diesen SQL-Anweisungen handelt es sich um DDL-Befehle. DDL steht für Data Definition Language – und die Datenbankrolle **db_ddladmin** erlaubt ihren Mitgliedern die Ausführung dieser Befehle. Der Datenbankentwickler muss also nur dieser Datenbankrolle zugeordnet werden und schon hat er die notwendigen Rechte zur Bereitstellung einer neuen Datenbankversion.

Beinhaltet die neue Datenbankversion eine detaillierte Rechtevergabe auf den Datenbankobjekten, braucht der Datenbankentwickler weitere Rechte. Mit **GRANT** vergibt er Rechte, mit **REVOKE** entzieht er diese und mit **DENY** verbietet er den Zugriff. Die Rechte für diese Befehle sind nicht in der Datenbankrolle **db_ddladmin** enthalten. Dafür gibt es die Datenbankrolle **db_securityadmin**. Ein Datenbankentwickler benötigt somit keine Zuordnung zur Rolle **db_owner**. Die beiden Datenbankrollen **db_ddladmin** und **db_securityadmin** sind für seine Aufgaben vollkommen ausreichend (siehe Bild 2).

Möglicherweise trägt der Datenbankentwickler in einer Tabelle die neue Versionsnummer der Datenbank ein. Dazu braucht er Lese- und Schreibrechte. Jedoch nur an der entsprechenden Tabelle. Die Rechtevergabe an Tabellen wird später beschrieben.

## Rechte für den Support

Eine Datenbank gehört zu einer Applikation. Ob es sich hierbei um eine Datenbank zu einer IT-Inventursoftware, einer Lagerverwaltung oder einem ERP-System handelt,

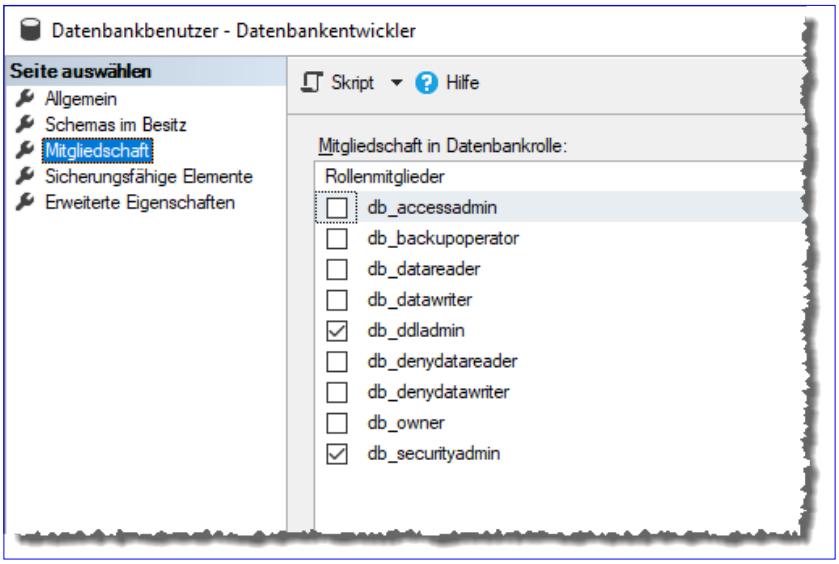



**Bild 2:** Datenbankrollen für Datenbankentwickler

die Anwender verwalten die Daten immer über ein zugehöriges Frontend. Für die Applikation selbst steht ein Support zur Verfügung. Dieser klärt Fragen, analysiert und behebt Fehler, verwaltet Zugriffsberechtigungen und führt Datenbanksicherungen durch.

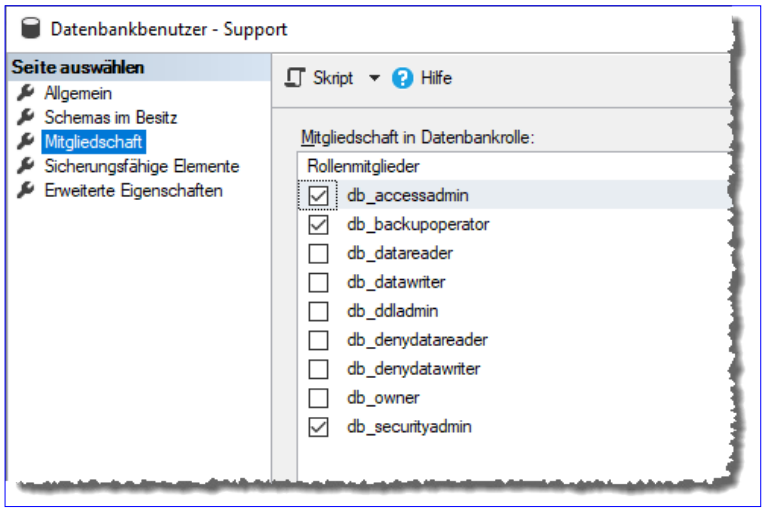



**Bild 3:** Datenbankrollen für Supportmitarbeiter

Einige Applikationen beinhalten eine eigene Benutzerverwaltung. Hier trägt der Support den neuen Kollegen als Anwender für die Applikation ein, worauf im SQL Server ein neuer Benutzer in der Datenbank mit entsprechenden Rechten angelegt wird. Vorausgesetzt, der Datenbankadministrator hat bereits die Anmeldung angelegt, mit der sich der neue Kollege am SQL Server authentifiziert. Diese Aufgabe findet außerhalb der Datenbank statt und obliegt somit weiterhin dem Datenbankadministrator. Innerhalb der Datenbank benötigt der Support die entsprechenden Rechte zur Verwaltung der Benutzer und deren Zugriffsrechte. Als Mitglied der bereits erwähnten Datenbankrolle **db_securityadmin** darf der Support die Zugriffsrechte direkt an den Datenbankobjekten wie auch über eigene Datenbankrollen vergeben. Für die Rechte zum Anlegen, Ändern und Löschen von Benutzern benötigt er zusätzlich noch die Zuordnung zur Datenbankrolle **db_accessadmin**.

Die Möglichkeit zur Sicherung und Wiederherstellung der Daten gehört bei vielen Applikationen ebenfalls zum Funktionsumfang. Hierzu benötigt der Support die Rechte zur Ausführung der T-SQL-Befehle **BACKUP** und **RESTORE**. Diese Rechte beinhaltet die Datenbankrolle **db_backupoperator**.

Als Mitglied dieser drei Datenbankrollen besitzt der Support alle Rechte die er für seine Aufgaben benötigt (siehe Bild 3). Die Zuordnung zur Datenbankrolle **db_owner** ist nicht erforderlich.

Sie fragen sich, wie der Support Fehler analysieren soll, wenn er die Daten der Datenbank nicht lesen darf. Die Antwort ist recht einfach. Die Fehleranalyse findet nicht in der Produktivdatenbank, sondern in einer Testdatenbank statt.

Dort kann der Support anhand von Testdaten das Szenario nachstellen und die Ursache für den Fehler finden. Durch diese Trennung verhindern Sie ein versehentliches Ändern der Echtdaten.

## Datenbankrolle public

Die Datenbankrolle mit der Zwangsmitgliedschaft. Jeder Benutzer einer Datenbank ist der Datenbankrolle **public** zugeordnet. Diese Zuordnung ist fest vorgegeben und lässt sich nicht ändern. Der ursprüngliche Sinn dieser Datenbankrolle liegt in der Definition von Zugriffsrechten, die für jeden Benutzer mindestens erforderlich sind. Eine solche Art der Rechtevergabe birgt jedoch einige Sicherheitsrisiken. Diese lernen Sie in einem der nächsten Beiträge kennen. Am besten folgen Sie der bewährten Sicherheitsempfehlung und ignorieren die Datenbankrolle **public**.

## Lese- und Schreibrechte

Es bleiben noch vier Datenbankrollen übrig. Diese beziehen sich auf die Lese- und Schreibrechte – die Rechte für die SQL-Anweisungen **SELECT**, **INSERT**, **UPDATE** und **DELETE**.

Mitglieder der Datenbankrolle **db_datareader** dürfen alle Daten der Datenbank lesen und Mitglieder der Datenbankrolle **db_datawriter** alle Daten ändern und löschen sowie neue hinzufügen. Für die meisten Anwender der Datenbank ist eine Zuordnung zu diesen beiden Datenbankrollen absolut ausreichend. Anwender, die Daten lediglich auswerten, benötigen sogar nur die Zuordnung zur Datenbankrolle **db_datareader**.

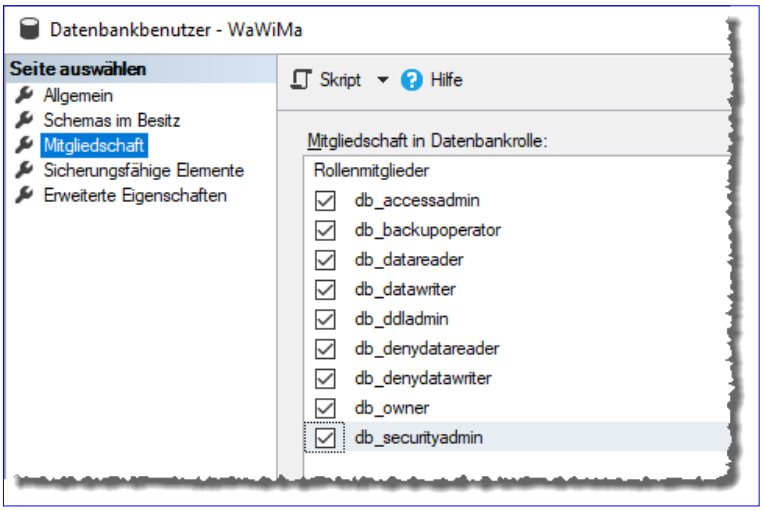



**Bild 4:** Viel hilft viel?

Nun haben Sie einen Überblick über die Rechte der einzelnen Datenbankrollen. Datenbankadministratoren ohne dieses Wissen wählen am Anfang gerne alle Datenbankrollen aus (siehe Bild 4).

Viel hilft viel – eine Aussage, die in diesem Fall nicht stimmt. Eine **SELECT**-Anweisung auf eine x-beliebige Tabelle der Datenbank liefert bei einer solchen Definition die Meldung aus Bild 5.

Mit dieser Rechtevergabe werden dem Benutzer nicht nur Rechte gewährt, sondern auch verweigert. Die Zuordnung zu den Datenbankrollen **db_denydatareader** und **db_denydatawriter** verbietet dem Benutzer jegliche Lese- und Schreibzugriffe. Im Hintergrund der beiden Datenbankrollen steht der Befehl **DENY**. Ein Verweigern per **DENY** überwiegt alle erteilten Rechte, sogar die der Datenbankrolle **db_owner**.

Die Auswahl aller Datenbankrollen ist nicht selten der Grund, warum die zum Benutzer gehörende Anmeldung letztendlich in der Serverrolle **sysadmin** landet. Für Mitglieder dieser Serverrolle gibt es im SQL Server keine Grenzen. Da verhindert auch kein **DENY** den Datenzugriff. Jedoch beinhaltet diese Zuordnung jegliche Rechte an den Datenbanken und vor allem an den dort gespeicherten Daten. Schneller können Sie Ihr mühsam erstelltes Sicherheitskonzept nicht zerstören.

Fehlende Zugriffsrechte in einer Datenbank lösen Sie bitte innerhalb der Datenbank. Die Serverrolle **sysadmin** ist in solchen Fällen für Sie tabu.

Wählen Sie die Datenbankrollen bewusst aus. Dies gilt insbesondere für die Datenbankrollen **db_denydatareader** und **db_denydatawriter**. Die beiden sollten Sie nur nutzen, wenn Sie den Lese- beziehungsweise Schreibzugriff unter allen Umständen verhindern möchten.

Mögliche Kandidaten für die Datenbankrolle **db_denydatawriter** sind Anwender, die Daten lediglich analysieren und auswerten. Um sicherzustellen, dass sie definitiv keine Daten hinzufügen, ändern oder löschen können, ordnen Sie den zugehörigen Benutzer der Datenbankrolle **db_denydatawriter** zu. Auf diese Weise können die Anwender in Access die Daten der SQL Server-Datenbank über eingebundene Tabellen auswerten, ohne Gefahr zu laufen, die Daten versehentlich zu ändern oder zu löschen.

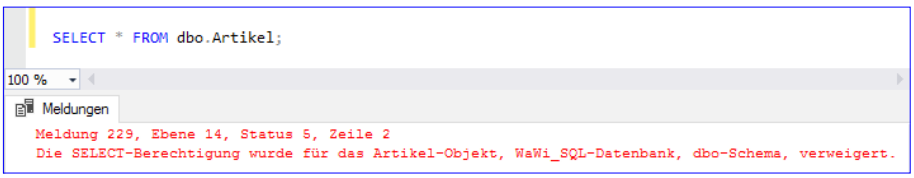



**Bild 5:** Die Folgen von »Viel hilft viel«

Wozu die Mühe? Reicht es nicht, wenn der Benutzer nur zur Datenbankrolle **db_datareader** gehört? Grundsätzlich schon. Erhält der Benutzer aber versehentlich das Recht zum Anlegen von Daten in einer Tabelle, verhindert dessen Mitgliedschaft in **db_denydatawriter** dieses Schreibrecht. Dies gilt ebenso für eine irrtümliche Zuordnung des Benutzers zu **db_datawriter** oder gar zu **db_owner**.

Das klingt recht sicher und durchdacht – zum Zeitpunkt der Rechtevergabe. Zu einem späteren Zeitpunkt ist das Konzept der Mitgliedschaft zu den Datenbankrollen **db_denydatareader** und **db_denydatawriter** vielleicht nicht mehr so einfach nachzuvollziehen. Deshalb sollten Sie sich den Grund für die Verwendung dieser beiden Datenbankrollen gut dokumentieren.

Angenommen, Sie müssen einem Anwender den lesenden Zugriff auf eine Tabelle gewähren. Das ist schnell getan. Sie gehen zur Tabelle, wählen dort über die Berechtigungen den zum Anwender gehörenden Benutzer aus und erteilen ihm das Recht für die **SELECT**-Anweisung. Bei dieser Art der Rechtevergabe sehen Sie die bestehende Zuordnung des Benutzers zur Datenbankrolle **db_denydatareader** nicht. Sie werden auch nicht darauf hingewiesen. Der Anwender möchte nun die Daten der Tabelle lesen. Er erhält jedoch nur Fehlermeldungen. Jetzt kommt Hektik auf, denn das Problem muss wie immer schnell gelöst werden. Sie überprüfen die Datenbankrollen des Benutzers und sehen die Zuordnung zur Datenbankrolle **db_denydatareader**. Dazu gab es bestimmt mal einen guten Grund. Nur ohne Dokumentation fehlt Ihnen die Argumentation, warum Sie diese Zuordnung nicht so ohne weiteres ändern können. Um das Problem zu lösen, ordnen Sie den Benutzer der Datenbankrolle **db_owner** zu. Dann kann er erst mal arbeiten. Um die tatsächlichen Rechte kümmern Sie sich später. Doch es bleibt bei den Fehlermeldungen, denn die Zuordnung zur Datenbankrolle **db_denydatareader** überwiegt den Rechten der Datenbankrolle **db_owner**. Eine Lösung muss her und zwar schnell. Also ordnen Sie die Anmeldung des Benutzers der Serverrolle **sysadmin** zu. Das Problem ist gelöst, denn mit diesen Rechten darf der Anwender im SQL Server alles – sogar die Daten der vorgesehenen Tabelle lesen. Natürlich ist diese Zuordnung nur vorübergehend. Sobald Sie Zeit haben, die Rechtevergabe neu zu überdenken und anzupassen, werden Sie die Anmeldung aus der Serverrolle **sysadmin** wieder entfernen. Dabei sollten Sie die Zuordnung zur Datenbankrolle **db_owner** nicht vergessen. Die Frage, wann Sie die Zeit haben und wie lange dieses vorübergehend anhält, dürfen Sie sich gerne selbst beantworten. Dieses Beispiel ist nicht erfunden. Vielmehr zeigt es den Grund, warum in der Praxis Anmeldungen in der Serverrolle **sysadmin** sowie Benutzer in der Datenbankrolle **db_owner** zu finden sind.

Erstellen Sie sich eine Dokumentation Ihres Berechtigungskonzepts. Diese beinhaltet nicht nur die Mitgliedschaft der Benutzer in den Datenbankrollen, sondern auch den Grund für diese Zuordnungen. Jegliche Anforderung, die Änderungen an den Zugriffsrechten erfordert, prüfen Sie zuerst gegen Ihre Dokumentation. Passen die Anforderungen nicht zu Ihrem Berechtigungskonzept, wehren Sie sich so gut wie möglich gegen die geforderten Rechte.

Sie haben den Überblick zur aktuellen Rechtevergabe und Sie als Datenbankadministrator sind verantwortlich für die Daten. Ob nun Daten versehentlich gelöscht oder bewusst von einem Anwender manipuliert wurden, Sie müssen gute Argumente haben, warum Sie dies nicht verhindern konnten. Mit dem Argument, Sie haben die Rechte nur vorübergehend erteilt, um die Anforderungen der Abteilung zu erfüllen, werden Sie nicht weit kommen. Sie sind der Fachmann und es gehört zu Ihren Aufgaben, die möglichen Folgen einer Rechtevergabe einzuschätzen und zu bewerten.

Es ist wie immer bei der Definition von Zugriffsrechten. Das Berechtigungskonzept sollte gut durchdacht und dokumentiert sein. Diese Dokumentation ist Ihr Leit-

faden. Jede neue Anforderung an Zugriffsrechte muss mit Ihrem Berechtigungskonzept vereinbar sein. Ist dies nicht der Fall, stellen Sie das aktuelle Konzept in Frage und definieren Sie bei Bedarf ein neues.

Zum Schluss ein kleiner Tipp für Ihr Berechtigungskonzept: Die Zuordnung eines Benutzers zu den Datenbankrollen **db_datareader** und **db_denydatareader** sowie **db_datawriter** und **db_denydatawriter** oder gar zu allen vieren ist schlicht und ergreifend sinnlos.

## Lese- und Schreibrechte für WaWiMa

Genug der Theorie. In der Beispieldatenbank **WaWi_SQL** ist der Benutzer **WaWiMa** aktuell der Datenbankrolle **db_owner** zugeordnet. Diese zu hohen Rechte werden Sie nun eingrenzen.

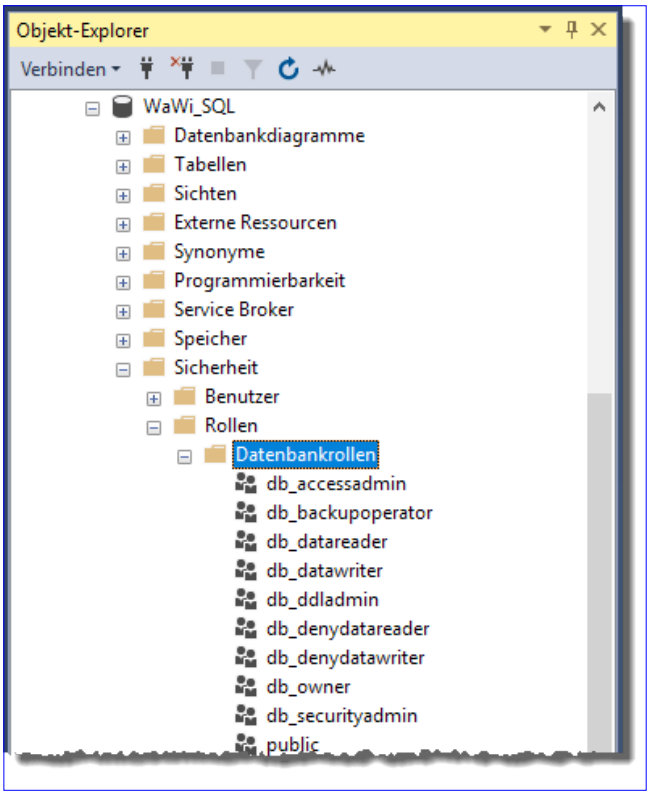



**Bild 6:** Datenbankrollen einer Datenbank

Mit diesem Beitrag gibt es eine neue Version der SQL Server-Beispieldatenbank und der Access-Applikation. Sollten Sie die Beispiele bereits installiert haben, löschen Sie diese und installieren Sie die neuen Versionen. Eine Installationsanleitung finden Sie am Ende des Artikels.

Öffnen Sie das SQL Server Management Studio und melden Sie sich an Ihrem SQL Server als Datenbankadministrator an. Danach erweitern Sie den Ordner der Datenbank **WaWi_SQL** und gehen zum Ordner **Sicherheit**. Hier gibt es jetzt zwei Möglichkeiten zur Rechtevergabe per Datenbankrollen. Sie können den Benutzer in den jeweiligen Datenbankrollen hinzufügen beziehungs-

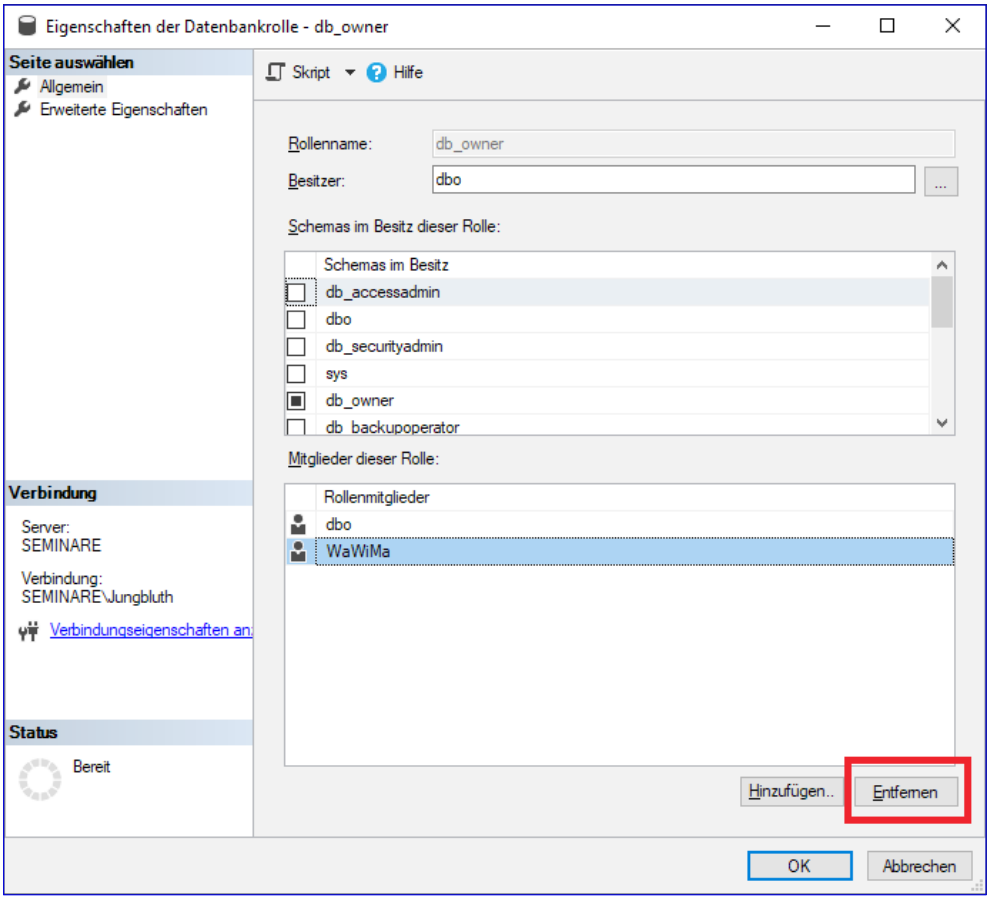



**Bild 7:** Verwalten der Mitglieder einer Datenbankrolle

weise entfernen oder beim Benutzer selbst die Zuordnung vornehmen.

Schauen Sie sich zunächst den Weg über die Datenbankrollen an. Dazu erweitern Sie den Ordner **Rollen** und danach den Ordner **Datenbankrollen** (siehe Bild 6). Öffnen Sie per Doppelklick den Dialog zur Datenbankrolle **db_owner**.

Dort können Sie unter Mitglieder dieser Rolle den Eintrag **WaWiMa** markieren und mit einem Klick auf die Schaltfläche Entfernen (siehe Bild 7) die Mitgliedschaft beenden. Danach besitzt der Benutzer **WaWiMa** in der Datenbank keinerlei Zugriffsrechte mehr. Sollten Sie dies in einer produktiven Umgebung vorhaben, stellen Sie sich schon einmal auf diverse Anrufe und E-Mails ein.

Eine weitaus bessere Vorgehensweise ist die Änderung der Mitgliedschaft über den Benutzer. Wechseln Sie dazu zum Ordner **Benutzer** und öffnen Sie mit einem Doppelklick auf **WaWiMa** den Dialog **Datenbankbenutzer**. In der Seite **Mitgliedschaft** sehen Sie die aktuelle Zuordnung des Benutzers zu den Datenbankrollen.

Entfernen Sie das Häkchen bei der Datenbankrolle **db_owner** und aktivieren Sie die Datenbankrollen **db_datareader** und **db_datawriter** (siehe Bild 8). Mit einem Klick auf die Schaltfläche **OK** bestätigen Sie die neuen Zugriffsrechte. Der Benutzer **WaWiMa** besitzt in der Datenbank nun keine administrativen Rechte mehr. Er darf nur noch Daten lesen und ändern.

In der Access-Beispielapplikation **WaWi** hat die Änderung der Zugriffsrechte keine Auswirkung. Die Verbindung zum SQL Server wird weiterhin über die Anmeldung **WaWiMa** hergestellt. Diese Anmeldung ist in der Datenbank **WaWi_SQL** dem gleichnamigen Benutzer zugeordnet. Durch die Mitgliedschaft des Benutzers in den Datenbankrollen **db_datareader** und **db_datawriter** darf der Anwender die Daten der Datenbank lesen und ändern.

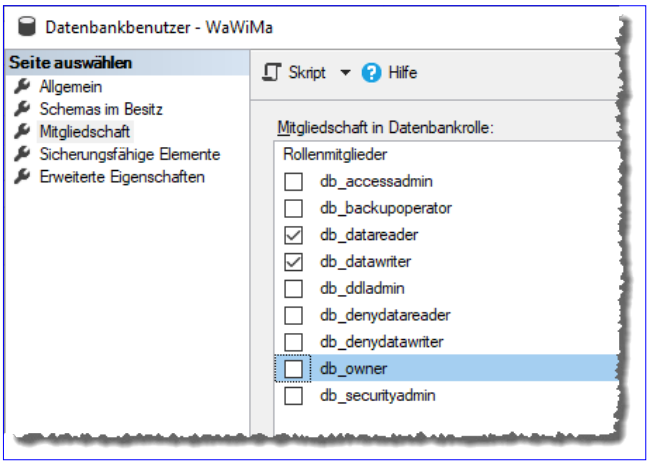



**Bild 8:** Verwalten der Datenbankrollen eines Benutzers

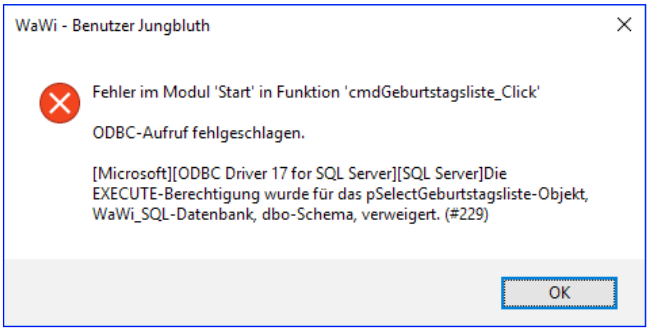



**Bild 9:** Fehler beim Datenzugriff

Testen Sie es einfach mal. Öffnen Sie die Beispielapplikation **WaWi**. Sie werden keinen Unterschied feststellen. Trotz der geringeren Rechte sehen Sie alle Daten.

Ein Klick auf die Schaltfläche **Kunden** liefert Ihnen die Daten der Tabelle **Kunden**. Sie können diese Daten nicht nur lesen, Sie dürfen auch neue Daten hinzufügen sowie bestehende ändern und löschen.

Die Anmeldung **WaWiMa** erlaubt Ihnen den Zugriff auf alle Daten der Datenbank **WaWi_SQL**. Ob nun per eingebundener Tabelle wie im Formular **Aufträge** oder per ADO wie im Formular **Artikel**, die Art und Technik des Datenzugriffs spielt dabei keine Rolle.

Es gibt jedoch Ausnahmen, wie ein Klick auf die Schaltfläche **Geburtstagsliste** zeigt (siehe Bild 9).

## EXECUTE-Rechte für WaWiMa

Das Formular **Geburtstagsliste** basiert auf der Pass Through-Abfrage **ptSelectGeburtstagsliste**, die wiederum die gespeicherte Prozedur **dbo.pSelectGeburtstagsliste** ausführt. Hier liegt die Ursache für den Fehler, genauer gesagt bei den Rechten zum Ausführen der gespeicherten Prozedur.

Der Benutzer **WaWiMa** besitzt mit der Zuordnung zur Datenbankrolle **db_datareader** das Recht für die SQL-Anweisung **SELECT** zum Lesen der Daten und mit der Datenbankrolle **db_datawriter** die Rechte zum Ändern der Daten per **INSERT**, **UPDATE** und **DELETE**. Keine der beiden Datenbankrollen beinhaltet das Recht für die SQL-Anweisung **EXECUTE** zur Ausführung von gespeicherten Prozeduren. Ergo muss der Benutzer **WaWiMa** zusätzlich das Recht für **EXECUTE** erhalten.

Bitte denken Sie jetzt nicht an die Datenbankrolle **db_owner** oder gar an die Serverrolle **sysadmin**. Wie schon erwähnt, wäre dies zwar eine schnelle Lösung des Problems, aber auch die mit Abstand schlechteste. Es ist nicht erforderlich, dem Anwender alle Rechte zu geben. Ihm fehlt ja nur ein Recht – das Recht für die SQL-Anweisung **EXECUTE**.

Microsoft empfiehlt den Einsatz von gespeicherten Prozeduren unter anderem aus Gründen der besseren Performance und der Datensicherheit. Da ist der Gedanke naheliegend, dass SQL Server eine entsprechende Datenbankrolle anbietet. Leider gibt es eine solche Datenbankrolle nicht, obwohl sie seit vielen Jahren vermisst und auch immer wieder gefordert wird.

Sie müssen sich selbst helfen. Konkret bedeutet dies, dass Sie eine eigene Datenbankrolle für diese Rechtevergabe anlegen. Also zurück zum SQL Server Management Studio und dort zum Ordner **Rollen** der Datenbank **WaWi_SQL**. Hier klicken Sie mit der rechten Maustaste auf den Ordner **Datenbankrollen** und wählen den Eintrag **Neue Datenbankrolle** (siehe Bild 10).

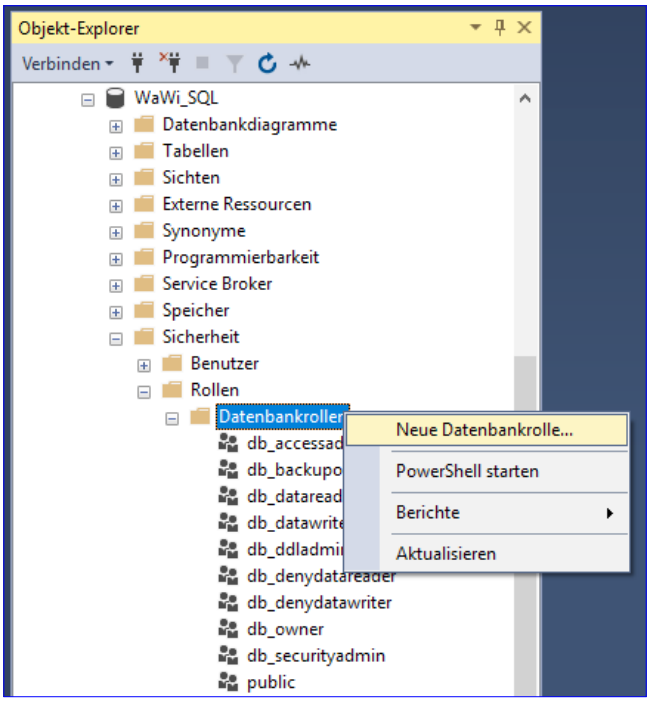



**Bild 10:** Eine eigene Datenbankrolle

Im Dialog **Datenbankrolle - Neu** geben Sie als erstes den Namen der Datenbankrolle im Feld **Rollenname** ein. Der Name ist frei wählbar. Sie können die Nomenklatur der fixen Datenbankrollen beibehalten und die Bezeichnung **db_execute** verwenden oder Sie legen für Ihre Datenbankrollen eine eigene Namensgebung fest. Wie wäre es mit der Bezeichnung **edb_execute**?

Die neue Datenbankrolle soll wie alle anderen Objekte der Datenbank dem Datenbankbesitzer gehören. Tragen Sie dazu **dbo** in das Feld **Besitzer** ein. Nun klicken Sie auf die Schaltfläche **Hinzufügen** und wählen im darauffolgenden Dialog über Durchsuchen den Benutzer **WaWiMa** aus (siehe Bild 11). Bestätigen Sie die Auswahl mit einem Klick auf **OK**. Der Benutzer **WaWiMa** ist jetzt Mitglied Ihrer Datenbankrolle.

Das war der erste Teil der Definition – und der einfachste. Jetzt müssen Sie noch jede Ihrer gespeicherten Prozeduren der Datenbankrolle zuordnen und diesen dann einzeln die SQL-Anweisung **EXECUTE** erlauben. Dazu wechseln Sie zur Seite **Sicherungsfähige Elemen-**

# Webcam-Bilder in Datenbank speichern

**Die Bilder einer Webcam kann man üblicherweise auf dem Rechner speichern, auf dem man diese erstellt hat. Gegebenenfalls geschieht das auch automatisch – so zum Beispiel mit der Kamera-App von Windows. Ein Leser fragte, ob sich das Aufnehmen von Bildern auch so steuern lässt, dass man die neuen Fotos direkt in einer Datenbank weiterverwenden kann – durch Speichern des Bildes oder des Speicherpfades in der Datenbank und das anschließende Anzeigen des Bildes direkt in einem Formular der Datenbank. Eine direkte Steuerung via VBA haben wir nicht realisiert, aber einen Workaround, der diese Aufgabe ebenso gut erfüllt.**

Die Aufgabe lautet also: Wir möchten von einem Access-Formular aus, das beispielsweise ein Produkt anzeigt, die Kamera-App von Windows öffnen, mit dieser ein Bild schießen und dieses dann in der Access-Anwendung so verfügbar machen, dass es direkt im aufrufenden Formular angezeigt werden kann.

Die Idee dazu lautet: Wir öffnen per Mausklick die Kamera-App und starten zum gleichen Zeitpunkt einen Formular-Timer, der alle paar Sekunden prüft, ob in einem bestimmten Ordner eine neue Datei hinzugefügt wurde. Ist das der Fall, soll die neue Datei als Bilddatei zum aktuellen Datensatz hinzugefügt werden.

Die Kamera-App speichert die Bilder nämlich in einem vorgegebenen Ordner (üblicherweise **c:\Users\<Benutzername>\Pictures\CameraRoll**).

Diesen können wir ändern oder ihn einfach übernehmen. Nachdem wir während der Ausführung der Timer-Ereignisprozedur ein neues Bild im angegebenen Ordner entdeckt haben, verarbeiten wir dieses und beenden das Auslösen des Timers. Außerdem soll das Bild nun als neues Bild im Formular angezeigt werden.

## Probleme mit der Kamera-App

Eines direkt vorab: Die Steuerung der Kamera-App ist ein wenig kompliziert und funktioniert mit der nachfolgend vorgestellten Methode nicht in allen Fällen perfekt. Zunächst einmal muss man wissen, dass die Kamera-App auch gestartet wird, wenn gar keine Kamera vorhanden ist – in diesem Fall erscheint zwar eine Anwendung, aber diese liefert lediglich die Meldung, dass keine Kamera gefunden werden konnte (siehe Bild 1). In diesem Fall

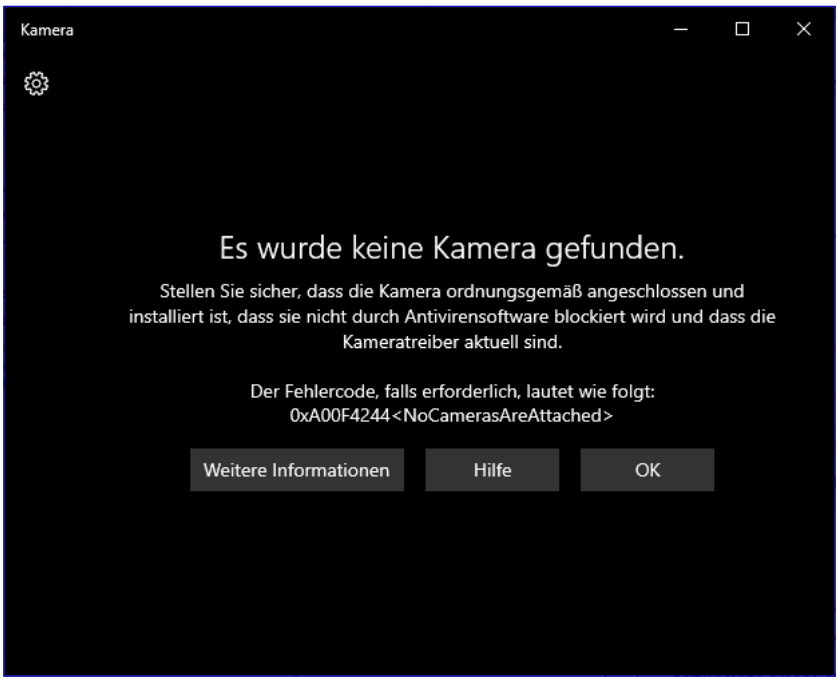



**Bild 1:** Meldung bei fehlender oder deaktivierter Kamera

wollen wir einfach abfangen, wenn der Benutzer die Meldung und somit die Kamera-App wieder schließt.

## Datenmodell der Anwendung

Für die Anwendung haben wir ein recht einfaches Datenmodell definiert. Wir wollen zu einem Datensatz der Tabelle **tblProdukte** ein oder mehrere Fotos in der Tabelle **tblBilder** speichern können und verknüpfen die Tabelle **tblBilder** wie in Bild 2 über das Fremdschlüsselfeld **ProduktID**.

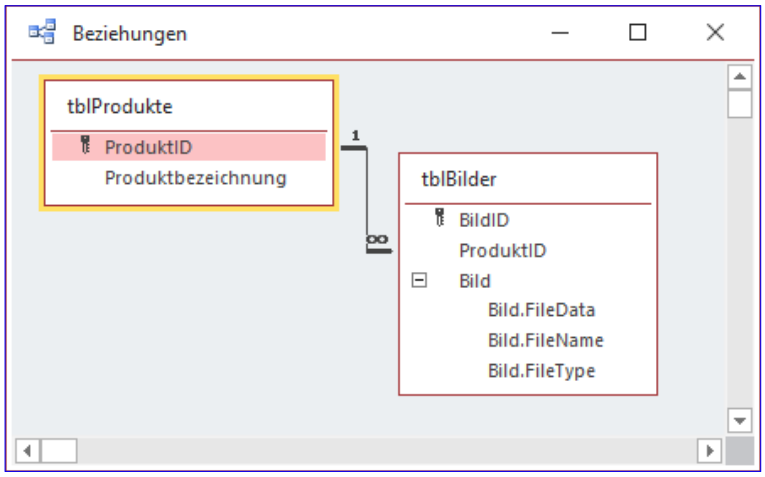



**Bild 2:** Datenmodell der Beispieldatenbank

Das Feld **Bild** der Tabelle **tblBilder** legen wir als Anlage-Feld an. Damit können wir darin nicht nur Bilder speichern, sondern diese auch gleich über das Anlage-Steuerelement im Formular anzeigen.

Neben diesen beiden Tabellen verwenden wir noch eine Tabelle namens **tblOptionen**, die nur ein Feld namens **Fotopfad** enthält.

## Formulare der Anwendung

Das erste Formular namens **frmOptionen** verwendet die Tabelle **tblOptionen** als Datensatzherkunft. Damit der Benutzer nur den vorhandenen Datensatz bearbeiten kann und weder neue Datensätze hinzufügen noch den bestehenden Datensatz löschen kann, stellen wir die Eigenschaften **Anfügen zulassen** und **Löschen zulassen** im Entwurf des Formulars auf **Nein** ein (siehe Bild 3).

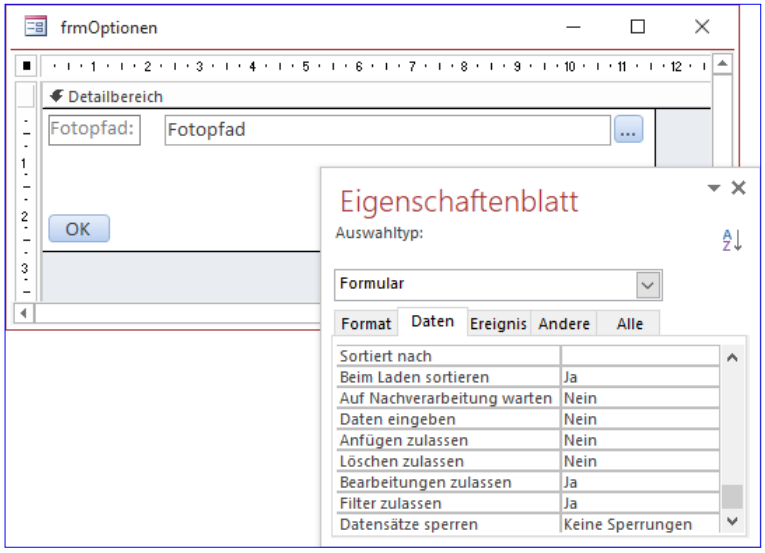



**Bild 3:** Entwurf des Formulars zum Auswählen des Fotopfades

Die Schaltfläche **cmdFotopfad** ruft die Prozedur aus Listing 1 auf. Diese verwendet die im Modul **mdlTools** definierte Funktion **OpenPathName**, um einen Dialog

```
Private Sub cmdVerzeichnisAuswaehlen_Click()
    Dim strFotopfad As String
    strFotopfad = OpenPathName(Nz(Me!txtFotopfad, CurrentProject.Path), "Fotoverzeichnis auswählen")
    If Not Len(strFotopfad) = 0 Then
        Me!txtFotopfad = strFotopfad
    End If
End Sub
```

**Listing 1:** Prozedur zum Auswählen eines Verzeichnisses

zum Auswählen eines Verzeichnisses anzuzeigen. Dabei übergibt sie als ersten Parameter den bisher in der Tabelle **tblOptionen** gespeicherten Ordner und als zweiten den anzuzeigenden Titel für den Dialog.

In der Formularansicht sieht das Formular wie in Bild 4 aus.

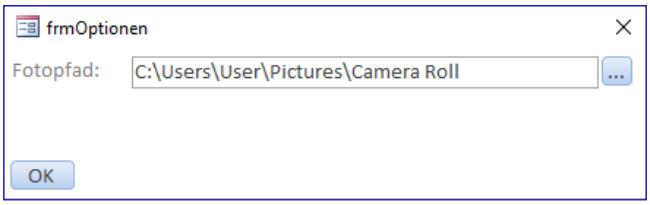



**Bild 4:** Dialog zum Einstellen der Optionen der Anwendung

## Formular zum Einlesen von Fotos

Das Formular zum Einlesen und Anzeigen der Fotos besteht aus einem Haupt- und einen Unterformular (siehe Bild 5).

Das Unterformular heißt **sfmProdukte** und verwendet die Tabelle **tblBilder** als Datensatzquelle und zeigt lediglich den Inhalt des Feldes **Bild** in einem **Anlage**-Steuerelement an. Nachdem wir dieses Formular angelegt und gespeichert haben, erstellen wir das Hauptformular.

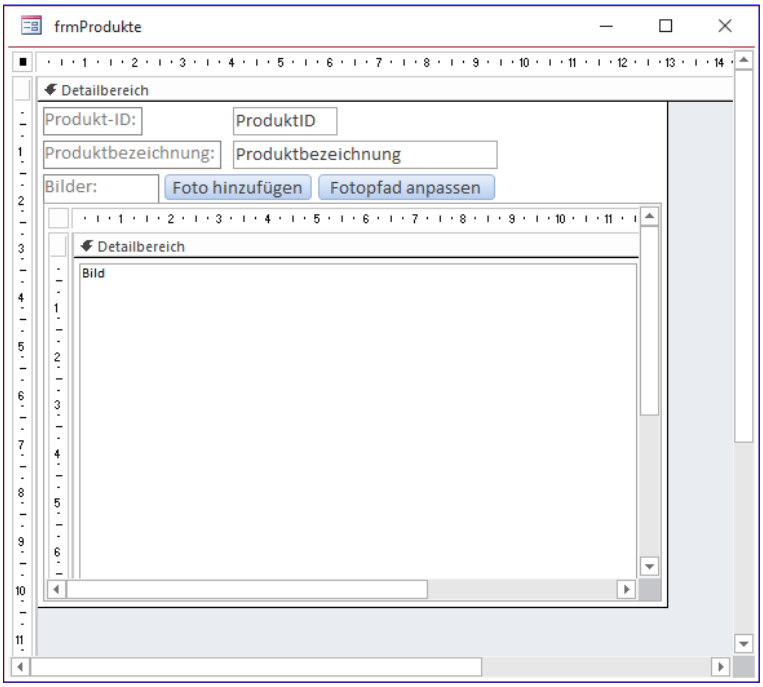



**Bild 5:** Entwurf des Formulars **frmProdukte**

In dieses ziehen wir das Unterformular **sfmProdukte**. Außerdem legen wir für das Formular **frmProdukte** die Tabelle **tblProdukte** als Datensatzquelle fest und fügen die beiden Felder **ProduktID** und **Produktbezeichnung** zum Detailbereich hinzu.

Über dem Unterformular platzieren wir die beiden Schaltflächen **cmdFotoHinzufuegen** und **cmdFotoPfadAnpassen**.

Letztere ruft die folgende Prozedur auf und öffnet so das Formular **frmOptionen** als modalen Dialog:

```
Private Sub cmdFotopfadAnpassen_Click()
    DoCmd.OpenForm "frmOptionen", WindowMode:=acDialog
End Sub
```

## Foto hinzufügen

Die Schaltfläche **cmdFotoHinzufuegen** soll dafür sorgen, dass die Kamera-App gestartet wird und dass nach dem Fotografieren die im Fotoverzeichnis gespeicherte neue Datei in die Tabelle **tblBilder** eingelesen und im Formular angezeigt wird.

Dazu benötigen wir eine modulweit deklarierte Variable, deren Funktion wir im Anschluss erläutern:

```
Dim datStart As Date
```

Die Prozedur, die durch einen Mausklick auf die Schaltfläche **Foto hinzufügen** ausgelöst wird, finden Sie in Listing 2. Diese füllt die Variable **datStart** mit dem Wert der Funktion **Now**, welche das aktuelle Datum und die aktuelle Uhrzeit liefert. Damit haben wir den Zeitpunkt gespeichert, an dem der Benutzer ein Foto aufnehmen möchte.

# Daten-Updates ausliefern

**Wenn Sie eine Anwendung bereitstellen wollen, die bereits Daten enthält und deren Daten sowohl von Ihnen als auch vom Benutzer der Anwendung erweitert werden sollen, müssen Sie einige Dinge beachten. Wir gehen zur Vereinfachung davon aus, dass nur neue Daten hinzukommen und keine vorhandenen Daten geändert werden – das würde die Aufgabe noch schwieriger gestalten. Als Beispiel soll eine Datenbank mit Produkten dienen, die sowohl aus einem Basiskatalog gefüttert werden, der gelegentlich erweitert wird, als auch vom Benutzer der Anwendung. Wie Sie das überhaupt machen, welche Probleme das mit sich bringt und wie Sie diese lösen, zeigen wir in diesem Beitrag.**

Im Beispiel verwenden wir zwei Tabellen, damit die möglichen Probleme schnell deutlich werden: Die Tabelle **tblProdukte** enthält die Produktdaten sowie die Zuordnung zu einer Kategorie. Die Kategorien werden in der zweiten Tabelle **tblKategorien** gespeichert (siehe Bild 1). Der Benutzer soll sowohl neue benutzerdefinierte Kategorien als auch Produkte anlegen können. Genauso soll der Basiskatalog von Produkten und Kategorien erweiterbar sein.

Angenommen, es gibt im Basiskatalog drei Kategorien mit den Primärschlüsselwerten **1**, **2** und **3**. Außerdem gibt es einige Produkte, die mit diesen Kategorien verknüpft sind. Nun fügt der Benutzer eine neue Kategorie hinzu, die standardmäßig den nächsten freien Autowert erhält, zum Beispiel den Wert **4**, und fügt außerdem einige Produkte hinzu, die mit dieser neuen Kategorie verknüpft sind.

Wenn der Herausgeber der Produktdatenbank nun dem Basiskatalog eine neue Kategorie hinzufügt, wird diese dort ebenfalls den Primärschlüsselwert **4** erhalten. Wenn wir die neuen Daten des Basiskatalogs nun in die bereits vom Anwender geänderte Datenbank übertragen wollen, müssen wir den Primärschlüsselwert einer der Kategorien ändern – entweder für die aus dem Basiskatalog oder für die vom Benutzer hinzugefügte Kategorie.

Da sowohl im Basiskatalog als auch im der vom Benutzer erweiterten Datenbank bereits Produkte enthalten sind, die mit der neuen Kategorie verknüpft sind, müssen wir folglich auch die Fremdschlüsselfelder dieser Einträge anpassen.

Man würde hier wohl die Daten des Benutzers anpassen, damit die Daten des Basiskatalogs in allen Kopien der Anwendung die gleichen Primärschlüsselwerte hat.

Das ist technisch kein Problem, allerdings ist es ein Aufwand, den man sich sparen könnte. Ein Weg ist, GUIDs als Primärschlüsselwerte zu verwenden. Allerdings bringt dies wieder eigene Probleme mit sich, die wir im Beitrag **GUIDs im Primärschlüsselfeld nutzen** in der folgenden Ausgabe zeigen (**www.access-im-unternehmen.de/1278**).

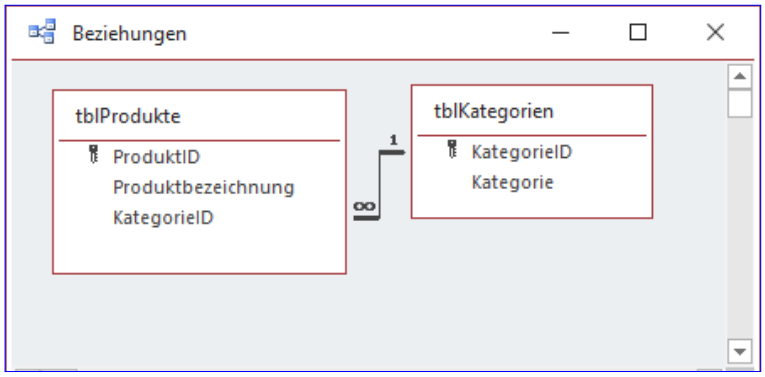



**Bild 1:** Datenmodell der Beispieldatenbank

### Alternative: Eigener Nummernkreis für den Anwender

Also wählen wir einen alternativen Weg: Wir gehen davon aus, dass die Anzahl der Produkte und Kategorien, die dem Basiskatalog hinzugefügt werden, endlich ist und wählen mit ausreichender Sicherheit eine maximale Menge aus – sagen wir 1.000.000.

Wenn der Benutzer einen neuen Produkt-Datensatz anlegt, soll dieser einen Autowert erhalten, der größer als 1.000.000 ist. Wie erledigen wir das?

Dazu fügen wir einfach einen Datensatz hinzu, dessen Primärschlüsselfeld den Wert **1.000.000** aufweist. Und keine Sorge: Sie müssen dazu nicht manuell die entsprechende Anzahl Datensätze eingeben. Wir fügen diesen Datensatz gezielt mit einer Prozedur wie der folgenden hinzu:

```
Public Sub AutowertSetzen()
    Dim db As DAO.Database
    Set db = CurrentDb
    db.Execute "INSERT INTO tblKategorien(KategorieID, 
       Kategorie) VALUES(1000000, 'Dummy')", dbFailOnError
End Sub
```

Die Prozedur führt eine **INSERT INTO**-Abfrage aus, die einen Datensatz hinzufügt, der den vorgegebenen Wert für das Feld **KategorieID** enthält. Das ist übrigens die einzige Methode, wie Sie trotz Autowert einen eigenen ID-Wert zu einer Tabelle hinzufügen können – mit der DAO-Methode **AddNew** ist das nicht möglich.

Wenn der Benutzer nun einen neuen Datensatz zur Tabelle **tblKategorien** hinzufügt, erhält dieser direkt den nächsthöheren Wert (siehe Bild 2).

Der Dummy-Datensatz hat nur einen Nachteil: Er erscheint beispielsweise im Nachschlagefeld zur Auswahl einer Kategorie. Wie können wir das ändern? Dazu gibt es zwei Möglichkeiten:

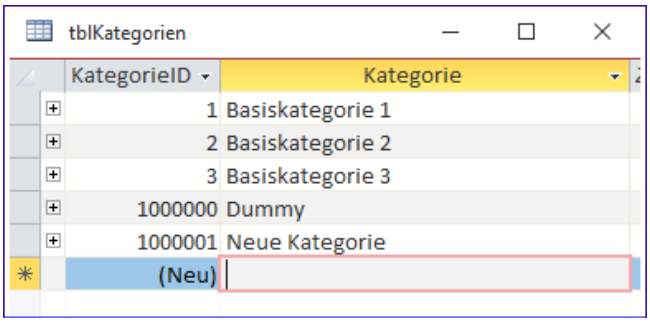



**Bild 2:** Neuer Benutzer-Datensatz

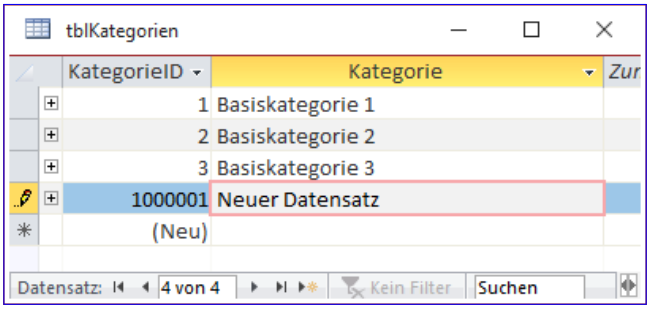



**Bild 3:** Neuer Benutzer-Datensatz ohne Dummy-Datensatz

- Wir definieren die Datensatzherkunft des Nachschlagefeldes in der Tabelle so, dass dieser Datensatz nicht angezeigt wird.

- Wir löschen den Datensatz vor der Weitergabe einfach wieder – denn solange der Benutzer die Datenbank nicht komprimiert, ist der zuletzt verwendete Autowert immer noch in der Datenbank gespeichert und es wird beim Anlegen eines neuen Datensatzes der nächste Autowert eingetragen (siehe Bild 3).

In der ersten Variante stellen wir die Eigenschaft **Datensatzherkunft** des Feldes **KategorieID** im Entwurf der Tabelle **tblProdukte** wie folgt ein:

```
SELECT [KategorieID], [Kategorie] FROM tblKategorien
WHERE NOT KategorieID = 1000000;
```

Damit erscheint der Eintrag nicht in diesem Nachschlagefeld und auch nicht in Kombinationsfeldern in Formularen, die auf Basis des Nachschlagefeldes angelegt werden (siehe Bild 4).

### ID vor Weitergabe löschen

Bei der zweiten Variante legen wir mit der oben genannten Prozedur einen Datensatz mit dem Wert 1.000.000 an und löschen diesen direkt wieder.

Dazu erweitern wir die Prozedur **AutowertSetzen** um die entsprechende **DELETE**-Anweisung:

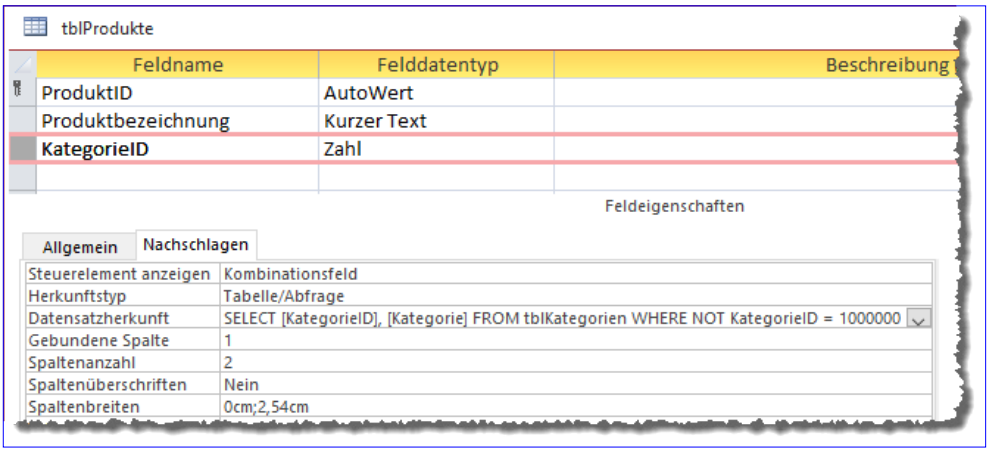



**Bild 4:** Anpassen der Datensatzherkunft des Nachschlagefeldes

```
Public Sub AutowertSetzen()
    ...
    db.Execute "DELETE FROM tblKategorien
                WHERE KategorieID = 1000000", dbFailOnError
End Sub
```

Da Access sich den höchsten angelegten Autowert merkt, auch wenn man diesen wieder löscht, wird der erste Datensatz des Benutzers in der Tabelle **tblKategorien** den Primärschlüsselwert **1.000.001** erhalten.

Wie aber können wir sicherstellen, dass der Benutzer die Datenbank nicht komprimiert und damit den Autowert-Zähler zurücksetzt – sodass der Autowert eines neuen Datensatzes für unser Beispiel den Wert **4** liefert?

Auch das lässt sich erledigen: Wir verlegen einfach den Aufruf der Prozedur **AutowertSetzen** an den Start der Anwendung. Allerdings erweitern wir diese noch leicht, indem wir eine Prüfung wie in Listing 1 voranstellen. Hier kontrollieren wir, ob der höchste Wert des Feldes **KategorieID** kleiner als **1.000.000** ist. Ist das der Fall, wurde die Datenbank offensichtlich komprimiert oder der Autowert wurde noch nicht auf **1.000.000** eingestellt. Also holen wir dies nach, indem wir einen neuen Datensatz mit diesem Autowert hinzufügen und diesen direkt wieder löschen. Damit erhält der erste vom Benutzer zur Tabelle **tblKategorien** hinzugefügte Datensatz den Wert **1.000.001**.

Diese Prozedur müssen wir nur noch beim Start der Anwendung aufrufen – dazu legen Sie beispielsweise ein Makro namens **AutoExec** an, das die Makroaktion

```
Public Function AutowertSetzen()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT Max(KategorieID) FROM tblKategorien", dbOpenDynaset)
    If rst(0) < 1000000 Then
        db.Execute "INSERT INTO tblKategorien(KategorieID, Kategorie) VALUES(1000000, 'Dummy')", dbFailOnError
        db.Execute "DELETE FROM tblKategorien WHERE KategorieID = 1000000", dbFailOnError
    End If
End Function
```

**Listing 1:** Setzen des Autowertes, wenn dieser nicht hoch genug ist

# XRechnung, Teil 1: Rechnungen generieren

**XRechnung ist ein Standard für das Einreichen von Rechnungen bei öffentlichen Aufträgen. Sinn und Zweck dieses Standards ist, das Übermitteln von Rechnungsdaten auf eine Weise zu erlauben, dass diese automatisiert eingelesen und verarbeitet werden können. Wie das X in XRechnung vermuten lässt, steckt dahinter ein Austausch der Daten im XML-Format. Die Vorlage für XRechnung liefert die europäische Norm EN 16931. Wie die Daten strukturiert werden müssen, haben wir dem Dokument »Spezifikation X … r ist. Als Grund… mliches Daten… igen wir, wie d… patibles XML-…**



Das Dok… t in ein **Extensi**… **…lidieren**. gen erfo… Struktur… …ei Beispiel- werden … …ieren und daher h… dass Sie…

https://… …; wir die nung-202… …m XML- …t wie in

Sollte d… fügbar s… Dokume… …Rech- … in der

### Validieren der XRechnung

Bevor wir beginnen, den Code zum Zusammenstellen der XRechnung zu programmieren, schauen wir uns an, wo wir die erstellen XRechnungen validieren können. Immerhin wollen wir wissen, ob das Ergebnis auch vom Rechnungsempfänger verarbeitet werden kann. Es gibt eine Reihe Online-Validatoren, zum Beispiel diesen hier:

https://www.epoconsulting.com/erechnung/xrechnung-validator

…XRechnung der Ansprechpartner in einem Feld angegeben wird, haben wir es uns hier einfach gemacht und auch nur ein Feld namens **Ansprechpartner** statt **Vorname** und **Nachname** hinterlegt. In dieser Tabelle sind auch die Kontodaten des jeweiligen Kontakts hinterlegt.

Die Tabelle **tblRechnungen** enthält die Basisdaten einer Rechnung. Dazu gehören das Rechnungsdatum, die Währung und ein Feld für die Kundenreferenz. Außerdem wählen wir einige Informationen aus Nachschlagefeldern

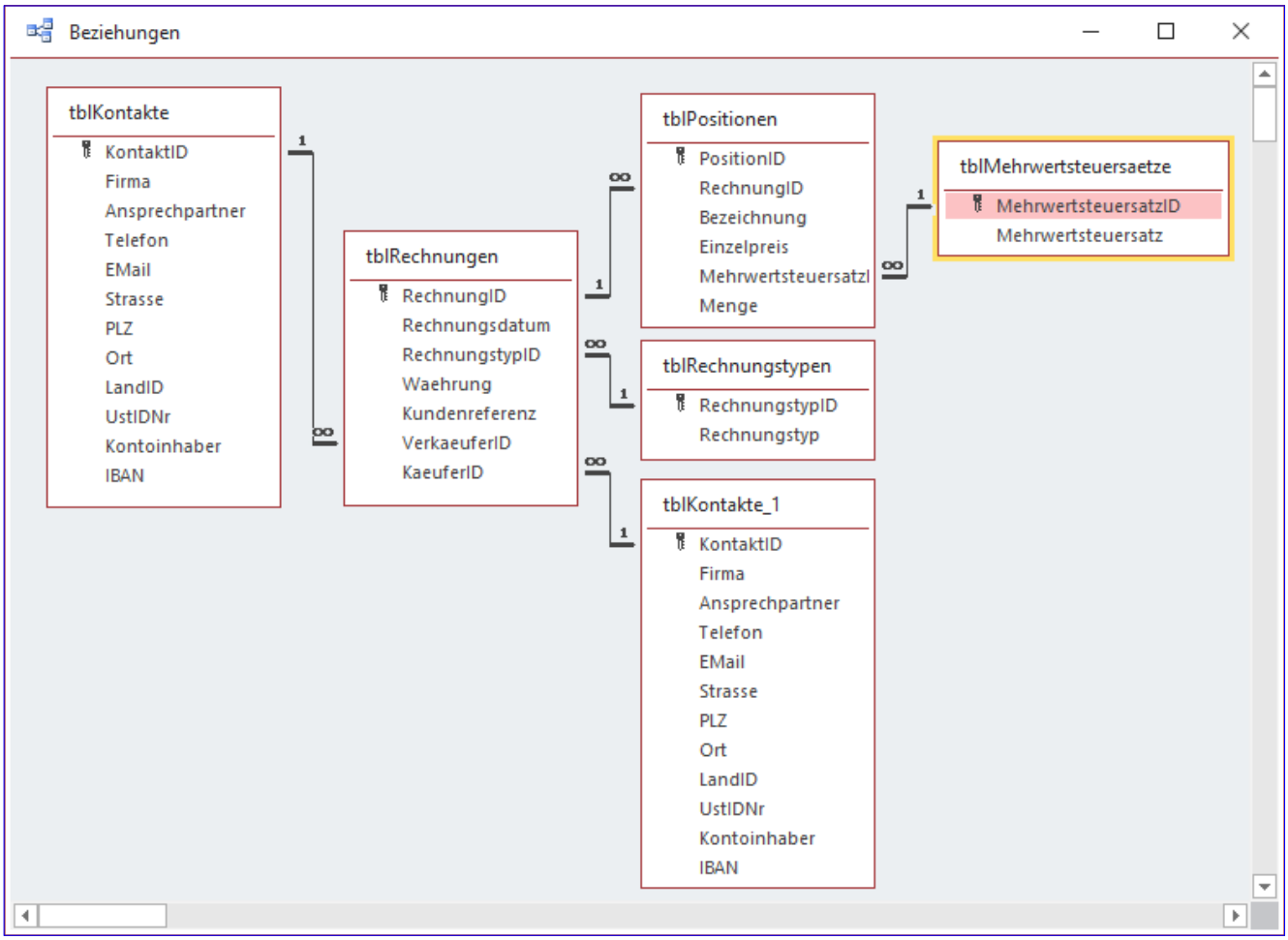



**Bild 1:** Datenmodell der Beispieldatenbank

für Fremdschlüsselfelder aus. Dazu gehört der Rechnungstyp. Die dafür verfügbaren Werte speichern wir in einer weiteren Tabelle namens **tblRechnungstypen** (siehe Bild 2).

Außerdem enthält die Tabelle **tblRechnungen** zwei Fremdschlüsselfelder, mit denen Daten aus der Tabelle **tblKontakte** ausgewählt werden können – und zwar **VerkaeuferID** und **KaeuferID**. Um dies im Beziehungen-Fenster abzubilden, haben wir die Tabelle **tblKontakte** zwei Mal hinzugefügt, einmal unter dem Originalnamen und einmal unter **tblKontakte_1**.

Die Tabelle **tblPositionen** nimmt die Rechnungspositionen auf. Zum Zuordnen einer Position zu einer Rechnung enthält sie das Fremdschlüsselfeld **RechnungID**. Außerdem speichert sie die Bezeichnung der Position, den Einzelpreis, den Mehrwertsteuersatz und die Menge. Den Mehrwertsteuersatz geben wir durch die Auswahl eines der Einträge der Tabelle **tblMehrwertsteuersaetze** für das Fremdschlüsselfeld **MehrwertsteuersatzID** an.

## Aufbau einer XRechnung

Wie bei XML-Dokumenten üblich, ist die XRechnung hierarchisch aufgebaut. Das Root-Element heißt dabei **Invoice** und definiert gleich noch die für das Dokument gültigen Namespaces:

```
<ubl:Invoice xmlns:ubl="urn:oasis:names:specification:ubl:
schema:xsd:Invoice-2" xmlns:cac="urn:oasis:names:specifica
```

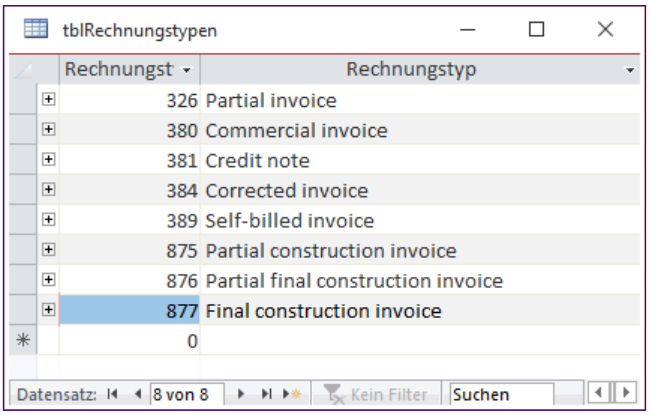



**Bild 2:** Tabelle mit den Rechnungstypen

```
tion:ubl:schema:xsd:CommonAggregateComponents-2" xmlns:c-
bc="urn:oasis:names:specification:ubl:schema:xsd:CommonBas
icComponents-2" xmlns:xsi="http://www.w3.org/2001/XMLSche-
ma-instance" xsi:schemaLocation="urn:oasis:names:specifica
tion:ubl:schema:xsd:Invoice-2 http://docs.oasis-open.org/
ubl/os-UBL-2.1/xsd/maindoc/UBL-Invoice-2.1.xsd">
```

Danach folgen einige Elemente, welche Basisdaten einer Rechnung abbilden. Da wir zunächst eine valide Rechnung erstellen und nicht direkt alle möglichen Elemente einbeziehen wollen, werfen wir zunächst einen Blick auf die notwendigen Elemente.

Das Dokument **XRechnung Standard und Extension** liefert im Kapitel **Detailbeschreibung** eine Übersicht der Elemente und vor allem der erwarteten Anzahl der Elemente. Diese wird dort in der Übersicht in eckigen Klammern angegeben. Die Rechnungsnummer zum Beispiel muss immer angegeben werden:

```
-Invoice number : Identifier [1]
```

Die Angabe des Zahlungsziels hingegen ist nicht verpflichtend:

```
-Payment due date : Date [0..1]
```

**[0..1]** bedeutet, dass das Element gar nicht oder einmal vorkommen darf.

Auf diese Weise finden wir heraus, ob Elemente überhaupt angegeben werden müssen und stellen so unsere minimal benötigten Elemente zusammen. Der Hintergrund ist, dass wir die dort einzufügenden Informatinen auch in unserem Datenmodell finden müssen.

Gegebenfalls könnten wir die Daten der Rechnung, die immer gleich sind, weil es sich um die Daten des Rechnungsstellers handelt, einfach fest im Code verdrahten. Aber wir wollen nicht, dass wir für jede Änderung den Entwurf des Codes anpassen müssen.

## Abweichungen zwischen Standard und Umsetzung

Wenn wir die Bezeichnungen der Felder im PDF-Dokument **XRechnung Standard und Extension** anschauen und diese mit denen vergleichen, die als Beispieldokumente unter **https://www.epoconsulting.com/erechnung/xrechnung-validator** bereitstehen, stellen wir einige Unterschiede fest. Diese sind zu erklären, da wohl das .xsl-Dokument unter folgendem Link zum Transformieren des einen in das andere Dokument verwendet wird:

```
https://github.com/itplr-kosit/xrechnung-visualization/
blob/master/src/xsl/ubl-invoice-xr.xsl
```

Wir werden also per VBA direkt in das Format der Beispieldokumente exportieren.

## Erster Test

Wir wollen uns an das finale Dokument herantasten und erstellen erstmal das **Invoice**-Element mit einigen Unterelementen. Dazu erstellen wir ein provisorisches Formular mit einigen Rechnungsdaten aus dem Datenmodell (siehe Bild 3). Das Hauptformular zeigt die Daten der Tabelle **tblRechnungen** an und erlaubt die Auswahl der verknüpften Daten für die Felder **RechnungstypID**, **VerkaeuferID** und **KaeuferID** per Kombinationsfeld.

Das Unterformular zeigt die Daten der verknüpften Tabelle **tblPositionen** an.

Ein weiteres Formular, das wir hier nicht abgebildet haben, heißt **frmKontakt** und erlaubt die einfache Eingabe neuer Kontakte.

Auf Komfort wie das Anzeigen dieses Formulars vom Formular **frmRechnung** haben wir an dieser Stelle verzichtet.

Für die Schaltfläche **cmdXRechnungErstellen** hinterlegen wir die Prozedur aus Listing 1. Diese verwendet zwei Hilfsfunktionen:

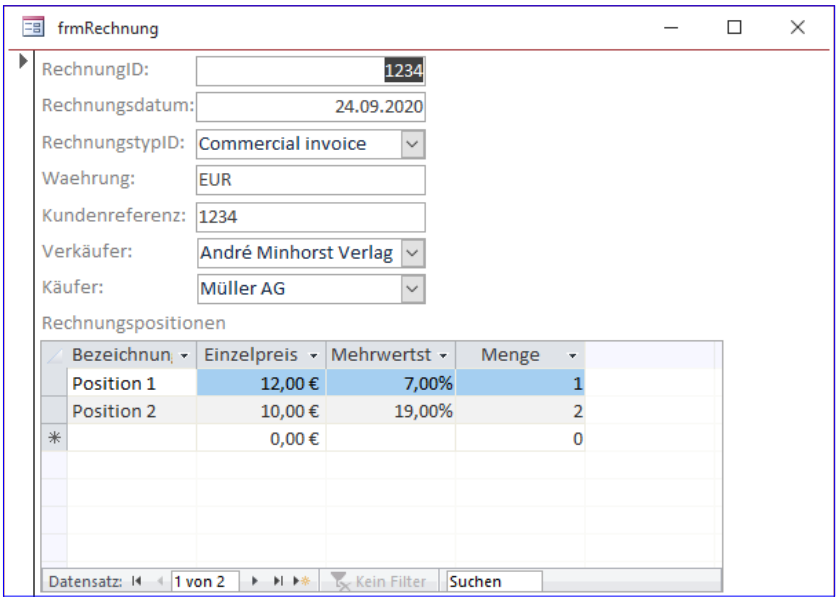



**Bild 3:** Formular zum Zusammenstellen einer Rechnung

- **CreateSubElement** legt ein neues Element unterhalb des mit dem ersten Parameter übergebenen Elements an. Der zweite Parameter gibt den Namen an, der dritte den Wert, soweit dieser zugewiesen werden soll. Die Funktion gibt das neue **IXMLDomNode**-Element als Ergebnis zurück.
- **CreateAttribute** legt ein Attribut für das mit dem ersten Parameter übergebene Element an. Der zweite Parameter nimmt den Namen des zu erstellenden Attributs entgegen, der dritte den Wert.

```
Private Sub cmdXRechnungErstellen_Click()
    Dim objXML As DOMDocument60
    Dim objInvoice As IXMLDOMNode
    Set objXML = New DOMDocument60
    Set objInvoice = CreateSubElement(objXML, "ubl:Invoice")
    CreateAttribute objInvoice, "xmlns:ubl", "urn:oasis:names:specification:ubl:schema:xsd:Invoice-2"
    CreateAttribute objInvoice, "xmlns:cac", "urn:oasis:names:specification:ubl:schema:xsd:CommonAggregateComponents-2"
    CreateAttribute objInvoice, "xmlns:cbc", "urn:oasis:names:specification:ubl:schema:xsd:CommonBasicComponents-2"
    CreateAttribute objInvoice, "xmlns:xsi", "http://www.w3.org/2001/XMLSchema-instance"
    CreateAttribute objInvoice, "xsi:schemaLocation", "urn:oasis:names:specification:ubl:schema:xsd:Invoice-2 " _
        & "http://docs.oasis-open.org/ubl/os-UBL-2.1/xsd/maindoc/UBL-Invoice-2.1.xsd"
    CreateSubElement objInvoice, "cbc:CustomizationID", "urn:cen.eu:en16931:2017#compliant#urn:" _
        & "xoev-de:kosit:standard:xrechnung_1.2"
    CreateSubElement objInvoice, "cbc:ID", Me!RechnungID
    CreateSubElement objInvoice, "cbc:IssueDate", Format(Me!Rechnungsdatum, "yyyy-mm-dd")
    CreateSubElement objInvoice, "cbc:InvoiceTypeCode", Me!RechnungstypID
    objXML.Save CurrentProject.Path & "\XRechnung.xml"
End Sub
```

**Listing 1:** Erster Test zum Zusammenstellung der XRechnung

Die Prozedur **cmdXRechnungErstellen_Click** erstellt ein neues XML-Dokument des Typs **DOMDocument60**. Damit dies gelingt, müssen wir noch einen Verweis auf die Bibliothek **Microsoft XML, v6.0** hinzufügen. Das erledigen Sie mit dem **Verweise**-Dialog, den Sie mit dem Menüeintrag **Extras|Verweise** des VBA-Editors öffnen (siehe Bild 4).

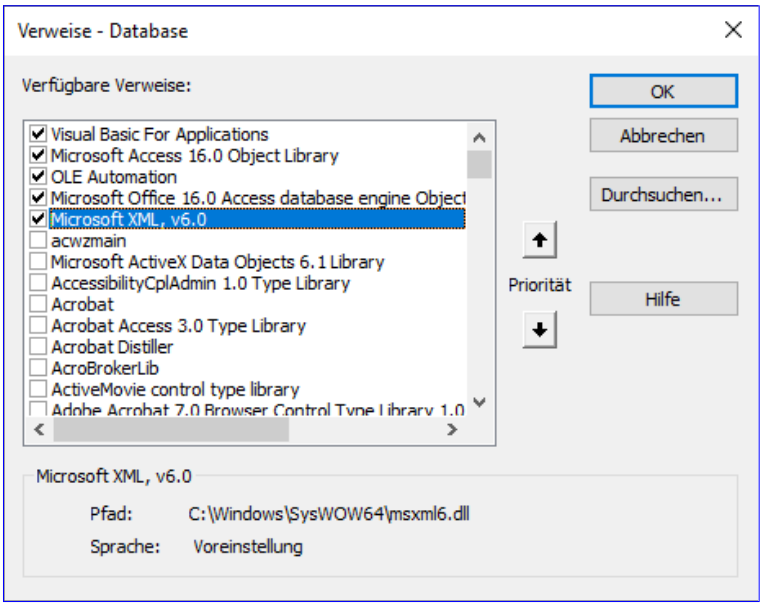



**Bild 4:** Verweis auf die XML-Bibliothek

Danach fügt die Prozedur mit der Funktion **CreateSubElement** ein neues Element des Typs **ubl:Invoice** hinzu und referenziert dieses mit der Variablen **objInvoice**.

Die Referenz benötigen wir, damit wir unter dem **ubl:Invoice**-Element noch weitere Attribute und Elemente hinzufügen können.

Das erledigen wir mit den folgenden Aufrufen der **CreateAttribute**-Funktion. Anschließend fügen wir die Elemente **CustomizationID** mit dem Wert hinzu, den wir in der Beispielrechnung gefunden haben.

Außerdem fügen wir aus der Tabelle **tblRechnungen** die Werte der Felder **RechnungID**, **Rechnungsdatum** und **RechnungstypID** hinzu. Das Datum formatieren wir noch entsprechend auf **yyyy-mm-dd**.

Danach speichern wir das neue XML-Dokument mit der **Save**-Methode von **objXML** im Verzeichnis der aktuellen Datenbank. Die so gespeicherte Datei laden wir dann zum Validieren auf die Seite aus Bild 5 hoch (oder die Seite eines anderen Anbieters, falls gewünscht).

Die Validierung dauert einige Sekunden länger, als man erwarten würde – vermutlich möchte der Anbieter verhindern, dass die Validierung automatisiert genutzt wird.

Das Ergebnis liefert uns eine Liste der Elemente, die noch fehlen beziehungsweise die ungültige Werte enthalten. Hieran wollen wir uns nun orientieren und die fehlenden Informationen hinzufügen beziehungsweise die falschen Daten korrigieren.

Allerdings zeigt sich schon nach dem Hinzufügen eines der fehlenden Elemente, dass die Meldung wohl immer angezeigt wird, wenn auch nur ein Element der aufgelisteten Elemente fehlt. Außerdem finden wir im Dokument **XRechnung Standard und Extension** heraus, dass das **Note**-Element beispielsweise gar kein Pflichtelement ist.

Das macht die Korrektur auf Basis des Validierungsergebnisses nicht leichter. Also arbeiten wir uns lieber anhand des Beispieldokuments, das erfolgreich validiert werden konnte, durch die geforderten Daten.

Wir fügen nun zunächst alle noch fehlenden Elemente hinzu, die im Dokument **XRechnung Standard und Extension** als Pflichtelement angegeben werden.

## Erweiterung um die fehlenden Elemente

Bevor wir die Funktionalität erweitern, gliedern wir die Anweisungen zum Erstellen der XRechnung in eine eigene Prozedur namens **XRechnungErstellen** aus, der wir lediglich den Primärschlüssel für die Rechnung übergeben.

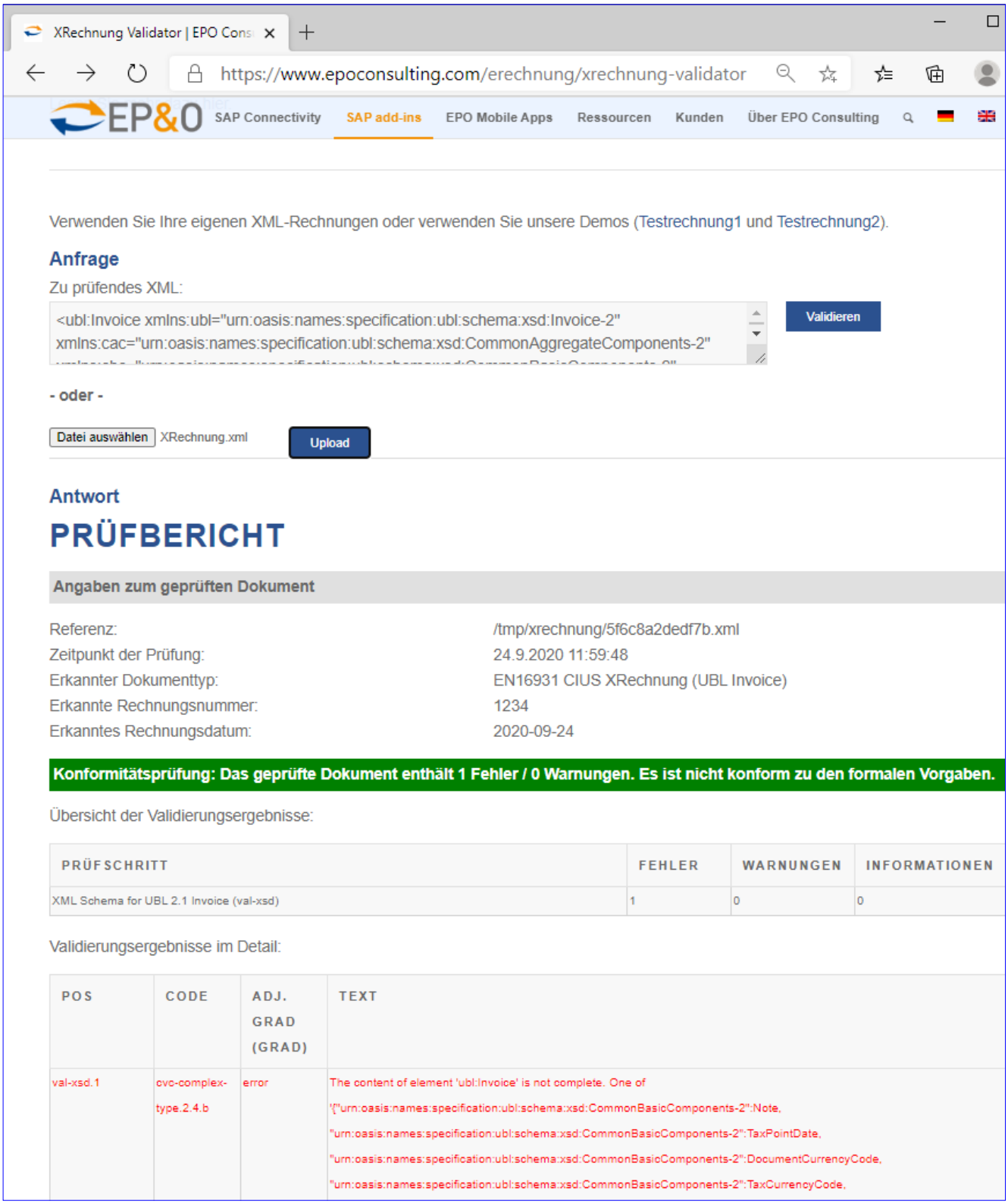



**Bild 5:** Validieren der Rechnung

Den Aufruf dieser Funktion fügen wir dann in der Ereignisprozedur ein:

```
Private Sub cmdXRechnungErstellen_Click()
    XRechnungErstellen Me!RechnungID
End Sub
```

Wir erweitern die Prozedur, die durch die Schaltfläche **cmdXRechnungErstellen_Click** aufgerufen wird, zunächst um einige Variablen.

Die meisten davon sollen Elemente des Typs **IXMLDomNode** aufnehmen – entweder, weil wir noch weitere Unterelemente anlegen wollen, für die wir das übergeordnete Element referenzieren müssen und/oder weil wir Attribute zu dem referenzierten Element hinzufügen wollen:

```
Public Sub XRechnungErstellen(lngRechnungID As Long)
    Dim objXML As DOMDocument60
    Dim objInvoice As IXMLDOMNode
    Dim objOrderreference As IXMLDOMNode
    Dim objContractDocumentReference As IXMLDOMNode
    Dim objProjectReference As IXMLDOMNode
    Dim objAccountingSupplierParty As IXMLDOMNode
    Dim objParty As IXMLDOMNode
    Dim objPostalAddress As IXMLDOMNode
    Dim objCountry As IXMLDOMNode
    Dim objPartyTaxScheme As IXMLDOMNode
    Dim objTaxScheme As IXMLDOMNode
    Dim objPartyLegalEntity As IXMLDOMNode
    Dim objContact As IXMLDOMNode
    Dim objAccountingCustomerParty As IXMLDOMNode
    Dim objPaymentMeans As IXMLDOMNode
    Dim objPayeeFinancialAccount As IXMLDOMNode
    Dim objPaymentTerms As IXMLDOMNode
    Dim objFinancialInstitutionBranch As IXMLDOMNode
    Dim objTaxTotal As IXMLDOMNode
    Dim objTaxAmount As IXMLDOMNode
    Dim objTaxSubtotal As IXMLDOMNode
    Dim objTaxableAmount As IXMLDOMNode
    Dim objTaxCategory As IXMLDOMNode
    Dim objLegalMonetaryTotal As IXMLDOMNode
    Dim objLineExtensionAmount As IXMLDOMNode
    Dim objTaxExclusiveAmount As IXMLDOMNode
    Dim objTaxInclusiveAmount As IXMLDOMNode
    Dim objPayableAmount As IXMLDOMNode
    Dim objInvoiceLine As IXMLDOMNode
    Dim objInvoicedQuantity As IXMLDOMNode
    Dim objItem As IXMLDOMNode
    Dim objSellersItemIdentification As IXMLDOMNode
    Dim objClassifiedTaxCategory As IXMLDOMNode
    Dim objPrice As IXMLDOMNode
    Dim objPriceAmount As IXMLDOMNode
    Dim db As DAO.Database
    Dim rstRechnung As DAO.Recordset
    Dim rstPositionen As DAO.Recordset
    Dim rstVerkaeufer As DAO.Recordset
    Dim rstKaeufer As DAO.Recordset
    Dim rstMehrwertsteuersaetze As DAO.Recordset
```

**rstPositionen** schließlich soll die Rechnungspositionen aufnehmen, die wir in einem Unterformular anzeigen. Diese durchlaufen wir, weil wir auch die Rechnungspositionen in Form geeigneter Elemente zum XML-Dokument hinzufügen wollen.

**rstVerkaeufer** nimmt die Daten des Eintrags der Tabelle **tblKontakte** auf, dessen Primärschlüsselfeld mit dem Wert des Feldes **VerkaeuferID** des aktuellen Datensatzes aus **rstRechnung** übereinstimmt.

Ähnlich sieht es beim Recordset **rstKaeufer** aus, der den passenden Datensatz zum Feld **KaeuferID** aus **rstRechnung** aufweist. Mit **rstMehrwertsteuersaetze** durchlaufen wir später beim Summieren der Mehrwertsteuer die Positionen verschiedener Mehrwertsteuersätze.

Zu diesem Zweck benötigen wir auch noch die folgenden drei Variablen. **curMwStGesamt** nimmt die Summe der Mehrwertsteuer für die Rechnung auf und mit **curBetragJeSatz** und **curMwStJeSatz** nehmen wir die Summe des steuerbaren Betrags und der Mehrwertsteuer je Mehr-