

# ACCESS

## IM UNTERNEHMEN

## ACCESS ERWEITERN

Erweitern Sie Access mit COM-Add-Ins um eigene Funktionen, die sogar über das Ribbon aufgerufen werden können (ab Seite 14).



### In diesem Heft:

#### SQL SERVER IM NETZWERK NUTZEN

Greifen Sie im Heimnetzwerk von einem Rechner auf den SQL Server eines anderen Rechners zu.

SEITE 33

#### ASSISTENT FÜR DOMÄNENFUNKTIONEN

Programmieren Sie einen Assistenten, mit dem Sie Domänenfunktionen leicht zusammenstellen können.

SEITE 48

#### KUNDENADRESSEN IN EIGENEN TABELLEN

Gliedern Sie Liefer- und Rechnungsadressen eines Kunden in separaten Tabellen aus.

SEITE 62

## Access erweitern

In diesen Tagen hat Microsoft eine neue Funktion zu Access hinzugefügt (zumindest vorerst für Office 365). Man kann nun Abfragen direkt in der SQL-Ansicht öffnen – eine Funktion, für die man zuvor die Abfrage in der Entwurfsansicht öffnen musste, um dann nochmal in die SQL-Ansicht zu wechseln. Wir haben dies aufgegriffen und noch eine Funktion hinzugefügt, mit der wir direkt Abfragen in dieser Ansicht erstellen können. Das zeigt: Wir können einfache (und auch kompliziertere) Funktionen mittlerweile einfach selbst programmieren. Wie das gelingt, zeigen wir in dieser Ausgabe!



Im Beitrag **Abfragen in der SQL-Ansicht öffnen** zeigen wir ab Seite 12, wie die neue Funktion von Access funktioniert. Viele Entwickler haben sich diese Funktion gewünscht, nun ist sie endlich da – und nicht nur für die Benutzeroberfläche, sondern wir können auch per VBA oder per Makro eine Abfrage direkt in der SQL-Ansicht öffnen.

Beim Schreiben dieses Beitrags ist uns aufgefallen, dass wir nun zwar Abfragen direkt in der SQL-Ansicht öffnen können, aber wenn wir eine neue Abfrage erstellen wollen, gelingt uns das nur in der Entwurfsansicht. Also haben wir ein COM-Add-In entwickelt, das diese Funktion über das Ribbon neben dem Button zum Erstellen einer Abfrage über die Entwurfsansicht hinzufügt. Sie brauchen dieses Add-In nur einmal installieren und können die Schaltfläche dann dauerhaft nutzen. Wie das funktioniert, erläutert der Beitrag **Abfragen direkt in der SQL-Ansicht erstellen** ab Seite 14.

Solche COM-Add-Ins muss man allerdings installieren oder zumindest registrieren. Damit das leicht von der Hand geht, beschreiben wir die grundlegenden Techniken ab Seite 40 im Beitrag **COM-Add-Ins und -DLLs installieren oder registrieren**.

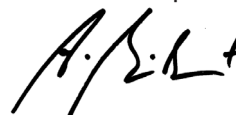
Fremdschlüsselfelder, die als Nachschlagefeld ausgeführt werden, zeigen die Daten aus anderen Tabellen an. Gelegentlich ist es sinnvoll, diese mit einem Standardwert zu versehen. Im Beitrag **Standardwerte für Fremdschlüsselfelder pflegen** zeigen wir ab Seite 2, wie wir diese Aufgabe komfortabel mit einem Formular erledigen können.

Unsere Musterlösung zur Rechnungsverwaltung geht in die nächste Runde. Diesmal schauen wir uns in zwei Beiträgen weitere Elemente an. Als Erstes passen wir das Datenmodell nochmals an. Wie das geschieht, erfahren Sie im Beitrag **Datenmodelle für die Rechnungsverwaltung, Teil 2** ab Seite 8. Außerdem besprechen wir im Beitrag **Rechnungsverwaltung: Kundenadressen ausgliedern** ab Seite 62, wie wir die Adressdaten aus einer Kundentabelle in zwei weitere Tabellen für Liefer- und Rechnungsadressen für mehr Flexibilität auslagern.

In weiteren Beiträgen zeigen wir, wie Sie ein Formular genau an der Mausposition öffnen können (siehe **Formular an Mausposition öffnen** ab Seite 26) und welche Gestaltungsmöglichkeiten es für modale Dialoge gibt (siehe **Modale Dialoge nach Wunsch gestalten** ab Seite 29).

Wenn Sie mal von einem Rechner im Heimnetzwerk auf die SQL Server-Datenbank auf einem anderen Rechner zugreifen wollen, erfahren Sie alles Wissenswerte ab Seite 33 unter dem Titel **SQL Server zwischen zwei Rechnern im Netzwerk**. Und schließlich erstellen wir ein **Access-Add-In zum Ermitteln von Domänenfunktionen** (siehe Assistent für Domänenfunktionen ab Seite 48).

Nun aber viel Spaß beim Stöbern in der neuen Ausgabe!



Ihr André Minhorst

## Standardwerte für Fremdschlüsselfelder pflegen

Für Felder in Tabellen kann man jeweils einen Standardwert im Tabellenentwurf festlegen, der beim Anlegen eines neuen Datensatzes in dieser Tabelle vorbelegt wird. Etwas anspruchsvoller ist es, wenn wir Standardwerte durch den Benutzer festlegen wollen. Noch einen Schritt weiter gehen wir, wenn der Standardwert für ein Fremdschlüsselfeld verwendet werden soll – also zur Auswahl eines Datensatzes aus einer verknüpften Tabelle. Einfache Beispiele sind Anreden, Zahlungsmethoden, Prioritäten oder Kategorien. Hier können wir jeweils einen Eintrag anwendungsweit festlegen. Es geht aber auch noch anspruchsvoller: Wenn wir beispielsweise Adressen für Kunden in einer separaten Tabelle speichern, in der mehrere Adressen je Kunde angelegt werden können und nur eine Adresse als Standard verwendet werden soll. Spätestens hier müssen wir auch beim Datenmodell umdenken. In diesem Beitrag schauen wir uns erst einmal an, wie wir Standardwerte für Nachschlagefelder komfortabel auswählen können.

### Einfache Standardwerte

In unserem Beitrag **Benutzerdefinierte Standardwerte** ([www.access-im-unternehmen.de/1466](http://www.access-im-unternehmen.de/1466)) haben wir bereits eine Lösung aufgezeigt, mit der wir für die Felder von Tabellen Standardwerte festlegen können. Normalerweise kann man Standardwerte direkt für die Eigenschaft **Standardwert** eines Feldes im Tabellenentwurf definieren. Diese werden beim Anlegen von gebundenen Feldern auf Basis dieser Tabelle in Formularen automatisch in die entsprechende Steuerelementeigenschaft übernommen. Allerdings möchten die Benutzer gegebenenfalls ihre eigenen Standardwerte definieren. Das lässt sich jedoch nicht mit der Eigenschaft **Standardwert** des Tabellenentwurfs realisieren, denn dann müsste man bei jeder Änderung den Entwurf der Tabelle anpassen. Also haben wir in obigem Beitrag eine Lösung gezeigt, wie wir diese Daten in einer eigenen Tabelle speichern und zur Laufzeit beim Verwenden gebundener Formulare abrufen und anwenden können.

Dabei haben wir Zahlen-, Ja/Nein- und Datumsfelder so behandelt, dass man den Wert in einem Popup-Formular eingibt, mit dem wir die als Standardwert eingegebenen Daten noch validieren konnten. Die Standardwerte von Textfeldern geben wir dort einfach so ein. Was noch fehlen würde, wäre eine komfortablere Eingabe von Standardwerten aus Nachschlagefeldern. Hier müsste man aktuell noch den Zahlenwert eingeben, welcher dem Primärschlüsselfeld des zu verknüpfenden Datensatzes entspricht (siehe Bild 1). Das ist recht unpraktisch, daher

Tabellenname	Feldname	Standardwert
tblAnreden	Anredebezeichnung	
tblAnreden	AnredeBrief	
tblKunden	AnredeID	
tblLieferadressen_Bestellung	AnredeID	2
tblLieferadressen_Kunde	AnredeID	
tblLieferadressen_Lieferung	AnredeID	
tblRechnungsadressen_Beste	AnredeID	
tblRechnungsadressen_Kunde	AnredeID	

Bild 1: Standardwerte für Nachschlagefelder müssen noch manuell eingegeben werden.

wollen wir im ersten Teil des vorliegenden Beitrags noch ein Popup-Formular für diesen Zweck hinzufügen und die Lösung zum Verwalten von Standardwerten entsprechend erweitern.

### Standardwerte aus Fremdschlüsselfeldern

Das Endergebnis sieht wie in Bild 2 aus: Der Benutzer klickt auf einen Standardwert, der im Datenmodell als Nachschlagefeld definiert ist und kann im nun erscheinenden Popup die vorhandenen Einträge auswählen und als Standardwert festlegen.

### Anpassungen des Datenmodells

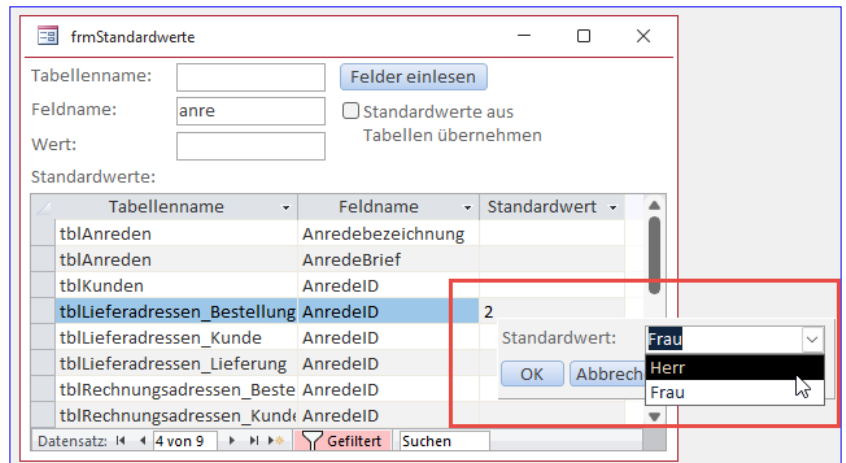
Um im Popup das Nachschlagefeld nachzubilden, das im Entwurf der Tabelle mit dem entsprechenden Feld enthaltenen ist, wollen wir ein paar Informationen zusätzlich zu diesem Feld in der Tabelle **tblStandardwerte** speichern. Diese nimmt bisher hauptsächlich den Namen der Tabelle und des Feldes auf, für das wir einen Standardwert speichern wollen. Hier kommen nun noch die wesentlichen Felder für die Anzeige eines Nachschlagefeldes hinzu, die wir direkt nach den entsprechenden Eigenschaften des Nachschlagefeldes benennen:

- **RowSource:** Tabelle, Abfrage oder SQL-Ausdruck mit der Datensatzherkunft des Nachschlagefeldes
- **BoundColumn:** Gebundene Spalte der Datensatzherkunft
- **ColumnWidths:** Spaltenbreiten, zum Beispiel **0;1440**
- **ColumnCount:** Anzahl der anzuzeigenden Spalten, zum Beispiel **2**

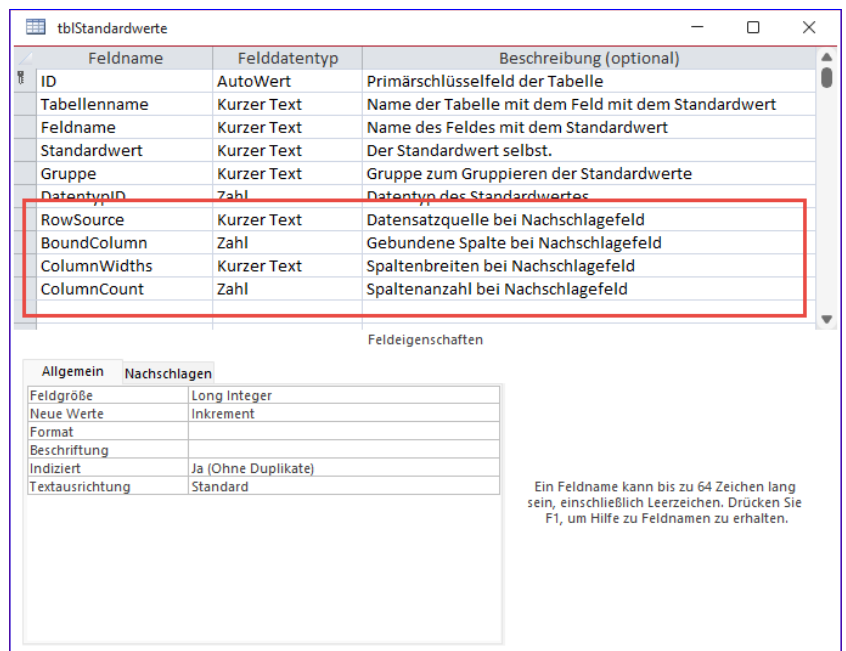
Diese Felder fügen wir dem Entwurf der Tabelle **tblStandardwerte** wie in Bild 3 hinzu.

### Anpassen des Einlesevorgangs

Die Daten zur Konfiguration der Nachschlagefelder wollen wir per Code in die Tabelle **tblStandardwerte** eintragen. Dazu passen wir die bisher verwendete Prozedur entsprechend an (siehe Listing 1). Wir fügen zunächst Variablen zum Speichern der betroffenen Informationen hinzu.



**Bild 2:** Die Auswahl per Kombinationsfeld wäre wesentlich ergonomischer.



**Bild 3:** Zusätzliche Informationen zum Anzeigen von Nachschlagefeldern

```
Public Sub FelderEinlesen(Optional bolStandardwerteUebernehmen As Boolean = False)
    Dim db As DAO.Database, tdf As DAO.TableDef, fld As DAO.Field, prp As DAO.Property, rst As DAO.Recordset
    Dim strRowSource As String, lngBoundColumn As Long, strColumnWidths As String, lngColumnCount As Long
    Dim i As Integer
    Set db = CurrentDb
    For Each tdf In db.TableDefs
        Select Case True
            Case tdf.Name Like "MSys*", tdf.Name Like "USys*", tdf.Name = "tblStandardwerte", tdf.Name = "tblDatentypen"
            Case Else
                For Each fld In tdf.Fields
                    strRowSource = ""
                    lngBoundColumn = 0
                    strColumnWidths = ""
                    lngColumnCount = 0
                    Set prp = Nothing
                    On Error Resume Next
                    Set prp = fld.Properties("DisplayControl")
                    On Error GoTo 0
                    If Not prp Is Nothing Then
                        If prp.Value = acComboBox Then
                            strRowSource = fld.Properties("RowSource")
                            lngBoundColumn = fld.Properties("BoundColumn")
                            strColumnWidths = fld.Properties("ColumnWidths")
                            lngColumnCount = fld.Properties("ColumnCount")
                        End If
                    End If
                    On Error Resume Next
                    If bolStandardwerteUebernehmen Then
                        db.Execute "INSERT INTO tblStandardwerte(Tabellenname, Feldname, Standardwert, DatentypID, " _
                            &"RowSource, BoundColumn, ColumnWidths, ColumnCount) VALUES('" & tdf.Name & "', '" _
                            & fld.Name & "', '" & fld.DefaultValue & "', " & fld.Type & "', '" & strRowSource & "', " _
                            & lngBoundColumn & "', '" & strColumnWidths & "', " & lngColumnCount & ")", dbFailOnError
                    Else
                        ...
                    End If
                    Select Case Err.Number
                        Case 3022
                            On Error GoTo 0
                            If bolStandardwerteUebernehmen Then
                                db.Execute "UPDATE tblStandardwerte SET Tabellenname = '" & tdf.Name & "', " _
                                    &"Feldname = '" & fld.Name & "', Standardwert = '" & fld.DefaultValue _
                                    & "', DatentypID = '" & fld.Type & "', RowSource = '" & strRowSource _
                                    & "', BoundColumn = '" & lngBoundColumn & "', ColumnWidths = '" & strColumnWidths _
                                    & "', ColumnCount = '" & lngColumnCount & "' WHERE Tabellenname = '" & tdf.Name _
                                    & "' AND Feldname = '" & fld.Name & "'", dbFailOnError
                            Else
                                ...
                            End If
                        Case 0
                        Case Else
                            MsgBox "Fehler " & Err.Number & ":" & vbCrLf & vbCrLf & Err.Description
                    End Select
                    On Error GoTo 0
                Next fld
            End Select
        Next tdf
    End Sub
```

**Listing 1:** Erweiterung der Prozedur zum Anlegen der Tabellen und Felder für die Standardwerte



## Hier geht es weiter ...

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

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

Alle Informationen hier – jetzt bestellen und direkt loslesen!

**ZUM SHOP**



## Datenmodelle für die Rechnungsverwaltung, Teil 2

Im ersten Teil dieser Beitragsreihe haben wir die grundlegenden Ideen vorgestellt, die wir zum Thema Bestellungen/Rechnungen/Lieferungen haben. Uns ist jedoch noch ein Detail entgangen, das wir in diesem Beitrag noch nachreichen wollen. Dabei geht es darum, eventuelle Inkonsistenzen beim Erstellen von Rechnungspositionen auf Basis von Bestellpositionen zu vermeiden. Normalerweise sollte mit dem aktuellen Datenmodell nichts schiefgehen – aber das Datenmodell sollte so viele Probleme wie möglich bereits durch die enthaltenen Restriktionen in Form von Beziehungen verhindern.

### Mögliche Inkonsistenz bei den Rechnungspositionen

Die Ausgabe 6/2023 war bereits fertig gesetzt, als unserem Lektor Carsten Gromberg eine mögliche Quelle für Inkonsistenzen in den Daten aufgefallen ist. Dabei geht es um den Teil, wo wir Rechnungen den Bestellungen zuweisen und Rechnungspositionen den Bestellposi-

tionen. Dabei haben wir die Tabellen **tblBestellungen** und **tblRechnungen** über die Verknüpfungstabelle **tblRechnungenBestellungen** miteinander verknüpft (siehe Bild 1). Hintergrund war, dass es sowohl vorkommen kann, dass eine Rechnung mehrere Bestellungen zusammenfasst als auch dass eine Bestellung in mehreren Rechnungen abgerechnet wird. Für die Beziehung zwi-

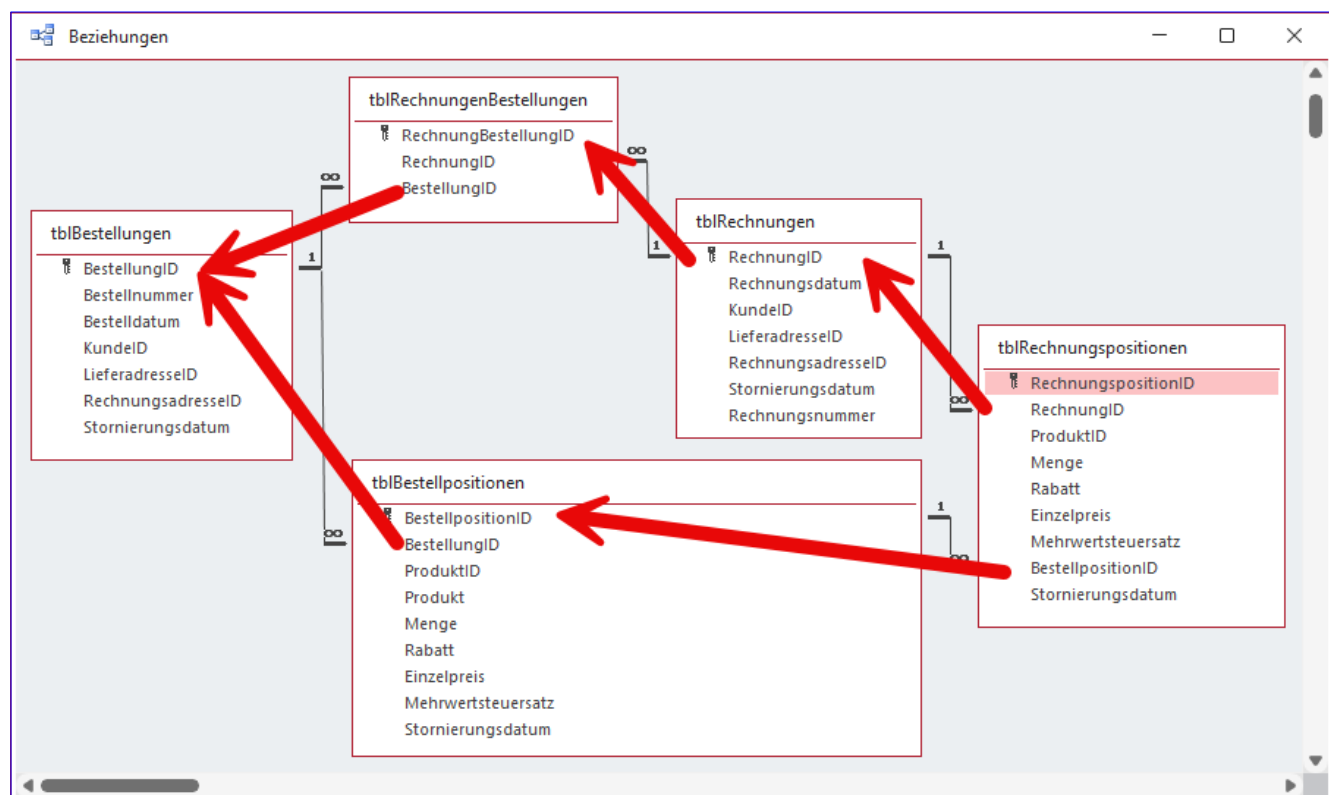


Bild 1: Auszug des Datenmodells mit der Möglichkeit von Inkonsistenzen

schen den Rechnungspositionen und den Bestellpositionen reichte uns eine einfache 1:n-Beziehung. Hier könnte man sich bereits die folgende Frage stellen:

Ist hier die 1:n-Beziehung der richtige Ansatz? Reicht nicht eine 1:1-Beziehung aus, weil wir zu jeder Bestellposition nur eine Rechnungsposition benötigen? Nein, die 1:1-Beziehung reicht nicht aus: Theoretisch soll es sogar möglich sein, eine Bestellposition mit einer Menge von mehr als eins auf mehrere Rechnungspositionen und auch Rechnungen aufzuteilen.

Es kann vorkommen, dass der Kunde zwei Exemplare des gleichen Produkts bestellt, aber nur eines lieferbar ist. Dann bekommt er zunächst nur die Lieferung des einen Exemplars und die entsprechende Rechnung.

Tatsächlich geht es um eine andere Frage: Angenommen, wir haben zwei Bestellungen **B1** und **B2**. **B1** enthält die Bestellpositionen **B1-1** und **B1-2**. **B2** enthält die Bestellposition **B2-1**. Wir erstellen darauf basierend nun eine Rechnung **R1**, die über die Tabelle **tblRechnungenBestellungen** mit der Bestellung **B1** verknüpft ist. Und wir verarbeiten die Bestellpositionen **B1-1** und **B1-2** in zwei Rechnungspositionen **R1-1** und **R1-2**.

Beide sind über das Fremdschlüsselfeld **RechnungID** der Tabelle **tblRechnungspositionen** mit der richtigen Rechnung verknüpft, nämlich **R1**. Und das Fremdschlüsselfeld **BestellpositionID** der gleichen Tabelle verknüpfen wir von **R1-1** zu **B1-1** und von **R1-2** zu **B1-2**.

Soweit, so gut: Es wäre aber theoretisch möglich, dass wir das Feld **BestellpositionID** der Rechnungsposition **R1-2** mit der Bestellposition **B2-1** verknüpfen, während diese Position über die Tabelle **tblRechnungen** und **tblRechnungenBestellungen** mit der Bestellung **B1** verknüpft ist.

Das wäre dann die mögliche Inkonsistenz. Aber wie wollen wir diese verhindern? Und benötigen wir die Zuordnung von Rechnungsposition zur Bestellposition überhaupt?

Letzteres ist notwendig, damit wir prüfen können, ob eine Bestellposition bereits in einer Rechnungsposition erfasst wurde – und gegebenenfalls auch abgleichen können, ob die in der Bestellposition angegebene Menge vollständig in Rechnung gestellt wurde.

Wie wir die Datenintegrität sicher stellen wollen, schauen wir uns im folgenden Abschnitt an.

### Verhindern der Inkonsistenz

Die Aufgabe lautet nun also: Wie können wir dafür sorgen, dass man für eine Rechnungsposition in der Tabelle **tblRechnungspositionen** keine Bestellposition für das Fremdschlüsselfeld **BestellpositionID** auswählen kann, die nicht zu der Bestellung aus der Tabelle **tblBestellungen** gehört, die über die Verknüpfungstabelle **tblRechnungenBestellungen** mit der zu der Rechnung aus **tblRechnungen** gehörenden Rechnung verknüpft ist?

Der Clou ist: Wir fügen der Tabelle **tblRechnungspositionen** ein weiteres Feld namens **BestellungID** hinzu. Dann erstellen wir eine Beziehung zwischen der Tabelle **tblRechnungspositionen** und **tblRechnungenBestellungen**, welche die beiden Felder **RechnungID** und **BestellungID** der beiden Tabellen umfasst und die im Fenster **Beziehungen bearbeiten** wie in Bild 2 aussieht.

Wechseln wir zurück zum **Beziehungen bearbeiten**-Fenster, sehen wir die Beziehung wie in Bild 3.

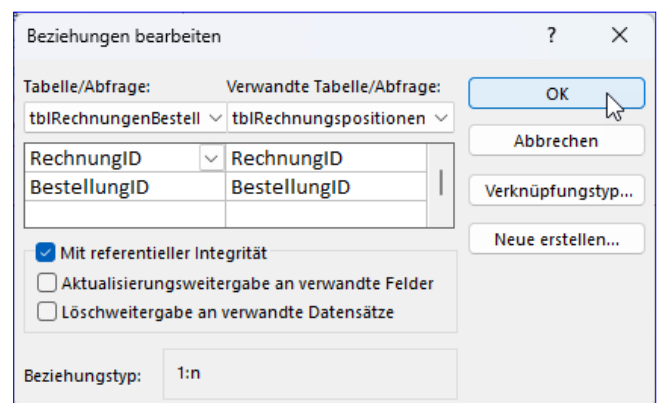


Bild 2: Anlegen einer Beziehung über zwei Felder



## Hier geht es weiter ...

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

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

Alle Informationen hier – jetzt bestellen und direkt loslesen!

**ZUM SHOP**



## Abfragen in der SQL-Ansicht öffnen

Seit vielen Jahren wünschen sich Access-Nutzer von Microsoft kleinste Veränderungen, die den Programmieralltag stark erleichtern könnten. Eine davon ist, Abfragen direkt in der SQL-Ansicht zu öffnen. Ja, es gibt Entwickler, die nicht ausschließlich in der Entwurfsansicht arbeiten – und manche Abfragetypen wie beispielsweise die UNION-Abfrage können nur in der SQL-Ansicht angezeigt werden. Bis vor kurzem musste man zur SQL-Ansicht einen Umweg über die Entwurfsansicht gehen. Wie dieser aussah und welche Möglichkeiten wir nun haben, zeigen wir in diesem Beitrag.

### Der steinige Weg zur SQL-Ansicht

Vor der Einführung des mit diesem Beitrag vorgestellten Features waren zur Anzeige einer Abfrage in der SQL-Ansicht unnötig viele Schritte erforderlich:

- Öffnen der Abfrage in der Entwurfsansicht per Rechtsklick auf den Eintrag im Navigationsbereich und anschließende Auswahl des Kontextmenü-Befehls **Entwurfsansicht**
- Auswahl des Ribbon-Befehls **Start|Ansichten|Ansicht|SQL-Ansicht** zum Wechsel in die SQL-Ansicht (siehe Bild 1).

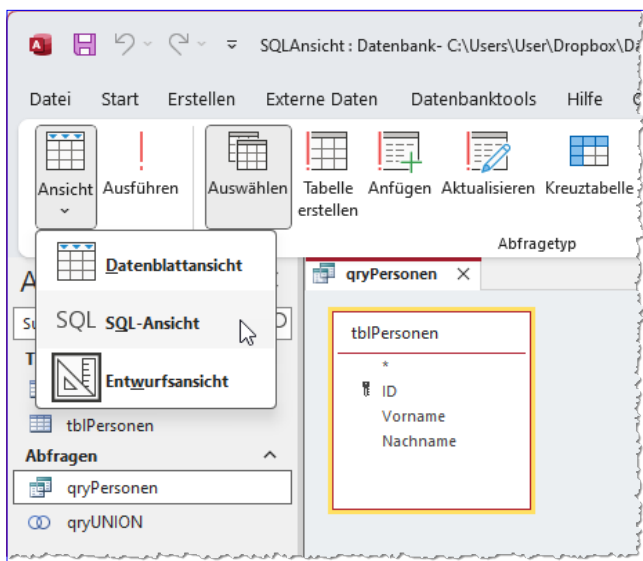


Bild 1: Wechsel zur SQL-Ansicht

- Erst danach finden wir die SQL-Ansicht der Abfrage vor (siehe Bild 2).

Wenn man standardmäßig mit dieser Ansicht arbeiten möchte, sind das recht viele zusätzliche Schritte, die im Laufe der Zeit zusammenkommen.

### Abfrage direkt in der SQL-Ansicht öffnen

Seit Ende November 2023 kann man in Access 2016 bis 2021 und in Office 365 Abfragen nun endlich direkt in der SQL-Ansicht öffnen. Dazu klicken wir einfach wie in Bild 3

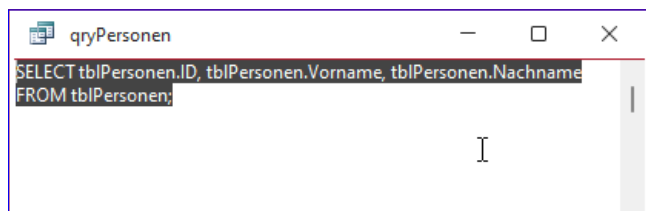


Bild 2: SQL-Ansicht der Abfrage

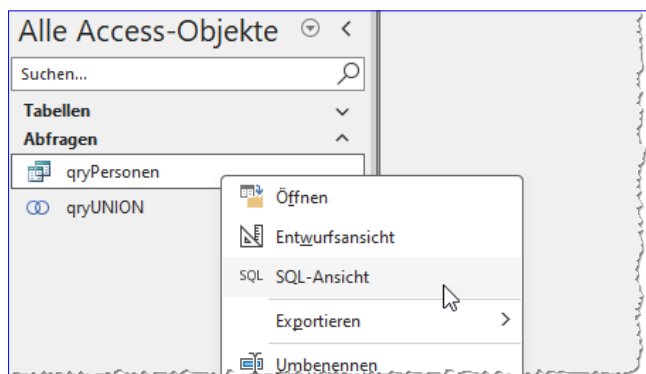


Bild 3: Abfrage direkt in der SQL-Ansicht öffnen

## Hier geht es weiter ...

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

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

Alle Informationen hier – jetzt bestellen und direkt loslesen!

**ZUM SHOP**



## Abfragen direkt in der SQL-Ansicht erstellen

Microsoft hat neuerdings einen Befehl zum direkten Anzeigen von Abfragen in der SQL-Ansicht. Damit können wir eine Abfrage direkt in der SQL-Ansicht öffnen statt den Umweg über den Abfrageentwurf zu wählen. Leider hat Microsoft keine entsprechende Funktion für das Erstellen von Abfragen hinzugefügt – also eine Schaltfläche im Ribbon, mit der wir eine neue SQL-Abfrage direkt in der SQL-Ansicht öffnen können. Das ist allerdings gar nicht schlimm, denn wir rüsten diese Funktion einfach selbst nach. Zunächst schauen wir uns an, welche Befehle dazu notwendig sind, danach erstellen wir ein COM-Add-In, das die Funktion in der Benutzeroberfläche verankert – gleich neben den Schaltflächen für den Abfrageassistenten und das Anlegen von neuen Abfragen über die Entwurfsansicht.

In dem Beitrag, der die neuen Befehle zum direkten Anzeigen vorhandener Abfragen in der SQL-Ansicht vorstellt, gehen wir bereits kurz auf die notwendigen Anweisungen ein, die zum Erstellen einer neuen Abfrage und ihrer Anzeige in der SQL-Ansicht nötig sind – siehe **Abfragen in der SQL-Ansicht öffnen** ([www.access-im-unternehmen.de/1482](http://www.access-im-unternehmen.de/1482)).

Dort stellen wir die folgende Prozedur vor:

```
Public Sub NewQueryInSQLView()  
    RunCommand acCmdNewObjectDesignQuery  
    RunCommand acCmdSQLView  
End Sub
```

Diese ruft die **RunCommand**-Anweisung mit zwei verschiedenen Parametern auf, die wir auch über die Benutzeroberfläche aufrufen können.

Der erste namens **acCmdNewObjectDesignQuery** öffnet zunächst eine neue Abfrage in der Entwurfsansicht.

Der zweite nutzt den Parameter **acCmdSQLView**, um die so erstellte Abfrage in der SQL-Ansicht anzuzeigen. Platzieren wir Access-Fenster und VBA-Editor nebeneinander und rufen die Prozedur auf, sehen wir ein kurzes Flackern

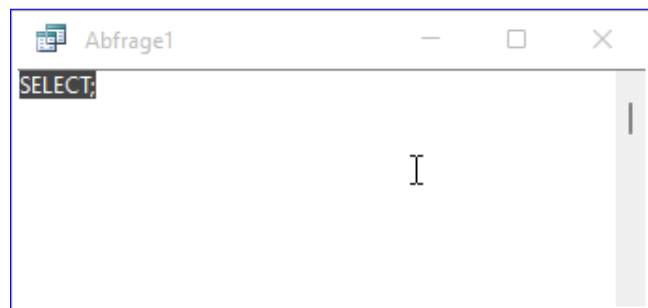


Bild 1: Neue, leere Abfrage per Code in der SQL-Ansicht

und erhalten anschließend eine weitgehend leere SQL-Ansicht der neuen Abfrage (siehe Bild 1).

Das kurze Flackern resultiert daraus, dass zuerst die Entwurfsansicht eingeblendet wird, zu der standardmäßig auch noch der Bereich **Tabellen hinzufügen** eingeblendet wird (in älteren Access-Versionen erscheint ein Fenster mit dem Titel **Tabelle auswählen**). Dieser verschwindet beim Wechsel zur SQL-Ansicht allerdings direkt wieder.

In älteren Versionen (zum Beispiel Access 2010) wird das Fenster **Tabelle auswählen** als modaler Dialog eingeblendet, den man erst schließen muss, bevor die Abfrage von der Entwurfsansicht in die SQL-Ansicht wechselt (siehe Bild 2).



In diesem Fall können wir nicht viel ausrichten – beim Anzeigen eines modalen Dialogs wird der aufrufende Code unterbrochen und folgende Anweisungen, mit denen wir den Dialog ausblenden könnten, würden nicht ausgeführt werden.

Benutzer älterer Access-Versionen müssen also diese Meldung noch schließen, bevor dann die SQL-Ansicht eingeblendet wird. Immerhin wird die Anweisung zum Wechseln zur SQL-Ansicht dann noch ausgeführt.

### Funktion per COM-Add-In zum Ribbon hinzufügen

Wenn wir diese aus zwei Anweisungen bestehende Funktion über das Ribbon aufrufen wollen, haben wir verschiedene Möglichkeiten. Die nachhaltigste ist die Erstellen eines COM-Add-Ins. Dieses wird einmal installiert und die Funktion ist dauerhaft in der Benutzeroberfläche verankert – egal, welche Datenbank Sie damit öffnen.

Das könnte beispielsweise wie in Bild 3 aussehen. Es schien uns logisch, dass die betroffene Schaltfläche in den Bereich **Erstellen/Abfragen** integriert werden sollte. Ein Mausklick auf diese Abfrage ruft die beiden oben vorge-

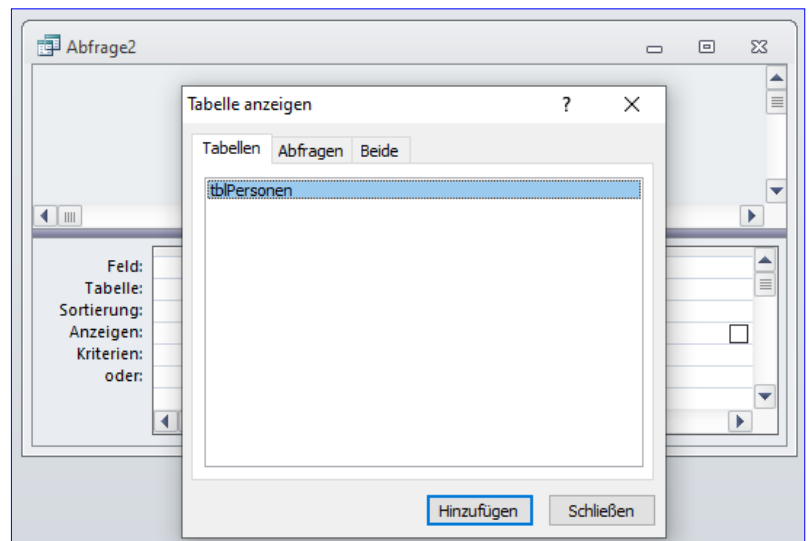


Bild 2: Dieses Fenster erscheint in älteren Access-Versionen.

stellten Anweisungen auf und öffnet so eine neue Abfrage in der SQL-Ansicht.

Ein solches COM-Add-In ist schnell erstellt. Wir nutzen dazu das schon oft referenzierte **twinBASIC** von Wayne Philips, das in der Version für das Erstellen von 32-Bit-Programmen kostenlos ist – erst die 64-Bit-Version ist kostenpflichtig (in DLLs, die mit der kostenlosen Version erstellt wurden, werden Hinweise auf twinBASIC eingeblendet). Die jeweils aktuelle Version kann hier heruntergeladen werden:

<https://github.com/twinbasic/twinbasic/releases>

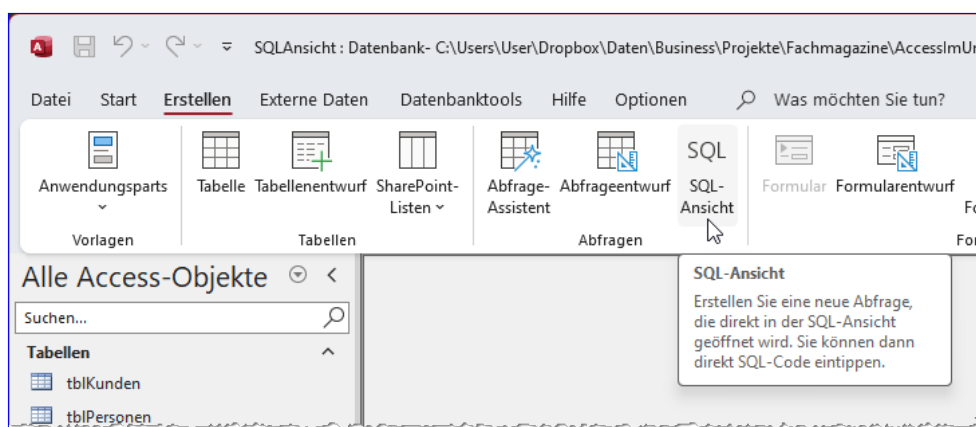


Bild 3: Neue Schaltfläche zum Anlegen einer Abfrage in der SQL-Ansicht

Nach dem Erstellen eines neuen Projekts auf Basis der Vorlage **Sample 5. MyCOMAddin** (siehe Bild 4) beginnen wir mit dem Speichern des Projekts unter dem von uns gewählten Pfad und Dateinamen, zum Beispiel **amvSQLView.twinproj**. Merken Sie sich



diesen Pfad, wir benötigen ihn später noch. Danach fahren wir mit dem Anpassen der Projekteinstellungen fort.

### Projekteinstellungen anpassen

Diese finden wir, wenn wir auf der linken Seite auf **Settings** klicken. Wir stellen nun den Projektnamen, die Projektbeschreibung und den Anwendungstitel auf den gewünschten Wert ein, beispielsweise **amvSQLView**. Das sieht dann für den Projekttitel wie in Bild 5 aus.

Danach fügen wir noch einen Verweis auf die Access-Bibliothek hinzu. Dazu scrollen wir unter **Settings** weiter nach unten bis zum Bereich **Library References**.

Hier wechseln wir zur Registerseite **Available COM References** und suchen nach Access. Die nun erscheinende Bibliothek fügen wir durch Setzen eines Hakens hinzu (siehe Bild 6).

### Registrierung der DLL programmieren

Damit kommen wir zu einem wichtigen Aspekt für ein COM-Add-In, das als DLL erstellt wird, nämlich die Registrierung. Damit Access beim Starten merkt, dass die Funktionen unseres COM-Add-Ins geladen werden sollen, müssen wir dies an bestimmten Stellen in der Registry vermerken.

Die Funktionen des Moduls aus Listing 1 erledigen genau dies. Im oberen Teil finden wir einige Konstanten mit wichtigen Informationen, die während der Installation benötigt werden. Die Funktion **DLLRegisterServer** schreibt die Einträge, die von Access beim Start abgefragt werden sollen und zum Laden des COM-Add-Ins führen, in einen bestimmten Bereich der Registry.

Die Funktion **DllUnregisterServer** entfernt diese Einträge wieder, falls gewünscht. Den Code des

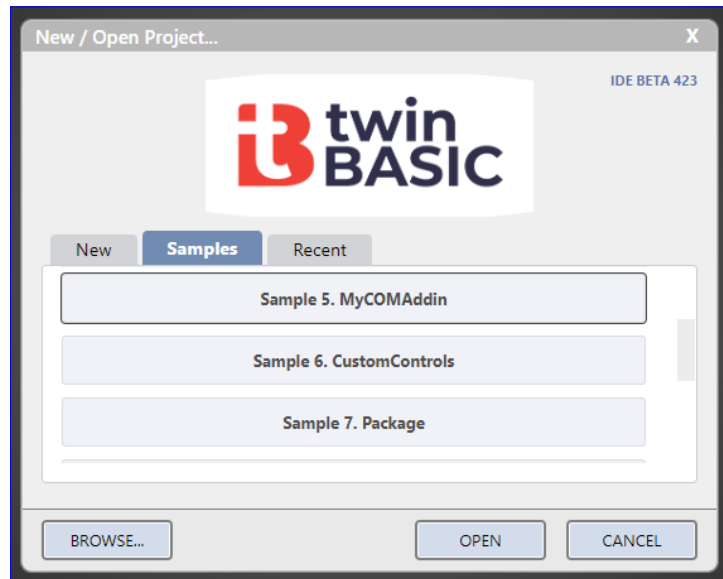


Bild 4: Neues Projekt auf Basis einer Vorlage

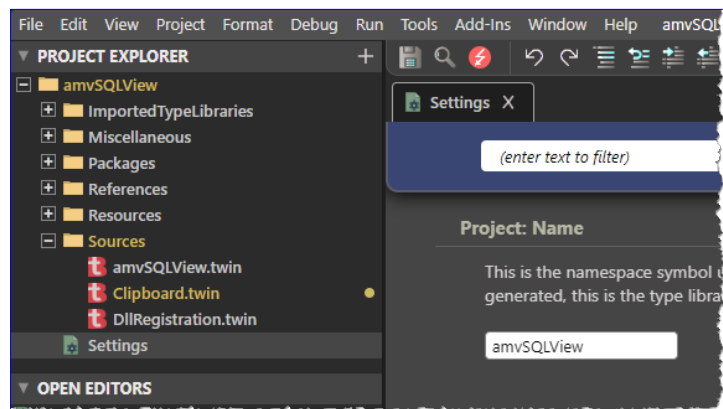


Bild 5: Projekteinstellungen anpassen

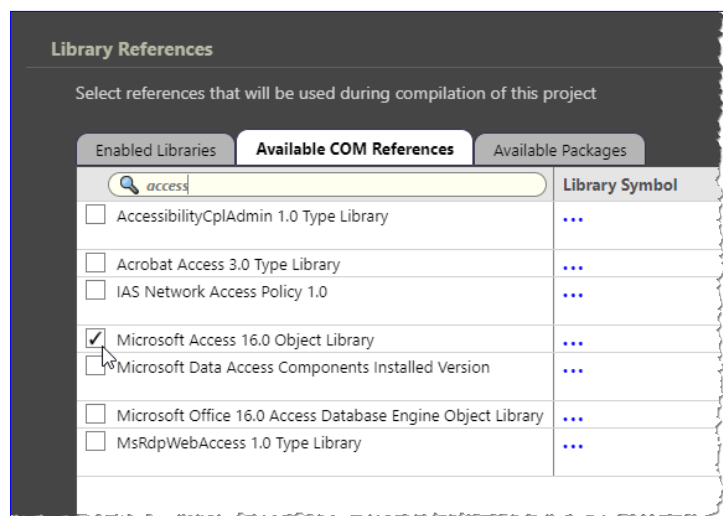


Bild 6: Hinzufügen eines Access-Verweises

```
Module DLLRegistration
    Const AddinProjectName As String = VBA.Compilation.CurrentProjectName
    Const AddinClassName As String = "amvSQLView"
    Const AddinQualifiedClassName As String = AddinProjectName & "." & AddinClassName
    Const RootRegistryFolder As String = "HKCU\SOFTWARE\Microsoft\Office\Access\Addins\" & AddinQualifiedClassName & "\"
    Public Function DLLRegisterServer() As Boolean
        On Error GoTo RegError
        Dim wscript As Object = CreateObject("wscript.shell")
        wscript.RegWrite RootRegistryFolder & "FriendlyName", AddinProjectName, "REG_SZ"
        wscript.RegWrite RootRegistryFolder & "Description", AddinProjectName, "REG_SZ"
        wscript.RegWrite RootRegistryFolder & "LoadBehavior", 3, "REG_DWORD"
        Return True
    RegError:
        MsgBox "DLLRegisterServer -- An error occured trying to write to the system registry:" & vbCrLf & _
            Err.Description & " (" & Hex(Err.Number) & ")"
        Return False
    End Function
    Public Function DLLUnregisterServer() As Boolean
        On Error GoTo RegError
        Dim wscript As Object = CreateObject("wscript.shell")
        wscript.RegDelete RootRegistryFolder & "FriendlyName"
        wscript.RegDelete RootRegistryFolder & "Description"
        wscript.RegDelete RootRegistryFolder & "LoadBehavior"
        wscript.RegDelete RootRegistryFolder
        Return True
    RegError:
        MsgBox "DLLUnregisterServer -- An error occured trying to delete from the system registry:" & vbCrLf & _
            Err.Description & " (" & Hex(Err.Number) & ")"
        Return False
    End Function
End Module
```

**Listing 1:** Modul für die Registrierung und die Deregistrierung des COM-Add-Ins

Moduls **DLLRegistration** des soeben erstellen Projekts können Sie durch den Code aus dem abgedruckten Listing ersetzen.

Wann werden diese Funktionen aufgerufen?

- Während der Entwicklung erfolgt der Aufruf der ersten Funktion, wenn wir das COM-Add-In durch einen Klick auf die **Build**-Schaltfläche oben rechts in der Symbolleiste von twinBASIC betätigen. Dann wird die DLL mit dem COM-Add-In erstellt, im Ordner **Build** abgelegt und registriert. Jedes Mal, wenn wir das tun, wird zuvor die Funktion **DLLUnregisterServer** aufgerufen, um die eventuell vorhandenen Einträge zuvor zu entfernen. Achtung: Wenn das COM-Add-In registriert und Access dieses nach dem Start geladen hat, kann dieses nicht deregistriert werden. Sie müssen Access zuvor schließen.
- Wir können die Funktion **DLLRegisterServer** auch manuell im Debugging-Modus aufrufen, indem wir auf den Pfeil über der ersten Zeile der Funktion klicken (siehe Bild 7). Auf die gleiche Weise starten wir **DLLUnregisterServer**, um die Registrierung der DLL aufzuheben.

- Die nächste Möglichkeit, für den Aufruf der Funktion **DLLRegisterServer** zu sorgen und das COM-Add-In zu registrieren, ist das Registrieren mit dem Kommandozeilentool **RegSvr32**. Diesem übergeben Sie den Pfad zur DLL als Parameter, was dazu führt, dass die enthaltene Funktion **DLLRegisterServer** ausgeführt und die DLL registriert wird. Der gleiche Befehl führt mit dem Parameter **-u** zum Aufruf der Funktion **DLLUnregisterServer**.
- Schließlich können wir den COM-Add-Ins-Dialog über die Access-Optionen öffnen (**Dateioptionen, Bereich Add-Ins**, unten **COM-Add-Ins** auswählen und auf **Los...** klicken). Im nun erscheinenden Dialog klicken wir auf **Hinzufügen...** und wählen die zuvor erstellte DLL aus, die dann registriert wird (siehe Bild 8). Durch Auswählen des neuen Eintrags und Anklicken der Schaltfläche **Entfernen** wird die DLL wieder deregistriert.
- Schließlich können wir in einem Inno Setup die DLL als Komponente im **Files**-Bereich angeben und mit dem Flag **regserver** versehen – siehe **COM-Add-Ins und -DLLs installieren oder registrieren, [www.access-im-unternehmen.de/1484](http://www.access-im-unternehmen.de/1484)**). Auch dies führt bei der Installation zum Aufruf der Funktion **DLLRegisterServer**.

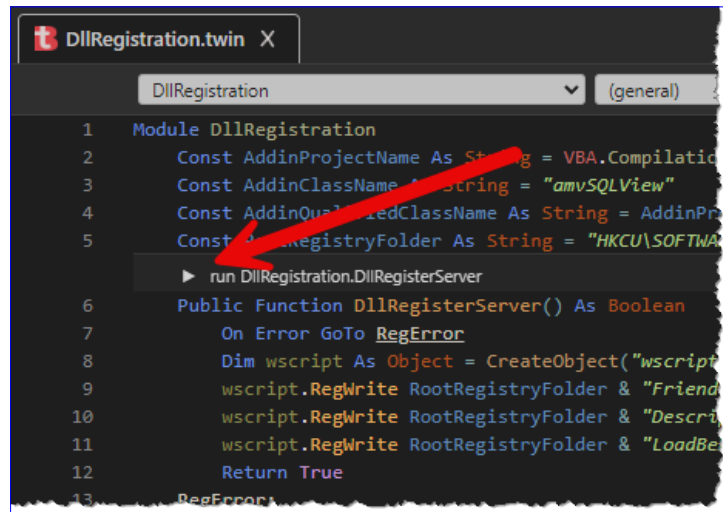


Bild 7: Start der Registrierung

weise beim Verbinden von Access mit dem COM-Add-In ausgelöst werden. Außerdem implementiert sie die Schnittstelle **IRibbonExtensibility**, die ein Ereignis bereitstellt, das beim Laden des Ribbons ausgelöst wird und das Anpassen des Ribbons ermöglicht. Wir deklarieren in der Klasse eine Variable, um die Access-Instanz zu referenzieren:

```
Private objAccess As Access.Application
```

In der beim Verbinden ausgelösten Ereignisprozedur **OnConnection** weisen wir dieser Variable den mit dem Parameter **Application** übergebenen Verweis auf die

## Programmieren der Ribbon-Erweiterung

Damit kommen wir zum Hauptteil der Arbeit, dem Klassenmodul **amv-SQLView** (siehe Listing 2). Dieses enthält eine eindeutige ID als Kennzeichnung.

Die Klasse implementiert die Schnittstelle **IDTExtensibility2**, die Ereignisse bereitstellt, die beispielsweise

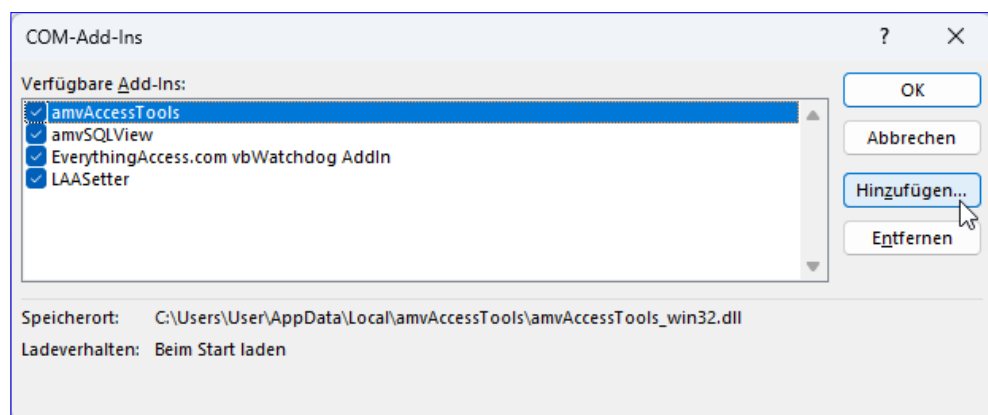


Bild 8: Registrierung über die Access-Optionen

## Hier geht es weiter ...

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

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

Alle Informationen hier – jetzt bestellen und direkt loslesen!

**ZUM SHOP**



## Formular an Mausposition öffnen

Verschiedene Anwendungszwecke machen es erforderlich, dass ein Formular an einer bestimmten Position geöffnet wird. Und selbst wenn es an sich nicht erforderlich ist, steigert es doch die Ergonomie, wenn zum Beispiel ein Popup-Formular an der Stelle erscheint, an der man den Befehl zum Öffnen betätigt hat – beispielsweise durch einen Klick auf eine Schaltfläche oder auf ein anderes Steuerelement. In diesem Beitrag schauen wir uns an, wie wir unabhängig von anderen Elementen ein Formular öffnen und an der Position des Mauszeigers positionieren können. Dazu benötigen wir lediglich ein paar Ereignisprozeduren und API-Funktionen.

### Ausgangsposition

Angenommen, wir wollen ein Popup-Formular (oder auch ein normales Formular) an der Stelle öffnen, an der sich gerade der Mauszeiger befindet. Das ist sinnvoll, denn wer möchte schon weite Wege mit der Maus zurücklegen, wenn er das zu öffnende Element direkt an der Mausposition anzeigen kann? Standardmäßig haben wir die Möglichkeit, für das Popup-Formular die Eigenschaft **Automatisch zentrieren** auf **Ja** einzustellen, dann würde das neue Formular zumindest in der Mitte des Access-

Fensters geöffnet werden. Ansonsten könnten wir mit **DoCmd.MoveSize** arbeiten, aber dies verwendet Koordinaten, die innerhalb des Access-Fensters liegen und die wir zuerst auch ermitteln müssen.

### Ziel dieses Beitrags

Am Ende wollen wir eine Lösung haben, mit der wir von einem Formular aus ein weiteres Formular per Mausklick öffnen können. Dabei soll die linke, obere Ecke des geöffneten Formulars sich an der Position des Mauszeigers befinden.

Dies könnte ungefähr wie in Bild 1 aussehen.

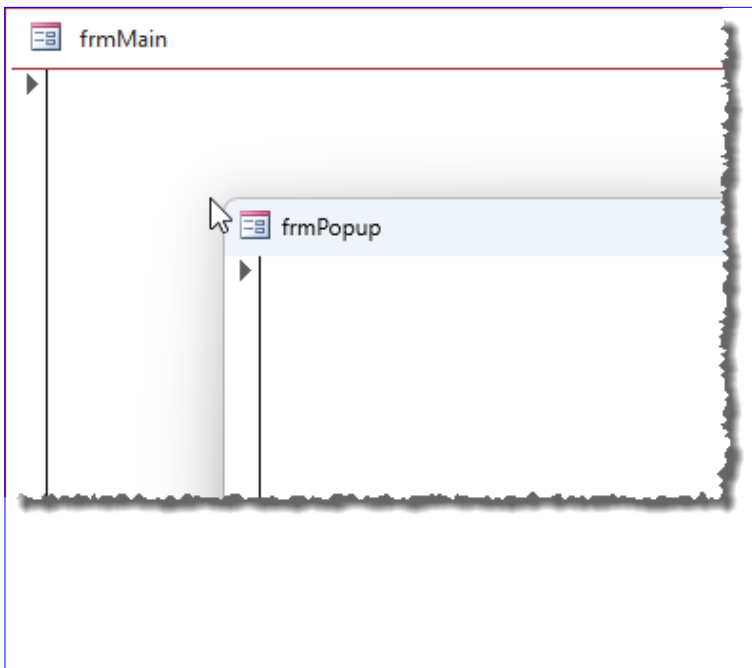


Bild 1: Öffnen eines Formulars an der Mausposition

### Lösung per API

Also nutzen wir API-Funktionen: Damit können wir nicht nur sehr schnell die aktuelle Position des Mauszeigers ermitteln, sondern auch noch mit einem Befehl die neue Position festlegen.

Dazu benötigen wir zuerst einen **Type** namens **RECT**, der die vier Eigenschaften **Left**, **Top**, **Right** und **Bottom** enthält. Wenn wir eine API-Funktion beispielsweise zum Ermitteln der aktuellen Position und Größe eines Fensters nutzen, liefert diese immer ein solches Konstrukt mit den entsprechenden Werten zurück:

```
Public Type RECT
    Left As Long
    Top As Long
```



```
Public Declare PtrSafe Function GetCursorPos Lib "user32" (lpPoint As POINTAPI) As Long
Public Declare PtrSafe Function GetWindowRect Lib "user32" (ByVal hwnd As LongPtr, lpRect As RECT) As Long
Public Declare PtrSafe Function MoveWindow Lib "user32" (ByVal hwnd As LongPtr, ByVal x As Long, ByVal y As Long, _
    ByVal nWidth As Long, ByVal nHeight As Long, ByVal bRepaint As Long) As Long
```

**Listing 1:** Die benötigten API-Funktionen

```
Right As Long
Bottom As Long
End Type
```

Daneben benötigen wir zum Ermitteln der Koordinaten eines Punktes wie zum Beispiel für die Position des Mauszeigers einen Type mit dem folgenden Aufbau:

```
Private Type POINTAPI
    X As Long
    Y As Long
End Type
```

Diese beiden Typen deklarieren wir öffentlich zugänglich in einem Modul namens **mdiAPI**.

Außerdem legen wir dort die drei API-Funktionen aus Listing 1 an. Diese sind bereits für die 64-Bit-Version von Office ausgelegt. Die API-Funktionen haben die folgenden Aufgaben:

- **GetCursorPos:** Ermittelt die aktuelle Position des Mauszeigers und schreibt diese in die Eigenschaften **X** und **Y** eines **POINTAPI**-Types.
- **GetWindowRect:** Ermittelt die Position und Größe eines Fensters, das mit dem Handle **hwnd** referenziert wird und schreibt diese Informationen in einen **RECT**-Typ.
- **MoveWindow:** Erlaubt das Einstellen der Position und der Größe des mit **hwnd** referenzierten Fensters. **x** und **y** nehmen die Koordinaten der linken, oberen Ecke entgegen und **nWidth** und **nHeight** die Höhe und die Breite. Der letzte Parameter **bRepaint** legt fest, ob das Fenster neu gezeichnet werden soll.

Neben diesen Elementen deklarieren wir im Modul **mdiAPI** noch die folgenden beiden Variablen:

```
Public lngX As Long
Public lngY As Long
```

Diese sollen die Position des Mauszeigers beim Aufrufen des zu öffnenden Formulars speichern und im geöffneten Formular zur Abfrage bereitstellen.

### Beispiel zum Öffnen des Formulars an der Position des Mauszeigers

Zu Beispielzwecken haben wir die ansonsten leeren Formulare **frmMain** und **frmPopup** angelegt. Wenn der Benutzer irgendwo in das Formular **frmMain** klickt, soll das Formular **frmPopup** an der Position des Mauszeigers erscheinen.

Im Klassenmodul des Formulars **frmMain** deklarieren wir eine Variable namens **pt** mit dem Typ **POINTAPI**:

```
Private pt As POINTAPI
```

Den Mausklick fangen wir gleich mit zwei Ereignissen ab – **Bei Maustaste ab** und **Bei Maustaste auf**. Die Frage ist: Wann soll das Formular erscheinen, beim Herunterdrücken der Maustaste oder erst beim Loslassen? Wir haben uns an dem Verhalten zur Anzeige von Kontextmenüs orientiert. Diese werden erst beim Loslassen des Mauszeigers eingeblendet. Und wo soll das neue Formular geöffnet werden – dort, wo sich der Mauszeiger beim Herunterdrücken befindet oder beim Loslassen? Hier gibt es durchaus unterschiedliche Varianten. In Windows-Anwendungen wie dem Windows Explorer oder auch auf dem Desktop wird das Kontextmenü dort geöffnet, wo die

## Hier geht es weiter ...

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

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

Alle Informationen hier – jetzt bestellen und direkt loslesen!

**ZUM SHOP**



## Modale Dialoge nach Wunsch gestalten

Modale Dialoge, also Formulare, die mit dem Befehl »DoCmd.OpenForm« mit dem Parameter »WindowMode:=acDialog« geöffnet wurden, sind in vielen Fällen hilfreich: Wir können den aufrufenden Code anhalten, bis das Formular geschlossen oder ausgeblendet wird und so gegebenenfalls auszulesende Werte ermitteln. Oder wir sorgen so dafür, dass der Benutzer nicht an anderen Formularen arbeiten kann, bevor er nicht die Eingabe in dieses Formular abgeschlossen hat. Einen Nachteil haben modale Dialoge allerdings: Wir können ihre Rahmenart nicht so einstellen, wie wir es von normal geöffneten Formularen gewohnt sind. Wollen wir also einen modalen Dialog einmal ohne Titelleiste anzeigen, weil er zum Beispiel einfach als Erweiterung neben einem anderen Steuerelement geöffnet werden soll, können wir das so nicht machen. Es gibt allerdings einen Workaround, den wir hier vorstellen.

### Modaler Dialog, der »normale« Weg

Der übliche Weg, ein Formular als modalen Dialog zu öffnen, ist dieser:

```
Private Sub cmdOpenForm_Click()
    DoCmd.OpenForm "frmPopup", WindowMode:=acDialog
    If IstFormularGeoeffnet("frmPopup") Then
        MsgBox "Eingebener Wert: " _
            & Forms!frmPopup!txtWert
        DoCmd.Close acForm, "frmPopup"
    Else
        MsgBox "Eingabe abgebrochen"
    End If
End Sub
```

Im aufgerufenen Formular fügen wir eine **OK**- und eine **Abbrechen**-Schaltfläche hinzu. **OK** blendet das Formular aus, was das Fortsetzen des aufrufenden Codes erlaubt, der dann den Wert der Steuerelemente des Popups auslesen kann, bevor er dieses schließt:

```
Private Sub cmdOK_Click()
    Me.Visible = False
End Sub
```

**Abbrechen** schließt das Formular, was wir in der aufrufenden Prozedur als Abbruch der Eingabe interpretieren können:

```
Private Sub cmdAbbrechen_Click()
    DoCmd.Close acForm, Me.Name
End Sub
```

Die aufrufende Prozedur prüft mit der Funktion **IstFormularGeoeffnet**, ob der Benutzer **OK** oder **Abbrechen**

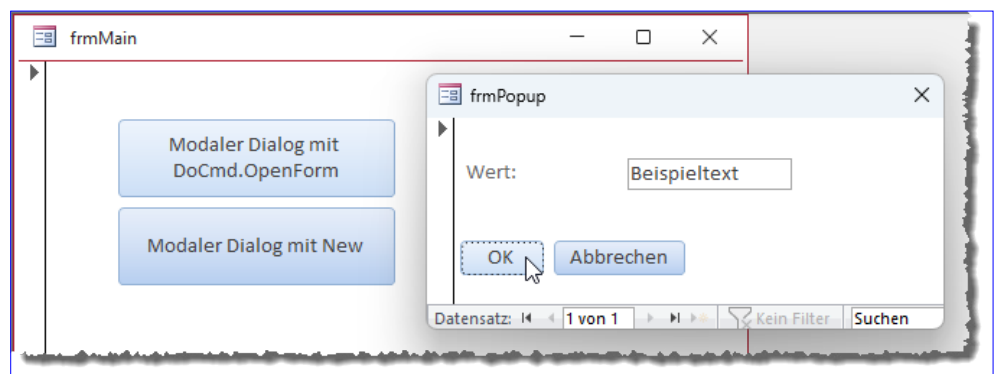


Bild 1: Beispiel für einen modalen Dialog

angeklickt hat. Das aufrufende Formular und das modale Formular sehen wie in Bild 1 aus.

#### Nachteil dieser Variante

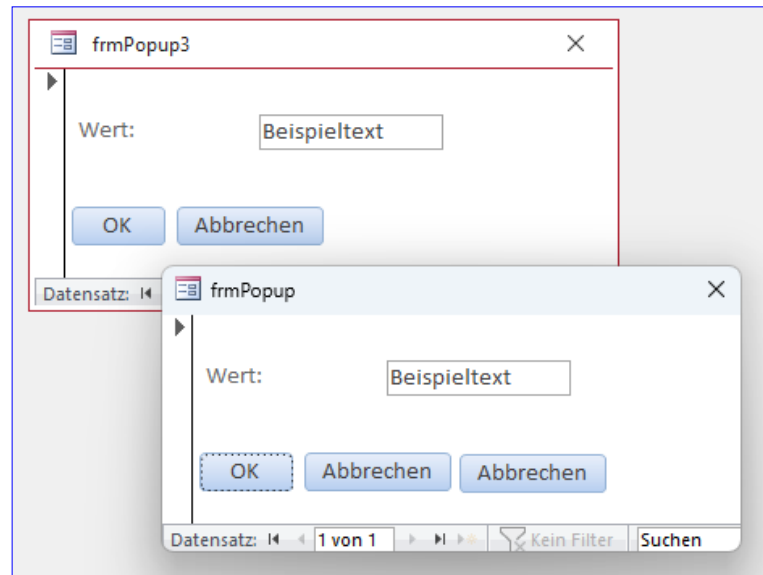
Der große Nachteil ist: Wir können bestimmte Dinge nicht beeinflussen, weil durch den Aufruf mit **WindowMode:=acDialog** einige Eigenschaften automatisch eingestellt werden – zum Beispiel die Rahmenart. Diese können wir zwar im Entwurf einstellen, aber letztlich wird das Formular immer mit der Rahmenart **Dialog** angezeigt – auch wenn wir den tatsächlich für diese Eigenschaft eingestellten Wert erhalten, wenn wir uns diesen beispielsweise per Meldungsfenster ausgeben lassen. Außerdem ist die Eigenschaft **Modal** für Formulare, die mit **WindowMode:=acDialog** geöffnet wurden, immer auf **Ja** eingestellt. Dies sorgt dafür, dass der Rahmen abgerundet dargestellt wird. Den Unterschied sehen wir in Bild 2.

#### Modale Formulare mit einstellbarer Rahmenart

Wenn wir zum Beispiel ein Formular mit der Einstellung **Keine** für die Eigenschaft **Rahmenart** als modalen Dialog anzeigen wollen, müssen wir einen ganz anderen Ansatz wählen. Dabei rufen wir das gewünschte Formular als neue Objektinstanz auf statt mit **DoCmd.OpenForm**.

Damit das Formular auch noch als modaler Dialog angezeigt wird, also die übrigen Formulare solange deaktiviert werden, bis das aufgerufene Formular wieder geschlossen ist, stellen wir nach dem Instanzieren des Formulars noch die Eigenschaft **Modal** auf den Wert **True** ein und machen es dann mit **Me.Visible = True** sichtbar:

```
Private Sub cmdNew_Click()
    Set frm = New Form_frmPopupNew
    With frm
        .Modal = True
        .Visible = True
    End With
```



**Bild 2:** Unterschied zwischen Rahmenart gleich Dialog und einem tatsächlich als Dialog geöffnetem Formular

End Sub

Die hier verwendete Objektvariable **frm** müssen wir unbedingt außerhalb der Prozedur deklarieren. Wenn wir sie in der Prozedur deklarieren, verliert sie bereits beim Beenden der Prozedur ihre Gültigkeit und auch das darin referenzierte Formular wird gelöscht. Deshalb deklarieren wir die Variable im Kopf des Formulars:

Dim frm As Form

Probieren wir dies aus, erleben wir einen Unterschied zum Öffnen von modalen Formularen mit **DoCmd.OpenForm**: Der Navigationsbereich wird eingeklappt. Dadurch wird auch das aufrufende Formular nach links verschoben. Wenn wir das Popup-Formular beispielsweise direkt neben einem anderen Steuerelement öffnen wollen, gelingt das nun nicht mehr so einfach, da wir die Koordinaten neu berechnen müssen. Und auch wenn wir eine professionelle Anwendung ohnehin ohne Anzeige des Navigationsbereichs ausliefern, wollen wir dieses Verhalten abschalten.

Das erledigen wir durch Einstellen der Eigenschaft **Popup**. Diese ist zwar auch unter VBA verfügbar, aber sie kann zur



## Hier geht es weiter ...

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

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

Alle Informationen hier – jetzt bestellen und direkt loslesen!

**ZUM SHOP**





## SQL Server zwischen zwei Rechnern im Netzwerk

Wer wie ich den SQL Server auf dem gleichen Desktop-Rechner installiert hat, auf dem er auch arbeitet, macht sich über den Einsatz im Netzwerk keine großen Gedanken. Neulich aber habe ich einen neuen Rechner aufgesetzt und es fehlte schlicht die Zeit, auch den SQL Server und die damit verwalteten Datenbanken auf dem neuen Rechner zu installieren. Also dachte ich, dass ich einfach über das Netzwerk auf den SQL Server des alten Rechners zugreifen könnte. Das geht jedoch nicht ohne Weiteres und je nach Konfiguration ist der eine oder andere Schritt nötig. Also schauen wir uns an, was bei mir zum Erfolg geführt hat.

In einem ersten, naiven Versuch habe ich einfach einmal die Frontend-Datenbank auf dem neuen Rechner geöffnet und versucht, auf die Tabellen der SQL Server-Datenbank auf dem alten Rechner zuzugreifen.

Das ist jedoch nicht gelungen – mein standardmäßig verwendetes Formular zum Einrichten und Testen von Verbindungen konnte die Verbindung nicht herstellen (siehe Bild 1).

Es gibt verschiedene Einstellungen auf Serverebene, die nötig sein können, damit eine Verbindung zu einem SQL Server auf einem anderen Rechner möglich sind. Diese schauen wir uns nachfolgend an. Es kann sein, dass Sie

nicht alle modifizieren müssen, um zugreifen zu können – aber bevor wir hier zu wenige Schritte abbilden, machen wir es lieber ein wenig ausführlicher. Die Schritte auf Serverebene lauten:

- Aktivieren von Remoteverbindungen im **SQL Server Management Studio**

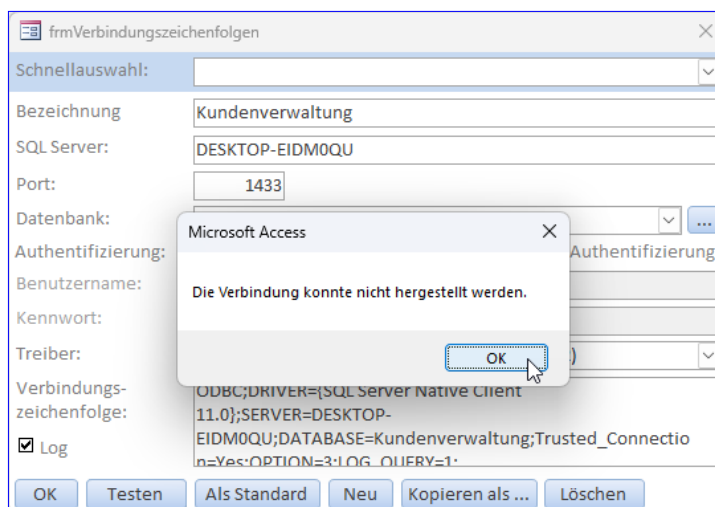


Bild 1: Der erste Zugriff ist misslungen.

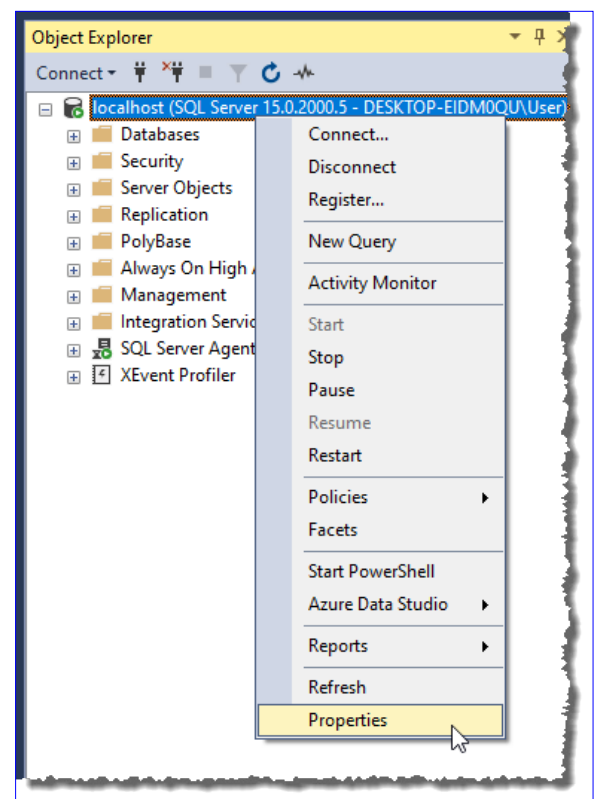


Bild 2: Öffnen der Eigenschaften des SQL Servers

- Port für den Zugriff auf den SQL Server in der Firewall öffnen
- SQL Server-Browser aktivieren
- TCP/IP aktivieren
- Windows-Benutzer angleichen

### SQL Server-Remoteverbindungen zulassen

Im ersten Schritt öffnen wir das **SQL Server Management Studio** auf dem alten Rechner.

Hier klicken wir mit der rechten Maustaste auf den Eintrag für den Server selbst und wählen aus dem Kontextmenü den Befehl **Properties** aus (in der deutschen Version **Eigenschaften**) – siehe Bild 2.

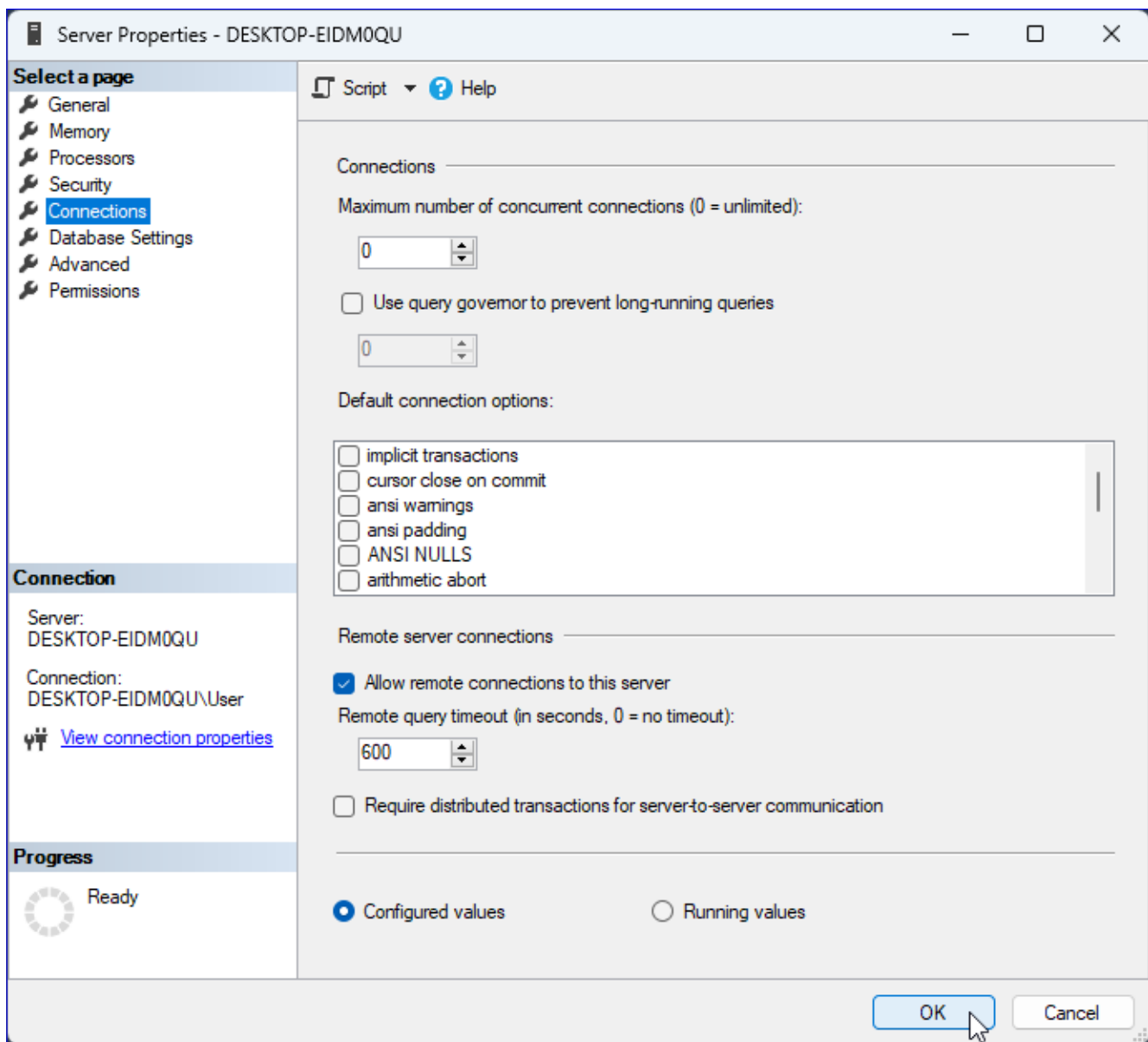


Bild 3: Aktivieren der Remoteverbindungen

Dies öffnet den Dialog **Server Properties** für den SQL Server. Hier wechseln wir zum Bereich **Connections** und aktivieren die Option **Allow remote connections to this server** (siehe Bild 3).

Damit erreichen wir Folgendes:

- Diese Einstellung bezieht sich darauf, ob der SQL Server Remoteverbindungen von Clientcomputern akzeptiert oder nicht.
- Wenn **Remote Server Connections** deaktiviert ist, können nur lokale Anwendungen auf dem Server selbst auf den SQL Server zugreifen. Dies ist die restriktivste Einstellung und verhindert, dass externe Clientcomputer Verbindungen zum SQL Server herstellen.
- Wenn **Remote Server Connections** aktiviert ist, können Remote-Clientcomputer Verbindungen zum SQL Server herstellen, vorausgesetzt, die anderen Netzwerk- und Sicherheitseinstellungen sind ebenfalls korrekt konfiguriert.

In unseren Fall schlägt der Test der Verbindung danach allerdings immer noch fehl, also gehen wir zum nächsten Schritt über.

## Firewall für den Zugriff auf den SQL Server öffnen

Wenn der Zugriff auf den SQL Server noch nicht klappt, kann es sein, dass die Firewall auf dem Rechner mit dem SQL Server dies verhindert.

Um die Firewall-Einstellungen vorzunehmen, öffnen wir zuerst die Systemsteuerung. Hier finden wir den Eintrag **Windows Defender**

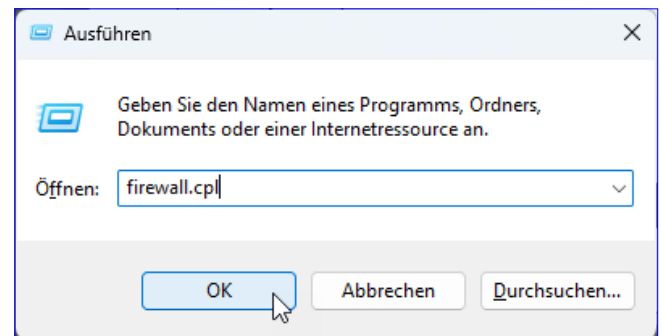


Bild 5: Starten der Systemsteuerung für die Firewall

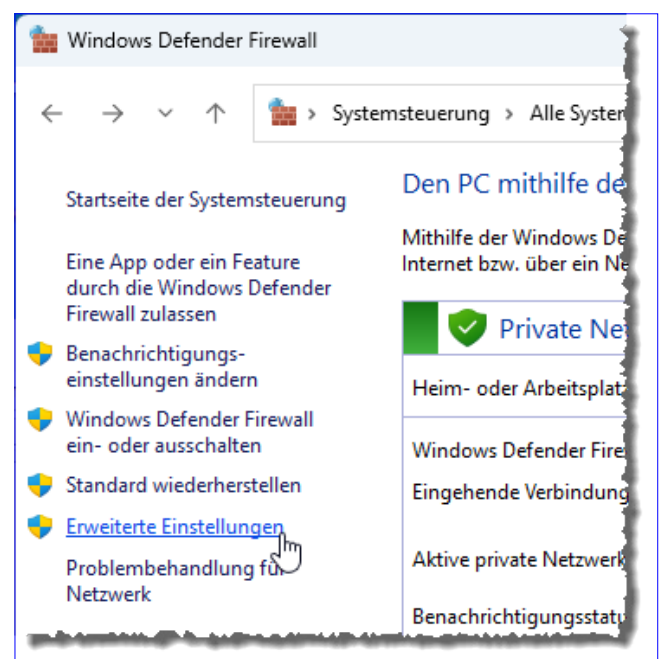


Bild 4: Anzeigen der erweiterten Einstellungen

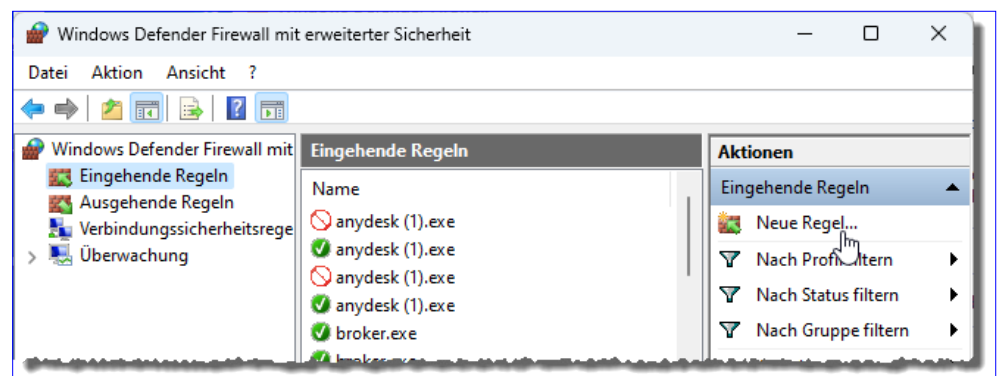


Bild 6: Anlegen einer neuen Regel

## Hier geht es weiter ...

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

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

Alle Informationen hier – jetzt bestellen und direkt loslesen!

**ZUM SHOP**



## COM-Add-Ins und -DLLs installieren oder registrieren

Wenn man eine COM-DLL oder ein COM-Add-In mit VB6 oder twinBASIC erstellt hat, muss dieses registriert werden, damit es in einer der Office-Anwendungen angezeigt wird oder unter VBA zur Verfügung steht. Die Informationen für die Registrierung werden in der Regel direkt in die jeweilige .dll-Datei integriert, sodass man die Registrierung nur noch von außen initialisieren muss. Dafür gibt es verschiedene Wege, die sich je nach Zweck (COM-DLL oder COM-Add-In) unterscheiden. Einer davon ist das Erstellen eines Setups mit Inno Setup. Dieses braucht der Anwender nur auszuführen und muss keine Verrenkungen machen, um die neuen Funktionen auf seinem Rechner nutzen zu können. Es gibt aber auch noch alternative Methoden, die wir ebenfalls in diesem Beitrag vorstellen.

Im Beitrag **Abfragen direkt in der SQL-Ansicht erstellen** ([www.access-im-unternehmen.de/1483](http://www.access-im-unternehmen.de/1483)) haben wir gezeigt, wie wir mit twinBASIC ein COM-Add-In erstellen können, das sich in die Access-Benutzeroberfläche integriert.

Auf die gleiche Weise können wir mit twinBASIC auch COM-DLLs erstellen, also DLLs, deren Funktionen wir in das VBA-Projekt einer Anwendung integrieren und diese dort aufrufen können.

In beiden Fällen definieren wir im twinBASIC-Projekt Funktionen namens **DLLRegisterServer** und **DLLUnregisterServer**, welche die Befehle zum Hinzufügen und Entfernen bestimmter Einträge in der Registry enthält.

Diese Funktionen werden durch verschiedene Aktionen ausgelöst, wobei eine Aktion nur für COM-Add-Ins zur Verfügung steht, nicht jedoch für COM-DLLs. Wir schauen uns diese nun an.

### Installieren von COM-Add-Ins

Für die Installation von COM-Add-Ins gibt es die folgenden Möglichkeiten:

- Registrierung per Eingabeaufforderung mit **RegSvr32.exe**

- Registrierung über den COM-Add-Ins-Dialog in Access
- Registrierung per Setup, beispielsweise mit Inno Setup zu erstellen

### Installation von COM-DLLs

Bei COM-DLLs, die ihre Befehle zur Verfügung stellen, wenn man sie als Bibliothek zu einem VBA-Projekt hinzufügt, gibt es nur zwei Möglichkeiten:

- Registrierung per Eingabeaufforderung mit **RegSvr32.exe**
- Registrierung per Setup, das beispielsweise mit Inno Setup erstellt werden kann

### Registrierung per Eingabeaufforderung

Für die Registrierung per Eingabeaufforderung braucht man nicht viel zu wissen – allein der Speicherort der zu registrierenden Datei ist wichtig.

Dafür öffnen wir als Erstes die Eingabeaufforderung von Windows. Dies allerdings nicht einfach so, sondern wir benötigen in den meisten Fällen den Administratormodus. Dazu geben wir **cmd** in das **Suche**-Fenster von Windows ein und klicken mit der rechten Maustaste auf den Eintrag **Eingabeaufforderung**. Aus dem nun erscheinenden Kon-



textmenü wählen wir den Eintrag **Als Administrator ausführen...** aus (siehe Bild 1).

Alternativ kann man auch den direkt eingeblendeten Befehl **Als Administrator ausführen** im rechten Bereich des **Suchen**-Bereichs anklicken.

Hier haben wir nun zwei Möglichkeiten: Entweder wir navigieren mit der **cd**-Anweisung in das Verzeichnis, in dem sich die zu registrierende **.dll**-Datei befindet, oder wir kopieren den Pfad aus dem Windows Explorer in die Zwischenablage, um ihn anschließend weiterzuarbeiten.

In jedem Fall müssen wir die **regsvr32.exe** ausführen und den Namen der **.dll**-Datei als Parameter übergeben.

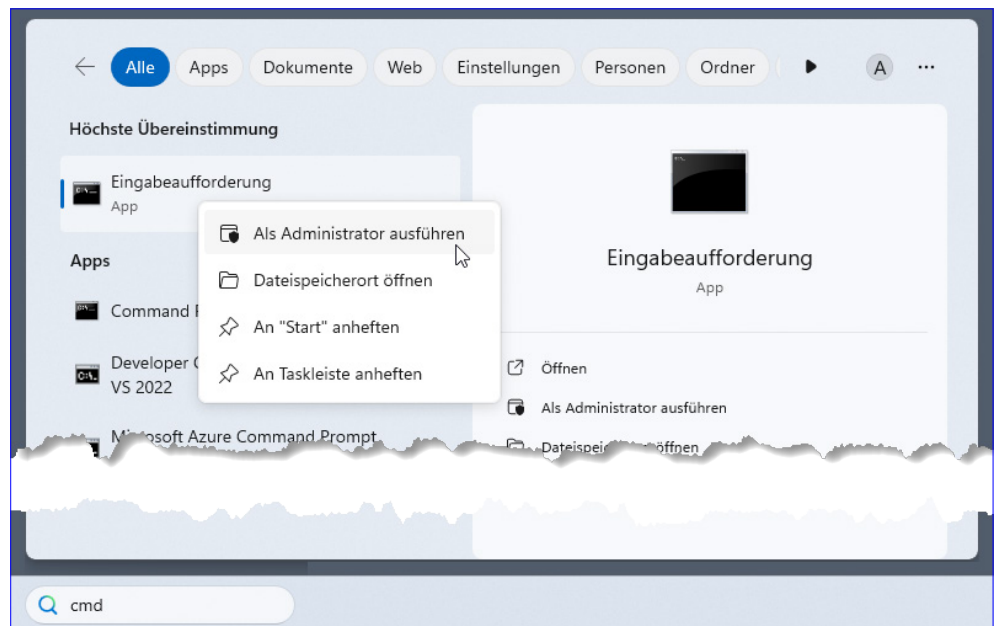
Wir sind hier zuerst mit der **cd**-Anweisung zu dem Verzeichnis navigiert, in dem wir das COM-Add-In gespeichert haben:

```
C:\Users\User\AppData\Roaming\Microsoft\AddIns
```

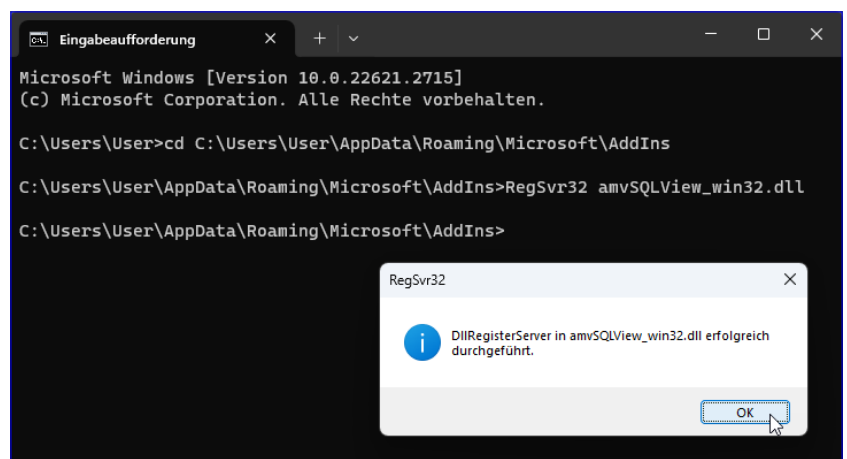
Dann haben wir den Befehl zum Registrieren eingegeben:

```
RegSvr32 amvSQLView_win32.dll
```

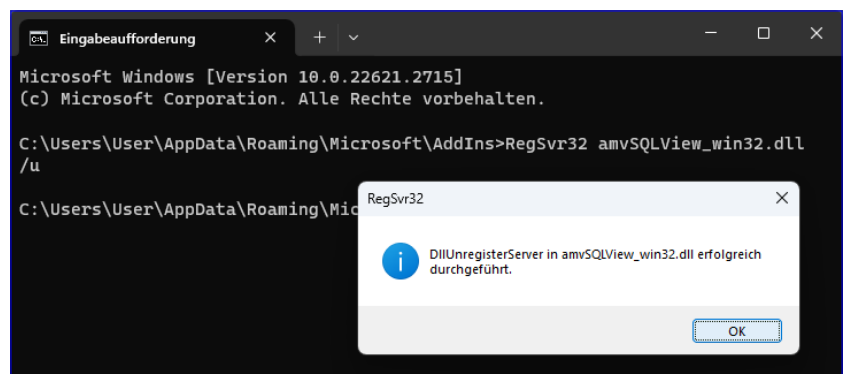
Das Ergebnis ist die Meldung aus Bild 2.



**Bild 1:** Öffnen der Eingabeaufforderung als Administrator



**Bild 2:** Registrieren einer DLL über die Eingabeaufforderung



**Bild 3:** Aufheben der Registrierung einer DLL über die Eingabeaufforderung

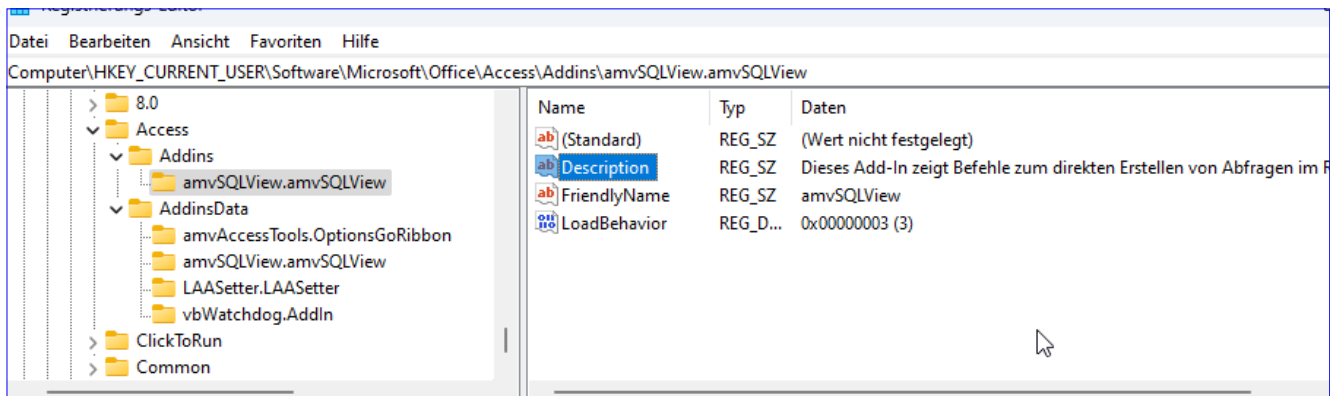


Bild 4: Hauptregistry-Einträge für das COM-Add-In

### Registrierung aufheben per Eingabeaufforderung

Wir können diese Registrierung auch wieder über die Eingabeaufforderung entfernen. Dazu geben wir zusätzlich zu dem soeben angegebenen Befehl noch den Parameter **/u** an. Die Bestätigung sieht ähnlich aus wie beim Registrieren der DLL (siehe Bild 3).

### Was geschieht beim Registrieren?

Beim Registrieren erfolgen genau die Schritte, die in der Funktion **DLLRegisterServer** angegeben sind. In diesem Fall legen wir einige Einträge in der Registry an. Diese können wir uns im Registry-Editor anschauen. Den Registry-Editor starten wir mit dem Befehl **Regedit**, den wir ebenfalls in das **Suchen**-Feld von Windows eingeben.

falls wichtig. So sehen wir in den hier angelegten Einträgen eigentlich noch keine wirkliche Information, wo sich die aufzurufende DLL befindet.

Genau genommen sagen die Werte folgendes aus:

- **Description:** Dieser Text wird in der Übersicht der Add-Ins einer Access-Anwendung angezeigt.
- **FriendlyName:** Wert, der im später vorgestellten Dialog COM-Add-Ins angezeigt wird.
- **LoadBehaviour:** Gibt an, wann das COM-Add-In geladen wird. **3** bedeutet beispielsweise: **Beim Start laden**.

Hier suchen wir mit **BearbeitenSuchen** nach der Registrierung nach dem Wert **amvSQLView**. Wir finden einige Vorkommen, aber die in der Funktion **DLLRegisterServer** angegebenen landen in dem Bereich aus Bild 4.

Die übrigen werden bei der Installation automatisch vorgenommen und sind für die Funktionsweise eben-

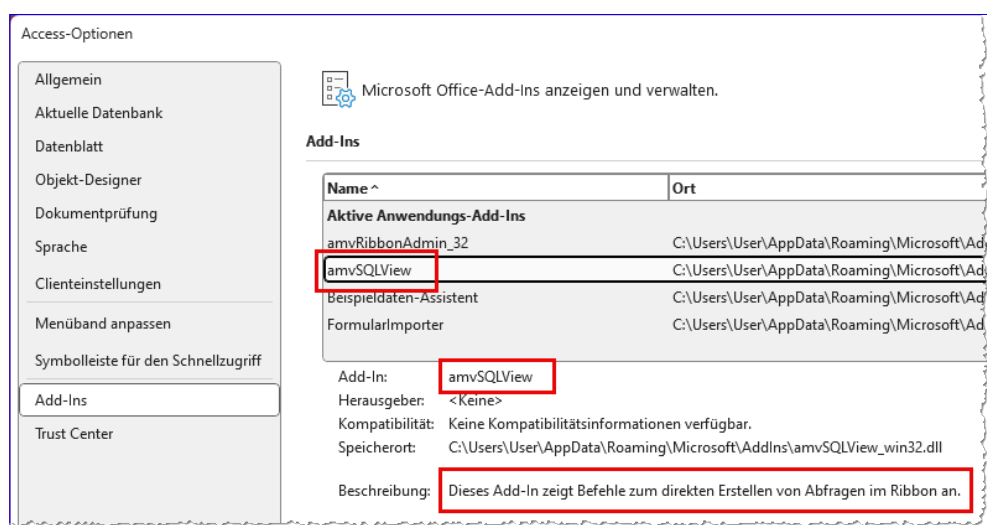
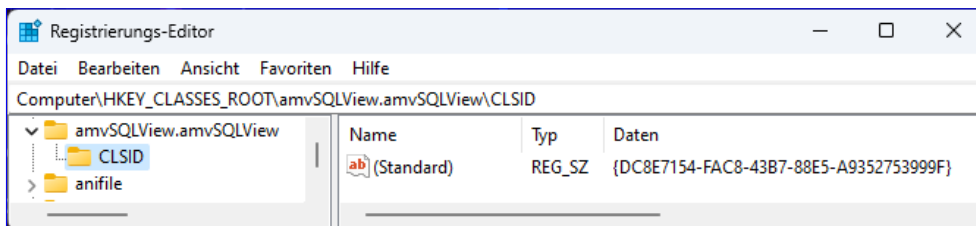


Bild 5: Informationen aus der Registry im Add-Ins-Bereich



**Bild 6:** Eintrag mit dem Namen von Projekt/Klasse und der GUID, die im Projekt als **ClassId** festgelegt ist

Wo diese Werte landen, sehen wir in Bild 5. Der Inhalt aus **FriendlyName** landet in der Liste der Add-Ins und unter **Add-In**. Der Wert aus **Description** ist unten unter **Beschreibung** zu finden.

### Welche weiteren Registry-Einträge werden angelegt?

Neben den hier genannten werden noch einige weitere Registry-Einträge bei jeder der hier vorgestellten Methode angelegt. Einer unter **Computer\HKEY\_CLASSES\_ROOT\amvSQLView.amvSQLView** weist beispielsweise dem Klassennamen die **ClassId** zu.

In diesem Fall entspricht die **ProgID** **amvSQLView.amvSQLView** der **CLSID** mit dem Wert **{DC8E7154-FAC8-43B7-88E5-A9352753999F}** (siehe Bild 6). Die **ClassId** wird übrigens normalerweise in der Hauptklasse der DLL zugewiesen, im Falle von **amvSQLView** also **amvSQLView.twin**.

In einem weiteren Eintrag unter **Computer\HKEY\_CLASSES\_ROOT\WOW6432Node\CLSID\{DC8E7154-FAC8-43B7-88E5-A9352753999F}\InProcServer32** werden die **ClassId** und unter **InProcServer** die zu verwendende

Datei zusammengeführt (siehe Bild 7).

### Wozu die Registry-Einträge?

Die Registry-Einträge dienen im Falle eines COM-Add-Ins dazu, dass die

Anwendung, der das COM-Add-In zugeordnet ist, davon erfährt, dass es das COM-Add-In gibt.

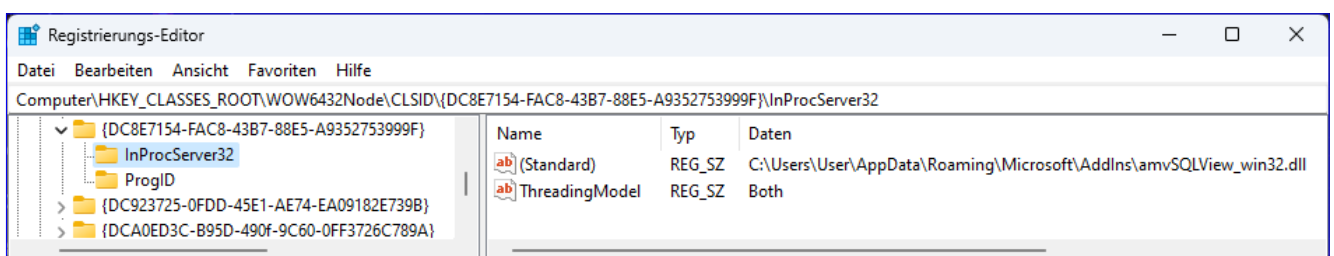
Dann kann es beim Öffnen auf das COM-Add-In zugreifen und darin enthaltene Prozeduren zum Initialisieren aufrufen.

Das sorgt beispielsweise dafür, dass die Ribbon-Erweiterung eines COM-Add-Ins beim Start von Access eingelesen wird. Bei COM-DLLs sorgt der Registry-Eintrag dafür, dass die DLL im **Verweise**-Dialog zur Auswahl angeboten wird.

### Registrierung per COM-Add-Ins-Dialog

COM-Add-Ins für Access können wir noch auf einem zusätzlichen Weg registrieren. Dies gilt nicht für COM-Add-Ins für den VBA-Editor.

Zum Registrieren eines COM-Add-Ins für Access starten wir Access und öffnen eine beliebige Datenbank. Dann rufen wir den **Optionen**-Dialog auf. Hier wechseln wir zum Bereich **Add-Ins**, wo wir das Auswahlfeld **Verwalten** finden. Hier ist bereits der Eintrag **COM-Add-Ins** vor-eingestellt, sodass wir nur noch die Schaltfläche **Los...** betätigen müssen (siehe Bild 8).



**Bild 7:** Zuweisung des Speicherortes zu der **ClassId**

## Hier geht es weiter ...

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

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

Alle Informationen hier – jetzt bestellen und direkt loslesen!

**ZUM SHOP**





## Assistent für Domänenfunktionen

Domänenfunktionen wie DomWert (DLookup), DomMax (DMax) oder DomAnzahl (DCount) sind praktische Helfer, wenn es um das schnelle Zugreifen auf verschiedene Informationen einer Datenbank geht. Wir können damit den Wert eines oder mehrerer Felder aus einem Datensatz mit einem bestimmten Kriterium ermitteln, die Gesamtanzahl von Datensätzen einer Tabelle mit oder ohne Kriterium oder auch Minimal- oder Maximalwerte. Der Assistent, den wir in diesem Beitrag vorstellen, hilft dabei auf verschiedene Arten. Die erste Option ist, dass Sie ihn einfach aufrufen können, um schnell die Werte für eine Domänenfunktion einzutippen und das Ergebnis auszulesen. Aber der Assistent kann noch mehr: Sie können ihn auch von Eigenschaftsfeldern heraus aufrufen, um die gewählte Domänenfunktion direkt dort einzutragen.

Damit Sie sich direkt einen Eindruck davon machen können, wie der fertige Assistent aussieht, schauen Sie sich Bild 1 an. Hier finden Sie im oberen Bereich die Eingabefelder für die Parameter der Domänenfunktionen.

Wir haben die Reihenfolge im Vergleich zur Funktion etwas umgestellt, sodass die Auswahl der Domäne der erste Schritt ist. Das hat den Hintergrund, dass wir nach der Auswahl der Tabelle oder Abfrage, aus der die zu

amvDomänenfunktion

Parameter:

Domäne:

Ausdruck:

Kriterium:

Ergebnisse: Ziel: ☒ Ausdruck ☐ VBA

<input type="radio"/> DomAnzahl (DCount):	8	DomAnzahl([Nachname] & ", " & [Vorname];tblAutoren;[Vorname] LIKE 'A*')
<input checked="" type="radio"/> DomErsterWert (DFirst):	Christie, Agatha	DomErsterWert([Nachname] & ", " & [Vorname];tblAutoren;[Vorname] LIKE 'A*')
<input type="radio"/> DomLetzterWert (DLast):	Minhorst, André	DomLetzterWert([Nachname] & ", " & [Vorname];tblAutoren;[Vorname] LIKE 'A*')
<input type="radio"/> DomMax (DMax):	W.H. Auden, Arundhati	DomMax([Nachname] & ", " & [Vorname];tblAutoren;[Vorname] LIKE 'A*')
<input type="radio"/> DomMin (DMin):	Christie, Agatha	DomMin([Nachname] & ", " & [Vorname];tblAutoren;[Vorname] LIKE 'A*')
<input type="radio"/> DomMittelwert (DAvg):	#Fehler	DomMittelwert([Nachname] & ", " & [Vorname];tblAutoren;[Vorname] LIKE 'A*')
<input type="radio"/> DomStAbw (DStDev):	#Fehler	DomStAbw([Nachname] & ", " & [Vorname];tblAutoren;[Vorname] LIKE 'A*')
<input type="radio"/> DomStAbwn(DStDevP):	#Fehler	DomStAbwn([Nachname] & ", " & [Vorname];tblAutoren;[Vorname] LIKE 'A*')
<input type="radio"/> DomSumme (DSum):	#Fehler	DomSumme([Nachname] & ", " & [Vorname];tblAutoren;[Vorname] LIKE 'A*')
<input type="radio"/> DomVarianz (DVar):	#Fehler	DomVarianz([Nachname] & ", " & [Vorname];tblAutoren;[Vorname] LIKE 'A*')
<input type="radio"/> DomVarianzen (DVarP):	#Fehler	DomVarianzen([Nachname] & ", " & [Vorname];tblAutoren;[Vorname] LIKE 'A*')
<input type="radio"/> DomWert (DLookup):	Christie, Agatha	DomWert([Nachname] & ", " & [Vorname];tblAutoren;[Vorname] LIKE 'A*')

✓ ✗

Bild 1: Der Domänenfunktions-Assistent in Aktion

ermittelnden Daten stammen sollen, die darin enthaltenen Felder per Kontextmenü zum Einfügen in die folgenden beiden Textfelder bereit halten wollen.

Nachdem wir beispielsweise eine Tabelle wie **tblAutoren** ausgewählt haben, liefert ein Rechtsklick auf eines der Textfelder für den Ausdruck oder das Kriterium die Liste der Felder dieser Tabelle (siehe Bild 2).

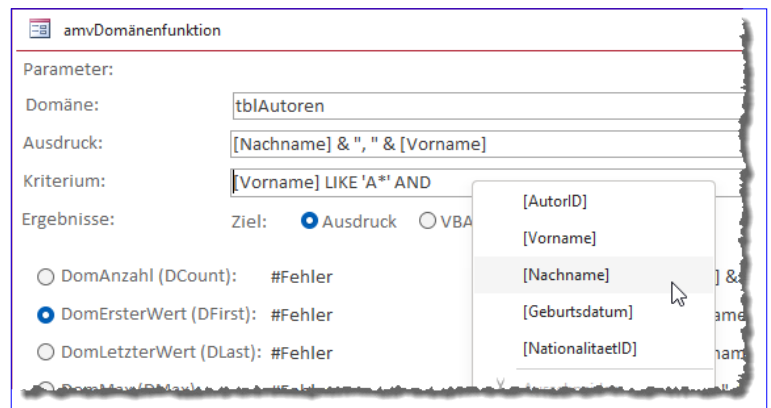


Bild 2: Auswahl der Felder per Kontextmenü

Der untere Bereich enthält jeweils eine Zeile für alle zwölf verfügbaren Domänenfunktionen. Hier sehen wir zuerst die deutsche und in Klammern die englische Bezeichnung, daneben folgt das Ergebnis für die Domänenfunktion mit den oben angegebenen Parametern.

Rechts davon finden wir den aktuellen Ausdruck für den Aufruf dieser Domänenfunktion, der die oben angegebenen Parameter berücksichtigt. Mit den Schaltflächen rechts daneben können wir die aktuelle Funktion in die Zwischenablage kopieren.

Da der Assistent die Domänenfunktionen in der deutschen Fassung (für die Benutzeroberfläche der deutschen Version von Access) sowie in der englischen Fassung (für VBA) bereitstellen soll, haben wir im oberen Bereich noch eine Option hinzugefügt, mit der wir wählen können, ob die

Funktion als Ausdruck oder als VBA-Funktion angezeigt werden soll.

### Programmierung des Formulars

In der Entwurfsansicht sieht das Formular wie in Bild 3 aus. Die oberen drei Steuerelemente heißen **cboDomae-**

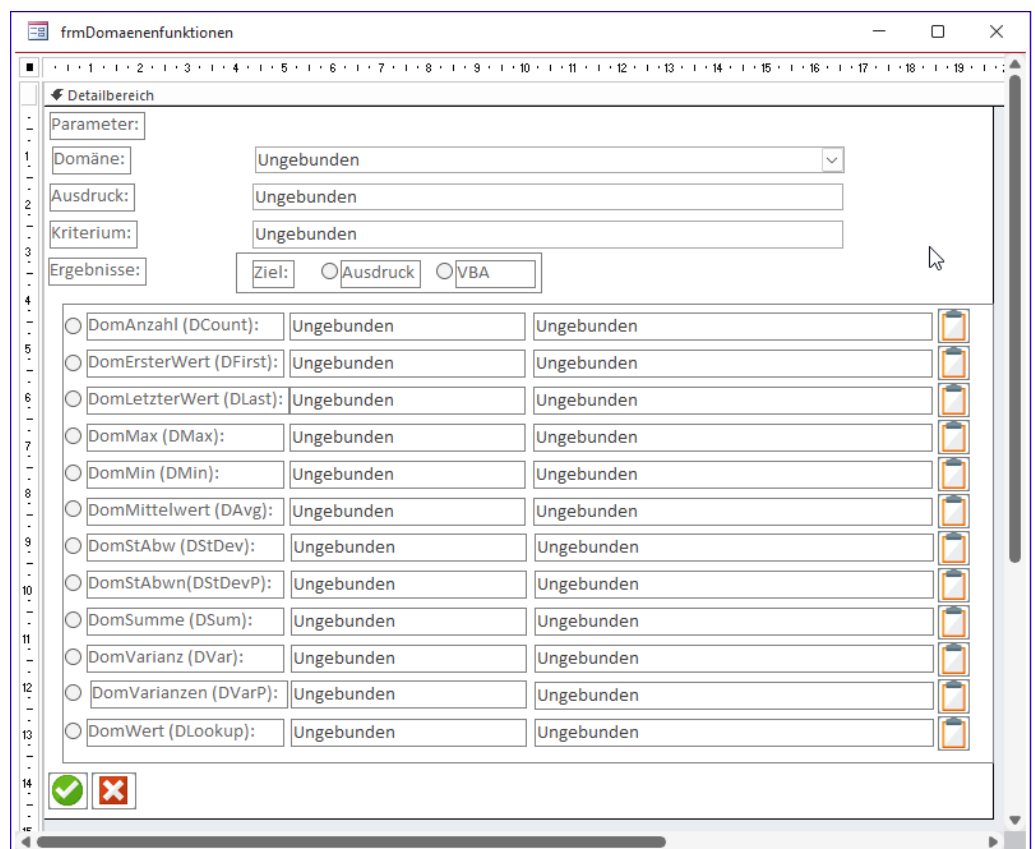


Bild 3: Entwurfsansicht des Formulars **frmDomaeenenfunktionen**

```
Private Sub Form_Load()
    Dim db As DAO.Database, rstDomaene As DAO.Recordset
    Dim strDomaenen As String, strAusdruck As String
    Dim bolHinzufuegen As Boolean
    Dim qdf As DAO.QueryDef, tdf As DAO.TableDef
    Set db = CurrentDb
    Set rstDomaene = db.OpenRecordset("SELECT Name, Type FROM MSysObjects WHERE Type IN (1,4,5,6) AND NOT Name " _
        & "LIKE 'f_*' AND NOT Name LIKE 'MSys*' AND NOT Name LIKE 'USys*' AND NOT Name LIKE '~*' ORDER BY Name", _
        dbOpenDynaset)
    Do While Not rstDomaene.EOF
        Select Case rstDomaene!Type
            Case 1, 4, 6
                bolHinzufuegen = True
            Case 5
                Set qdf = db.QueryDefs(rstDomaene!Name)
                On Error Resume Next
                If qdf.Parameters.Count = 0 Then
                    bolHinzufuegen = True
                Else
                    bolHinzufuegen = False
                End If
                If Not Err.Number = 0 Then
                    bolHinzufuegen = False
                End If
                On Error GoTo 0
            End Select
        End Select
        If bolHinzufuegen Then
            If Len(strAusdruck) = 0 Then
                Select Case rstDomaene!Type
                    Case 1, 4, 6
                        Set tdf = db.TableDefs(rstDomaene.Fields(0))
                        strAusdruck = tdf.Fields(0).Name
                    Case 5
                        Set qdf = db.QueryDefs(rstDomaene.Fields(0))
                        strAusdruck = qdf.Fields(0).Name
                End Select
            End If
            strDomaenen = strDomaenen & "'''' & rstDomaene!Name & ''';"
        End If
        rstDomaene.MoveNext
    Loop
    Me!cboDomaene.RowSource = strDomaenen
    Me!cboDomaene = Me!cboDomaene.ItemData(0)
    Me!txtAusdruck = "[" & strAusdruck & "]"
    Me!txtKriterium = ""
    Me!ogrZiel = 1
    UpdateControls Nz(Me!txtAusdruck, ""), Nz(Me!cboDomaene, ""), Nz(Me!txtKriterium, "")
End Sub
```

**Listing 1:** Prozedur, die beim Laden des Formulars aufgerufen wird

ne, **txtAusdruck** und **txtKriterium**. Das Kombinationsfeld **cboDomaene** wird beim Laden des Formulars durch das Ereignis aus Listing 1 gefüllt. Hier öffnen wir mit **OpenRecordset** ein Recordset auf Basis der Tabelle **MSysObjects**. Diese enthält alle Objekte einer Access-Datenbank. Wir benötigen nur die mit den Typen **1, 4, 5** und **6**. Die Typen **1, 4** und **6** liefern die verschiedenen Tabellenarten (lokale Tabelle, verknüpfte Tabelle und ODBC-Tabelle) und der Typ **5** die Abfragen. Außerdem wollen wir Systemtabellen und ähnliche Objekte ausschließen, die nicht vom Benutzer angelegt wurden und legen dazu entsprechende Kriterien fest.

Das Kombinationsfeld **cboDomaene** soll eine Wertliste als Datensatzherkunft erhalten, deshalb stellen wir die Eigenschaft **Herkunftsart** auf **Wertliste** ein. Um den Rest kümmern sich die folgenden Zeilen der Prozedur **Form\_Load**. Hier durchlaufen wir alle Datensätze des Recordsets. In einer **Select Case**-Bedingung prüfen wir den Typ, also ob es sich um eine Tabelle oder um eine Abfrage handelt. Warum ist das nötig? Weil es vorkommen kann, dass eine Abfrage Parameter enthält. In diesem Fall können wir nicht per Domänenfunktion darauf zugreifen, daher wollen wir solche Einträge direkt auslassen. Um festzulegen, ob eine Domäne Parameter enthält, verwenden wir die Variable **bolHinzufuegen**. Diese wird bei Tabellen immer auf **True** eingestellt. Bei einer Abfrage erstellt die Prozedur ein **QueryDef**-Objekt und weist diesem die aktuell untersuchte Abfrage zu. Für dieses können wir mit der Eigenschaft **Parameters.Count** ermitteln, ob die Abfrage Parameter enthält. Ist das nicht der Fall, stellen wir **bolHinzufuegen** auf **True** ein, sonst auf **False**.

Diesen Vorgang haben wir auch noch bei deaktivierter eingebauter Fehlerbehandlung ausgeführt, da dieser Zugriff bei einigen weiteren Abfragen zu Fehlern führte. Ist dies der Fall, stellen wir ebenfalls **bolHinzufuegen** auf **False** ein.

Hat **bolHinzufuegen** danach den Wert **True**, können wir die Tabelle oder Abfrage zur Liste der Domänen in **strDomaenen** hinzufügen. Der Variablen **strAusdruck** weisen

wir den Namen des ersten Feldes der Tabelle oder Abfrage zu, denn wir wollen das Kombinationsfeld mit dem ersten Eintrag vorbelegen und das Textfeld **txtAusdruck** mit dem ersten Feld dieser Tabelle oder Abfrage – so sieht der Benutzer direkt Beispiele für die anzugebenden Werte. Den Namen des ersten Feldes lesen wir wieder in einer **Select Case**-Bedingung ein, in der wir testen, ob es sich um eine Tabelle oder Abfrage handelt. Abhängig davon öffnen wir ein **TableDef**- oder **QueryDef**-Objekt, dem wir dann den Namen des ersten Feldes entnehmen.

Nach dem Durchlaufen aller Recordsets stellen wir die Eigenschaft **Datensatzherkunft (RowSource)** auf die Liste aus **strDomaenen** ein und setzen den Wert des Kombinationsfeldes auf den ersten verfügbaren Eintrag.

Dann füllen wir das Textfeld **txtAusdruck** mit dem Wert aus **strAusdruck**, also dem ersten Feld der gewählten Tabelle oder Abfrage. **txtKriterium** leeren wir hingegen. Schließlich stellen wir noch die Optionsgruppe zum Auswählen der auszugebenden Ausdrücke auf **1** ein, damit zunächst die deutschen Versionen der Domänenfunktionen erscheinen.

Danach folgt Arbeit für das Add-In – wir rufen die Prozedur **UpdateControls** auf, welche die übrigen Steuerelemente des Formulars aktualisiert. Dabei übergeben wir die Werte der drei Steuerelemente **cboDomaene**, **txtAusdruck** und **txtKriterien** als Parameter.

### Aktualisieren der Steuerelemente

Die Prozedur **UpdateControls** erwartet die Werte der drei Steuerelemente **cboDomaene**, **txtAusdruck** und **txtKriterien** als Parameter. Sie füllt zunächst die Textfelder mit den Ergebnissen der jeweiligen Domänenfunktion (siehe Listing 2). Dies erledigen wir jeweils unter Deaktivierung der eingebauten Fehlerbehandlung. Somit verhindern wir eine bei einem Fehler ausgegebene Fehlermeldung von Access selbst und können stattdessen einfach den Wert **#Fehler** in das jeweilige Textfeld schreiben – genau so, wie es auch bei einer als Ausdruck verwendeten Domä-



```
Private Sub UpdateControls(strAusdruck As String, strDomaene As String, strKriterium As String)
    On Error Resume Next
    Me!txtDCount = DCount(strAusdruck, strDomaene, strKriterium)
    If Not Err.Number = 0 Then
        Me!txtDCount = "#Fehler"
    End If
    ...
    Me!txtDLookup = DLookup(strAusdruck, strDomaene, strKriterium)
    If Not Err.Number = 0 Then
        Me!txtDLookup = "#Fehler"
    End If
    On Error GoTo 0
    Select Case ogrZiel
        Case 1
            Me!txtDCountCall = "=DomAnzahl(""" & strAusdruck & """;"" & strDomaene & _
                IIf(Len(strKriterium) = 0, "", """" & strKriterium & """)"
            Me!txtDFirstCall = "=DomErsterWert(""" & strAusdruck & """;"" & strDomaene & _
                IIf(Len(strKriterium) = 0, "", """" & strKriterium & """)"
            Me!txtDLastCall = "=DomLetzterWert(""" & strAusdruck & """;"" & strDomaene & _
                IIf(Len(strKriterium) = 0, "", """" & strKriterium & """)"
            Me!txtDMaxCall = "=DomMax(""" & strAusdruck & """;"" & strDomaene & _
                IIf(Len(strKriterium) = 0, "", """" & strKriterium & """)"
            Me!txtDMinCall = "=DomMin(""" & strAusdruck & """;"" & strDomaene & _
                IIf(Len(strKriterium) = 0, "", """" & strKriterium & """)"
            Me!txtDAvgCall = "=DomMittelwert(""" & strAusdruck & """;"" & strDomaene & _
                IIf(Len(strKriterium) = 0, "", """" & strKriterium & """)"
            ...
            Me!txtDLookupCall = "=DomWert(""" & strAusdruck & """;"" & strDomaene & _
                IIf(Len(strKriterium) = 0, "", """" & strKriterium & """)"
        Case 2
            Me!txtDCountCall = "DCount(""" & strAusdruck & """, "" & strDomaene & _
                IIf(Len(strKriterium) = 0, "", """" & strKriterium & """)"
            Me!txtDFirstCall = "DFirst(""" & strAusdruck & """, "" & strDomaene & _
                IIf(Len(strKriterium) = 0, "", """" & strKriterium & """)"
            Me!txtDLastCall = "DLast(""" & strAusdruck & """, "" & strDomaene & _
                IIf(Len(strKriterium) = 0, "", """" & strKriterium & """)"
            Me!txtDMaxCall = "DMax(""" & strAusdruck & """, "" & strDomaene & _
                IIf(Len(strKriterium) = 0, "", """" & strKriterium & """)"
            Me!txtDMinCall = "DMin(""" & strAusdruck & """, "" & strDomaene & _
                IIf(Len(strKriterium) = 0, "", """" & strKriterium & """)"
            Me!txtDAvgCall = "DAvg(""" & strAusdruck & """, "" & strDomaene & _
                IIf(Len(strKriterium) = 0, "", """" & strKriterium & """)"
            ...
            Me!txtDLookupCall = "DLookup(""" & strAusdruck & """, "" & strDomaene & _
                & IIf(Len(strKriterium) = 0, "", """" & strKriterium & """)"
    End Select
End Sub
```

**Listing 2:** Prozedur, welche die Parameter in Ergebnisse umwandelt

nenfunktion geschehen würde. Der Ablauf sieht für alle zwölf Domänenfunktionen ähnlich aus:

```
Me!txtDCount = DCount(strAusdruck,
strDomaene, strKriterium)
If Not Err.Number = 0 Then
    Me!txtDCount = "#Fehler"
End If
```

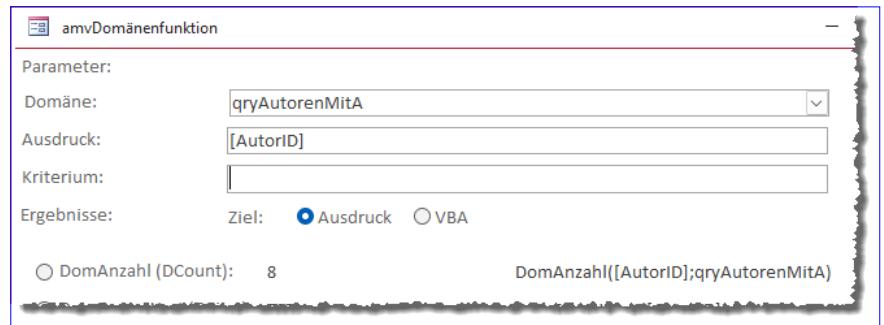


Bild 4: Anzeige nach dem Laden des Assistenten

Anschließend folgt eine **Select Case**-Bedingung, die prüft, ob der deutsche Ausdruck oder die englische VBA-Funktion ausgegeben werden soll. Der erste Teil weist jeweils dem rechten Textfeld die Funktion zu. Diese setzen wir aus dem Funktionsaufruf und den in Klammern eingefassten Parametern zusammen, die wir den drei Steuerelementen **cboDomaene**, **txtAusdruck** und **txtKriterium** entnehmen – hier am Beispiel der Funktion **DomAnzahl (DCount)**:

```
Me!txtDCountCall = "DomAnzahl(" & strAusdruck & ";" _
    & strDomaene & IIf(Len(strKriterium) = 0, "", ";") _
    & strKriterium) & ")"
```

Für die VBA-Funktion sieht der zusammengestellte Ausdruck nur wenig anders aus.

Die beiden Unterschiede sind die englische statt der deutschen Bezeichnung und die Verwendung des Kommas statt des Semikolons als Trennzeichen:

```
Me!txtDCountCall = "DCount(" & strAusdruck & "," _
    & strDomaene & IIf(Len(strKriterium) = 0, "", "," _
    & strKriterium) & ")"
```

Nach dem Laden sehen die Werte der drei oberen Steuerelemente wie in Bild 4 aus.

### Aktualisieren der Funktionen bei Eingabe

Nach dem Initialisieren zeigt das Formular also die erste Abfrage oder Tabelle nach alphabetischer Sortierung an

sowie die Ergebnisse der einzelnen Domänenfunktionen samt Definition.

Die Ergebnisse und Definitionen sollen nun direkt bei Änderung durch den Benutzer aktualisiert werden, unmittelbar nach der Eingabe eines jeden Zeichens. Dazu nutzen wir das Ereignis **Bei Änderung** der verschiedenen Steuerelemente.

Für das Textfeld **txtAusdruck** fügen wir folgende Prozedur hinzu, welche wieder die Prozedur **UpdateControls** aufruft. Um den aktuell im Textfeld angezeigten Text mit dem ersten Parameter zu übergeben, verwenden wir dessen **Text**-Eigenschaft. Anderenfalls würde nämlich nur der gespeicherte Wert übergeben, aber wir wollen ja den aktuell eingegebenen Wert:

```
Private Sub txtAusdruck_Change()
    UpdateControls Me!txtAusdruck.Text, _
        Nz(Me!cboDomaene, ""), Nz(Me!txtKriterium, "")
End Sub
```

Eine ähnliche Ereignisprozedur hinterlegen wir für das Textfeld **txtKriterium**. Der Unterschied ist, dass wir hier die **Text**-Eigenschaft von **txtKriterium** als dritten Wert übergeben und den Wert von **txtAusdruck** als ersten:

```
Private Sub txtKriterium_Change()
    UpdateCalls Nz(Me!txtAusdruck, ""), _
        Nz(Me!cboDomaene, ""), Me!txtKriterium.Text
End Sub
```

## Hier geht es weiter ...

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

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

Alle Informationen hier – jetzt bestellen und direkt loslesen!

**ZUM SHOP**



## Rechnungsverwaltung: Kundenadressen ausgliedern

In der bisherigen Version der Rechnungsverwaltung haben wir nur eine Kundenadresse in der Tabelle »tblKunden« gespeichert. Im Bestreben, diese Rechnungsverwaltung flexibler zu gestalten, wollen wir in diesem Beitrag zwei Dinge durchführen: Das Aufteilen der Kundentabelle in eine Tabelle mit den Basisdaten des Kunden und mehrere weitere Tabellen zum Speichern der Adressen dieses Kunden sowie das Anpassen des Formulars »frmKundenDetails« an diese Änderung des Datenmodells. Dies soll die Grundlage bilden, mehrere Adressen je Kunde zu speichern und diese dann nach Bedarf in Bestellungen, Lieferungen, Rechnungen und weitere Vorgänge zu übernehmen.

Das bisherige Datenmodell sah eine Tabelle namens **tblKunden** vor, die genau einen Satz von Adressdaten enthielt. Das wird den modernen Anforderungen nicht mehr gerecht, sodass wir unsere Rechnungsverwaltung, noch anpassen möchten, bevor wir die auf den Kundendaten aufbauenden Tabellen und Formulare für die Bestelldaten und die Rechnungs- und Lieferdaten erstellen.

Grundlage dafür sind einige Überlegungen aus dem Beitrag **Datenmodelle für die Rechnungsverwaltung** ([www.access-im-unternehmen.de/1459](http://www.access-im-unternehmen.de/1459)), die wir hier nicht genau, aber ansatzweise übernehmen. Im bisherigen Datenmodell sah die Tabelle unserer Beispiellösung wie in Bild 1 aus. Hier haben wir noch nicht einmal verschiedene Adressen für die Rechnung und die Lieferung gepflegt – es war eben ein recht einfaches Datenmodell.

### Aufteilung der Kundendaten auf mehrere Tabellen

Nun fügen wir zwei Tabellen zum Verwalten von Rechnungsadressen und Lieferadressen hinzu,

die wir über zwei Fremdschlüsselfelder mit der Tabelle **tblKunden** verbinden. Aus der Tabelle **tblKunden** entfernen wir einige Felder, die hier nicht mehr gebraucht werden. Welche Daten man in dieser Tabelle noch benötigt, ist ohnehin wie vieles andere vom Anwendungsfall abhängig. In unserem Fall behalten wir die Felder **Kundennummer**, **Firma**, **Vorname**, **Nachname**, **E-Mail** und **Telefon** in dieser Tabelle.

Außerdem legen wir zwei neue Tabellen namens **tblLieferadressen\_Kunde** und **tblRechnungsadressen\_Kunde** an. Diese erhalten jeweils ein Fremdschlüsselfeld namens **KundeID**, mit dem wir die Beziehung der jeweiligen Adresse mit dem Kundendatensatz aus der Tabelle **tblKunden** herstellen können. Außerdem enthalten diese

tblKunden		
Feldname	Felddatentyp	Beschreibung (optional)
ID	AutoWert	Primärschlüsselfeld der Tabelle
Kundennummer	Kurzer Text	Individuelle Kundennummer
Firma	Kurzer Text	Firma des Kunden
AnredeID	Zahl	Fremdschlüsselfeld zur Tabelle tblAnreden
Vorname	Kurzer Text	Vorname des Kunden
Nachname	Kurzer Text	Nachname des Kunden
Strasse	Kurzer Text	Strasse des Kunden
PLZ	Kurzer Text	PLZ des Kunden
Ort	Kurzer Text	Ort des Kunden
LandID	Zahl	Fremdschlüsselfeld zur Tabelle tblLaender
E-Mail	Kurzer Text	E-Mail des Kunden
UstIDNr	Kurzer Text	Umsatzsteuer-Identifikationsnummer des Kunden

Bild 1: Ausgangstabelle



Tabellen die üblichen Adressfelder, aber auch nochmal **Firma**, **AnredeID**, **Vorname** und **Nachname**.

Die Tabelle **tblRechnungsadressen\_Kunde** enthält außerdem noch das Feld **UstIDNr**. Warum hier und nicht in der Tabelle **tblKunden**? Weil es durchaus sein kann, dass ein Kunde verschiedene Firmen mit unterschiedlichen Umsatzsteuer-Identifikationsnummern hat, für die er Rechnungen benötigt.

### Standardadresse für Rechnung und Lieferung festlegen

Die Tabelle **tblRechnungsadressen\_Kunde** sieht in der Entwurfsansicht wie in Bild 2 aus. Ein weiteres wichtiges Feld neben den genannten ist das Feld **Standardadresse**. Es ist ein **Ja/Nein**-Feld, mit dem wir für jeden Kunden eine Standardlieferadresse und eine Standardrechnungsadresse festlegen können.

Die Pflege dieses Feldes erfordert Aufwand, den wir nicht über das Datenmodell erledigen können: Wir müssen sicherstellen, dass bei mehreren Adressen je Kunde immer genau eine Adresse den Wert **True** in diesem Feld aufweist. Das erledigen wir später beim Programmieren der Benutzeroberfläche.

Das Datenmodell für die Kundenadressen sieht nun wie in Bild 3 aus. Die beiden Tabellen **tblLieferadressen\_Kunde** und **tblRechnungsadressen\_Kunde** sind über Fremdschlüsselfelder mit der Tabelle **tblKunden** ver-

Feldname	Felddatentyp	Beschreibung (optional)
ID	AutoWert	Primärschlüsselfeld der Tabelle
Firma	Kurzer Text	Firma des Kunden
AnredeID	Zahl	Fremdschlüsselfeld zur Tabelle tblAnreden
Vorname	Kurzer Text	Vorname des Kunden
Nachname	Kurzer Text	Nachname des Kunden
Strasse	Kurzer Text	Strasse des Kunden
PLZ	Kurzer Text	PLZ des Kunden
Ort	Kurzer Text	Ort des Kunden
EMail	Kurzer Text	EMail des Kunden
Telefon	Kurzer Text	Telefon des Kunden
Standardadresse	Ja/Nein	Gibt an, ob dies die Standardadresse des Kunden ist.
KundeID	Zahl	Fremdschlüsselfeld zur Tabelle tblKunden
UstIDNr	Kurzer Text	Umsatzsteuer-Identifikationsnummer des Kunden

Bild 2: Die Tabelle für die Rechnungsadressen der Kunden

knüpft. Auf diese Weise können wir nun mehrere Adressen für Rechnungen und Lieferungen für jeden Kunden anlegen und jeweils eine als Standardadresse definieren.

### Formulare für die Kunden und ihre Adressen

Nun benötigen wir ein Formular zur Darstellung der Sachverhalte in diesen Tabellen. Ein Teil scheint offensichtlich zu sein: Ein Hauptformular wird an die Tabelle **tblKunden** gebunden und zeigt die Daten dieser Tabelle an.

Für die Rechnungs- und Lieferadressen gibt es jedoch sehr viele Möglichkeiten zur Darstellung, von denen viele nicht ergonomisch sind. Was genau wollen wir in dem Formular erledigen?

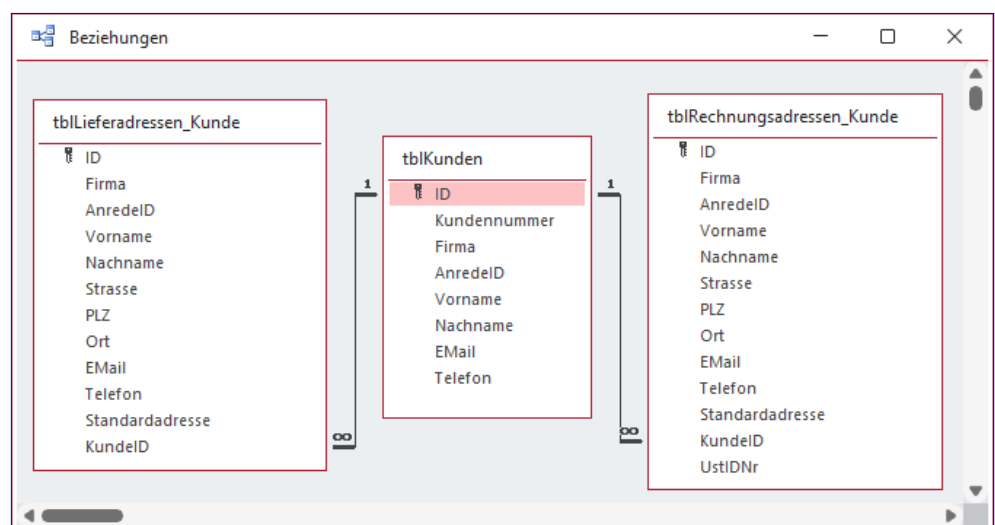


Bild 3: Die Kundendaten liegen nun in drei verschiedenen Tabellen.

- Übersichten der Rechnungsadressen und Lieferadressen anzeigen
- Details jeweils einer Rechnungs- und Lieferadresse anzeigen
- Wechsel zu den übrigen Detaildatensätzen für Rechnungen und Lieferungen auswählbar machen
- Neue Rechnungs- und Lieferadressen anlegen
- Bestehende Adressen bearbeiten
- Jeweils eine der Adressen als Standardadresse für Rechnungen und Lieferungen einstellen

The screenshot shows a software window titled 'frmKundeDetails'. It contains a form for customer details. At the top, it displays 'Kundennummer: 99000001'. Below this, there are fields for 'Firma: Minhorst und Minhorst GbR', 'Anrede: Herr', 'E-Mail: andre@minhorst.com', 'Vorname: André', 'Telefon: 0203-4495577', and 'Nachname: Minhorst'. There are two tabs: 'Lieferadresse' (selected) and 'Rechnungsadresse'. Under 'Lieferadresse', there is a dropdown 'Adresse auswählen:' with 'Minhorst und Minhorst GbR (Standard)' selected. Below this is a sub-form with fields for 'Firma: Minhorst und Minhorst GbR', 'AnredeID: Herr', 'PLZ: 47137', 'Vorname: André', 'Ort: Duisburg', 'Nachname: Minhorst', 'E-Mail: buchhaltung@minhorst.com', 'Strasse: Borkhofer Str. 17', and 'Telefon: 0203-4495577'. A checkbox 'Standardadresse' is checked. At the bottom of the form are buttons 'Adresse löschen' and 'Neue Adresse'. Below the form is a table 'Bestellungen:' with columns: 'BestellNr', 'Bestellt am', 'Rechnung am', 'Zahlungsziel', 'Bezahlt am', and 'Storniert am'. The table contains three rows of data. At the bottom of the window are buttons 'OK', 'Bestellung anzeigen', 'Bestellung löschen', and 'Neue Bestellung'.

BestellNr	Bestellt am	Rechnung am	Zahlungsziel	Bezahlt am	Storniert am
11000062	21.04.2022	27.04.2022	18.05.2022	04.05.2022	
11000219	01.07.2021	10.07.2021	31.07.2021	11.07.2021	
99999999	01.06.2022				

**Bild 4:** So sollen die Kundendetails im Formular angezeigt werden

- Standardadresse für Rechnungen und Lieferungen für neue Bestellungen übernehmen
- Rechnungs- und Lieferadressen löschen

Rechnungs- und Lieferadressen löschen? Aber brauchen wir die nicht dauerhaft? Nicht mit dem Modell, das wir auf den hier genannten Tabellen aufbauen. Erstellen wir eine Bestellung, werden die aktuellen Standardadressen für Rechnung und Lieferung automatisch in weitere Tabellen übernommen, die wiederum mit der Bestellung verknüpft sind.

Dementsprechend können wir die Adressen für Rechnungen und Lieferungen für einen Kunden löschen, denn

diese befinden sich ja in den Datensätzen für die Bestellungen, Rechnungen und Lieferungen.

### Umsetzung des Kundendetail-Formulars

Wie aber können wir das Formular so gestalten, dass die obigen Anforderungen alle erfüllt werden können? Dies erledigen wir wie in Bild 4. Im Gegensatz zu der Version, die wir im Beitrag **Rechnungsverwaltung: Kundendetails** ([www.access-im-unternehmen.de/1383](http://www.access-im-unternehmen.de/1383)) vorgestellt haben, haben wir die Adressdaten aus dem Hauptformular herausgenommen und stellen diese nun in zwei Unterformularen dar.

Diese beiden fügen wir wiederum auf zwei Seiten eines Registerstueurelements ein, sodass diese nicht gleich-

zeitig angezeigt werden müssen.

Diese Unterformulare zeigen alle Liefer- und Rechnungsadressen zum Kunden aus dem Hauptformular an. Wir können über die Navigationsschaltflächen durch diese Datensätze navigieren, aber auch eine der Adressen über das Kombinationsfeld oberhalb des Unterformulars auswählen. Außerdem wollen wir ermöglichen, eine neue Adresse durch die Auswahl des Eintrags **Neu...** zu erzeugen (siehe Bild 5).

Bild 5: Neuanlage einer Adresse per Auswahlfeld

Bild 6: Entwurf des Unterformulars für die Lieferadressen

Schauen wir uns nun an, wie wir dies auf Basis der vorhandenen Tabellen im Formular abbilden.

Bild 7: Hinzufügen eines Moduls zum Unterformular

### Unterformular für die Lieferadressen

Das erste Unterformular namens **sfmKunde\_Lieferadressen** stellen wir mit der Tabelle **tblLieferadressen\_Kunde** als Datensatzquelle aus. Wir ziehen alle Felder aus der Feldliste in den Detailbereich und ordnen die Felder wie in Bild 6 an.

Es gibt einige Ereignisse im Unterformular, auf die wir reagieren wollen und die sich meist auf die Aktualisierung von Steuerelementen auswirken. Um die Ereignisse zur Steuerung des gesamten Formulars samt Unterformularen im Hauptformular zu belassen, können wir im dortigen Klassenmodul Objektvariablen deklarieren, die wir mit Verweisen auf die Unterformulare füllen.

Dadurch können wir die Ereignisse der Unterformulare im Klassenmodul des Hauptformulars implementieren und haben den kompletten Code in einem Modul. Im Unterformular müssen wir dazu nur einen Schritt durchführen,

nämlich die Eigenschaft **Enthält Modul** auf **Ja** einzustellen (siehe Bild 7).

Damit sind die Arbeiten am Unterformular für die Lieferadressen bereits abgeschlossen und wir können uns den Rechnungsadressen zuwenden.

### Unterformular für die Rechnungsadressen

Bei diesem Unterformular bestehen die gleichen Anforderungen wie beim Unterformular für die Lieferadressen. Allerdings fügen wir hier die Tabelle **tblRechnungsadressen** als Datensatzquelle hinzu und speichern das Formular unter dem Namen **sfmKunden\_Rechnungsadressen**.

Bild 8: Entwurf des Unterformulars für die Rechnungsadressen

Bild 9: Erste Seite des Registersteuerelements

Warum können wir nicht einfach nur ein solches Unterformular verwenden und per Code zur Laufzeit die Datensatzquelle anpassen? Weil die Tabelle mit den Rechnungsadressen ein zusätzliches Feld namens **UstIDNr** enthält, das wir ebenfalls abbilden müssen (siehe Bild 8). Auch dieses Formulars enthält keinen eigenen Code, aber wir wollen die Eigenschaft **Enthält Modul** aktivieren, damit wir im Hauptformular die Ereignisse für dieses Unterformular implementieren können.

### Registersteuerelement für die Unterformulare

Im Hauptformular legen wir nun ein Registersteuerelement namens **regAdressen** an. Dieses soll zwei Registerseiten mit den Beschriftungen **Lieferadresse** und **Rechnungsadresse** aufweisen. Sie haben den Seitenindex **0** und **1**.

Um der ersten Registerseite das Unterformular für die Lieferadressen zuzuweisen, markieren wir diese Seite durch einen Klick auf den Registerreiter. Es sollte nun ein Rahmen unterhalb des Registerreiters erscheinen. In diesen Bereich können wir nun das Formular **frm\_KundeLieferadressenaus** dem Navigationsbereich hineinziehen. Wenn sich beim Hineinziehen der Hintergrund dieser Registerseite schwarz färbt, können wir das Unterformular fallen lassen. Nun fügen wir den Registerseiten noch weitere Steuerelemente hinzu, aber auf unterschiedliche Weise:

- Im oberen Bereich legen wir ein Kombinationsfeld an, mit dem wir die Auswahl einer der Lieferadressen ermöglichen wollen. Dieses soll ebenfalls auf der Registerseite abgelegt werden und den Namen **cboRech-**



## Hier geht es weiter ...

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

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

Alle Informationen hier – jetzt bestellen und direkt loslesen!

**ZUM SHOP**

